Application of Statistical Pattern Recognition To Document Segmentation and Labelling

by

Kevin Laven

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Application of Statistical Pattern Recognition To Document Segmentation and Labelling

Kevin Laven

Master of Science

Graduate Department of Computer Science

University of Toronto

2005

In the field of computer analysis of document images, the problems of physical and logical layout analysis have been approached through a variety of heuristic, rule-based, and grammar-based techniques. In this paper we investigate the effectiveness of statistical pattern recognition algorithms for solving these two problems. Using a new software environment for manual page image segmentation and labelling, a dataset containing 932 page images from academic journals has been created. Several physical layout analysis algorithms have been implemented, including a new algorithm based on a logistic regression classifier. Three statistical classifiers were applied to the logical layout analysis problem, with encouraging results. A new model for how ink is laid out on a page was used to develop a prototype combined segmentation and labeling system. Finally, several applications have been investiaged, and rudimentary implementations demonstrated. Results indicate that statistical pattern recognition approaches to these problems will be very fruitful.

Acknowledgements

I would first like to thank my supervisor Sam Roweis for sharing his wealth of knowledge and enthusiasm for machine learning with me, as well as for his assistance finding focus and direction in this research.

In addition, I would like to thank the rest of the Machine Learning Group at the University of Toronto for their suggestions and criticisms, as well as their assistance in learning the tools necessary to perform this research. I would like to thank Liam, Kier, and Rama for helping me learn to work with Linux, Matlab, and LaTex. I would also like to thank Ben and Khash for their help in understanding the statistical models and techniques used in this thesis. Finally, I would like to thank Rich Zemel for acting as second reader for this thesis.

A tremendous amount of effort went into preparing the software environment and ground-truth data for this thesis. My thanks goes to Scott Leishman for putting in this effort, and again to Sam Roweis for providing me with the opportunity to contribute to this project. I wish Scott and Sam great success in their plans to continue this line of research.

On a personal note, I would like to thank my friends, family, and co-workers for supporting me through this effort. Thanks to Tracey, Ofer, Stephanie, Richard, and Natalie for encouraging me to seek the healthy distractions of sports and exercise, as well as to Neil, Chris, Rob, Marina, and Lindsay for providing me with ample opportunity for relaxation. Thanks also to The Pressure Pipe Inspection Company for providing me with part-time employment, and to Mohan for making this arrangement work out. Most of all, my deepest thanks to Tracey for all her support and encouragement throughout my pursuit of this degree.

Contents

Chapter 1

Introduction

With the advent and growth of electronic sources of information in the past decades, a debate has emerged as to the relative merits of paper versus electronic publication. Electronic publication means making a paper primarily available in a text-searchable electronic form. Electronic publication has several disadvantages, such as loss of archival stability, and reduction of the ability of the author and publisher to control and charge for the distribution of the materials. On the other hand, electronic publishing offers a wide variety of advantages over paper. These include a lower cost of publication [21], a wider variety of methods of distribution, and the added content-value derived from the ability to hyperlink to other documents.

One particular area of interest for this debate is the publication of academic journals. Unlike many published materials, the main goal of the authors of papers in academic journals is rarely to earn the maximum possible revenue from the publication. More often, the author's primary goal is to maximize the impact of their research on its field. One measure of the impact of a paper is the number of citations it receives. Steve Lawrence has shown that the most-cited papers of the past several years have been published electronically, and makes a convincing case that electronic publication increases the impact of a paper, as measured by number of citations [13].

Many of these advantages held by electronic publications are growing more significant over time, leading both economists and academics to ask not if it will come to pass that all scholarly journals are published electronically, but when [22], and how. In fact, Robert Cameron [2] argues that the added value of cheap distribution and linking between documents implies that in the near future, not only will all scholarly journals every published be available electronically, but a full database of citations between them will be compiled.

One great difficultly in achieving this goal is the fact that many older papers currently exist only on paper. Scanned images of the pages achieve some of the benefits of electronic versions, including electronic distribution and allowing other documents to link to them. Scanned images of the page, however, do not make the paper text-searchable (either in terms of searching for the paper, or searching within it), do not allow for effective compression of the stored information, and do not allow for easy linking to documents cited by the paper.

These advantages can all be gained through manual effort. Locating the text regions, labeling them according do what type of text they are (for example, identifying references so that links can be made to the referenced document), and then converting them to electronic text, provides a text-searchable electronic copy of the original paper, and all the benefits that come with it.

Unfortunately, these manual tasks are time-consuming and expensive. The JSTOR project[1] is a large project whose goal is to create full electronic copies of paper journals. Odlyzko cites their costs as of 1996 as being between $1 and $2 per page [20], despite the use of OCR systems.

The challenge of extracting this information from page images is addressed by the field of Document Understanding. This field can be subdivided into three major functions. *Page segmentation* is the process of dividing a page into regions of ink that should be considered together. This problem is also know as *page physical structure analysis*, as well as *layout analysis*. *Labeling* is the process of assigning labels to each region of ink that describe what it

[1]http://www.jstor.org

is, such as a block of text, a graph, or an equation. This step is alternately referred to as *page logical structure analysis*, *region classification*, and *block classification*. Finally, encoding is the process of taking each labeled region and converting it to the most appropriate electronic representation. For example, text regions can be encoded as electronic text using an optical character recognition system. These steps need not be performed in this order, or need they all be performed separately, but they all need to be done in some manner to reach an ideal representation of the paper.

This thesis addresses the problems of page segmentation and labeling in the domain of academic journals. Several existing methods for segmentation are evaluated in this domain. These methods are generalized and extended using machine learning techniques. Machine learning techniques are also applied to the labeling problem. Finally, a new model for combined segmentation and classification is proposed and tested.

1.1 Motivation

As optical character recognition is by far the most well-known branch of document understanding, some explanation of the motivation for page segmentation and labeling is necessary.

While optical character recognition provides the actual text information, the results can be improved by identifying individual regions of text, rather than simply applying optical character recognition to the entire page. One reason for this is the presence of non-text regions on the page. Some optical character recognition systems will attempt to match characters to these regions, adding large amounts of garbled text to the results. Even more problematic, some OCR systems will attempt to adapt their font models to fit the ink they find on the page. A second problem is raised by layouts such as multi-column pages in which text appears in horizontal rows, but does not belong together. This can result in OCR results that, while they are actual text, no longer reflect the content of the original document.

A second independent benefit of segmentation is that some applications require contextual information for the text as well as the text itself. An example of this would be allowing detailed searches of journals, such as searching for all articles with the phrase "segmentation errors" in a figure caption. In most cases, applications attempt to reconstruct this contextual information from the text after OCR is applied. An alternative approach is to use the segmentation and labeling information to extract this information directly.

Additionally, some applications can be best performed using the segmentation and labeling information alone. An example of this is re-flowing a document from single-column to double-column layout, as discussed in Chapter 7.2. This can be done by using OCR to convert the text-images to text, and then creating a new layout for this text, but this is likely to introduce errors from the OCR process. With the segmentation and labeling information alone, however, blocks of text can be identified, the individual word images can be found, and a new layout created without ever using OCR.

Chapter 2

Background

The focus of this thesis is the application of Machine Learning techniques to the problems of document segmentation and labeling. As such, an understanding of these techniques is necessary. This chapter provides an explanation of the machine learning classification techniques used. An understanding of the principles of probability and statistics is assumed. Classification is the process of automatically assigning an item (called a sample) to one of several pre-defined classes. It can equivalently be thought of as applying one of several pre-defined labels to a sample.

2.1 K Nearest Neighbors

The K Nearest Neighbors (KNN) classification technique uses a simple principle: when presented with a new test sample, compare it to all of the samples for which we know the correct class, and assign to it the same class as those to which it is most similar.

A K Nearest Neighbors classifier, like all the other classifiers we will describe, sees each sample as a vector of numbers, which are referred to as features. A feature may be an aspect of the sample itself (such as the height and width of an ink region), or may be the result of functions applied to the sample (such as the pixel density of the ink region).

Using a vector of features (called a feature vector) to describe each sample lends itself to a geometric interpretation. A vector with N elements is equivalent to a point in an N-dimensional space, with each element of the vector describing the point's position in one dimension of the space. The range of all possible values of the features is known as the *feature space* of the problem.

The distance between two samples represented by feature vectors x_1 and x_2 in this N-dimensional feature space can be measured by any distance function. One popular choice is the Squared Euclidean Distance formula, where i is an index over the dimensions, and $x_{1,i}$ represents the i'th element of the feature vector x_1:

$$distance(x_1, x_2) = \sum_i (x_{1,i} - x_{2,i})^2 \tag{2.1}$$

The nearest neighbors of a given point can be found by comparing the distances between one point and each of the others, and selecting those with the smallest distance measures.

The K Nearest Neighbors algorithm begins with a collection of samples for which the correct class is known, called the training data. When a new sample is presented, this sample's class is predicted by examining the K nearest neighbors of that new sample. Whichever label appears the most frequently among those K neighbors is the label that is predicted for the new sample. In the case of a tie, the value of K can be temporarily increased or decreased, examining additional (or fewer) neighbors one by one until the tie is broken.

Another way to think of the K Nearest Neighbors algorithm is in terms of the training samples in the "neighborhood" of a new sample. The new point is placed in the feature-space, and a hypersphere is drawn around this point. This hypersphere defines the neighborhood of the new sample, and all the training points within it are the new sample's neighbors. The hypersphere is expanded until it contains exactly K neighbors, and the training sample is assigned the same class as the largest number of its neighbors hold. In the case of a tie, the hypersphere can be either expanded or contracted until the tie is broken. This view of the algorithm leads to a related one, in which a hypersphere of fixed radius is used, rather than a fixed value of K.

One challenge in this algorithm is selecting an appropriate value of K. A set of training samples, along with a value of K, defines areas in the feature-space where a sample will be assigned each class label. With larger values of K, these areas in the feature space tend to be "smoother", with the occasional errant training sample not affecting the areas very much. Larger values of K can be said to reduce the variance of the system. However, as values of K get large, the classes with large numbers and high densities of training samples will tend to push their borders outward, enveloping areas that they should not. This tends to bias the classifier in favor of these classes with large numbers of training samples. Choosing an appropriate value of K is thus an example of a bias-variance trade off.

Results can often be improved by using a more accurate measure of distance. The simple Euclidean distance does not account for the fact that some features will vary much more than others. For example, two features used in attempting to classify regions of ink on a page were the region's pixel density, which vary between 0 and 1, and its area (in pixels), which was often over 10,000. If simple Euclidean distance were used, any differences in pixel density would be unnoticeable when added to the differences in area. In order to account for these different degrees of variance, each feature can be divided by the variance of the sample data along that dimension when calculating distance, giving the following improved distance formula:

$$Variance(i) = \frac{1}{N} \sum_{n} (z_i^n - \overline{z_i})^2 \tag{2.2}$$

$$distance(x, z) = \sum_{i} \left(\frac{(x_i - z_i)^2}{Variance(i)} \right) \tag{2.3}$$

Another method of improving the distance measure is to find some estimate of how effectively each dimension differentiates between the various classes. These estimates can be used to weight the distances along each dimension.

2.2 Gaussian Naive Bayes

The Gaussian Naive Bayes classifier is a simple generative classifier. At training time, it learns a joint distribution of the training samples, represented by X, and their correct labels, represented by Y, by factorizing the joint distribution as shown in Equation 2.4, and finding that parameter values θ that maximize this for the training data.

$$P(X, Y) = P(X \mid Y)\, P(Y) \tag{2.4}$$

It uses a Gaussian distribution to model $P(X \mid Y)$, and simply counts the number of labels of each type in the training set to model $P(y)$. At test time, it applies Bayes rule to find the most likely label assignments Y given the features X and model parameters θ, as shown in Equation 2.5.

$$P(Y|X, \theta) = \frac{P(X \mid Y, \theta)\, P(Y \mid \theta)}{P(X \mid \theta)} \tag{2.5}$$

Gaussian classifiers assume that the class-conditional distribution (that is, the distribution for each group when separated by class) of the samples along each feature (in other words, in each dimension of the feature-space) is a Normal distribution, which can be described by a Gaussian curve. Since a Gaussian distribution is fully described by just its mean vector and its covariance matrix, Gaussian classifiers use the collection of samples for which the correct label is known (the training data) to build a prototype for each label. Labels can be assigned to new samples based on their distance from these prototypes.

If x is an m-dimensional feature vector representing a sample, and μ_y and Σ_y are the mean and covariance matrix respectively of the prototype for label y, then the estimate of the likelihood that y will generate x is:

$$P(x|y) = \frac{1}{(2\pi)^{\frac{m}{2}} \|\Sigma\|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_y)^T \Sigma_y^{-1} (x-\mu_y)} \tag{2.6}$$

Restrictions are often placed on the covariance matrix to simplify the learning process and improve the generalization of the model. A popular such restriction is to make the *naive* assumption that each feature is independent of all the others, given the class of the sample, forcing the covariance matrix to be diagonal:

$$\Sigma_y = \begin{bmatrix} \sigma_{y,1} & & & \\ & \sigma_{y,2} & & \\ & & \cdots & \\ & & & \sigma_{y,m} \end{bmatrix}$$

Under the naive assumption, if x_i represents the i'th dimension of the feature vector x, and $\mu_{y,i}$ the i'th dimension of the mean for class y, then the likelihood that a given prototype y will generate a sample x is given by:

$$P\left(x \mid y\right) = \prod_i P\left(x_i \mid y_i\right)$$

$$P\left(x \mid y\right) = \prod_i \frac{1}{\sqrt{2\pi\sigma_{y,i}^2}} e^{\left(-\frac{(x_i-\mu_{y,i})^2}{2\sigma_{y,i}^2}\right)} \tag{2.7}$$

The Gaussian Naive Bayes classifier is a generative classifier. Generative classifiers learn a joint distribution over the samples and their labels for a training set, and then apply Bayes Rule to estimate the likelihood of the label given the sample at test time.

In the case of a Gaussian Naive Bayes classifier, learning the joint distribution means finding the means $\mu_y$and variances σ_y for each label y that maximize the product of the likelihoods of all the training samples. Let X represent a matrix of training samples, and Y represent a vector of their correct labels. Let x^n represent feature vector of the n'th sample, and y^n represent the label assigned to the n'th sample. If the samples are independent and identically distributed (iid), then the likelihood of the entire training set is the product of the likelhoods of its members. Thus the Gaussian Naive Bayes classifier assumes each pair

(x^n, y^n) is iid allows, and aims to find the values of μ_y and σ_y that maximize:

$$P(X,Y) = \prod_n P(x^n, y^n)$$

Where

$$P\left(x^n, y^n\right) = P\left(x^n \mid y^n\right) P\left(y^n\right)$$

The first term $P\left(x^n \mid y^n\right)$ expands as shown in equation 2.7. Conveniently, the values of μ_y and σ_y that maximize this joint likelihood of the training samples are simply the mean and standard deviation of the training samples with label y^n.

The second term represents an estimate of how likely it is that a randomly chosen sample would have the same label as y^n. The simplest way to estimate this is by using the fraction of the training samples that have the same label as y^n. To improve generalization, smoothing can be added to the second term. A common way to accomplish this is to add some constant γ to the number of training samples with each label.

At test time, the Gaussian Naive Bayes classifier finds the label y that maximizes the likelihood that the prototype for y would produced the sample x. This likelihood is expressed mathematically as $P\left(y \mid x\right)$, and is calculated using Bayes Rule:

$$P\left(y|x\right) = \frac{P\left(x \mid y\right) P\left(y\right)}{P\left(x\right)}$$

If j is an index over the possible values of y (making y_j the j'th possible label), then $P(x)$ can be expressed as the sum $P(x \mid y_j)$ for all possible values of j:

$$P(x) = \sum_j P\left(x \mid y_j\right)$$

Where $P(x \mid y_y)$ expands as shown above. However, since the goal of a classifier is generally to find the value of y that maximizes $P(y \mid x)$, and since the value of $P(x)$ is independent of which label is assigned to y, this term can generally be ignored.

2.3 Logistic Regression

Discriminative classifiers do not attempt to model the joint distribution $P(X,Y)$; they simply attempt to learn the posterior distribution $P(Y \mid X)$ directly. These classifiers are often more effective than generative classifiers, especially when large amounts of training data are available, although they can be more difficult to train [19].

Logistic Regression classifiers are a simple yet effective form of discriminative classifier. Let $\omega_y = [\omega_{y,0}, \omega_{y,1}, \ldots, \omega_{y,m}]$ be a vector of m+1 dimensions corresponding to the label y, and let the feature vector x be augmented with a constant value 1, such that $x = [1, x_1, \ldots, x_m]$. Logistic Regression assumes that the posterior distribution of the likelihood can be represented in the following form, where j is an index over all possible labels:

$$P(y \mid x) = \frac{e^{\omega_y^T x}}{\sum_j e^{\omega_j^T x}} \tag{2.8}$$

Logistic regression makes use of the fact that $\log(a) > \log(b)$ iff $a > b$, and the fact that we are always attempting to maximize the function $P(y \mid x)$. Instead of trying to maximize $P(Y \mid X)$ directly, we maximize the function $l(Y \mid X) = \log(P(Y \mid X))$, which also provides the solution for which $P(Y \mid X)$ is maximized. At training time, the logistic regression classifier searches for the values of ω_y that maximize the product of the likelihoods over the matrix of training samples X, with the known labels Y. As was the case with the Gaussian Naive Bayes classifier, the logistic regression classifier assumes the samples are iid, making the total likelihood of the training set the product of the likelihoods of the individual members. Since maximizing $P(Y \mid X)$ is equivalent to maximizing $l(Y \mid X)$, the classifier maximizes the sum of the log likelihoods of the n samples in the training data:

$$l\left(Y \mid X\right) = \sum_{n} l\left(y^n \mid x^n\right) \tag{2.9}$$

We can take the log of Equation 2.8 to expand the term $l\left(y^n \mid x^n\right)$, yielding Equation 2.10. Since Equation 2.9 is differentiable with respect to the ω_y values, any number of optimization algorithms are available for finding the optimal values.

$$l\left(y^n \mid x^n\right) = \omega_{y^n}^T x^n - \log\left(\sum_j e^{\omega_j^T x^n}\right) \tag{2.10}$$

At test time, finding the best label to assign to a new sample x is simply a matter of calculating $l\left(y \mid x\right)$ for each possible label y. The maximum likelihood label is the one then assigned to x.

2.4 Label Dependence

All of the classifiers discussed thus far assume that each new sample is independent of the others. For the problem being studied in this thesis, however, this assumption is not valid. The locations of a region on a page, and its associated label, is related to the locations and labels of other regions on a page. As a simple example, consider a single line of text, left-aligned, that stretches about 2/3 of the way across the page. If this is placed directly below a figure, then it is obviously a figure caption. If it is instead below a text paragraph, it is more likely to be a section heading.

Considering the items as an ordered sequence allows us to take advantage of the wide variety of classifiers which take the sequence of labels into account. One popular group of such classifiers is called Markov Models. These are so named because they adhere to the Markov Assumption, which states that all of the prior information useful in predicting the label of a given sample is contained in some finite number k of most recent labels [1]:

[1]The notation $y_{a:b}$ represents the sequence of labels from y_a through y_b

$$P\left(y_n|x_n, y_{1:n-1}\right) = P\left(y_n|x_n, y_{n-k:n-1}\right)$$

The value of k used for a given model is called the Markov Order of the model.

One of the most popular extensions of Markov models, the Hidden Markov Model, or HMM, makes a slightly different assumption. Rather than assume that all the required information about the history of labels is held in the k most recent labels, it assumes there is some variable separate from the features and the label (a “hidden variable” or “latent variable”) which encodes all the information about the state of the system.

Hidden Markov models are generative models, making the assumption that each feature vector x_n is generated exclusively by the corresponding label y_n [9]. This assumption makes HMM’s inappropriate for the purposes of segmentation and classification.

2.4.1 Maximum Entropy Markov Models

One type of Markov model that is appropriate is the discriminative cousin of the HMM, the Maximum Entropy Markov Model, or MEMM. This model combines the principles of Maximum Entropy used in the Logistic Regression classifier with the Markov Assumption. This thesis makes use of an MEMM which assumes all the relevant information about the past labels is contained in the single previous label, making it a first-order Maximum Entropy Markov Model.

An MEMM is not conceptually much different from a logistic regression classifier. Instead of maximizing $P(y|x)$, we are now maximizing $P(y|x, y')$ (where y’ is the most recent label assigned). There are two changes that need to be made to the classifier accomplish this. First, we need to find some way of including y’ in the training process. Second, we need to consider the fact that when assigning labels to a sequence of samples, we can no longer consider each sample in isolation: we are now searching for the maximum likelihood sequence of labels.

There are two different ways to include information about previous labels in an MEMM. The traditional way of doing this is to train a separate logistic regression classifier for each value of y'. Conceptually, this means that rather than thinking of each label generating samples, we now think of it as each pair of labels (the previous label and the current label) generating samples. This places a high degree of emphasis on the sequence of labels, as compared to the features. The other way is to simply augment the features with the previous label information, allowing the underlying logistic regression classifier to make use of this information. This method is more flexible, and is the method used on the region labeling problem in this thesis.

The training of the logistic regression classifier is altered as follows. A collection of boolean variables representing whether or not the previous label is each of the possibilities is appended to the feature vector $\vec{x}$. For example, if the label set were text, equation, figure, heading, and the previous label were *equation*, the following features might be added:

Feature Name	Feature Value
is_text	0
is_equation	1
is_figure	0
is_heading	0

Each page is considered to be a separate sequence. Extra dummy samples with the labels *start of page* and *end of page* are appended to the beginning and end of page, to allow the system to make use of the fact that certain labels are much more likely than others to appear at the beginning and end of page.

In order to find the maximum likelihood sequence of labels for a new page, we now need to effectively consider all possible sequences of labels. While this problem would be intractable if approached naively, the application of dynamic programming through the Viterbi algorithm [16] allows us to evaluate only a small subset of the possible sequences directly, while guaranteeing that the best one will be among those evaluated.

The intuition behind this algorithm can be easily understood using a visual aid. Consider the problem of assigning one of K labels to a sequence of N regions as finding a path through a grid from left to right, where the n'th column represents the available labels for the n'th region, as illustrated by Table 2.1.

	Region 1	Region 2	Region 3	Region 4	Region 5	Region 6
Text		YES			YES	
Equation						YES
Figure			YES	YES		
Heading	YES					

Table 2.1: An illustration of the intuition behind the Viterbi algorithm.

In this diagram, likelihoods are assigned not to each cell, but to each pair of cells in adjacent columns. This is another way of expressing the First-Order Markov assumption that the likelihood of a given label being assigned to one region depends on the label assigned to the previous region. Since the likelihood depends on both the region itself, and the label of the region before it, assigning likelihoods to the transitions between cells in adjacent columns captures all this information.

In terms of this diagram, the intuition behind the Viterbi algorithm is that the best path that goes through a given cell must follow the best sequence of labels to that cell. This is true because we have used a first-order MEMM, which follows the Markov assumption that only the most recent label is relevant in determining the value of a given label. The Viterbi algorithm iterates over the columns of the gird, maintaining a list of the best paths up to each cell in the most recent column, and using this list to find the best path to each cell in the next column. The algorithm is as follows:

```
For each sample s on the page:
  For each possible label LAB:
    For each possible previous label LAB':
      Evaluate the best sequence up to LAB', adding LAB to this sequence.
      If the likelihood of this sequence is the best found for LAB so far:
        Store this as the sequence to LAB.
      end;
    end;
end;
```

Since the last sample has only one possible label *end of page*, the final iteration of this loop will produce the best sequence beginning at *start of page*, and ending at *end of page*. Thus this algorithm finds the best sequence of labels in $O(|s| \times |l|^2)$ time.

A small improvement was made to the Viterbi algorithm that is worth noting. The algorithm can be improved by using a variant of the "Beam Search" technique. This technique treats the search as a tree, in which only the most promising branches are followed down the tree. The resulting search path is a narrow "beam" down the tree, rather than the full tree.

In practice, for most samples, most of the labels are extremely unlikely. In fact, for most individual samples, many of the label assignments will be less likely than the entire best sequence of label assignments. A reasonable best guess at the sequence of label assignments can be made by finding the individual labels that would be selected by a simple logistic regression classifier. Evaluating the log likelihood LL of this path under the MEMM provides us with a simple criteria for deciding which paths on the search tree to consider at any point in the algorithm: if the log likelihood of this path is already less than LL, then we know it will not turn out to be the best path, since the log likelihood can only decrease further as more label assignments are added to the path. This means that we can disregard this potential path, ignoring it in future steps. The modified algorithm works as follows:

```
For each sample s on the page:
  For each possible label LAB:
    For each possible previous label LAB':
      Evaluate the best sequence up to LAB', adding LAB to this sequence.
      If the likelihood of this sequence is the best found for LAB so far:
        Store this as the sequence to LAB.
    end;
    If the sequence up to LAB has a lower total likelihood than LL, then
    remove it from the list of possible values of LAB' for the next sample
    to be considered.
  end;
end;
```

Essentially, this modification allows us to discount paths as soon as we know they cannot be judged the most likely. In practice, this modification was found to reduce the runtime of the algorithm on our data by about half.

Chapter 3

Ground Truth Data

In the field of document understanding, ground truth data refers to page images with manually created or verified records of the correct segmentation and labeling, and possibly the actual text of each region. This data is used for both the training of machine-learning algorithms, as well as for performance evaluation of all algorithms.

Preparation of ground truth data is extremely time consuming. It requires the acquisition or development of a software environment for preparation of the data, as well as standards for how this data will be recorded. It also requires the meticulous construction of the data itself, ideally with multiple passes of error-checking.

Several collections of ground truth data for document understanding, as well as systems for constructing this data, are mentioned in the literature. These include the Pink Panther environment by Berrin Yanikoglu and Luc Vincent [26], the University of Washington's "UW-III English/Technical Document Image Database" [1], and others.

None of these sets of data were appropriate for this study. While many of them do deal at least partially with journal articles, all of these classified ink regions into a small number of categories [3]. This study aims to be able to successfully classify into 25 different categories. A new set of ground-truth data was thus required. The software for producing this data,

[1]http://www.science.uva.nl/research/dlia/datasets/uwash3.html

as well as some of the data itself, were prepared by Scott Leishman in the summer of 2003, working at the Computer Science Department in the University of Toronto.

3.1 Description of Ground-Truth Data

One of the challenges of page segmentation and classification is the subjectivity in defining "correct" performance. A given page can, in fact, have a wide variety of segmentations and classifications that are each correct according to some definition. These different definitions of correctness may arise with a particular application in mind, or from abstract principles whose elegance inspires researchers to declare them correct. This proliferation in versions of "correct" has, unfortunately, led to a lack of standards to follow in the preparation of ground-truth data.

As there are no standards to follow, this chapter explains the motivation, standards, methodology used to prepare this set of ground truth data in detail.

3.1.1 Motivation For Methodology

The standards and methodology for preparation of this ground truth data were developed with a particular application in mind. This application is the improvement of targeted retrieval of electronic articles from scholarly journals, especially those for which the original version is an image, rather than an electronic document. In particular, allowing searches to target not just a particular string, but a particular string in a certain type of text (such as a heading, abstract, reference, or figure caption). This information about the type of text could be used not just for targeted retrieval, but for assessing the relevance of the text for generic retrieval. For example, finding a given word in the abstract could be considered more significant than finding it in a footnote.

This motivation heavily influenced all aspects of the preparation of the ground-truth data, and keeping this in mind may provide insight as to why things were done the way they were.

3.1.2 Page Images Used in Ground Truth Data

The page images used for the ground truth data were taken from sources available online, such as the proceedings of the Neural Information Processing Systems conference (NIPS)[2] and the Journal of Machine Learning Research (JMLR)[3], and several others. A variety of such articles were downloaded, and the pages were converted to .tif files 2156 pixels high and 1728 pixels wide, which are then stored locally.

It should be noted that these page images are assumed to come from electronic original documents, as opposed to being scanned images. This has several significant implications. First, the pages are entirely free of noise. Noise in the page makes the task of page segmentation more challenging, and can have varying effects on different algorithms. Second, the pages are assumed to be free of skew. A skewed page is one in which the ink on the page is “tilted”, as compared to the page itself, as demonstrated in figure 3.1. As with noise on the page, skewed pages are more difficult to segment, and skew will affect some algorithms more than others.

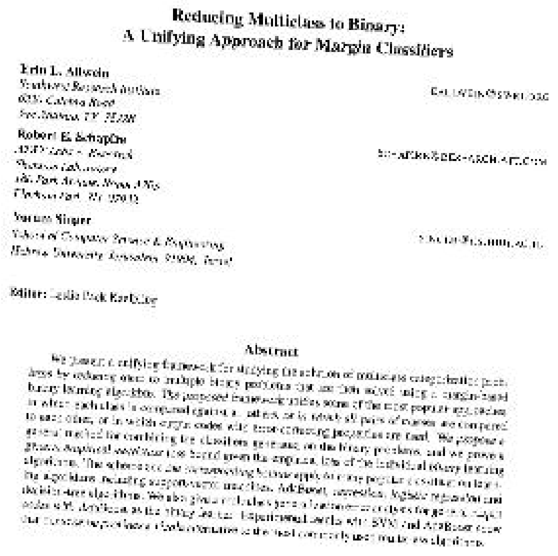
Many document understanding systems rely on pre-processors to perform deskewing and noise removal. Cattoni et al. provide a review of deskewing techniques in [3]. To extend the work done in this paper to the processing of scanned pages, deskewing and noise removal pre-processors would need to be added.

A wide variety of layout complexities are used in scholarly journals. Several attributes of the layout scheme determine its complexity. These are often expressed in terms of the “regions” on the page, which are blocks of ink which appear to belong together, such as a figure, a block of text, or an equation.

- Are all of the regions polygonal?

[2]http://www.nips.cc/

[3]http://jmlr.csail.mit.edu/



Reducing Multiclass to Binary:
A Unifying Approach for Margin Classifiers

Erin L. Allwein
Southwest Research Institute
6220 Culebra Road
San Antonio, TX 78238
EALLWEIN@SWRI.ORG

Robert E. Schapire
AT&T Labs − Research
Shannon Laboratory
180 Park Avenue, Room A203
Florham Park, NJ 07932
SCHAPIRE@RESEARCH.ATT.COM

Yoram Singer
School of Computer Science & Engineering
Hebrew University, Jerusalem 91904, Israel
SINGER@CS.HUJI.AC.IL

Editor: Leslie Pack Kaelbling

Abstract

We present a unifying framework for studying the solution of multiclass categorization problems by reducing them to multiple binary problems that are then solved using a margin-based binary learning algorithm. The proposed framework unifies some of the most popular approaches in which each class is compared against all others, or in which all pairs of classes are compared to each other, or in which output codes with error-correcting properties are used. We propose a general method for combining the classifiers generated on the binary problems, and we prove a general empirical multiclass loss bound given the empirical loss of the individual binary learning algorithms. The scheme and the corresponding bounds apply to many popular classification learning algorithms including support-vector machines, AdaBoost, regression, logistic regression and decision-tree algorithms. We also give a multiclass generalization error analysis for general output codes with AdaBoost as the binary learner. Experimental results with SVM and AdaBoost show that our scheme provides a viable alternative to the most commonly used multiclass algorithms.

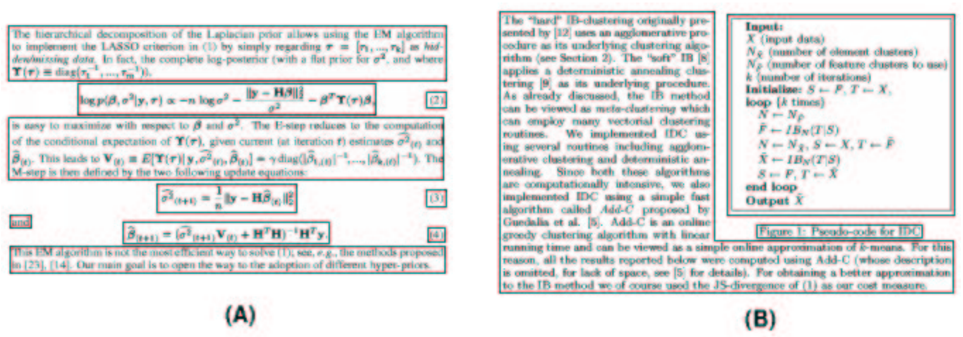
Figure 3.1: An example of a skewed page image.

- Are all of the regions rectangular?
- Are all the regions aligned in the same direction, or are some skewed?
- Is a single column or multi-column layout used?

The majority of the pages used for this study follows a single-column Manhattan layout scheme. Under a Manhattan layout scheme, all of the regions are required to be non-overlapping page-aligned rectangles. Figure 3.2 demonstrates Manhattan and more complex layouts.

3.1.3 Segmentation of Ground Truth Data

The first step in preparing a page as ground truth data is to segment the page. In order to ensure consistency, a set of rules were set out and followed during the manual segmentation. These rules are intended to produce a segmentation scheme that is both appropriate for the application of targeted search, and that one could reasonably expect a good algorithm to be able to reproduce.


The hierarchical decomposition of the Laplacian prior allows using the EM algorithm to implement the LASSO criterion in (1) by simply regarding $\boldsymbol{\tau} = [\tau_1, ..., \tau_k]$ as *hidden/missing data*. In fact, the complete log-posterior (with a flat prior for σ^2, and where $\boldsymbol{\Upsilon}(\boldsymbol{\tau}) \equiv \text{diag}(\tau_1^{-1}, ..., \tau_m^{-1})$).

$$\log p(\boldsymbol{\beta}, \sigma^2 | \mathbf{y}, \boldsymbol{\tau}) \propto -n \log \sigma^2 - \frac{\|\mathbf{y} - \mathbf{H}\boldsymbol{\beta}\|_2^2}{\sigma^2} - \boldsymbol{\beta}^T \boldsymbol{\Upsilon}(\boldsymbol{\tau})\boldsymbol{\beta}, \quad (2)$$

is easy to maximize with respect to $\boldsymbol{\beta}$ and σ^2. The E-step reduces to the computation of the conditional expectation of $\boldsymbol{\Upsilon}(\boldsymbol{\tau})$, given current (at iteration t) estimates $\widehat{\sigma^2}_{(t)}$ and $\widehat{\boldsymbol{\beta}}_{(t)}$. This leads to $\mathbf{V}_{(t)} \equiv E[\boldsymbol{\Upsilon}(\boldsymbol{\tau}) | \mathbf{y}, \widehat{\sigma^2}_{(t)}, \widehat{\boldsymbol{\beta}}_{(t)}] = \gamma\, \text{diag}(|\widehat{\beta}_{1,(t)}|^{-1}, ..., |\widehat{\beta}_{k,(t)}|^{-1})$. The M-step is then defined by the two following update equations:

$$\widehat{\sigma^2}_{(t+1)} = \frac{1}{n} \|\mathbf{y} - \mathbf{H}\widehat{\boldsymbol{\beta}}_{(t)}\|_2^2 \quad (3)$$

and

$$\widehat{\boldsymbol{\beta}}_{(t+1)} = (\widehat{\sigma^2}_{(t+1)} \mathbf{V}_{(t)} + \mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}. \quad (4)$$

This EM algorithm is not the most efficient way to solve (1); see, *e.g.*, the methods proposed in [23], [14]. Our main goal is to open the way to the adoption of different hyper-priors.

(A)

The "hard" IB-clustering originally presented by [12] uses an agglomerative procedure as its underlying clustering algorithm (see Section 2). The "soft" IB [8] applies a deterministic annealing clustering [9] as its underlying procedure. As already discussed, the IB method can be viewed as *meta-clustering* which can employ many vectorial clustering routines. We implemented IDC using several routines including agglomerative clustering and deterministic annealing. Since both these algorithms are computationally intensive, we also implemented IDC using a simple fast algorithm called *Add-C* proposed by Guedalia et al. [5]. Add-C is an online greedy clustering algorithm with linear running time and can be viewed as a simple online approximation of k-means. For this reason, all the results reported below were computed using Add-C (whose description is omitted, for lack of space, see [5] for details). For obtaining a better approximation to the IB method we of course used the JS-divergence of (1) as our cost measure.

```
Input:
$X$ (input data)
$N_{\tilde{X}}$ (number of element clusters)
$N_{\tilde{F}}$ (number of feature clusters to use)
$k$ (number of iterations)
Initialize: $S \leftarrow F$, $T \leftarrow X$,
loop ($k$ times)
    $N \leftarrow N_{\tilde{F}}$
    $\tilde{F} \leftarrow IB_N(T|S)$
    $N \leftarrow N_{\tilde{X}}$, $S \leftarrow X$, $T \leftarrow \tilde{F}$
    $\tilde{X} \leftarrow IB_N(T|S)$
    $S \leftarrow F$, $T \leftarrow \tilde{X}$
end loop
Output $\tilde{X}$
```

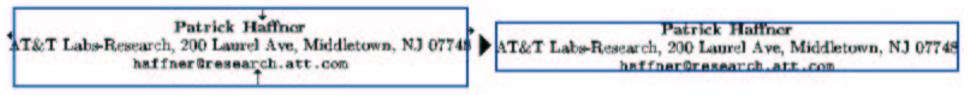
Figure 1: Pseudo-code for IDC

(B)

Figure 3.2: Examples of (A) Manhattan and (B) non-Manhattan layouts.

This dataset requires that all pages follow a Manhattan layout scheme. In cases where the articles do not use a purely Manhattan layout (such as when equations are embedded in a text region, and figures with irregular edges), a Manhattan segmentation that approximates the actual layout was used. While it may result in some oversimplification of the segmentation of pages, assuming all regions use a Manhattan layout allows each region to be specified by just four parameters: the top, left, bottom, and right. This simplifies many aspects of working with the data.

A software tool was used to assist in the manual markup of the data. The bounding boxes of the regions were selected by hand. These regions were then automatically "snapped" to the boundaries of the contained ink along all four edges, as demonstrated in figure 3.3. This snapping process ensures both that the borders of regions are consistently placed correctly, and are neither left in whitespace several pixels away from ink, nor are they placed across a mark of ink. This provides consistency to the boundaries of the regions, which is extremely valuable in later classification tasks.


Patrick Haffner
AT&T Labs-Research, 200 Laurel Ave, Middletown, NJ 07748
haffner@research.att.com

Patrick Haffner
AT&T Labs-Research, 200 Laurel Ave, Middletown, NJ 07748
haffner@research.att.com

Figure 3.3: Region boundaries were automatically "snapped" to the edge of the ink they bound.

Since the page images used all came from electronic originals, and there is no noise in the images, the added constraint was used that each non-background pixel must be contained by exactly one of the regions, preventing ink from being left outside of the regions, or being contained by overlapping regions. The general rule for selecting the regions was that regions should be made as large as possible, without violating any of the following rules:

- Regions of different classes should not be grouped together, unless one region completely contained another region of a different class, such as a small equation set inside a block of text.
- Regions should never be grouped together when doing so would degrade the results of applying OCR. For example adjacent columns of text should be separated into different regions, as an OCR system operating left-to-right would produce incorrect results if the regions were kept together.
- Regions clearly separated by significant whitespace should be separated.
- Each reference in a bibliography should be contained in its own region. This rule was added with consideration to citation extraction applications.

To further ensure consistency, all of the segmentation has been checked over by one individual. This means that any divergence from the rules set forth should at least be consistent, as should the results of any judgement calls in situations where the rules do not clearly dictate what should be done.

3.1.4 Classification of Ground Truth Data

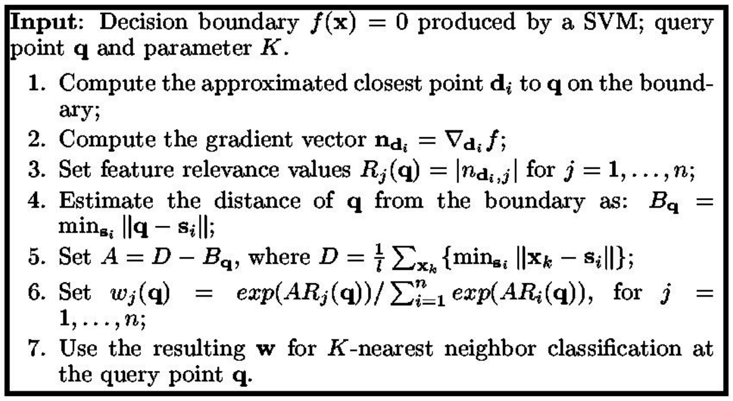
Page images that have been segmented and double-checked are ready for classification, or labeling (note that the terms “class” and “label” can be used interchangeably in this context). Each region must be assigned the most appropriate label from those listed in Table 3.1. Like the segmentation scheme, this set of labels was chosen with the application of targeted document retrieval, in mind. An additional consideration was that a good classifier should

Label	Description
text	Normal body text.
header	A header at the top of a page.
section heading	A top-level heading, often numbered or lettered.
figure label	The label assigned to a figure, such as “Figure 3”.
figure caption	The caption describing a figure.
subsection heading	A second or lower level heading, often numbered or lettered.
references	A single reference entry.
abstract	The abstract of a paper, possibly including the word “Abstract”.
bullet item	A single item in a bulleted list.
pg number	The page number.
main title	The main title of the paper.
footer	A footer repeated at the bottom of every (or every second) page.
table label	The label assigned to a table, such as “Table 2.4”.
table caption	The caption describing a table.
editor list	One or more entries in the listing of editors for the paper.
eq number	The reference number assigned to an equation.
author list	One or more entries in the author list at the beginning of a paper.
footnote	A footnote used to provide additional details.
decoration	Ink used to decorate or demarcate.
equation	An equation that is separated from the body of the text.
table	A table.
image	A single picture, possibly transformed or modified.
graph	A single graph, including a drawn (or implied) set of axes.
figure	Any figure that is not an image, graph, or code block.
code block	A block of code or pseudo-code.

Table 3.1: The 25 fine-grained labels applied to the ink regions.

be able to reproduce the classification. A detailed description of each of the labels is provided in Appendix A.

It is worth noting that the correct label for many regions is somewhat ambiguous, as authors will often mix several of the elements listed above, as demonstrated in Figure 3.4. While all of the classification was checked over by a single individual in an attempt to achieve consistency in treatment of ambiguous regions, it is expected that some questionable labellings still exist in the dataset.



Input: Decision boundary $f(\mathbf{x}) = 0$ produced by a SVM; query point $\mathbf{q}$ and parameter K.

```
1. Compute the approximated closest point $\mathbf{d}_i$ to $\mathbf{q}$ on the boundary;
2. Compute the gradient vector $\mathbf{n}_{\mathbf{d}_i} = \nabla_{\mathbf{d}_i} f$;
3. Set feature relevance values $R_j(\mathbf{q}) = |n_{\mathbf{d}_i,j}|$ for $j = 1, \ldots, n$;
4. Estimate the distance of $\mathbf{q}$ from the boundary as: $B_\mathbf{q} = \min_{\mathbf{s}_i} \|\mathbf{q} - \mathbf{s}_i\|$;
5. Set $A = D - B_\mathbf{q}$, where $D = \frac{1}{l} \sum_{\mathbf{x}_k} \{\min_{\mathbf{s}_i} \|\mathbf{x}_k - \mathbf{s}_i\|\}$;
6. Set $w_j(\mathbf{q}) = exp(AR_j(\mathbf{q})) / \sum_{i=1}^{n} exp(AR_i(\mathbf{q}))$, for $j = 1, \ldots, n$;
7. Use the resulting $\mathbf{w}$ for $K$-nearest neighbor classification at the query point $\mathbf{q}$.
```

Figure 3.4: Example of a region whose class is ambiguous.

Journal	Subset	Articles	Pages	Regions
JMLR	Training	10	294	3521
JMLR	Test	5	178	2035
JMLR	Total	15	472	5556
NIPS*	Training	39	309	3445
NIPS*	Test	19	151	1473
NIPS*	Total	58	460	4918
Both	Total	73	932	10474

Table 3.2: Details of the contents of the data set created.

3.1.5 Data Set Details

The ground truth data includes a total of 932 page images and the associated ground truth data, representing 73 articles from two different sources: one journal, and one set of conference proceedings. For each source, the dataset was divided into two subsets: a training set, which was used for optimizing parameters and training machine learning algorithms, and a test set, which was used exclusively for performance evaluation of completed algorithms. Table 3.2 provides a summary breakdown of both the JMLR and NIPS data sets[4].

[4]The NIPS data is divided roughly evenly between the 2001 and 2002 proceedings, and the JMLR data is taken from the 2000 through 2003 journals.

3.2 Software Environment

A software system named JTAG (Journal TAGger) was developed by Scott Leishman at the University of Toronto during the summer of 2003 for the preparation of ground-truth data. This system consists of three parts. First, the .jtag file format provides a manner for storing this tagging information, and exchanging it with other software. Second, the actual JTAG program is a collection of TCL scripts that provide a user-interface for viewing and tagging individual pages of a journal. Finally, a collection of Matlab scripts are used to automatically segment and label pages. When these scripts are used, the pages need only be manually verified and corrected, which speeds up the process of producing ground truth data.

3.2.1 The .jtag File Format

The .jtag file format was developed for storing the segmentation and labeling information for page images, as well as for exchanging this information with other software. This format is specific to storing rectangular page-aligned regions, with a single label applied to each.

Each .jtag file contains the segmentation and classification information for a single page image. The .jtag file has the same name as the image file, with the image file's extension replaced by .jtag. The .jtag files are in the ASCII text format, and can easily be parsed in any programming language. Each .jtag file contains the following information about a single page image:

1. The name, location, and checksum value of the associated image file.
2. The program settings used when this file was tagged.
3. The location and class of each region in the image, along with information about how and when the region was created.

A more detailed documentation of the .jtag file format is available in Appendix B.

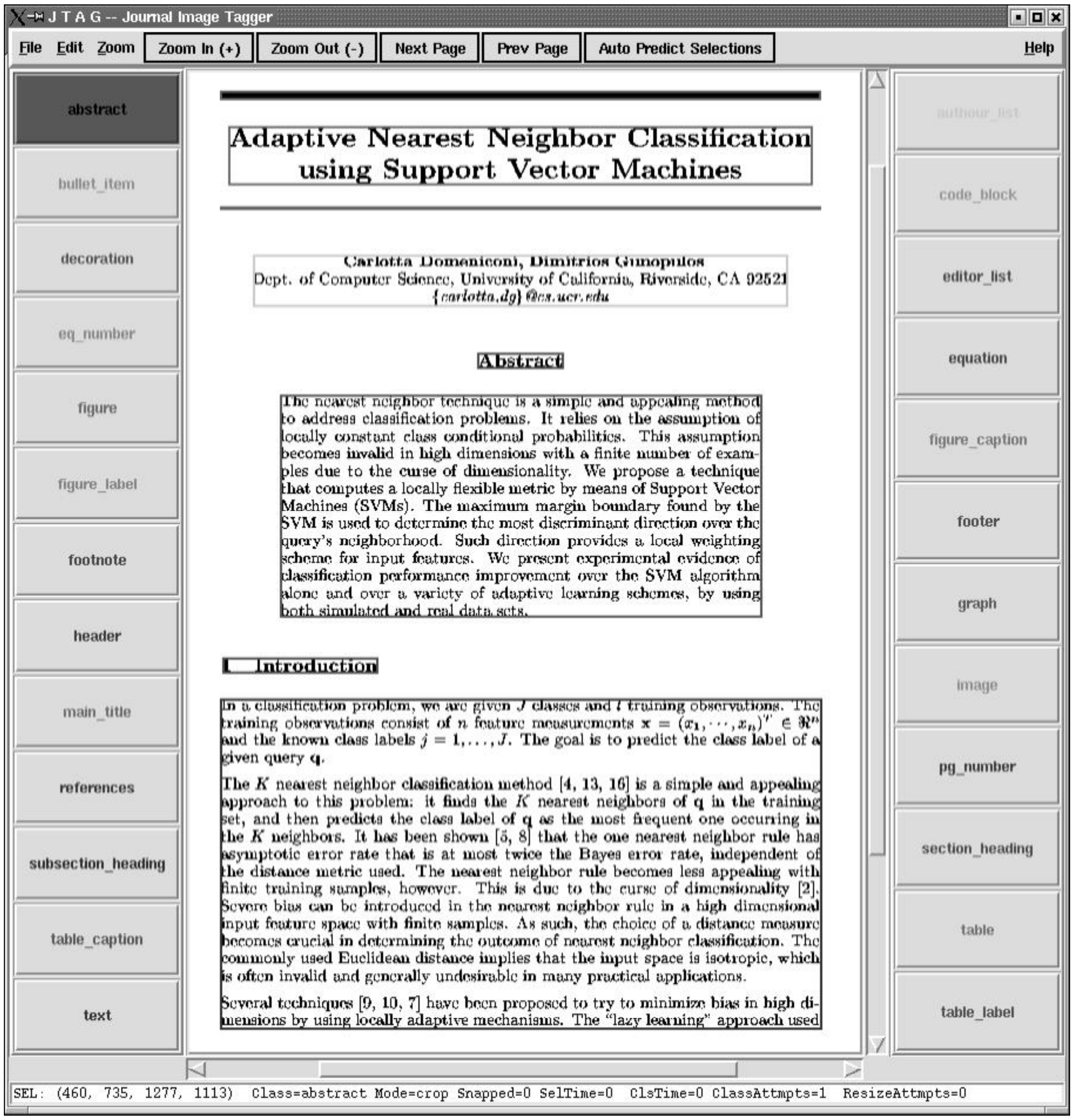
3.2.2 The JTAG Program

The JTAG program consists of a collection of TCL scripts that load and display a page image, along with its associated segmentation and classification information. A screen shot of the program is provided in Figure 3.5.

Sufficient functionality is provided by the program for easy creation of ground truth data. The following functions allow the user to manipulate this tagging information:

1. Manually create a new region bounding rectangle by dragging the mouse across the page image.
2. Manually label a region by dragging it to the appropriate class "bucket" displayed beside the page image.
3. Split a region in two, either vertically or horizontally.
4. Merge two regions into a single region.
5. Manually adjust a region's borders.
6. Automatically snap a region's bounding box to the edge of the contained ink.
7. Delete a region.
8. Send the page and tagging information to Matlab for automatic segmentation and labeling.

These functions, along with navigation functions, allow the user to manually segment and label large data sets very rapidly, preparing them for use as training and test data. Tagging rates of about 1 minute per page were achieved. Note that the JTAG software works only with rectangular regions that are aligned with the page. This restriction allows for much faster creation of ground truth data, as well as for simpler specification of the regions.



Adaptive Nearest Neighbor Classification using Support Vector Machines

Carlotta Domeniconi, Dimitrios Gunopulos
Dept. of Computer Science, University of California, Riverside, CA 92521
{carlotta,dg}@cs.ucr.edu

Abstract

The nearest neighbor technique is a simple and appealing method to address classification problems. It relies on the assumption of locally constant class conditional probabilities. This assumption becomes invalid in high dimensions with a finite number of examples due to the curse of dimensionality. We propose a technique that computes a locally flexible metric by means of Support Vector Machines (SVMs). The maximum margin boundary found by the SVM is used to determine the most discriminant direction over the query's neighborhood. Such direction provides a local weighting scheme for input features. We present experimental evidence of classification performance improvement over the SVM algorithm alone and over a variety of adaptive learning schemes, by using both simulated and real data sets.

1 Introduction

In a classification problem, we are given J classes and l training observations. The training observations consist of n feature measurements $\mathbf{x} = (x_1, \cdots, x_n)^T \in \Re^n$ and the known class labels $j = 1, \ldots, J$. The goal is to predict the class label of a given query $\mathbf{q}$.

The K nearest neighbor classification method [4, 13, 16] is a simple and appealing approach to this problem: it finds the K nearest neighbors of $\mathbf{q}$ in the training set, and then predicts the class label of $\mathbf{q}$ as the most frequent one occuring in the K neighbors. It has been shown [5, 8] that the one nearest neighbor rule has asymptotic error rate that is at most twice the Bayes error rate, independent of the distance metric used. The nearest neighbor rule becomes less appealing with finite training samples, however. This is due to the curse of dimensionality [2]. Severe bias can be introduced in the nearest neighbor rule in a high dimensional input feature space with finite samples. As such, the choice of a distance measure becomes crucial in determining the outcome of nearest neighbor classification. The commonly used Euclidean distance implies that the input space is isotropic, which is often invalid and generally undesirable in many practical applications.

Several techniques [9, 10, 7] have been proposed to try to minimize bias in high dimensions by using locally adaptive mechanisms. The "lazy learning" approach used

```
SEL: (460, 735, 1277, 1113)  Class=abstract Mode=crop Snapped=0 SelTime=0  ClsTime=0 ClassAttmpts=1  ResizeAttmpts=0
```

Figure 3.5: Screenshot of the JTAG software.

3.2.3 The Matlab Software

A collection of Matlab scrips was used for the automated segmentation and labeling of individual image files. The entry-point to these scripts is a single function that takes as input the full path of the .jtag file, the names of the segmentation and labeling functions to use, as well as any required parameters. This function loads the .jtag file, applies the appropriate segmentation and labeling functions, saves the updated .jtag file, and returns control to the calling program. Note that while it operates on only one image file at a time, the naming scheme used for the image files allows the scripts to determine the number of pages in that article, as well as the position of the current page within the article.

All of the segmentation and classification algorithms have parameters that need to be optimized in order to be effective. Optimizing these parameters requires a set of prepared ground-truth data upon which to train different values. By manually preparing a small amount of data, and then training these algorithms on this data, the preparation process can be sped up. The algorithms trained on small amounts of data will not be extremely accurate, but will nonetheless provide a good “first guess” at the segmentation and classification, speeding up the process of developing further ground truth data. Successive cycles of preparing more data, and then re-training the algorithms, can be used to produce large amounts of training data quickly.

The bulk of this study focused on developing, optimizing, and testing automated segmentation and labeling algorithms. Detailed descriptions of the file formats and data structures used by each of these are provided in the sections in which the algorithm is discussed.

Chapter 4

Segmentation

4.1 Problem Description

Segmentation is the process of dividing a page into discrete regions of ink that belong together.

A major challenge of the segmentation problem is that it is ill-posed, leaving a great deal of room of subjectivity. For many page layouts, there is no single, objectively correct segmentation. Several philosophies exist as to what constitutes the "correct" segmentation. Some suggest that the correct segmentation is the one that forms the largest possible regions without including bits of ink that should be classified differently (such as text and figures). Another philosophy is that the goal should be to re-construct the original regions that were put on the page by the layout program (such as LaTex). A third would be that any segmentation is correct, provided that it allows some particular application to work correctly. Finally, some authors give up on the concept of a single correct segmentation altogether, and consider the segmentation of a page to include nested regions describing multiple levels of segmentation.

An example of these different viewpoints is the separation of references at the end of a paper. If we accept that the correct segmentation is the one that produces the largest regions of the same type, this would imply that all of the references should be grouped together into the same region. On the other hand, having separate regions for each reference may be useful for some applications, such as making document images searchable. The particular segmentation conventions used in this study are tailored to the application of targeted document searching, and are described in detail in Chapter 3.

Most academic journals use relatively simple layout schemes. Regions of ink tend to be rectangular, following what is known as the Manhattan Layout scheme, in which all regions are rectangular and aligned to the page, with no overlapping allowed. This study works on the explicit assumption that all documents in the dataset follow a Manhattan layout scheme.

Assuming that all ink regions follow the Manhattan scheme simplifies the segmentation problem in several ways. First, a region can be fully described by just four parameters: top, left, bottom, and right. In non-Manhattan layout schemes, a more complex method of describing regions needs to be used. Second, most segmentation algorithms have simplified forms, often with much faster run-times and higher levels of accuracy, when the Manhattan layout assumption is used. For example, algorithms based on the idea of making cuts to divide up the page need only consider vertical and horizontal lines, and will still be able to come up with the correct segmentation.

4.2 Evaluation of Segmentation Performance

Given that there is no clear agreement as to what constitutes the correct segmentation of a page, it should come as no surprise that a plethora of performance metrics for segmentation exist in the literature. Many of these techniques are the logical consequence of the author's selection of what constitutes a correct segmentation, and some are in turn connected to a particular application.

A performance metric is required for several reasons. First, a method is required for comparing the performance of many different algorithms. Most existing performance metrics have been developed for the purpose of comparing the performance of various techniques, with this motivation. Second, several segmentation algorithms contain parameters that affect how they operate. While selecting values for these parameters is often done either by hand, or using some heuristic, there is no reason to assume that this results in the best possible values. Far better would be an automated search of the parameter-space to find their optimal settings. Any performance metric that provides a single number that represents the quality of a segmentation of a page is suited to use for optimizing parameters.

4.2.1 Existing Performance Metrics

Cattoni et al. [3] provide a useful division of performance metrics into text-based methods, which measure the accuracy of OCR applied to the results of segmentation, and region-based methods, which measure the accuracy of the segmentation itself .

Text-based metrics are focused on the application of page segmentation as on OCR pre-processor. The quality of a segmentation is measured, in some manner, based on the number of errors the segmentation would induce in the results of applying OCR to the segmented document. For example, merging two adjacent columns of text together would create multiple OCR errors, as the two columns would end up interleaved, with the first row of the first column being followed by the first row in the second column. Text-based error metrics ignore non-text regions entirely, and do not penalize for grouping consecutive text regions of different types. They are thus not appropriate for this study.

Region-based metrics focus on measuring how well the regions produced by the segmentation algorithm match the actual regions in the ground-truth data. Liang et al. provide a method for creating a single number that estimates this matching for a page [14]. Several numbers are calculated from the two sets of regions (the ground-truth regions G, and the detected regions D):

1. Correct: Regions in G that map “exactly” to a single region in D

2. Misses: Regions in G that do not map even partially to any region in D
3. False alarms: Regions in D that do not even partially map to any region in G
4. Splits: Regions in G that map to multiple regions in D
5. Merges: Several regions in G map fully to a single region in D
6. Spurious: Any other regions

$$Cost = \frac{W_{miss}N_{miss} + W_{fal}N_{fal} + W_{spl}(N^G_{spl} + N^D_{spl}) + W_{meg}(N^G_{meg} + N^D_{meg}) + W_{spu}(N^G_{spu} + N^D_{spu})}{|G| + |D|}$$

A penalty coefficient is assigned to each type of error, and the cost for a segmentation is the weighted sum of the errors divided by the total number of regions. The fact that a region in G being split implies a region in D being merged is ignored in the particular cost function used, but this does not affect the merit of the overall approach.

A similar approach is used by Antonacopoulos et al. for the 2003 Page Segmentation Competition at the International Conference on Document Analysis and Recognition [1]. Instead of assigning a penalty for errors, however, they provide a score based on the number of correct matches, split-matches, and merged-matches, with coefficients being assigned once again to all three types of matches.

4.2.2 Metrics Considered

Several factors must be considered when selecting a performance metric. First, it should be appropriate for the particular ground truth data in use. Since regions in this study are assumed to be rectangular and page-aligned, performance metrics that make this same assumption can be considered. Second, the performance metric should reflect anticipated performance on the target application. As this study is focused on the application of targeted document retrieval, performance metrics were used that are expected to be good indicators of performance for this task. Finally, preference is given to performance metrics that are fast and simple, to ease the reproducibility of all experiments done.

Several performance metrics were considered.

The simplest performance metric considered is how many of the hand-segmented regions were correctly identified. A region was considered correctly identified if, for some predicted region, each of the boundaries (left, right, top and bottom) differed by less than p pixels. For our particular images, a value of $p = 5$ was found to be appropriate. This metric works relatively well when the segmentation algorithms being evaluated are not able to create overlapping regions. If this is not the case, the "best" segmentation algorithm is simply one that makes every possible combination of contiguous pixels a region, since this is guaranteed to include a match for every hand-segmented region.

Even when used to evaluate an algorithm that does not allow overlapping regions, this metric tends to favor algorithms that over-segment over those that under-segment. If a hand-labeled region is incorrectly divided into 2 regions, this only counts as one error, since only one hand-labeled region will not be matched. Likewise, even if the hand-labeled region is divided into 8 regions, this metric still counts that as only one error. Two hand-segmented regions that are kept together as a single region, however, will be counted as two errors, since two hand-segmented regions will not be matched. As a result, segmentation algorithms trained on this error metric have a tendency to over-segment.

Another performance metric was developed to compensate for this. In this metric, the hand-segmented regions and the predicted regions are treated as two sets. Each region that is a member of one set, but not the other, is considered to be an error. This means that an over-segmentation in which a hand-labeled region is incorrectly divided into two regions will produce 3 errors: one for each of the three regions that does not find a match. An under-segmentation in which two hand-labeled regions are kept together in a single region will also produce 3 errors. Algorithms optimized for this error metric did not show a consistent tendency to over-segment.

Both of these error metrics ignore the fact that there can be multiple valid segmentations of a page. Take as an example a large block of text with full lines of whitespace between paragraphs. It is open for debate as to whether a better segmentation considers each paragraph to be its own region, or whether the entire block should be a single region. Likewise for a series of references, figures, or bullet items. Unfortunately, developing a performance metric that takes this into account requires knowledge of the labels of the regions, which is not necessarily available.

4.3 Past Approaches to Segmentation

A growing variety of approaches to segmentation exist in the literature. Most of the older approaches can be divided into top-down and bottom-up algorithms . The top-down algorithms start with the entire page as one segment, and divide the page into smaller and smaller segments. Bottom-up algorithms start with many small regions and merge regions together to form the final segmentation. Most often, the starting point is individual "marks" of ink, where a mark of ink is a contiguous group of black pixels. These approaches tend to be heuristic-based, with a small number of tunable parameters usually set by hand [15]. Several other approaches with produce a full segmentation in a single step (rather than cutting or merging regions), and thus do not fall into either of these categories are also considered.

4.3.1 Taxinomy of Past Techniques

Several attempts have been made to survey and analyze the diverse array of existing page segmentation algorithms. The most common division is between top-down algorithms, which start at the level of the entire page and gradually break it down into smaller regions, and bottom-up algorithms, which start with individual marks of ink, merging them together to form regions.

Page segmentation algorithms all attempt to accomplish the same fundamental task: to decide which bits of ink belong together. As such, every page segmentation algorithm must contain two core elements:

1. A structural algorithm by which bits of ink are selected for evaluation as to whether they belong together.

2. A decision criteria by which these bits of ink are evaluated.

Segmentation algorithms can be effectively grouped based on the first criteria.

Top-down algorithms start with all the ink on the page together in a single block, and cut it into smaller regions until some stopping criteria is met. The cuts can be made recursively, making just a single cut each time the recursive function is called, or concurrently, as done by [12], evaluating and applying any number of cuts in each pass. The cuts can be restricted to straight lines, and can be forced to be page-aligned. Page-aligned cuts may consider only whitespace as a separator, or may consider lines of ink as well [4].

Bottom-up algorithms start with very small blocks of ink as separate regions, merging them together to form regions. They may start at the level of pixels, at contiguous marks of ink (such as individual characters) [11], at words, or even at lines of text. They may be applied recursively, merging small regions into increasingly large ones, or single-pass, considering all possible mergings at the same time. Additional restrictions can be added preventing nested regions or forcing the region to be rectangular and page-aligned.

These different algorithms can be arranged in a hierarchy of increasing restrictions on which bits of ink will be considered as candidates for belonging together. Each such restriction has a corresponding assumption about the layout of the page which must hold in order for the algorithm to be able to reach the correct segmentation. In general algorithms with more restrictions will run faster, and will perform better *so long as the page layout fits their assumptions.* However, if the assumptions for a given restriction do not hold, imposing this restriction will significantly degrade results. When selecting an algorithm structure for a given task, one should thus use as many restrictions as possible, so long as their respective assumptions hold for the class of documents being examined.

It is more difficult to form a taxonomy based on the decision criteria for whether bits of ink belong together. One natural criteria for classifying decision criteria is the information upon which decisions are based. This information is often drawn from one or more of several sources, including (but not limited to):

- The layout (location, size, distance between, etc.) of bits of ink and/or regions [23], [11], [18].
- Statistical properties of the ink distribution [5], such as which pixels are off and on, or (in our case) the density of on pixels.
- The results of OCR applied to regions.
- The labels assigned to bits of ink by a classification system [17].
- The stage of progress through the structural algorithm [23].

These do not form any clear hierarchy, and any number of them may be used by a given classifier. They may also be used in elaborate combinations, forming such constructions as page grammars [12] and geometric decision trees [6].

A segmentation algorithm can be fully described by specifying just its structural algorithm, as well as its decision criteria for evaluating whether regions belong together. The full structural and decision aspects of the algorithms used in this research program will be described in the following sections.

4.3.2 Recursive XY Cuts

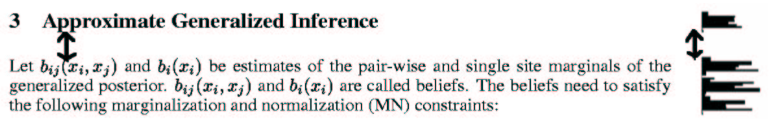
Approaches that start with the entire page as a single region and repeatedly divide this into smaller regions are termed top-down approaches. The most popular top-down approach is the Recursive XY Cuts algorithm proposed by Nagy et al. [18]. Recursive XY Cuts is a simple algorithm that makes horizontal and vertical cuts through existing regions, creating smaller regions until a stopping condition is met.

This algorithm is loosely based on a model of how pages are laid out. This model is that the page image is a series of nested tables. The page is successively dividing the page up in rows or columns separated by whitespace, and laying down regions of ink once this process is complete. The Recursive XY Cuts algorithm attempts to replicate this process.

The model that the XY Cuts algorithm is based on makes several simplifying assumptions about the layout of the page. First, it assumes that all boundaries between regions are straight lines. Second, it assumes that all regions are rectangular and page-aligned, so that only horizontal and vertical cuts need to be considered. Third, it assumes that regions are not nested, with one completely surrounding another. Finally, it assumes that the page is laid out like a table. This allows algorithms to consider only cuts that go all the way across a region, dividing it completely; it disallows cyclical regions, where a group of regions is cyclical if no two regions can be divided by a line such that the line does not intersect another region. This last assumption is often not stated explicitly, as it is rare to find a page that contains only rectangular page-aligned regions that are not nested, but does not fit this assumption.

The Recursive XY Cuts algorithm begins by considering the entire page as a single region. It considers places where the region could be cut without the cut intersecting any ink, and searches for the best cut in that region in both the x and the y directions. It makes that cut, creating two new regions. The algorithm is then applied recursively, searching for and making cuts in all of the remaining regions. The algorithm terminates when none of the remaining regions have any valid cuts left to be made.

Potential locations for cuts are located using horizontal and vertical projections of the ink. Any position where the fraction of black pixels shown by the projection is below some threshold (known as “whitespace”, even though it may contain some non-white pixels) is a candidate for cutting. Our implementation assumes that there is no pixel noise in the image,


3 Approximate Generalized Inference

Let $b_{ij}(x_i, x_j)$ and $b_i(x_i)$ be estimates of the pair-wise and single site marginals of the generalized posterior. $b_{ij}(x_i, x_j)$ and $b_i(x_i)$ are called beliefs. The beliefs need to satisfy the following marginalization and normalization (MN) constraints:

Figure 4.1: The width of the cut is identical to the length of the valley in the projection profile.

and hence uses the “whitespace threshold” of 0. The quality of a cut candidate is judged exclusively on the width of the whitespace region created by the cut, which corresponds to the length of the valley in the projection profile, as illustrated in Figure 4.1. The wider the cut, the better it is judged to be, regardless of whether it is a horizontal or vertical cut.

Pseudo-code for the algorithm can be found below [1]:

```
function xycut(segment):segments
    h_cut = find_widest_horizontal_cut(segment)
    v_cut = find_widest_vertical_cut(segment)
    if (v_cut.width >= v_threshold) & ((v_cut.width > h_cut.width) |
       (h_cut.width < h_threshold))
        halves = make_cut(v_cut)
        return [xycut(halves(1)),xycut(halves(2))]
    elseif (h_cut.width >= h_threshold) & ((h_cut.width > v_cut.width) |
           (v_cut.width < v_threshold))
        halves = make_cut(h_cut)
        return [xycut(halves(1)),xycut(halves(2))]
    else
        return [segment]
```

Cesarini et al. [4] extend this algorithm to allow it to cut along solid black lines, as well as through whitespace. This is a sensible extension, as lines are often used to separate

[1]The notation *foo(a,b):c* is used throughout this paper, and indicates that the function *foo* takes inputs a and b, and returns output c.

regions in densely formated pages, where whitespace is at a premium.

The Recursive XY Cuts algorithm has several points in its favor. It is simple to code, and quite fast. It performs quite well on documents that fits the assumptions that the model is based on, and in which the regions are generally separated by whitespace (as opposed to solid lines, or some other method). This algorithm also allows the intermediate regions to be stored, creating a tree-like data structure representing the process of dividing the page. Since the assumptions made in the construction of the ground truth data for this study are the same as those made by the XY Cuts algorithm's model for page layout, it serves as a valuable reference point for comparison.

Two significant improvements to the XY Cuts algorithm were made. First, an observation about its recursion function has resulted in a more efficient version of the algorithm. Second, a method of parameter selection that provides optimal performance was devised.

The recursive XY Cuts algorithm works by successively dividing each region into two regions, always selecting the widest cut first, until no cut wider than a certain minimum threshold is reached. However, it should be noted that making a cut can never reduce the width of another potential cut, it can only be increased [2] or left the same. This implies that every cut candidate that is wider than the minimum threshold at a given level of recursion will continue to be wider than that threshold at all future levels of recursion. Since all cut candidates wider than the minimum threshold will eventually be reached in the recursion, it follows that at any given stage in the recursion, all cut candidates that are wider than the minimum threshold will eventually be cut by the algorithm.

This observation leads to a new version of the XY Cuts, termed Concurrent XY Cuts. In this version of the algorithm, rather than select only the widest cut at a given level of recursion, all cuts wider than the minimum threshold are immediately made. In the recursive version, the algorithm is often required to compute the width of certain cut candidates

[2]Imagine a 2-column layout in which 1cm and 2cm gap in the left and right columns respectively are lined up, creating a horizontal cut candidate between these two gaps 1cm wide. Separating the columns will leave this horizontal cut candidate 1cm wide in the left column, but will increase it to 2cm in the right column.

	NIPS Train	NIPS Test	JMLR Train	JMLR Test
# Segments	3445	1473	3521	2085
Cost Metric 1	0.39	0.41	0.21	0.20
Cost Metric 2	0.73	0.81	0.35	0.32
Cost Metric 3	0.22	0.18	0.18	0.18

Table 4.1: Segmentation results for the XY Cuts algorithm. Results are given as number of "errors" according to each cost metric per actual region in that data set.

multiple times, since only a single candidate is selected at a given level of recursion. The concurrent version produces the same results as the recursive, as can be seen from the following simple argument. Cutting a region can never *decrease* the width of a cut candidate. This means that, in the recursive version of XY Cuts, any cut candidate with a width greater than the threshold will eventually be cut. Cutting them all at the same time simply speeds up the algorithm, eliminating the waste of resources inherent in considering the same candidate multiple times. There is still a benefit, however, to the recursive version of the algorithm, in cases where the tree structure produced by the successive cuts is used for later analysis.

The second improvement relates to the two tunable parameters for the XY Cuts algorithm. These are the thresholds for the minimum cut width in each of the horizontal and vertical directions. Reasonable values for these parameters are between 10 and 20 pixels for horizontal cuts, and between 30 and 60 pixels for vertical cuts, at 2156 by 1728 pixel resolution. In most of the literature, these values are tuned by hand. Since the range of reasonable values is relatively small, it is often reasonable to perform an exhaustive search of the parameter space by testing the performance of a range of values on a collection of hand-segmented training data. The second segmentation performance metric described in Chapter 4.2.2 was selected for optimization. This metric is the best estimate available of the number of incorrect cuts made by the algorithm, and was thus appropriate for this purpose.

Table 4.1 shows the performance of the XY Cuts algorithm. Since the values of cost metric 2 are all close to twice those of cost metric 1, we can deduce that XY Cuts achieves a balance between over-segmentation and under-segmentation errors. This is expected, as the algorithm was optimized for the second performance metric (total errors), which we would expect to be minimized when the two types of errors are relatively balanced. The fact that performance is much stronger as measured by cost metric 3 indicates that a majority of the errors result from whitespace between ink regions with the same label. These regions are especially challenging, as they involve subjective decisions on the part of the person creating the ground-truth data as to whether the ink on either side of the whitespace belongs together or not.

Visual inspection of results shows two particular trouble areas. First is the use of lines of ink to separate regions, rather than whitespace. This is frequently done at the bottom of a page, where a footnote will be separated from the body of the page by a horizontal line. This problem is well known, and a solution has been suggested [4]. Second is the handling of references at the end of the paper. A decision was made to treat each reference as a single region, regardless of how close together they are. This has proven very difficult for XY Cuts to replicate. Finally, the handling of consecutive paragraphs of text has resulted in many errors. Most of these papers were likely put together using LaTeX, a document preparation system that dynamically sets the space between paragraphs, as well as before and after other regions (such as section headings) to produce aesthetically appealing layouts. The lack of consistency in the spacing of these regions makes it very difficult for XY Cuts to accurately reproduce them.

4.3.3 The Smearing Algorithm

The Smearing algorithm is another simple approach to segmentation. This is what is known as a bottom-up approach to the segmentation problem, in which the system begins with each individual mark of ink as a separate region, and then groups regions together to produce a segmentation.

As the name implies, this algorithm works by "smearing" the ink in each direction, and merging any regions that become connected by this process. There are a wide range of variants on this approach, including the well known Run Length Smoothing Algorithm [25].

For this study, a very simple version was used in which two images are created, representing the original image smeared in the x and y directions respectively. A logical OR operation is applied to these two images, creating a final image in which every pixel that is black in either of the smeared images is black in this new image. The connected components in this new image are identified, and represent the regions.

The smearing technique is able to find non-rectangular regions in this manner. Since the data in this study is restricted to rectangular, page-aligned regions, two post-processing step were applied. First, rectangular page-aligned boxes are drawn around each of the connected components, to take advantage of this restriction. Second, any overlapping regions are merged into a single region, so that all ink will be contained by exactly one region.

As with the XY Cuts algorithm, the simple smearing algorithm has two tunable parameters: how far to smear across the page, and how far to smear up and down the page. Following the same techniques outlined in Chapter 4.3.2, an exhaustive search of reasonable values for each journal was performed on the training data, yielding optimal parameters for this journal.

The simple smearing algorithm had comparable results to the recursive XY cuts algorithm. This is as expected, since the two algorithms are actually closely related. This becomes clear when the algorithms are viewed not as processes for splitting and merging regions, but as systems for answering the question "should this ink be together with that ink." The XY Cuts algorithm answers this question explicitly, deciding whether neighboring blocks should be kept together based on the width of the potential cut between them. The smearing algorithm implicitly answers this questions for any two adjacent marks of ink, deciding that they should be joined together based on the minimum distance between them in each of the x and y directions.

Note that if the XY Cuts algorithm were modified to cut based on the minimum distance across the cut, rather than the width of the cut according to the projection profile, they

	NIPS Train	NIPS Test	JMLR Train	JMLR Test
# Segments	3445	1473	3521	2085
Cost Metric 1	0.56	0.54	0.25	0.24
Cost Metric 2	0.75	0.75	0.41	0.39
Cost Metric 3	0.43	0.41	0.22	0.23

Table 4.2: Segmentation results for the simple smearing algorithm. Results are given as number of "errors" according to each cost metric per actual region in that data set.

would be working with identical "belongs-together" criteria. The XY Cuts algorithm would recursively separate regions until no separation of the minimum width remained. The smear algorithm would merge together any regions closer than the minimum separation, leaving all (and only) the separations greater than the minimum range.

While this observation about the similarity of the XY Cuts algorithm and the simple smearing algorithm may seem trivial, it is a vital understanding that will bring about a new approach to segmentation.

Table 4.2 provides the results for the simple smearing algorithm. Comparison with Table 4.1 shows that the smearing algorithm did not perform as well. Performance was similar by the second error metric, but noticeably worse by the first and third. This indicates that the information used by the smearing algorithm is not quite as effective as that used by XY Cuts. In particular, the errors made by the smearing algorithm are more often unambiguous errors, rather than subjective differences as to whether consecutive blocks of ink of the same type should be considered together or not.

4.3.4 The Voronoi Diagram Technique

The Area Voronoi Diagram technique is introduced by Kise et al. [11], where it is discussed in detail.

This is a single-pass bottom-up technique for grouping individual marks together. An "Area Voronoi Diagram" is constructed, in which for each each "off" pixel, the mark of ink to which it is closest is identified. The "Voronoi Area" for a mark of ink is the group of "off" pixels for which it is the closest mark. "Voronoi Edges" are the edges between different Voronoi Areas, and these edges are used to determine which marks of ink are adjacent to each other. Two sets of criteria are provided to allow adjacent marks to be joined together. Let a and b represent two adjacent marks on a page. Let D(a,b) be the minimum distance between a and b. Let A(a) be the area of a (measured in pixels). Adjacent marks are merged if either of these two criteria are met:

1.
$$\frac{D(a,b)}{T_{d1}} < 1$$

2.
$$\frac{D(a,b)}{T_{d2}} + \frac{\left(\frac{Max(A(a),A(b))}{Min(A(a),A(b))}\right)}{T_a} < 1$$

Essentially, if the marks are close enough together, they are automatically merged. If the marks are reasonably close together, and similar in size, they are also merged.

Kise et al. recommend that the value of T_a be set at 40, which is roughly the largest area ratio between characters of the same font [11]. The values of T_{d1} and T_{d2} are heavily dependent upon aspects of the layout of the page, such as line spacing and font size. A method for estimating reasonable values is provided in the paper.

It should be noted that this algorithm is related to both the XY Cuts and the simple smearing algorithm. Once again, the algorithm seeks to ask the question "do these bits of ink go together". Like the smearing algorithm, the Voronoi Diagram technique asks this question of each pair of neighboring marks of ink.

The Voronoi Diagram technique has several advantages over the XY Cuts and smearing techniques. First, it provides a method for estimation of the correct values of the two tunable parameters. Since, however, an exhaustive search of the parameter-space can be used to find optimal values for a given journal, this benefit is largely negated. Second, it evaluates each pair of regions based on two criteria, rather than just one (while XY Cuts and simple smearing do both have 2 parameters, only one generally comes into play for a

	NIPS Train	NIPS Test	JMLR Train	JMLR Test
# Segments	3445	1473	3521	2085
Cost Metric 1	0.90	0.84	0.68	0.67
Cost Metric 2	1.07	1.04	0.92	0.91
Cost Metric 3	0.88	0.82	0.58	0.54

Table 4.3: Segmentation results for the Area Voronoi Diagram technique. Results are given as number of "errors" according to each cost metric per actual region in that data set.

given pair of regions), creating the possibility of more flexible performance.

The Voronoi Diagram technique does, however, have two major disadvantages over XY Cuts and simple smearing. First, it runs much more slowly, as bottom-up approaches tend to. Second, it treats the horizontal and vertical directions as being equivalent. In scholarly journals, vertical spaces are much more likely to indicate a separation of regions than horizontal spaces are. The XY Cuts and simple smearing algorithms are able to take this into account by using different constants for each direction. The Voronoi Diagram technique considers only the distance between two items. We propose that the Voronoi Diagram technique could be improved by adding a third parameter A which would effectively stretch the y dimension, making the distance between two points now:

$$dist([x_1, y_1], [x_2, y_2]) = \sqrt{(A * (x_1 - x_2))^2 + (y_1 - y_2)^2}$$

Like the simple smearing algorithm, the Voronoi Diagram technique finds non-rectangular regions. A similar set of post-processing was applied to the results, in which rectangular bounding boxes are placed around regions, and overlapping regions are merged together.

Results for the Voronoi Diagram technique are shown in Table 4.3. This technique proved ineffective on our data. The results of the second performance metric are hardly higher than the first. This means that the heavy majority of errors were under-segmentation. The number of errors using the second error metric are close to, or even above, the total number of segments. These scores are not much better than would be achieved by simply selecting the entire page as a single segment. These two facts combined suggest that the Voronoi technique achieved its best performance, based on the second error metric, by grouping very large blocks of ink together, and thus reducing the number of errors it made by simply reducing the number of predicted segments.

4.3.5 Generalization of Past Approaches

As noted in Chapters 4.3.2 through 4.3.4, each of these past techniques attempts to provide a method for answering the question “do these bits of ink belong together?”. Most of the algorithms used to approach the segmentation problem can be characterized with just two pieces of information.

1. The criteria for whether bits of ink belong together.

2. The mechanism by which bits of ink are selected for comparison.

The bottom-up (or merge-based) versus top-down (or split-based) division of segmentation techniques reflects the second of these two criteria. Bottom-up techniques, such as simple smearing and Voronoi Diagrams, perform their comparison on each of the many neighboring pairs of marks. Top-down techniques, such as XY Cuts, make use of two shortcuts. First, they consider many marks concurrently, deciding whether groups of marks go together, rather than individual marks. Second, the top-down nature of the search means that many parts of the search tree are never reached, skipping the unnecessary evaluation of many pairs of marks.

While a top-down structure is generally more computationally efficient, such a structure generally requires assumptions to be made about the layout of the page. For example, the structure used in the XY Cuts algorithm requires that the page contain only rectangular page aligned regions, and that they are laid out in a table-like manner, so that the correct segmentation can be reached by making successive cuts. If this assumption does not hold, then the performance of any algorithm using this top-down approach will degrade.

Algorithm	Criteria For Marks Belonging Together	Structure
XY Cuts	Cut width (projection profile)	Top-down
smear	Cut width (minimum ink distance)	Bottom-up
Voronoi	1. Distance between marks. OR 2. Distance between marks + area ratio	Bottom-up

Table 4.4: Essential elements of past approaches to segmentation used in this study.

The selection of a top-down or bottom-up structure for an algorithm should, then, be regarded as a speed/accuracy trade-off, where an appropriate choice should be made based on what assumptions can be made about the structure of the pages being analyzed.

Table 4.4 provides a summary of the past approaches examined.

Two generalized structures for segmentation can thus be prepared: one for the top-down structure, and one for bottom-up. Different criteria can simply be plugged in to these algorithms by using a different "compare" function, allowing for easy testing of a variety of "belonging together" criteria.

```
function top_down(seg,dir):segments
  cut_cands=get_cut_cand(seg,dir);
  for each cut_cand in cut_cands:
    if should_cut(cut_cand):
      valid_cuts.add(cut_cand);
  tmp_segs=make_all_cuts(valid_cuts);
  for each s in tmp_segs:
    segments.add(top_down(s,~dir));
  return segments;

function bottom_up(image):segments
  Find individual ink marks.
  Find neighboring marks.
  For each 2 neighboring marks m1 and m2:
    If belong_together(m1,m2):
      should_merge.add(m1,m2)
  For each pair of marks m1,m2 in should_merge:
    Merge m1 and m2 into a new mark m3, recording the fact that m1 and m2
    are now stored in m3, so that m1 and m2 can still be merged with
    other regions.
  Create segments by drawing boundaries around the resulting marks.
  Return the resulting segments.
```

4.4 Learning-Based Approaches

The "belonging together" criteria for all of these past techniques all make use of very little information. Since little information is used, these algorithms require very few tunable parameters. There is a significant advantage to having few such parameters, in that the optimal values for these parameters can be found through an exhaustive search of the parameter space. The disadvantage of these techniques, however, is that the small amount of information they examine is simply not enough to make correct decisions consistently.

As Chapter 4.3.5 has shown, the core of each of the traditional segmentation algorithms discussed is the criteria for whether two marks belong together. This observation raises the possibility of treating segmentation as a supervised classification problem, in which marks (or groups of marks) are classified as "belonging together" or "not belonging together". Treating this as a classification problem allows for much more information to be used in the consideration of whether marks belong together. The remainder of this chapter will provide the details of two attempts at turning segmentation into a supervised classification problem.

As discussed in Chapter 4.3.5, the top-down versus bottom-up distinction is essentially a structural decision. Since the data set used in this study fits the assumptions of the top-down algorithm structure, a top-down approach shall be followed in these two attempts at turning segmentation into a supervised classification problem.

The first algorithm, called “concurrent learn-to-cut”, will follow the obvious approach of treating each potential cut as an isolated classification problem, labeling each cut candidate as *valid* or *invalid*. The second algorithm, called “concurrent learn-to-segment”, will consider cuts in pairs, rather than individually. Essentially, the second algorithm will be comparing the segments that could be created by a given set of cuts, rather than evaluating the cuts themselves.

4.4.1 Preparing Training Data

A *cut candidate* is any line through which a segment of the page could be cut without intersecting any ink. Some slightly more stringent conditions for cut candidates may be used to speed up the algorithm, at the risk of a loss of accuracy. For this study, “cut candidates” are only considered if the cut width is at least some minimum number of pixels. The thresholds we selected were 1 pixel for horizontal cuts (ensuring all horizontal cut candidates are considered), and 5 pixels for vertical cuts. The selection of the threshold of 5 pixels for vertical cuts prevents each individual letter from being considered as a region, while missing very few actual cuts.

A *cut feature* is a real number that describes some aspect of the cut itself. The term feature is used here in exactly the same manner as in Chapter 2. The values used by past cutting algorithms (such as the width of the whitespace created by the cut) are prime examples of cut features. All of the cut features from a given cut can be joined together into a feature vector for the cut.

Before a learning algorithm can be applied to this problem, a large number of training samples must be extracted from the hand-labeled training data. The *Concurrent Correct Cuts* [3] algorithm described here provides a simple method for doing this. As long as the sample pages follow a Manhattan layout, this algorithm will examine each cut candidate

[3]The term *concurrent* is used to indicate that this algorithm allows multiple cuts to be made in each pass, a practice not used in most cuts-based algorithms in the literature.

exactly once, extracting the required cut features from it for training.

The idea behind the algorithm is simple: recursively make all of the cuts in one direction that do not intersect any actual region in the hand-tagged version of the page (these are termed *valid* cuts), alternating between vertical and horizontal. Each time a segment is examined for cuts, each cut candidate is passed to a feature extraction function, creating a sample for the learning algorithm. Pseudo-code for the algorithm is provided below.

```
function correct_cuts(pixels, current_segment, vertical_pass) : samples
  If this is a vertical pass
    Find the vertical cut candidates
  If this is a horizontal pass
    Find the horizontal cut candidates
  Generate samples by extracting features from each cut candidate
  For each cut candidate
    If this cut candidate intersects a region in the actual segmentation:
      Remove this cut candidate from the list
  If any cut candidates remain on the list
    Generate N+1 new segments from the N remaining cut candidates
    For each new segment
      Call correct_cuts(pixels, this_segment, ~vertical_pass), and
      add what it returns to the list of samples
```

4.4.2 The Concurrent Learn-To-Cut Algorithm

The Concurrent Learn-To-Cut algorithm assumes that whether or not to make a given cut can be determined using only information about the cut itself, ignoring the context of other potential cuts and segments. This assumption is enforced by allowing the cut features to be constructed from only the following information:

- The raw page image.
- The location of the cut.
- The parameters of the segment before this round of cuts
- The current stage of page cutting
- The page number

A full listing of the features used is available in Table 4.5.

Description	**Values**	**# Feats**
How wide of a valley is being cut		1
How long is the cut		1
Total cut area (width * height)		1
Min ws separation above & below cut, to a max of NN pixels each way	80,40,30,25,20,15,10,5	7
Average ws height in a block extended NN pixels above the cut.	80,40,30,25,20,15,10,5	7
Total ws area in a block extended NN pixels above the cut.	80,40,30,25,20,15,10,5	7
Average ws height in a block extended NN pixels below the cut	80,40,30,25,20,15,10,5	7
Total ws area in a block extended NN pixels below the cut.	80,40,30,25,20,15,10,5	7
Percent whitespace on a row NN pixels above cut	-20 through 20	41
Pct ws in the NN% last cols in the cut, extended 80 pixels up & down	80,40,30,25,20,15,10,5	7
Pct ws in the NN% middle cols in the cut, extended 80 pixels up & down	80,40,30,25,20,15,10,5	7
Pct ws in the NN% first cols in the cut, extended 80 pixels up & down	80,40,30,25,20,15,10,5	7
Boolean: Is this on the first page of the article		1
Boolean: Is this on the last page of the article		1
Boolean: Is this on the last 15% of the pages in the article		1
Fraction of the way through the article's pages		1
Total number of features		**104**

Table 4.5: The 104 features used for Concurrent Learn-To-Cut. The first column describes the type of feature. For features where a range of different values for some property were used to create multiple features, the second column lists the different values used. The third column indicates how many features of that type were created. The features were normalized to ensure that each ranged from [0,1] in the training data.

The hand-labeled pages are passed to the Concurrent Correct Cuts algorithm, generating a collection of feature vectors that will act as training samples, as described in Chapter 4.4.1. An extra "always 1" feature (known as a bias feature) is appended to each feature vector. These samples are passed to a logistic regression learning (described in Chapter 2), which finds the linear hyperplane in the feature-space which best separates the *valid* and *invalid* classes. This hyperplane is described by a single vector orthogonal to the hyperplane, called the weight vector. A new sample can be multiplied by the weight vector, and if the result is greater than 0, then it has been classified as *valid*, and if it is less than 0, as *invalid*.

The same weight vector need not be used for all cuts. The valid-cut criteria for vertical cuts are clearly different from those for horizontal cuts, so these were given separate weight vectors. Additionally, it was noted that cuts made in the first pass tend to be through long, broad areas of whitespace (such as the separation between columns of text, or between the main title of a paper and the abstract), whereas cuts in further passes tend to be through smaller regions of whitespace (such as the space between a figure and its caption). From these observations, it was decided to use four different weight vectors for this study, each corresponding to one of these four categories of cut candidates:

- First vertical pass
- First horizontal pass
- Other vertical passes
- Other horizontal passes

Cut candidates from each category were passed separately to the logistic regression learner, generating four different weight vectors to be used in different contexts.

The segmentation algorithm for Concurrent Learn-To-Cut is almost identical to the Concurrent Correct Cuts algorithm. The only difference is that instead of looking at the hand-labeled segmentation information to decide whether or not to make a given cut, Concurrent

Learn-To-Cut uses the logistic regression weight vector to decide whether or not each cut should be made.

```
function learn_to_cut(pixels,segment,lr_weights,vertical_pass):segments
  If this is a vertical pass
    Find the vertical cut candidates
  If this is a horizontal pass
    Find the horizontal cut candidates
  For each cut candidate
    Run the feature extraction function, creating a feature vector
    Multiply the feature vector by the lr_weights.
    If the result is < 0, then
      Remove this cut candidate from the list
  If any cut candidates remain
    Generate N+1 new segments from the N remaining cut candidates
    For each new segment
      Call learn_to_cut(pixels,new_segment,~vertical_pass), and
      add what it returns to the list of segments
  Else no cuts remain
    Add segment to the list of segments
  Return the list of segments
```

It should be noted that while logistic regression will find the weight vector that maximizes the log likelihood of the training samples, it is unlikely to find the weight vector that yields the best test-time performance on the training data. The reason for this is that certain errors in deciding whether or not to cut (the term “cutting errors” shall be used) will cause more segmentation errors than other cutting errors. Consider the example in Figure 4.2. If the cut shown with a dashed line is missed, then a cut separating the two regions above will never

	NIPS Train	NIPS Test	JMLR Train	JMLR Test
# Segments	3445	1473	3521	2085
Cost Metric 1	0.21	0.29	0.11	0.14
Cost Metric 2	0.41	0.56	0.22	0.28
Cost Metric 3	0.15	0.19	0.08	0.11

Table 4.6: Segmentation results for the Concurrent Learn-To-Cut Algorithm. Results are given as number of "errors" according to each cost metric per actual region in that data set.

even be considered as a cut candidate. This is in fact a very typical example of a cutting error, demonstrating that one cutting error will often result in multiple segmentation errors.

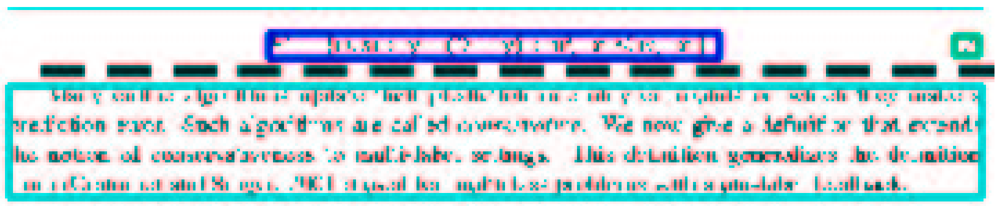


Figure 4.2: If the cut shown with a dashed line is not made, then a cut separating the two regions above it will never even be considered. In this case, a single cutting error will cause three actual segments not to be found by a top-down algorithm.

If the number of segmentation errors induced by each cutting error could be considered while performing logistic regression training, the Concurrent Learn-To-Cut algorithm could be significantly improved. How to consider the number of segmentation errors induced by one cutting error in the learning process is left as an open problem.

Results of Concurrent Learn-To-Cut are presented in Table 4.6. Comparing them with the optimized results for XY Cuts in Table 4.1 shows that the Learn-To-Cut algorithm performed extremely well. Despite the problems with the algorithm noted above, Concurrent Learn-To-Cut outperformed XY Cuts in nearly all categories. This indicates that the features provided do actually contain additional relevant information that is not used by the XY Cuts algorithm.

4.4.3 Concurrent Learn-To-Segment

The Concurrent Learn-To-Segment algorithm is closely related to the Concurrent Learn-To-Cut algorithm. It follows the pattern of examining all cut candidates in either the horizontal or vertical direction in each sweep. It does not, however, make the assumption that the information needed to decide whether or not to make a cut is contained exclusively in that cut.

The Concurrent Learn-To-Segment algorithm assumes that the information to decide whether or not to make a cut is contained within the new segments that would be created by making a cut.

The feature extraction function for Concurrent Learn-To-Segment is allowed to use the following information:

- The raw page image.
- The locations of the two cuts being considered.
- The parameters of the segment before this round of cuts
- The current stage of page cutting
- The page number

The sample collection algorithm is a slightly modified version of the Concurrent Correct Cuts algorithm. The only modification is in the creation of feature vectors from the cut candidates. Rather than use each cut candidate to create a single feature vector, all possible pairs of cut candidates are considered. Additional cut candidates are added at the beginning and end of the page, to ensure that the first and last segments are considered. The space between each pair of cut candidates makes up the "segment" to be considered. Each such segment is considered valid if its edges do not intersect any actual regions in the hand labeled data, and the segment is as finely divided as can be done at this stage of the segmentation (in other words, if there are no other valid cut candidates parallel to the cuts being made that are contained by the region). The segment is hence considered to be valid if and only if both of these properties hold:

1. The two cut candidates that define the segment are valid cut candidates (ie. they do not intersect any regions in the correct segmentation).
2. There are no valid cut candidates between them (in other words, the segment is as small as can be created in the current pass).

A large number of feature vectors can be created from each page. Since each possible pair of segments is considered, each pass creates a number of feature vectors proportional to the square of the number of cut candidates.

As with Concurrent Learn-To-Cut, these feature vectors are labeled as *valid* and *invalid*, and passed to a logistic regression learning algorithm to create four pairs of weight vectors:

- First vertical pass (valid and invalid)
- First horizontal pass (valid and invalid)
- Other vertical passes (valid and invalid)
- Other horizontal passes (valid and invalid)

Performing segmentation on a new page is significantly more involved than with Concurrent Learn-To-Cut, since the validity of any given cut depends on the locations of the other cuts. In Concurrent Learn-To-Cut, the best assignment of *valid* and *invalid* labels over each pass is found simply by assigning each cut candidate the label (either cut or nocut) that has the highest log likelihood according to the appropriate pair of weight vectors. Concurrent Learn-To-Segment has two additional complications:

1. It is not immediately clear what constitutes the “best” assignment of cut/nocut labels.

2. The best assignment of *valid* and *invalid* labels over each pass now needs to consider entire sequences of *valid* and *invalid* labels, since the labeling of each cut is affected by the locations of other cuts.

It is not entirely clear how to decide what constitutes the "best" segmentation for a pass in the Concurrent Learn-To-Cut algorithm.

The intuitive choice would be to select the combination of segments (pairs of cuts) that produces the maximum sum of the log-likelihoods of the *valid* labels for those segments. This is, however, incorrect, and would encourage the system to under-segment. Since likelihood estimates fall between 0 and 1, all log-likelihoods must be less than zero. To maximize the sum of the log-likelihoods, the system would tend to choose segmentations that reduce the number of segments.

Consider the typical example of a page with 6 equations, with several lines of text separating the equations. Let us consider two alternate segmentations of this page: one in which the entire page is considered to be a single segment, and another in which each block of text and equation is a separate segment. Since the latter is the correct segmentation, this is what we would want the system to select. Suppose, however, that the entire-page segment was given a likelihood of 0.1 (very unlikely), and each correct segment was given a likelihood of 0.8 (quite likely). The sum of the log likelihoods for the single-segment version is -2.30, whereas the sum of the log likelihoods for the correct segmentation is notably lower at -2.68. Clearly this is not the correct manner of evaluating the quality of a sequence of cut/nocut labels.

In fact, the correct manner of evaluating the quality of a sequence of *valid/invalid* labels is by summing up the log likelihoods of *all possible* segments, with their attached (segment/not-a-segment) labels under the current sequence of cut/nocut labels. The fastest algorithm for doing this will first compute the log likelihoods of all possible segments in the sequence, and then proceed to find the best possible combination of them.

This leaves only the challenge of searching the possible sequences for the one that produces the best score. The number of possible sequences grows exponentially with the number of cut candidates, making an exhaustive search intractable.

Since the algorithm is restricted to examining pairs of cuts, we can use a dynamic programming algorithm modeled loosely on the Viterbi algorithm [16] to find the best sequence of cuts in a runtime proportional to the square of the number of cut candidates. The intuition behind this algorithm is that the best sequence in which cut candidate k is labeled *valid* must contain the best sub-sequence of labels before cut candidate k in which cut candidate k is labeled *valid*. The recursion is shown below:

Notation:

- A *path* is a sequence of (*valid*/*invalid*) labels assigned to cut candidates.
- $score(path)$ is the sum of the log likelihoods for all of the *valid* and *invalid* regions for *path*.
- $bestpath(i)$ is the best path up to candidate i, in which candidate i is considered a *valid* cut.
- $padpath(subpath, i)$ is a path beginning with *subpath*, and then padded with *invalid* labels until candidate i, which is labeled *valid*.

Recursion:

$$prev - cut(i) = argmax[j < i](score(padpath(bestpath(j), i))$$

$$bestpath(i) = padpath(bestpath(prev - cut(i)), i)$$

An implementation of this recursion, which will be referred to throughout this document as the Best Cut Sequence (BCS) Algorithm, is provided in Figure 4.3. Note that this

```
variables:
  -path is a sequence of valid or invalid assignments to cut candidates.
   Note that [path;valid] means the sequence path with an additional
   valid tag added to the end.
  -paths is a list of possible sequences of valid or invalid assignments
   under consideration.  At any given time, paths(i) represents the
   highest scoring path in which the i'th cut candidate is the last valid
   cut candidate.
------------------------------------------------------

function path = padpath(path,pathlen)
  for i = (length(path) + 1) to (pathlen - 1)
    path = [path;invalid]
  path = [path;valid]
  return path
------------------------------------------------------

function path = BestCutSequence(segment,cut_cands,pixels)
  Set path = [valid]
  Add path to path_list
  For i = 2 to length(cut_cands):
    Set best_score = -infinity
    For j = 1 to length(cut_cands) each path in paths:
      path = padpath(path,i)
      If score(path,pixels) > best_score:
        best_score = score(path,segment,pixels)
        paths(i) = [path;valid]
  return paths(length(cut_cands)
```

Figure 4.3: The function BCS returns the best path in which the first and final cuts are assumed to be valid. The algorithm assumes that cut_cands includes candidates located at the first and last pixel in the segment, which translates into assuming that the segment boundaries are valid locations for cuts. The validity of this assumption must be assured by the way the algorithm is used.

implementation assumes that the list of cut candidates has already been "padded" with cuts at the very beginning and very end of the region begin examined.

As with the Concurrent Learn-To-Cut algorithm, the Concurrent Learn-To-Segment algorithm neglects the fact that certain cutting errors will induce more segmentation errors than others. The illustration in Figure 4.2 demonstrates this. Adapting the training procedure for Concurrent Learn-To-Segment to take into account the number of segmentation errors induced by each cutting error is left as an open problem.

Unfortunately, the results of the Concurrent Learn-To-Segment algorithm were very poor. The most likely cause of this was a poor set of features used in the algorithm. The features used in this algorithm were adapted from those used in Chapter 5 on the labeling problem. While these features proved very effective in distinguishing among the various types of regions, they proved very poor at telling which regions are actually regions.

Given the poor performance and processor-intensiveness of the algorithm, time considerations prevented detailed quantitative study of its performance.

Chapter 5

Labeling

5.1 Problem Description

The labeling problem is known by several names in the literature, including Block Classification [3], Region Classification [1], and Document Logical Structure Analysis [15]. To tie the labeling problem in with Machine Learning literature, it should be noted that it can also be described as a classification problem, in which each region is a unique datum which needs to be classified into one of several categories. The terms "class" and "label" will be used interchangeably throughout this paper.

The essence of the problem is quite simple: given a page, and a division of this page into regions, assign the correct labels to each region.

There are several points of subjectivity in the labeling problem. To further complicate matters, the subjective aspects of the labeling problem are tied in to the subjective aspects of the segmentation problem.

The first point of subjectivity in the labeling problem is the question of what labels to use. There is no single set of clearly correct labels to assign to regions. Cattoni et al. provide a summary of the sets of labels used by various different authors on [3]. In addition to the minor problem of some regions having several possible names (such as *text* versus *paragraph*), there are the major problem of deciding how finely to divide the classes, and how finely to segment the page (which influences which classes are used). For example, much of the literature works with a very small set of classes based on a coarse segmentation of the page, such as *text*, *pictures*, and *background*. Others, however, work with finer segmentations of the page, and use classes such as *words* and *text lines*.

A small set of classes is appropriate for many applications, such as acting as a pre-processor for optical character recognition. For such applications, there are usually only a small number of distinct steps to be taken in dealing with different types of regions. For example, an OCR pre-processing application needs only to distinguish text regions, which are left in the original image, from other regions, such as figures and images, which are removed from the page image and saved to a separate image file.

Other applications, however, require more detailed information about the labels assigned to regions. For example, a search engine pre-processor would like to provide as finely grained a classification scheme as possible. This would allow user to search, for example, for all documents with the words “segmentation” and “comparison” in a figure caption.

This fine-grained classification can be attempted by performing optical character recognition on the text regions, and then using some sort of lexical analysis to further subdivide the *text*. The merits of performing this fine-grained classification directly on the page image are, however, worth investigating.

The hand-labeled test data for this project has been labeled using a very fine-grained system. In addition, a mapping from this fine-grained labeling to a more coarse-grained labeling has been provided. This allows the sample set to be used for both fine-grained and coarse-grained labeling. The fine-grained (or *precise*) labels, along with their corresponding coarse-grained (or *generalized*) counterparts, are shown in Table 5.1, and a detailed description of the labels used is provided in Appendix A.

Precise Category	Generalized Category
text	*TEXT*
header	*TEXT*
section heading	*TEXT*
subsection heading	*TEXT*
figure label	*TEXT*
figure caption	*TEXT*
references	*TEXT*
abstract	*TEXT*
bullet item	*TEXT*
page number	*TEXT*
main title	*TEXT*
footer	*TEXT*
table label	*TEXT*
table caption	*TEXT*
editor list	*TEXT*
equation number	*TEXT*
author list	*TEXT*
footnote	*TEXT*
decoration	*OTHER*
equation	*EQUATION*
table	*FIGURE*
image	*FIGURE*
graph	*FIGURE*
figure	*FIGURE*
code block	*FIGURE*

Table 5.1: The 25 fine-grained labels used, and the 4 generalized labels into which they were aggregated. Descriptions of the labels can be found in Table 3.1.

5.2 Approaches to Labeling

In their review of past approaches to segment labeling, Mao et al. note that nearly all past approaches to the labeling problem have used rule-based systems and decision trees to label the regions [15]. As rule-based systems alone generally prove too rigid to adequately discriminate between the labels, recent work has turned to using tree structures to represent pages, as well as page grammars to resolve ambiguity in the rules.

Krishnamoorthy et al. [12] observe that the segmentation and labeling problems can be tackled concurrently. They use an algorithm structure very similar to the generalized top-down approach presented in Chapter 4.3.5. At each stage of segmentation, either the horizontal or vertical profile (depending on the stage of segmentation) is used to create a string of tokens for a context-free grammar. These tokens are then parsed, disambiguating any non-terminal tokens, leaving the regions with assigned labels. These labels are used at the next stage in the recursive cutting process to determine which grammars should be used. While the rules for this system make use only of the projection profile information, the use of grammars is expected to improve performance, since it increases the information used in the labeling process. This system has a major weakness, however, in that "correct" grammars need to be specified by hand for each type of journal.

Dengel works with a tree-based description of document classes, referred to as a "Geometric Tree" [6], or "GTree" [7]. This tree acts as a model for a class of documents. Each non-terminal node in the tree represents a partially-specified document layout. Each terminal node represents a fully specified document layout. It is termed a "Geometric Tree" because the children of a given node on the tree are formed by dividing some segment of the document either horizontally or vertically. The non-terminal nodes in the tree also act as nodes in a decision tree, specifying rules for which child nodes to consider. Some work has been done in developing systems to build GTrees in an automated manner based on examples [8].

Imade et al. made use of neural networks to discriminate between printed characters, handwritten characters, photographs, and painted images using a narrow set of features [10]. For each region, one or more square blocks of pixels are selected at random, and histograms were formed for the luminosity (32 bin histogram) and gradient vector direction (24 bin histogram) of the pixels contained. The values in these histograms formed 54 numerical values passed to the input layer of the neural network. While Imade reports classification success rates of 100%, these appear to refer to success rates of multiple attempts to classify the same 4 regions (one of each type) using different randomly selected pixel blocks within those regions, rather than classification success rates on a collection of different regions.

Summers et al. use a variant on a rule-based system that allows for some deviation from the rules [24]. A collection of attributes are specified for each region, along with distance measures for each attribute. The distance between two segments is defined as the average of the distances between the attributes of those two segments. A "prototype" segment is specified by hand for each type of label. Each new label to be classified has its distance computed from each prototype, and is assigned the label of the nearest prototype. Their implementation of this system requires that the document's segmentation be specified in terms of a tree, in which each node in the tree represents a partially segmented region. Labels are assigned to all nodes in the tree (not just the leaves), and the parent and child information for a given node is used in labeling. The labels used by Summers et al. are relatively similar to those used in this research, consisting of 16 "precise" categories, and an unspecified (but smaller) number of "generalized" categories. This similarity of labels makes this work appropriate for comparison to our results.

5.3 Labeling As A Supervised Learning Problem

Machine Learning is the discipline of teaching computers to recognize patterns. One of the best-known sub-disciplines is supervised classification. The goal of supervised classification is to come up with an algorithm that can learn to assign the correct class labels to items simply by examining a collection of pre-classified examples.

When a sufficiently large ground truth dataset is available, region labeling clearly falls into the category of a supervised classification problem. Once the individual regions have been identified, labeling these regions is equivalent to assigning each region to a class.

A variety of supervised classification techniques are outlined in Chapter 2. All of these techniques require that the items to be classified be described by a fixed number of numerical values, called features. Features can either be real-valued, taking on any of a range of values, or discrete, with only certain values allowed. Each region will be presented to the classification algorithms exclusively in terms of these features, thus the effectiveness of any algorithm depends heavily on the use of features that lend themselves to distinction between the different classes.

In this thesis, 59 numerical features were extracted from each region. Each feature was drawn from one of four broad types: the location of the region on the page (and those of its neighbors), the density of the ink inside the region, the number of the page that the region appears on, and the individual marks of ink within the region. The majority of the features are real-valued, such as the area of the region. The remainder of the features are binary, with values of 0 and 1 allowed. A detailed listing of the features extracted can be found in Table 5.2.

The ground truth data was divided into two subsets: a set of training data from which the algorithms are expected to learn how to classify regions, and a set of test data, upon which the performance of the trained algorithms was measured. For each of the two data sets studied (NIPS and JMLR), about 2/3 of the data was placed in the training set, and the remaining 1/3 was placed in the test set. Individual articles were not split up and placed partly in each; each article was placed either entirely in the training set or in the test set. It should be noted that this actually makes the learning problem somewhat more challenging, since each author may have their own unique divergence from the standard formatting for a given journal. If that author's work is entirely in the training set, it may confuse the algorithm during training, resulting in worse performance on articles that follow the standard format. If, however, the author's work is entirely in the test set, then there is no way for the algorithm to have learned that author's peculiarities. This particular division of test and training data was done in an attempt to accurately simulate how such a system may actually be used in the future: a small number of hand-tagged articles from a given journal would be taken as a training set, and the system would then be expected to work

Feature Description	**# Feats**
Distance from each edge of the rectangle to the corresponding edge of the page	4
Distance from each edge of the rectangle to the next ink in that direction	4
Width, height, area, and log of the aspect ratios	5
Boolean: is it within 10 pixels of being centered	1
Is there any ink between it and each edge of the page	4
Is there any ink closer to each edge of the page than in this region	4
Distance from each edge of the rectangle to the nearest other region box	4
Fraction of the neighbor in each direction covered by a projection of this region	4
Fraction of this region covered by a projection of the neighbor in each direction	4
Fraction of the way through the estimated reading order of the page	1
Fraction of "on" pixels in the region (ink density)	1
"Snapped region" ink density	1
Horizontal Sharpness (helps distinguish text from non-text)	1
Boolean: does it contain a horizontal line across the region	1
Boolean: does it contain a vertical line across the region	1
Fraction of ink in the leftmost quarter of the region	1
Number of ink marks in the region	1
Number of marks per 100 pixels	1
Number of marks per pixel wide	1
Number of marks per pixel high	1
Average number of black pixels per mark	1
Standard deviation of number of black pixels per mark	1
Hight, width, area, and number of pixels in the largest mark	4
Height and width of the highest and widest marks	4
Boolean: Is this the first page in the article	1
Boolean: Is this the last page in the article	1
Boolean: Is this page in the last 15% of the pages in the article	1
Fraction of the way through the article	1
Total # of Features	59

Table 5.2: The 59 features used for the labeling problem. The first column describes the type of feature, and the second column indicates how many features of that type were created. The features were normalized to ensure that each ranged from [0,1] in the training data.

well on other papers from that journal.

Many of the learning algorithms used are sensitive to the scale of the values used. In other words, they behave slightly differently with features whose numerical magnitude is large than with those whose magnitude is small. To ensure that all features were considered equally, all of the features have been normalized so that their values are expected to fall in the range [0,1]. The maximum and minimum values of each feature was found for each set of training data. These values were stored and used to normalize all feature values using the formula:

$$normalizedValue = \begin{cases} (originalValue - min) \, / \, (max - min) \, , \; if \; max > min \\ (originalValue - min) \, /1, \; if \; max = min \end{cases}$$

The max and min drawn from the training set were used to normalize all feature values, at both training time and at test time. While this raises the possibility of values outside the range [0,1] at test time, it ensures consistency of normalization across the full data set.

5.4 Evaluation of Performance

Performance evaluation is performed in a standard manner for machine learning classification problems. A set of test data is prepared, and the performance of the algorithm is based on how many of these test data are correctly and incorrectly classified.

This study performs a more precise classification than is common in the literature. Most studies classify the regions into a small number of classes, as shown in [3]. To facilitate comparison with these studies, two different methods of classification performance evaluation have been used.

Each of the classes used in the test data is a member of one of the four generalized classes *text*, *equation*, *figure*, and *other*. Performance numbers are provided both in terms of the

25 fine-grained classes, as well as in terms of the 4 generalized classes. The mapping of fine-grained classes to the generalized classes is shown in Table 5.1.

5.5 Classification Methods Considered

Three different classification methods were considered for this thesis: K Nearest Neighbors, logistic regression, and maximum entropy Markov models. The following chapters describe how these methods were applied, and the results achieved.

5.5.1 K Nearest Neighbors

This is one of the simplest supervised learning methods available, yet often produces reasonable results. It thus serves as a good benchmark for more complex learning algorithms. A detailed description of how K Nearest Neighbors works is found in Chapter 2.

One benefit of K Nearest Neighbors is that there is no real training phase for the algorithm. Testing simply consists of comparing new data points to the data points in the training set. Thus assembling the training set in a usable form is all that needs to be done prior to testing. In this particular case, the feature extraction code takes several seconds to run on each page, thus the most usable form of the data is a large matrix, each row of which represent the pre-calculated feature vector for a single region. This matrix thus represents the training data for the K Nearest Neighbors algorithm.

A related advantage of K Nearest Neighbors is that there is only one parameter whose value needs to be determined: the value of K. A technique called Leave One Out Cross Validation was used to determine a reasonable value of this parameter. With this technique, a single row is removed from the training matrix. That row is then classified using the K Nearest Neighbors technique with the remainder of the training matrix. The result is recorded, and the row is then returned to the training matrix. This process is repeated for each row in the training matrix, providing with an estimate of how well the algorithm will work on an arbitrary region when that one region is left out of the training matrix. This is repeated for several values of K between 1 and 10, with the best results coming from K=3, and results getting gradually worse from 4 to 10.

The largest drawback of K Nearest Neighbors is its relatively slow performance and memory-intensiveness at test time. For example, the training set for NIPS has a total of 3445 regions, each of which has 59 real-valued features. In order to classify a new data point, the distance to each of these 3445 training regions must be calculated, with each calculation considering all 59 values in the feature vector. While this slow performance is not of consequence in a lab setting, it could prove to be a major barrier to industrial use of this technique. Fortunately, several techniques for improving the speed of K Nearest Neighbors are available that could be used in an industrial setting.

K Nearest Neighbor Results

The K Nearest Neighbor algorithm produced good results, with an average fine-grained accuracy of 82.2% on test data, and an overall generalized accuracy of 96.7% on the test data. Recall that the fine-grained classification involves a total of 25 classes, whereas the generalized classification reflects whether they were correctly classified in the 4 broader categories of *Text*, *Equation*, *Figure* (including tables), and *Other*.

A *confusion matrix* is a grid that indicates the full distribution of the items' correct and predicted classes. This is a simple method to see where a given algorithm is making mistakes. Several confusion matrices for the K Nearest Neighbors algorithm are provided in the tables below. These summarize both the coarse and fine grained results for both the test set and the training set (using Leave-One-Out Cross Validation) for each of the two data sets used.

The errors that are fairly predictable: the algorithm is tending to confuse relatively similar items. The most common errors include confusing Text and References regions, confusing Section Headings and Subsection Headings with each other, as well as with Text, and confusing Text and Bullet Items.

	text	authour list	section heading	main title	decoration	footnote	abstract	eq number	equation	graph	table	table caption	figure caption	references	subsection heading	image	bullet item	code block	figure	figure label	table label	header	editor list	pg number	footer
text	917	0	7	0	1	21	0	6	8	1	0	4	31	6	4	0	19	8	9	0	0	0	0	0	0
authour list	0	69	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
section heading	17	0	236	0	0	0	0	0	1	2	0	0	0	1	48	0	1	0	0	0	0	0	0	0	0
main title	0	0	0	39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
decoration	0	0	0	0	157	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
footnote	25	0	0	0	0	48	0	0	0	0	0	0	2	1	0	0	1	0	0	0	0	0	0	0	0
abstract	1	0	0	0	0	0	39	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
eq number	3	0	0	0	0	0	0	358	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0
equation	6	0	0	0	0	0	1	1	413	0	1	1	7	0	0	0	0	0	2	0	0	0	0	0	0
graph	2	0	1	0	2	0	0	0	2	71	2	0	3	0	0	3	0	2	21	0	0	0	0	0	0
table	0	0	0	0	0	0	0	0	0	1	8	0	0	0	0	0	0	4	3	0	0	0	0	0	0
table caption	7	0	0	0	0	0	0	0	3	1	0	3	3	0	0	0	2	1	1	0	0	0	0	0	0
figure caption	22	0	0	0	0	4	0	2	4	2	0	3	48	1	0	1	0	1	3	0	0	0	0	0	0
references	7	0	0	0	0	1	0	0	1	0	0	2	5	432	0	0	1	0	0	0	0	0	0	0	0
subsection heading	12	0	50	0	0	0	0	0	0	0	0	0	0	0	52	0	2	0	0	0	0	0	0	0	0
image	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0
bullet item	12	0	1	0	0	1	0	0	5	0	0	2	1	0	1	0	12	0	0	0	0	0	0	0	0
code block	4	0	0	0	0	0	0	0	0	3	6	0	1	0	0	0	0	4	2	0	0	0	0	0	0
figure	0	0	0	0	1	0	0	0	1	14	3	1	2	0	0	0	0	2	25	0	0	0	0	0	0
figure label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
header	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
editor list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pg number	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
footer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Success %	89	100	80	100	98	64	98	97	94	72	40	19	44	98	50	60	32	18	38	0	0	0	0	0	0

Table 5.3: Full confusion matrix for NIPS K Nearest Neighbors results on the training data. The top row indicate the correct label, the left column indicates the predicted label.

	text	authour list	section heading	main title	decoration	footnote	abstract	eq number	equation	graph	table	table caption	figure caption	references	subsection heading	image	bullet item	code block	figure	figure label	table label	header	editor list	pg number	footer
text	986	0	12	0	0	8	0	0	15	0	0	0	0	8	16	0	30	5	7	1	0	0	0	0	0
authour list	0	39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
section heading	15	0	37	0	0	0	0	0	0	0	0	0	0	1	27	0	5	0	0	0	0	0	1	0	0
main title	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
decoration	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
footnote	9	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
abstract	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
eq number	3	0	0	0	0	0	0	193	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
equation	9	0	0	0	0	0	0	0	504	0	1	0	1	1	0	0	0	1	2	0	0	0	0	0	0
graph	0	0	0	0	0	0	0	0	0	59	1	0	0	0	0	0	0	1	8	0	0	0	0	0	0
table	0	0	0	0	0	0	0	0	2	0	24	0	0	0	0	0	0	0	1	0	0	0	0	0	0
table caption	2	0	0	0	0	0	0	0	0	0	0	10	13	0	0	0	0	0	0	2	0	0	0	0	0
figure caption	2	0	0	0	0	0	0	0	2	0	0	14	81	0	0	0	0	0	2	1	0	0	0	0	0
references	10	0	3	0	0	0	0	0	2	0	0	0	0	265	0	0	0	0	1	0	0	0	0	0	0
subsection heading	28	0	40	0	0	0	0	0	0	0	0	0	0	0	62	0	6	0	0	0	0	0	0	0	0
image	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bullet item	30	0	2	0	0	0	0	0	2	0	0	0	0	0	3	0	29	0	1	0	0	0	0	0	0
code block	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	2	0	0	0	0	0	0
figure	3	0	0	0	0	0	0	0	0	7	2	0	0	0	0	0	0	2	33	0	0	0	0	0	0
figure label	4	0	0	0	0	0	0	0	1	0	1	3	1	0	0	0	0	0	0	91	1	0	0	0	0
table label	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0	0	0	0
header	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	304	0	0	0
editor list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0
pg number	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	284	0
footer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10
Success %	89	100	39	100	91	67	100	100	95	89	83	37	84	96	57	0	41	47	58	96	96	100	90	100	100

Table 5.4: Full confusion matrix for JMLR K Nearest Neighbors results on training data. The top row indicate the correct label, the left column indicates the predicted label.

	text	authour list	section heading	main title	decoration	footnote	abstract	eq number	equation	graph	table	table caption	figure caption	references	subsection heading	image	bullet item	code block	figure	figure label	table label	header	editor list	pg number	footer
text	390	0	5	0	0	5	1	0	3	0	0	6	22	4	2	3	12	2	5	0	0	0	0	0	0
authour list	0	29	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
section heading	3	0	118	0	0	0	0	0	1	1	1	0	1	1	24	0	3	2	0	0	0	0	0	0	0
main title	0	2	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
decoration	0	0	0	0	53	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
footnote	7	0	0	0	0	10	0	0	1	0	0	0	3	0	0	0	1	0	0	0	0	0	0	0	0
abstract	1	0	0	0	0	0	16	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
eq number	0	0	0	0	0	0	0	120	0	0	0	0	0	0	0	1	0	0	2	0	0	0	0	0	0
equation	2	0	0	0	0	0	0	0	151	0	0	1	3	3	0	0	2	1	1	0	0	0	0	0	0
graph	0	0	0	0	0	0	0	0	0	10	4	0	5	0	0	4	0	0	20	0	0	0	0	0	0
table	0	0	0	0	0	0	0	0	0	1	11	0	0	0	0	0	0	1	0	0	0	0	0	0	0
table caption	2	0	0	0	0	0	0	0	1	0	0	3	3	1	0	0	1	0	0	0	0	0	0	0	0
figure caption	8	0	0	0	0	0	0	0	1	0	0	3	10	1	0	5	1	0	3	0	0	0	0	0	0
references	5	0	1	0	0	0	0	0	0	1	0	1	3	206	0	0	1	0	1	0	0	0	0	0	0
subsection heading	7	0	20	0	0	0	0	0	0	0	0	0	0	0	24	0	1	0	0	0	0	0	0	0	0
image	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
bullet item	5	0	1	0	0	0	0	0	4	0	0	1	0	0	1	1	4	0	0	0	0	0	0	0	0
code block	0	0	0	0	0	0	0	0	1	0	2	0	1	0	0	0	0	3	3	0	0	0	0	0	0
figure	3	0	0	0	0	0	0	0	0	7	1	0	1	0	0	4	0	0	11	0	0	0	0	0	0
figure label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
header	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
editor list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pg number	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
footer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Success %	90	94	81	100	100	67	84	100	92	50	55	20	19	95	47	5	15	33	24	0	0	0	0	0	0

Table 5.5: Full confusion matrix for NIPS K Nearest Neighbors results on test data. The top row indicate the correct label, the left column indicates the predicted label.

	text	authour list	section heading	main title	decoration	footnote	abstract	eq number	equation	graph	table	table caption	figure caption	references	subsection heading	image	bullet item	code block	figure	figure label	table label	header	editor list	pg number	footer
text	574	0	10	0	0	12	0	0	7	0	0	12	22	33	17	0	16	4	10	0	0	0	0	0	0
authour list	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
section heading	11	0	20	0	0	0	0	0	0	0	0	0	0	1	21	0	3	0	0	0	0	0	0	0	0
main title	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
decoration	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
footnote	10	0	0	0	0	15	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
abstract	2	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
eq number	0	0	0	0	0	0	0	90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
equation	9	0	0	0	0	0	0	0	285	1	0	1	5	0	0	0	1	1	1	0	0	0	0	0	0
graph	0	0	0	0	0	0	0	0	0	51	3	0	0	0	0	0	0	0	6	0	0	0	0	0	0
table	0	0	0	0	0	0	0	0	0	1	22	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table caption	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
figure caption	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
references	32	0	1	0	0	0	0	0	0	0	0	5	0	112	0	0	1	0	0	0	0	0	0	0	0
subsection heading	16	0	16	0	0	0	0	0	0	0	0	0	0	0	37	0	5	0	0	0	0	0	0	0	0
image	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bullet item	33	0	4	0	0	0	0	0	2	0	0	3	1	5	7	0	10	0	0	0	0	0	0	0	0
code block	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2	0	0	0	0	0	0	0
figure	1	0	0	0	2	0	0	0	0	9	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
figure label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
table label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
header	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	183	0	0	0
editor list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0
pg number	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	173	0
footer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5
Success %	84	100	39	100	95	56	100	100	97	81	81	52	79	74	41	0	28	29	34	96	86	100	100	100	100

Table 5.6: Full confusion matrix for JMLR K Nearest Neighbors results on test data. The top row indicate the correct label, the left column indicates the predicted label.

	text	equation	figure	other
text	1121	12	28	0
equation	11	151	2	0
figure	10	1	83	0
other	0	0	1	53
Success %	98.2	92.1	72.8	100

Table 5.7: Confusion matrix for NIPS test results using K Nearest Neighbors. The top row indicate the correct label, the left column indicates the predicted label.

	text	equation	figure	other
text	1608	9	17	1
equation	10	285	3	0
figure	4	0	105	1
other	0	0	1	41
Success %	99.1	96.9	83.3	95.3

Table 5.8: Confusion matrix for JMLR test results using K Nearest Neighbors. The top row indicate the correct label, the left column indicates the predicted label.

The performance in terms of the 4 generalized categories was very strong. Most of the errors represent cases where an unusual layout was used, such as figure captions being placed beside (rather than underneath or above) a figure, or cases where the layout of one region coincidentally matched the normal layout for another type (such as an equation that happened to be exactly the right width to appear to be a fully-justified text region).

5.5.2 Logistic Regression

The Logistic Regression algorithm, discussed in detail in Chapter 2, is a simple yet effective discriminative learner. This algorithm finds parameters that maximize the label-conditional log likelihood of the feature vectors (augmented with an always-1 feature, or bias feature) extracted from the training samples, learning linear hyperplanes that separate the different classes.

One aspect of the Logistic Regression algorithm specific to the particular data set is the selection of the smoothing factor. This can have a significant impact on the performance of the algorithm. If too large a smoothing factor is chosen, performance will be poor on both the training and test data. While smaller smoothing factors will never harm performance on the training data, if too small a smoothing factor is chosen, the algorithm will over-fit to the training data, degrading performance on the test data.

To determine a reasonable smoothing factor, the NIPS training data was divided into two subsets: one for training, and one for validation. A logistic regression learning was trained with each of this set of smoothing factors: [0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3].

	text	authour list	section heading	main title	decoration	footnote	abstract	eq number	equation	graph	table	table caption	figure caption	references	subsection heading	image	bullet item	code block	figure	figure label	table label	header	editor list	pg number	footer
text	1022	0	0	0	0	0	0	0	0	0	0	4	21	2	0	0	5	1	1	0	0	0	0	0	0
authour list	0	69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
section heading	0	0	293	0	0	0	0	0	0	1	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
main title	0	0	0	39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
decoration	0	0	0	0	161	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
footnote	0	0	0	0	0	75	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
abstract	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
eq number	1	0	0	0	0	0	0	368	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
equation	2	0	0	0	0	0	0	0	439	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
graph	0	0	0	0	0	0	0	0	0	96	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
table	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table caption	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0
figure caption	4	0	0	0	0	0	0	0	0	0	0	2	86	0	0	0	0	0	0	0	0	0	0	0	0
references	2	0	0	0	0	0	0	0	0	0	0	0	1	439	0	0	0	0	0	0	0	0	0	0	0
subsection heading	2	0	2	0	0	0	0	0	0	0	0	0	0	0	103	0	0	0	0	0	0	0	0	0	0
image	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
bullet item	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	0	0	0
code block	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21	1	0	0	0	0	0	0
figure	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	63	0	0	0	0	0	0
figure label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
header	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
editor list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pg number	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
footer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Success %	99	100	99	100	100	100	100	100	100	98	100	62	80	100	98	100	84	95	95	0	0	0	0	0	0

Table 5.9: Full confusion matrix for NIPS Logistic Regression results on training data. The top row indicate the correct label, the left column indicates the predicted label.

Results for the test data were compared, with the best results coming from a smoothing factor of 0.001, which was used for the remainder of the study.

Logistic Regression Results

The Logistic Regression classifier performed quite well for both data sets, with an average fine-grained accuracy rate of 89.4% on the test data, and an average generalized accuracy rate of 97.5% on the test data.

Several confusion matrices for the Logistic Regression algorithm are provided in the tables below. These summarize both the coarse and fine grained results for both the test set and the training set for each of the two data sets used.

	text	authour list	section heading	main title	decoration	footnote	abstract	eq number	equation	graph	table	table caption	figure caption	references	subsection heading	image	bullet item	code block	figure	figure label	table label	header	editor list	pg number	footer
text	1092	0	0	0	0	0	0	0	1	0	0	0	0	5	2	0	16	0	0	0	0	0	0	0	0
authour list	0	39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
section heading	0	0	89	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
main title	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
decoration	0	0	0	0	44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
footnote	0	0	0	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
abstract	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
eq number	0	0	0	0	0	0	0	193	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
equation	2	0	0	0	0	0	0	0	527	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
graph	0	0	0	0	0	0	0	0	0	66	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table	0	0	0	0	0	0	0	0	0	0	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table caption	0	0	0	0	0	0	0	0	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0
figure caption	0	0	0	0	0	0	0	0	0	0	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0
references	4	0	0	0	0	0	0	0	0	0	0	0	0	270	0	0	0	0	0	0	0	0	0	0	0
subsection heading	3	0	5	0	0	0	0	0	0	0	0	0	0	0	103	0	0	0	0	0	0	0	0	0	0
image	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bullet item	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	54	0	0	0	0	0	0	0	0
code block	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0
figure	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	57	0	0	0	0	0	0
figure label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	95	0	0	0	0	0
table label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0
header	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	304	0	0	0
editor list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0
pg number	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	285	0
footer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10
Success %	99	100	95	100	100	100	100	100	100	100	100	100	100	98	95	0	77	100	100	100	100	100	100	100	100

Table 5.10: Full confusion matrix for JMLR Logistic Regression results on training data. The top row indicate the correct label, the left column indicates the predicted label.

	text	authour list	section heading	main title	decoration	footnote	abstract	eq number	equation	graph	table	table caption	figure caption	references	subsection heading	image	bullet item	code block	figure	figure label	table label	header	editor list	pg number	footer
text	414	0	2	0	0	0	2	1	0	0	0	8	9	2	1	1	13	1	3	0	0	0	0	0	0
authour list	0	31	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
section heading	1	0	137	0	0	0	0	0	1	0	1	0	2	0	7	0	0	1	0	0	0	0	0	0	0
main title	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
decoration	0	0	0	0	53	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
footnote	3	0	0	0	0	15	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
abstract	0	0	0	0	0	0	15	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
eq number	0	0	0	0	0	0	0	119	2	0	0	0	0	0	0	3	0	0	1	0	0	0	0	0	0
equation	0	0	0	0	0	0	0	0	153	0	0	0	1	0	0	0	1	0	2	0	0	0	0	0	0
graph	0	0	1	0	0	0	0	0	0	15	4	0	2	0	0	0	0	0	10	0	0	0	0	0	0
table	0	0	0	0	0	0	0	0	1	1	10	0	0	0	0	0	0	1	1	0	0	0	0	0	0
table caption	3	0	0	0	0	0	0	0	0	0	0	3	6	0	0	0	4	0	0	0	0	0	0	0	0
figure caption	1	0	0	0	0	0	0	0	2	0	0	2	30	0	0	6	0	0	14	0	0	0	0	0	0
references	7	0	0	0	0	0	0	0	0	0	0	1	0	214	0	3	0	0	0	0	0	0	0	0	0
subsection heading	3	0	5	0	0	0	0	0	0	0	0	0	0	0	43	0	1	0	0	0	0	0	0	0	0
image	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	3	0	0	0	0	0	0	0	0	0
bullet item	0	0	0	0	0	0	0	0	2	0	0	1	0	0	0	0	6	0	0	0	0	0	0	0	0
code block	0	0	0	0	0	0	1	0	0	2	2	0	0	0	0	0	1	4	1	0	0	0	0	0	0
figure	1	0	0	0	0	0	0	0	0	2	3	0	0	0	0	2	0	1	14	0	0	0	0	0	0
figure label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
header	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
editor list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pg number	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
footer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Success %	96	100	94	100	100	100	79	99	93	75	50	20	58	99	84	16	23	44	30	0	0	0	0	0	0

Table 5.11: Full confusion matrix for NIPS Logistic Regression results on test data. The top row indicate the correct label, the left column indicates the predicted label.

	text	authour list	section heading	main title	decoration	footnote	abstract	eq number	equation	graph	table	table caption	figure caption	references	subsection heading	image	bullet item	code block	figure	figure label	table label	header	editor list	pg number	footer
text	643	0	2	0	0	4	0	0	0	0	0	0	0	38	1	0	18	0	0	1	0	0	0	0	0
authour list	0	18	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
section heading	3	0	48	0	0	0	0	0	0	0	0	0	0	0	15	0	3	0	0	0	0	0	0	0	0
main title	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
decoration	0	0	0	0	42	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
footnote	5	0	0	0	0	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
abstract	2	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
eq number	0	0	0	0	0	0	0	89	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
equation	3	0	0	0	0	0	0	0	292	0	0	0	0	1	0	0	2	1	2	0	0	0	0	0	0
graph	0	0	0	0	0	0	0	0	0	40	3	0	0	0	0	0	0	0	6	0	0	0	0	0	0
table	0	0	0	0	0	0	0	0	0	2	22	0	0	0	0	0	0	0	1	0	0	0	0	0	0
table caption	0	0	0	0	0	0	0	0	0	0	0	14	1	0	0	0	0	0	0	0	0	0	0	0	0
figure caption	0	0	0	0	0	0	0	0	0	0	0	6	27	0	0	0	0	0	2	0	0	0	0	0	0
references	24	0	1	0	1	0	0	0	0	0	0	3	0	113	1	0	1	0	0	0	0	0	0	0	0
subsection heading	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65	0	1	0	0	0	0	0	0	0	0
image	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bullet item	7	0	0	0	0	0	0	0	2	0	0	0	0	1	0	0	11	0	0	0	0	0	0	0	0
code block	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	5	0	0	0	0	0	0	0
figure	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	1	14	0	0	0	0	0	0
figure label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	27	0	0	0	0	0
table label	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	22	0	0	0	0
header	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	183	0	0	0
editor list	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0
pg number	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	173	0
footer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5
Success %	93	100	94	100	98	78	100	99	99	63	81	61	96	74	79	0	31	71	48	96	100	100	100	100	100

Table 5.12: Full confusion matrix for JMLR Logistic Regression results on test data. The top row indicate the correct label, the left column indicates the predicted label.

	text	equation	figure	other
text	1132	10	35	0
equation	2	153	2	0
figure	8	1	76	0
other	0	0	1	53
Success %	99.1	93.3	66.7	100

Table 5.13: Confusion matrix for NIPS test results using Logistic Regression. The top row indicate the correct label, the left column indicates the predicted label.

	text	equation	figure	other
text	1615	2	5	1
equation	6	292	3	0
figure	1	0	116	0
other	0	0	2	42
Success %	99.6	99.3	92.1	97.7

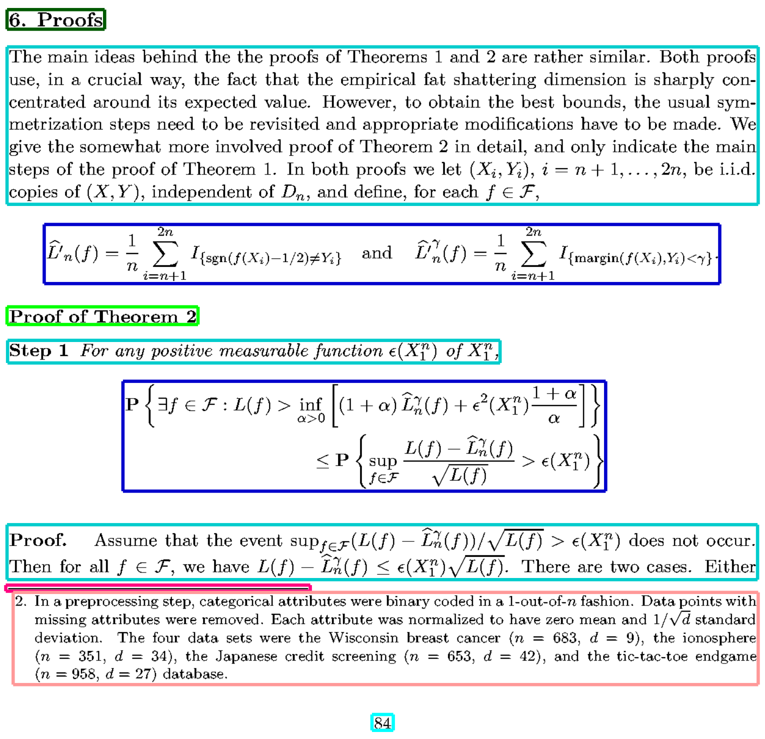
Table 5.14: Confusion matrix for JMLR test results using Logistic Regression. The top row indicate the correct label, the left column indicates the predicted label.

The errors made by the Logistic Regression classifier are similar to those made by the K Nearest Neighbors classifier. Similar classes, such as section are being confused with each other. It should be noted that some of these regions really are ambiguous as to their class, with human readers disagreeing as to what the correct label would be. An example of such as ambiguous regions is shown in Figure 5.1, with possible labels of *text* or *subsection heading*.

Performance on the four broader classes was extremely strong. Most of the errors present resulted from the small number of images and figures in the NIPS training data. There were only 10 images among the 3445 regions in the NIPS training data. The algorithm correctly labeled these 10 regions in the training data, but mislabeled 13 of the 16 images in the test data. This suggests the algorithm overfit the training data, finding a specific *coincidental* linear combination of variables that fit the training data very well, but was not characteristic of images in general. In fact, this was the case, as all 10 images in the training data happened to come at the top of the last page of an article. The algorithm gave a heavy positive weight to the *is_last_page* feature in the column of the weight vector corresponding to the *image* label, which had the effect of allowing images in the test data to be correctly classified only if they happened to occur on the last page.

5.5.3 Maximum Entropy Markov Model

Maximum Entropy Markov Models are discussed in detail in Chapter 2.4.1. MEMM's offer a major advantage in that they make use of the order of the labels. One drawback of MEMM's is their slow performance at test time, since sequences of labels need to be considered, rather



6. Proofs

The main ideas behind the the proofs of Theorems 1 and 2 are rather similar. Both proofs use, in a crucial way, the fact that the empirical fat shattering dimension is sharply concentrated around its expected value. However, to obtain the best bounds, the usual symmetrization steps need to be revisited and appropriate modifications have to be made. We give the somewhat more involved proof of Theorem 2 in detail, and only indicate the main steps of the proof of Theorem 1. In both proofs we let (X_i, Y_i), $i = n+1, \ldots, 2n$, be i.i.d. copies of (X, Y), independent of D_n, and define, for each $f \in \mathcal{F}$,

$$\widehat{L'}_n(f) = \frac{1}{n} \sum_{i=n+1}^{2n} I_{\{\mathrm{sgn}(f(X_i)-1/2) \neq Y_i\}} \quad \text{and} \quad \widehat{L'}^{\gamma}_n(f) = \frac{1}{n} \sum_{i=n+1}^{2n} I_{\{\mathrm{margin}(f(X_i),Y_i)<\gamma\}}.$$

Proof of Theorem 2

Step 1 *For any positive measurable function $\epsilon(X_1^n)$ of X_1^n,*

$$\mathbf{P}\left\{\exists f \in \mathcal{F} : L(f) > \inf_{\alpha>0}\left[(1+\alpha)\widehat{L}^{\gamma}_n(f) + \epsilon^2(X_1^n)\frac{1+\alpha}{\alpha}\right]\right\} \leq \mathbf{P}\left\{\sup_{f\in\mathcal{F}} \frac{L(f) - \widehat{L}^{\gamma}_n(f)}{\sqrt{L(f)}} > \epsilon(X_1^n)\right\}$$

Proof. Assume that the event $\sup_{f\in\mathcal{F}}(L(f) - \widehat{L}^{\gamma}_n(f))/\sqrt{L(f)} > \epsilon(X_1^n)$ does not occur. Then for all $f \in \mathcal{F}$, we have $L(f) - \widehat{L}^{\gamma}_n(f) \leq \epsilon(X_1^n)\sqrt{L(f)}$. There are two cases. Either

2. In a preprocessing step, categorical attributes were binary coded in a 1-out-of-n fashion. Data points with missing attributes were removed. Each attribute was normalized to have zero mean and $1/\sqrt{d}$ standard deviation. The four data sets were the Wisconsin breast cancer ($n = 683$, $d = 9$), the ionosphere ($n = 351$, $d = 34$), the Japanese credit screening ($n = 653$, $d = 42$), and the tic-tac-toe endgame ($n = 958$, $d = 27$) database.

84

Figure 5.1: The subsection_heading region 2/3 of the way down the page could well have been labeled as text. The context of the document suggest a subsection_heading label, as a large number of proofs are included, one every few pages, each separated by a similar “Proof of Theorem NN” heading, typed in bold-face font.

than individual labels in isolation.

Maximum Entropy Markov Models require the data to be presented as a linear sequence of items to be classified. The natural way to sequence the regions on a page is by their reading order. This ordering can be approximated for single-column documents by running from top-to-bottom, left-to-right. A simple formula illustrated by Figure 5.2 based on the top left corner of each region provides a scalar value by which the regions can be ordered. In this formula, α^{-1} represents the slope of a line across the page. The regions are ordered by sliding this line down the page, ordering the regions by when their top-left corner is reached. We found a value of $\alpha = 30$ to be effective, and used this value in the experiments conducted for this paper.

$$f(top, left) = (\alpha * top) + left$$

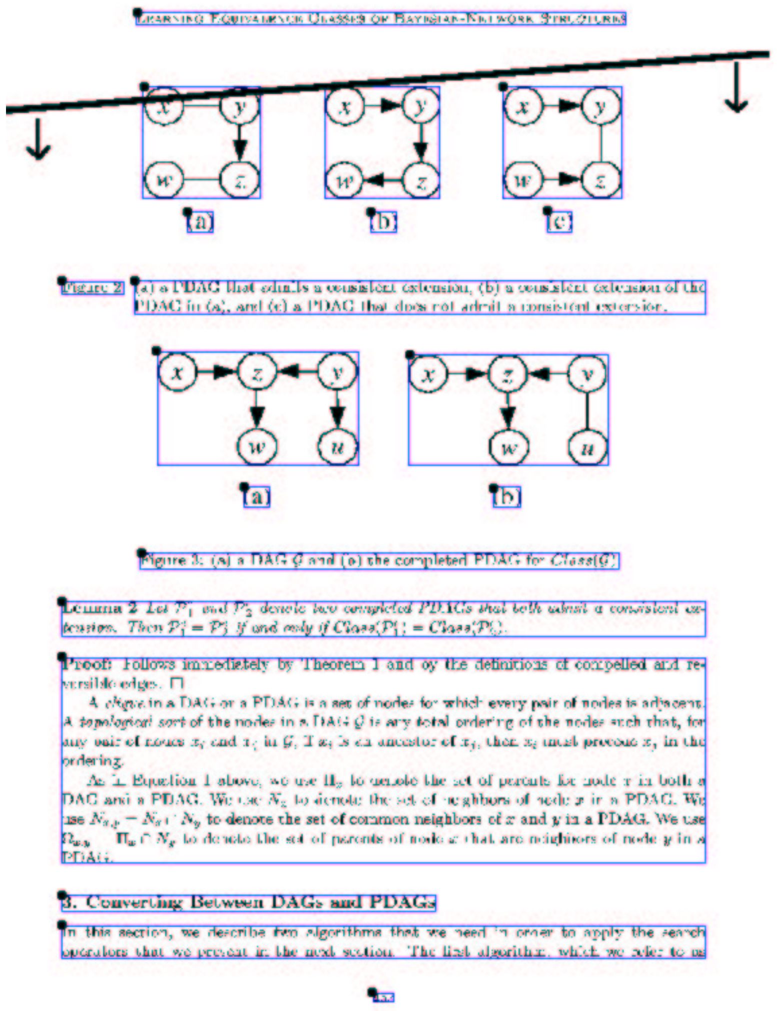

LEARNING EQUIVALENCE CLASSES OF BAYESIAN-NETWORK STRUCTURES

Figure 2 (a) a PDAG that admits a consistent extension, (b) a consistent extension of the PDAG in (a), and (c) a PDAG that does not admit a consistent extension.

Figure 3: (a) a DAG $\mathcal{G}$ and (b) the completed PDAG for $Class(\mathcal{G})$

Lemma 2 *Let $\mathcal{P}_1^c$ and $\mathcal{P}_2^c$ denote two completed PDAGs that both admit a consistent extension. Then $\mathcal{P}_1^c = \mathcal{P}_2^c$ if and only if $Class(\mathcal{P}_1^c) = Class(\mathcal{P}_2^c)$.*

Proof: Follows immediately by Theorem 1 and by the definitions of compelled and reversible edges. □

A *clique* in a DAG or a PDAG is a set of nodes for which every pair of nodes is adjacent. A *topological sort* of the nodes in a DAG $\mathcal{G}$ is any total ordering of the nodes such that, for any pair of nodes x_i and x_j in $\mathcal{G}$, if x_i is an ancestor of x_j, then x_i must precede x_j in the ordering.

As in Equation 1 above, we use Π_x to denote the set of parents for node x in both a DAG and a PDAG. We use N_x to denote the set of neighbors of node x in a PDAG. We use $N_{x,y} = N_x \cap N_y$ to denote the set of common neighbors of x and y in a PDAG. We use $\Omega_{x,y} = \Pi_x \cap N_y$ to denote the set of parents of node x that are neighbors of node y in a PDAG.

3. Converting Between DAGs and PDAGs

In this section, we describe two algorithms that we need in order to apply the search operators that we present in the next section. The first algorithm, which we refer to as

Figure 5.2: The region ordering function used is equivalent to sliding an angled line down the page, and numbering the regions as their upper-left corner is reached.

Once the regions have been sequenced, a standard MEMM can be used, as described in Chapter 2.4.1.

Our MEMM is based on an underlying logistic regression classifier. As with the logistic regression classifier, the feature vectors were augmented with an always-1 feature, or bias feature. In addition, the smoothing factor found to be effective for logistic regression (through cross-validation) was used in our MEMM.

MEMM Results

The MEMM classifier was run with a value of $\alpha = 30$, and performed very well for both data sets, with an average fine-grained accuracy of 95% on the test data, and an overall generalized accuracy of 98.0% on the test data. Notably, the MEMM achieved average fine-grained and generalized accuracy rates of 99.4% and 99.9% respectively on the training data, suggesting that the majority of the errors in the test data are from the learning overfitting the training data.

Several confusion matrices for the MEMM algorithm are provided in the tables below. These summarize both the coarse and fine grained results for both the test set and the training set for each of the two data sets used.

The MEMM classifier exhibited slightly better performance than the logistic regression classifier. It performed significantly better on the training data, making use of the region ordering information to eliminate over 2/3 of the errors made by the logistic regression classifier. However, these improvements did not generalize to the test data. The MEMM classifier suffered from overfitting, with test error rates comparable to those achieved by the logistic regression classifier. For example, as with the logistic regression classifier, the MEMM classifier was successful on all 10 *image* regions in the NIPS training data. However, it misclassified all 19 of the image regions in the test data. It gave strong weight to the fact that in the test data, 9 of the 10 image regions came after another image region, encouraging the classifier to label only sequences of image regions with the *image* label, so that it incorrectly handled the isolated images occurring in the test data. These overfitting problems could likely be alleviated through the use of more training data.

	text	authour list	section heading	main title	decoration	footnote	abstract	eq number	equation	graph	table	table caption	figure caption	references	subsection heading	image	bullet item	code block	figure	figure label	table label	header	editor list	pg number	footer
text	1031	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	2	0	0	0	0	0	0	0	0
authour list	0	69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
section heading	0	0	292	0	0	0	0	0	0	1	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
main title	0	0	0	39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
decoration	0	0	0	0	161	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
footnote	1	0	0	0	0	75	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
abstract	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
eq number	1	0	0	0	0	0	0	368	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
equation	1	0	0	0	0	0	0	0	439	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
graph	0	0	0	0	0	0	0	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table caption	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0
figure caption	0	0	0	0	0	0	0	0	0	0	0	0	108	0	0	0	0	0	0	0	0	0	0	0	0
references	0	0	0	0	0	0	0	0	0	0	0	0	0	439	0	0	0	0	0	0	0	0	0	0	0
subsection heading	1	0	3	0	0	0	0	0	0	0	0	0	0	0	102	0	0	0	0	0	0	0	0	0	0
image	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
bullet item	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35	0	0	0	0	0	0	0	0
code block	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0
figure	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	66	0	0	0	0	0	0
figure label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
header	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
editor list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pg number	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
footer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Success %	100	100	99	100	100	100	100	100	100	98	100	100	100	100	97	100	92	100	100	0	0	0	0	0	0

Table 5.15: Full confusion matrix for NIPS MEMM results on training data. The top row indicate the correct label, the left column indicates the predicted label.

	text	authour list	section heading	main title	decoration	footnote	abstract	eq number	equation	graph	table	table caption	figure caption	references	subsection heading	image	bullet item	code block	figure	figure label	table label	header	editor list	pg number	footer
text	1099	0	0	0	0	0	0	0	1	0	0	0	0	2	0	0	8	0	0	0	0	0	0	0	0
authour list	0	39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
section heading	0	0	90	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
main title	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
decoration	0	0	0	0	44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
footnote	0	0	0	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
abstract	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
eq number	0	0	0	0	0	0	0	193	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
equation	0	0	0	0	0	0	0	0	525	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
graph	0	0	0	0	0	0	0	0	0	66	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table	0	0	0	0	0	0	0	0	0	0	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table caption	0	0	0	0	0	0	0	0	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0
figure caption	0	0	0	0	0	0	0	0	0	0	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0
references	0	0	0	0	0	0	0	0	0	0	0	0	0	273	0	0	0	0	0	0	0	0	0	0	0
subsection heading	2	0	4	0	0	0	0	0	0	0	0	0	0	0	105	0	0	0	0	0	0	0	0	0	0
image	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bullet item	4	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	62	0	0	0	0	0	0	0	0
code block	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0
figure	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	57	0	0	0	0	0	0
figure label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	95	0	0	0	0	0
table label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0
header	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	304	0	0	0
editor list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0
pg number	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	285	0
footer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10
Success %	99	100	96	100	100	100	100	100	99	100	100	100	100	99	97	0	89	100	100	100	100	100	100	100	100

Table 5.16: Full confusion matrix for JMLR MEMM results on training data. The top row indicate the correct label, the left column indicates the predicted label.

	text	authour list	section heading	main title	decoration	footnote	abstract	eq number	equation	graph	table	table caption	figure caption	references	subsection heading	image	bullet item	code block	figure	figure label	table label	header	editor list	pg number	footer
text	417	0	1	0	0	0	2	2	0	0	0	5	3	9	1	3	11	2	1	0	0	0	0	0	0
authour list	0	30	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
section heading	2	0	134	0	0	0	0	0	0	0	1	0	2	1	5	0	0	1	0	0	0	0	0	0	0
main title	0	1	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
decoration	0	0	0	0	53	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
footnote	0	0	0	0	0	15	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
abstract	0	0	0	0	0	0	15	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
eq number	0	0	0	0	0	0	0	113	2	0	0	0	0	0	0	2	0	0	4	0	0	0	0	0	0
equation	0	0	0	0	0	0	0	0	150	0	0	0	1	0	0	0	4	0	0	0	0	0	0	0	0
graph	1	0	1	0	0	0	0	0	0	16	4	0	2	0	0	0	0	2	14	0	0	0	0	0	0
table	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	3	0	0	0	0	0	0	0
table caption	3	0	0	0	0	0	0	0	1	0	0	6	4	1	1	0	1	0	0	0	0	0	0	0	0
figure caption	1	0	0	0	0	0	0	1	0	0	0	3	37	1	0	6	0	0	9	0	0	0	0	0	0
references	6	0	2	0	0	0	0	3	4	0	0	0	0	204	0	0	0	0	1	0	0	0	0	0	0
subsection heading	1	0	6	0	0	0	0	0	0	0	0	0	0	0	43	0	2	0	0	0	0	0	0	0	0
image	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bullet item	1	0	1	0	0	0	0	0	5	0	0	1	0	0	1	0	7	0	0	0	0	0	0	0	0
code block	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
figure	1	0	0	0	0	0	0	1	0	3	2	0	2	0	0	7	0	0	17	0	0	0	0	0	0
figure label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
header	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
editor list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
pg number	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
footer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Success %	96	97	92	100	100	100	79	94	91	80	65	40	71	94	84	0	27	11	37	0	0	0	0	0	0

Table 5.17: Full confusion matrix for NIPS MEMM results on test data. The top row indicate the correct label, the left column indicates the predicted label.

	text	authour list	section heading	main title	decoration	footnote	abstract	eq number	equation	graph	table	table caption	figure caption	references	subsection heading	image	bullet item	code block	figure	figure label	table label	header	editor list	pg number	footer
text	673	0	1	0	1	0	0	0	0	0	0	0	1	37	1	0	16	1	0	1	0	0	0	0	0
authour list	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
section heading	3	0	48	0	0	0	0	0	0	0	0	0	0	0	13	0	5	0	0	0	0	0	0	0	0
main title	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
decoration	0	0	0	0	42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
footnote	0	0	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
abstract	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
eq number	0	0	0	0	0	0	0	90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
equation	0	0	0	0	0	0	0	0	293	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
graph	0	0	0	0	0	0	0	0	0	32	4	0	0	0	0	0	0	0	3	0	0	0	0	0	0
table	0	0	0	0	0	0	0	0	0	3	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0
table caption	0	0	0	0	0	0	0	0	0	0	0	17	2	0	0	0	0	0	0	0	0	0	0	0	0
figure caption	0	0	0	0	0	0	0	0	0	0	0	3	25	0	0	0	0	0	3	0	0	0	0	0	0
references	11	0	0	0	0	0	0	0	0	0	0	3	0	116	1	0	0	0	0	0	0	0	0	0	0
subsection heading	0	0	1	0	0	0	0	0	0	0	0	0	0	0	67	0	0	0	0	0	0	0	0	0	0
image	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bullet item	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0
code block	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	5	0	0	0	0	0	0	0
figure	0	0	0	0	0	0	0	0	0	27	0	0	0	0	0	0	0	1	18	0	0	0	0	0	0
figure label	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	5	25	2	0	0	0	0
table label	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	20	0	0	0	0
header	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	183	0	0	0
editor list	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0
pg number	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	173	0
footer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5
Success %	98	100	94	100	98	100	100	100	100	51	78	74	89	76	82	0	36	71	62	89	91	100	100	100	100

Table 5.18: Full confusion matrix for JMLR MEMM results on test data. The top row indicate the correct label, the left column indicates the predicted label.

	text	equation	figure	other
text	1128	14	30	0
equation	5	150	0	0
figure	9	0	83	0
other	0	0	1	53
Success %	98.8	91.2	72.3	100

Table 5.19: Confusion matrix for NIPS test results using Maximum Entropy Markhov Model. The top row indicate the correct label, the left column indicates the predicted label.

	text	equation	figure	other
text	1619	1	10	1
equation	2	293	0	0
figure	1	0	116	0
other	0	0	0	42
Success %	99.8	99.7	92.1	97.7

Table 5.20: Confusion matrix for JMLR test results using Maximum Entropy Markhov Model. The top row indicate the correct label, the left column indicates the predicted label.

5.5.4 Summary of Labeling Results

The labeling problem is a natural fit to the supervised learning paradigm. Once ink regions have been identified, a classifier can be used to assign the appropriate label, based on a vector of numerical features. The classifier can be trained with examples from the labeled training set. Three algorithms were evaluated: K Nearest Neighbors, logistic regression, and maximum entropy Markov models[16].

For each ink region, a vector of 59 features describing it was created based on its position in the page, ink density, ink marks contained, as well as the page number on which it appeared compared to the total number of pages in the article. All features were normalized such that they ranged between zero and one. For the MEMM, which requires labels to be presented in discrete sequences, the regions on each page were ordered top-to-bottom, left-to-right, in an attempt to approximate their reading order. Each page was treated as a single sequence.

Dataset	KNN	LR	MEMM
NIPS: Training	85.3%	98.3%	99.5%
NIPS: Testing	80.7%	88.1%	87.6%
JMLR: Training	88.8%	97.7%	99.3%
JMLR: Testing	83.6%	90.6%	92.4%

Table 5.21: Success rates for finegrained classification among 25 precise labels.

Success rates in terms of the 25 fine-grained labels, as well as the 4 generalized labels, are shown in Tables 5.21 and 5.22 respectively. All three algorithms showed strong performance, especially among the 4 generalized categories. As expected, KNN had the weakest performance, and MEMM had the strongest.

Dataset	KNN	LR	MEMM
NIPS: Training	97.3%	99.8%	99.9%
NIPS: Testing	95.6%	96.0%	96.0%
JMLR: Training	98.1%	99.9%	99.9%
JMLR: Testing	97.8%	99.0%	99.3%

Table 5.22: Success rates for coarse classification among the 4 generalzed labels.

As a point of comparison, Summers[24] used a complex rule-based system to achieve an average "precise accuracy" (comparable to our "fine-grained accuracy") of 85.5% among 16 categories, and a "generalized accuracy" of 86.0% on an unspecified (but smaller) number of categories.

The majority of the errors came from three sources: confusing similar items (such as *section heading* and *subsection heading*), classifying ambiguous items differently than the human tagger chose to, and learning algorithms overfitting the training data.

Chapter 6

Combined Segmentation and Labeling

In this chapter, a new approach to the both the segmentation and labeling problems is proposed. Rather than treating the problems separately and sequentially, a single model for the way ink is laid out on the page was created. This model contains parameters that can be extracted from hand-segmented pages. The resulting parameters can be used to simultaneously segment and classify new pages. The algorithm developed to implement this model is called Recursive Generative Segmentation, or RGS.

6.1 Recursive Generative Segmentation

As with other segmentation algorithms examined, the description of RGS can be broken down into two elements: a decision process for evaluating which bits of ink belong together, and a structural algorithm for reaching these decisions. Since this is a learning algorithm, the decision process is best described in two parts: a training algorithm, and a test-time procedure.

6.1.1 Model Concept

Recursive Generative Segmentation provides a model for how the ink was laid out on the page. It attempts to maximize the joint likelihood of the region locations, the region labels, and the on/off state of each pixel on the page. Let LAB represent the n region labels $\{lab_1, lab_2, ...lab_n\}$for a page, let SEG represent feature vectors describing the layouts of the n ink regions (or segments) $\{seg_1, seg_2, ..., seg_p\}$, and let PIX represent the m pixel states $\{pix_1, pix_2, ..., pix_m\}$. This model then attempts to maximize the joint likelihood $P(LAB, SEG, PIX)$ for each page. The joint likelihood is factored as shown in Equation 6.1.

$$P(LAB, SEG, PIX) = P(LAB)P(SEG|LAB)P(PIX|SEG, LAB) \tag{6.1}$$

The model makes a collection of simplifying assumptions. First, it assumes that the likelihoods of the labels are marginally independent of the other labels.

$$P(LAB) = \prod_i P(lab_i) \tag{6.2}$$

Second, it assumes that the likelihood of each segment's layout information is label-conditionally independent of the other layout of the other segments, and of the labels of the other segments, with the exception of the rule that no two segments may overlap.

$$P(SEG|LAB) = \prod_i P(seg_i|lab_i) \tag{6.3}$$

Each segment will be represented with a feature vector containing features drawn from its layout, as was done with the labeling problem, restricting the features to those that describe the layout of the region [1]. The Naive assumption will be used, assuming that the values of each of the r elements in the feature vector $seg_i = \{seg_{i,1}, seg_{i,2}, ...seg_{i,r}\}$are independent of

[1]Our implementation erroneously included features that are functions of the pixels contained in the region. This should be corrected in future work

one another, given the label.

$$P(seg_i|lab_i) = \prod_j P(seg_{i,j}|lab_i) \tag{6.4}$$

The values of $P(seg_{i,j}|lab_i)$ will be determined by modeling continuous-valued features as Gaussian distributions (as in a Gaussian Naive Bayes classifier), and by using normalized counts of the distribution for variables with a small number of discrete values.

Finally, it assumes that the likelihood of the on/off state of each of the k pixels depends only on the label assigned to the segment that contains it; in other words, it is independent of the other pixels, of the locations of segments, and of labels other than that assigned to the segment that contains it (denoted by lab_k).

$$P(PIX|LAB, SEG) = \prod_k P(pix_k|lab_k) \tag{6.5}$$

This is one of the simplest possible models for $P(LAB, SEG, PIX)$, and is intended as a proof of concept for the idea of using a generative model for page images.

6.1.2 RGS Structural Algorithm

Ideally, to ensure that we have found the assignment of segments and labels that maximizes the page likelihood at test time according to the model of $P(LAB, SEG|PIX)$, we would try every possible segmentation, along with every possible assignment of labels for each segmentation.

$$[BEST\_SEG, BEST\_LAB] = ArgMax_{SEG,LAB} P(LAB, SEG|PIX) \tag{6.6}$$

The assumptions illustrated by Equations 6.2 and 6.3 ensure that the label assigned to one segment will not affect the likelihood estimates for any other segments or labels. This means that assigning the maximum likelihood label to each segment is guaranteed to maximize the product of the likelihoods of all the segments in the segmentation, allowing us to treat the label assignments in isolation once a segmentation has been proposed, as shown in Equation 6.7.

$$ArgMax_{SEG,LAB}P(LAB, SEG|PIX) = ArgMax_{SEG}\prod_{i} P\left(ArgMax_{lab_i}P(lab_i|seg_i), seg_i \mid PIX\right) \tag{6.7}$$

The question of how to consider the possible segmentation must also be considered. The number of possible segmentations of a page is enormous. Consider just the number of possible sizes for the upper-left-most segment on a 1000 by 1500 pixel page: over one million. Some structured algorithm is necessary to limit the number of segmentations considered. As with the label assignments, some simplifying assumptions will need to be made.

First, we shall assume that a Manhattan layout is used, as is assumed for many of the other segmentation algorithms considered.

Second, we shall assume that regions are separated by bands of whitespace of a certain minimum width, as was implicitly assumed by the XY Cuts and Concurrent Learn To Cut algorithms. As with the Concurrent Learn To Cut algorithm, an appropriate minimum cut width must be selected to speed up the algorithm, while discounting a minimal number of actual cuts. Thresholds of 1 pixel for horizontal cuts and 5 pixels for vertical cuts were selected by examining how many actual cuts in the training data would be discounted at various thresholds.

If these two assumptions hold, then starting with the entire page as one region and successively cutting regions either vertically or horizontally describes a search tree that is guaranteed to contain the correct segmentation. Since we know that the regions are label-conditionally independent of one another, we can make use of this cutting process to define a recursive cutting function that will find the optimal segmentation and label assignment. This recursive function forms the basis for the RGS structural algorithm.

The intuition behind this algorithm is simple. We know that the best segmentation can be reached by starting with the entire page as a single region, and successively dividing regions with alternating vertical and horizontal cuts. Candidate locations for cutting (termed *cut candidates*) were found using the projection profile. Considering all possible pairings of cut candidates allows us to find all possible segments created in the current level of recursion. Once the likelihood of each of these segments is calculated, the Best Cut Sequence algorithm presented in Figure 4.3 can be used to find the best segmentation at this level of recursion.

The real trick lies in calculating the likelihood of a segment. This can be defined recursively as larger value of the maximum likelihood label assigned to that segment, or the product of the likelihoods of the best segmentation produced by the next pass of the algorithm on that segment. This trick allows the Best Cut Sequence algorithm described in Figure 4.3 to be applied at each level of recursion. The recursion terminates on a given segment when a pass is run with no cut candidates found.

The following pseudo-code illustrates the algorithm. Note that several simple functions are defined before the actual RGS function.

```
--------------------------------

function score = evaluate(segment): tests all labels on segment, and
returns the log-likelihood of the label with the highest likelihood.

--------------------------------

function subsegs find_all_possible_subsegs(segment,vert_pass): Finds all
possible subsegments that can be created by making cuts in segment. The
boolean variable vert_pass indicates whether vertical or horizontal cuts
should be considered in this pass.

--------------------------------

function [score, segmentation] = rgs(segment,vert_pass);
  score = evaluate(segment)
  subsegs = find_all_possible_subsegs(segment,vert_pass)
  for i = 1:length(subsegs)
    [subscore(i),subsegmentation(i)] = rgs(subseg(i),~vert_pass)
  end
  Here, we use the dynamic programming trick to find the best
  possible set of subsegs to use, and set:
    segmentation = best segmentation
    score = product(subscores of best segmentation)
  return [score, segmentation]
--------------------------------
```

This recursive structural algorithm allows us to overcome the problem of the massive number of possible segmentations of a page. The following chapter describes the implementation of the function.

6.1.3 RGS Segment Scoring

The test-time decision criteria for RGS are conceptually simple: find the segmentation and assignment of labels that maximize the likelihood of $P(LAB, SEG|PIX)$. This chapter outlines how the model assumptions can be used to provide a manner of estimating $P(LAB, SEG|PIX)$ using a set of parameters Θ.

First, it should be noted that Bayes Rule can be applied to the test-time objective function, expressing it as:

$$P(LAB, SEG|PIX) = \frac{P(LAB, SEG, PIX}{P(PIX)}$$

Since $P(PIX)$ is marginally independent, this factor will be the same for all segmentations and label assignments. This allows us to ignore this factor at test time, and use the joint distribution $P(LAB, SEG, PIX)$ as the objective function at test time. Recall that the objective function is broken down by applying the chain rule:

$$P(LAB, SEG, PIX) = P(LAB)P(SEG|LAB)P(PIX|SEG, LAB) \tag{6.8}$$

The assumptions laid out above can now be used to provide methods of estimating each of the three terms above. The first term $P(LAB)$ can be understood as representing the likelihood that a given set of labels will be used on a page. The second term $P(SEG|LAB)$

represents the likelihood that a certain segmentation of a page would be used for a known set of labels. Finally, the third term $P(PIX|SEG, LAB)$ can be thought of as an estimate of how likelihood it is that a certain set of pixel on/off states would be found on a page that used a given segmentation and set of labels. The assumption expressed by Equation 6.2 allows us to treat each of the label likelihoods independently.

A single parameter α_k can be used to represent the estimated likelihood that any given region will be assigned the label k. Essentially, $P(LAB)$ for a given assignment of labels can be found by multiplying together all of the α values corresponding to the assigned labels.

$$P(LAB) = \prod_i (\alpha_{lab_i}) \tag{6.9}$$

The model assumptions of label and segment independence, along with the assumption of label-conditional feature independence for the feature vectors of the segments, allow us to break the second term $P(SEG|LAB)$ down into the product of the likelihoods of the individual features, given the labels, as shown in Equation 6.10. Note that this equation does not encorporate the restrictions of manhatten layout and minimal whitespace separation between segments. These assumptions must be enforced in some other manner. A listing of the segment features used can be found in Table 6.1.

$$P(SEG|LAB) = \prod_i \left(\prod_j (P(seg_{i,j}|lab_i) \right) \tag{6.10}$$

The method used for estimating $P(seg_{i,j}|lab_i)$ was dependent on the type of variable for the j'th element in the feature vector. For boolean variables, $\beta_{j,k}$ represents the likelihood that the j'th variable will take on the value *true*, given that the corresponding label is k. For continuous-valued variables, a Gaussian probability-density estimation approach is used, assuming that each feature j with the label k is distributed about a mean $\mu_{j,k}$ with standard deviation $\sigma_{j,k}$.

Feature Description	**# Feats**
Distance from each edge of the rectangle to the corresponding edge of the page	4
Distance from each edge of the rectangle to the next ink in that direction	4
Width, height, area, and log of the aspect ratios	5
Boolean: is it within 10 pixels of being centered	1
Is there any ink between it and each edge of the page	4
Is there any ink closer to each edge of the page than in this region	4
Fraction of "on" pixels in the region (ink density)	1
"Snapped region" ink density	1
Horizontal Sharpness (helps distinguish text from non-text)	1
Boolean: does it contain a horizontal line across the region	1
Boolean: does it contain a vertical line across the region	1
Fraction of ink in the leftmost quarter of the region	1
Boolean: Is this the first page in the article	1
Boolean: Is this the last page in the article	1
Boolean: Is this page in the last 15% of the pages in the article	1
Fraction of the way through the article	1
Total # of Features	32

Table 6.1: The 32 features used for the Recursive Generative Segmentation algorithm. The first column describes the type of feature, and the second column indicates how many features of that type were created.

$$P(seg_{i,j}|lab_i) = \begin{cases} \beta_{j,lab_i}, type(j) = boolean\&seg_{i,j} = true \\ (1-\beta_{j,lab_i}), type(j) = boolean\&seg_{i,j} = false \\ \frac{1}{\sqrt{2\pi\sigma^2_{j,lab_i}}} e^{\left(\frac{\left(seg_{i,j}-\mu_{j,lab_i}\right)^2}{2\sigma^2_{j,lab_i}}\right)}, type(j) \neq boolean \end{cases} \tag{6.11}$$

Finally, since the likelihood of each pixel's on-off state is independent of everything except the label of the region containing it, only a single variable ρ_k is required to represent the likelihood that a given pixel inside a region with label k is on. The likelihood of the third term $P(PIX|SEG, LAB)$ can be expressed as:

$$P(PIX|SEG,LAB) = \prod_i \left(\prod_j \begin{cases} (\rho_{lab_i}), pix_j = on \\ (1-\rho_{lab_i}), pix_j = off \end{cases} \right) \tag{6.12}$$

Where i iterates over all the regions, and j iterates over all pixels within each region. Note that the structural algorithm prevents any pixels in the on state from falling outside of the regions, allowing us to ignore these pixels. If this were not the case, pixels not contained by any region (whether off or on) would need to be considered in a similar manner, with some very small likelihood ρ_{none} of being on.

Since our algorithm will be maximizing the value of $P(LAB, SEG, PIX)$, and it is known that maximizing the log of a function is equivalent to maximizing the function itself, we are able to work with log-likelihoods, rather than likelihoods. The above equations 6.8, 6.9, 6.11, and 6.12 can be converted to their logs, for working purposes.

$$l(LAB,SEG,PIX) = l(LAB) + l(SEG|LAB) + l(PIX|SEG,LAB) \tag{6.13}$$

$$l(LAB) = \sum_i \log(\alpha_{lab_i}) \tag{6.14}$$

$$l(SEG|LAB) = \sum_i \sum_{,j} \begin{cases} \log\left(\beta_{j,lab_i}\right), type(j) = boolean \& seg_{i,j} = true \\ \log\left(1-\beta_{j,lab_i}\right), type(j) = boolean \& seg_{i,j} = false \\ -\log\left(\sqrt{2\pi\sigma^2_{j,lab_i}}\right) - \frac{\left(seg_{i,j} - \mu_{j,lab_i}\right)^2}{\left(2\sigma^2_{j,lab_i}\right)}, type(j) \neq boolean \end{cases} \tag{6.15}$$

$$l(PIX|SEG,LAB) = \sum_i \left(\left(\rho_{lab_i}\right)\left(pix\ on\ in\ seg_i\right) + \left(1-\rho_{lab_i}\right)\left(pix\ off\ in\ seg_i\right)\right) \tag{6.16}$$

Note that each of the three terms in the sum begin with a summation over each region

i. Equation 6.13 can hence be rearranged as

$$l(LAB, SEG, PIX) = \sum_i (l(lab_i) + l(seg_i|lab_i) + l(pix\ in\ i)|seg_i, lab_i)) \tag{6.17}$$

Given a set of values for the model parameters Θ, these equations allow for simple estimation of the log-likelihood of each segment, and its assigned label. These can be summed together to provide and estimate of the total log-likelihood of a segmentation and label assignment for an entire page. This allow us to evaluate one or more segments against the objective function, providing an implementation of the function "score=evaluate(segments)".

6.1.4 RGS Parameter Estimation

This section outlines how the optimal (given the model assumptions) values for the model parameters Θ can be estimated from a set of training data. Since a very simple generative model has been used, parameter estimation is actually nearly trivial.

The optimal values for the label likelihoods can be found through counting the fraction of the regions in the training data with each label. A constant value of 1 was added to the count of regions of each class to improve generalization.

$$\alpha_k = \frac{count(label = k) + 1}{\sum_j (count(label = j) + 1)} \tag{6.18}$$

For a Gaussian distribution, the optimal mean and standard deviation are simply the mean and standard deviation of the data. Hence the optimal values of $\mu_{j,k}$ and $\sigma_{j,k}$ are the mean and standard deviation of the j'th element in the feature vector for all regions in the training data with label k. A small regularization amount λ was added to each standard deviation, to improve generalization.

Estimation of the label-conditional likelihoods $\beta_{j,k}$ for binary features can be done through simple counts of the values from the training data. As with the label likelihoods, a constant of 1 was added to the count of both the true and false results for each label-conditional feature, to improve generalization.

$$\beta_{j,k} = \frac{count\left((lab_i = k)\ \&\ (seg_{i,j} = true)\right) + 1}{count\left(lab_i = k\right) + 2} \tag{6.19}$$

Estimation of ρ_k was similarly simple. For each class k, the optimal value of ρ_k is simply the fraction of the pixels in all the regions in the training data with label k that are in the on state. Once again, a small amount of regularization was applied to improve generalization.

$$\rho_k = \frac{count(on\, pixels\ in\ regions\ with\ label\ k) + 1}{count(all\, pixels\ in\ regions\ with\ label\ k) + 2} \tag{6.20}$$

All these model parameters can be estimated through a single pass through a set of training data, making for extremely simple, fast training.

6.1.5 Results and Discussion

Preliminary results with this algorithm were quite poor. These poor results were likely caused by two problems found with the model, as well as by a violation of the model assumptions in our implementation. Our implementation contained features in the representation of the segments that were functions of the pixel distribution, violating the assumption expressed in Equation 6.5. Additionally, two problems with the model were identified. First, the test-time performance of the model was, while tractable, nonetheless very computationally expensive; on the computer systems available for this research, segmentation of a single page would take upward of twenty minutes. Second, the construction of the model ended up vastly favoring over-segmentations of the page, for reasons that will be discussed below.

Two aspects of the model independently favored over-segmentation. The first was the assumption of pixel independence made when estimating $P(PIX|LAB, SEG)$. The second was the use of Gaussian probability density estimation for continuous-valued features in $P(SEG|LAB)$.

The first cause of over-segmentation was the fact that the third term in the objective function, $P(PIX|LAB, SEG)$, tended both to favor over-segmentation, and was by far the most significant term. It is simple to see how this term favored over-segmentation. For most region labels k, the value of ρ_k (the fraction of pixels that were on in the training data inside regions with label k) was between 0.1 and 0.4. This means that for each "off" pixel that is inside a region at test time, the likelihood estimate for that region (and hence of the objective function) is multiplied by a value of, say, 0.9. An extremely small cut separating two regions may create a block of whitespace, 100 pixels by 10 pixels, moving 1,000 pixels from inside a region to outside a region (a more common cut size would be around 1000 pixels by 20 pixels). This increases the likelihood score of the page by a factor of $1/0.9$ for each of these pixels, for a total likelihood increase of 5.7×10^{45}. The sheer number of pixels present on in even a small area of the page allows this term to effectively swamp all the others.

An ideal way to eliminate with this problem would be to optimize $P(LAB, SEG|PIX)$, rather than $P(LAB, SEG, PIX)$. The difference is subtle, yet significant: if we were to optimize for $P(LAB, SEG|PIX)$, we would need to normalize for all possible distributions of the pixels, as shown in Equation 6.21.

$$P(LAB, SEG|PIX) = \frac{P(LAB, SEG, PIX)}{P(PIX)} \tag{6.21}$$

For the model we used of $P(LAB, SEG, PIX)$, we are simply finding the parameter values Θ that simply maximize the likelihood of the joint distribution of the sample data. The optimal set of parameters Θ for a model of $P(LAB, SEG|PIX)$ would, however, be those that maximize the ratio of the sum of the log-likelihoods of the actual segmentation and labeling examples seen to the sum of the log likelihoods of all possible labeling and segmentation schemes for the given set of training data pages (and their associated pixel states). Such regularization presents an ideal opportunity for future work.

The second reason for over-segmentation was the use of Gaussian probability density estimate for continuous-valued features. The probability density for a given value of a feature is directly proportional to the probability itself. When simply comparing the likelihoods for a single feature with different possible labels, this fact allows us to use the probability density estimate as an effective relative likelihood estimate, as is done with Gaussian Naive Bayes classifiers. The fact that it is possible, when the standard deviation is small, to achieve likelihood estimates greater than 1, is not significant in the case of ordinary classification, since it effectively means multiplying the likelihood estimates for all candidates by the same constant amount. In our case, however, the number of items being classified is not constant - it varies depending on the segmentation chosen. The constant amount by which each likelihood estimate is effectively multiplied has a tremendous effect on the segmentation aspect of our model. In the case of our data, the majority of segments did in fact have a label assignment that yielded a likelihood estimate greater than 1. The result of this was a tendency for the system to choose the segmentation with the largest total number of regions, hence over-segmenting the page. This problem could be eliminated in future work by discretizing all variables, and eliminating the use of Gaussian probability density estimation.

In order to compensate for the tendency toward over-segmentation, a "third term scaling factor" τ and an additional "per-region penalty" ϕ was added to the objective function.

$$l(LAB, SEG, PIX) = \sum_i (l(lab_i) + l(seg_i|lab_i) + \tau l(pix\ in\ i)|seg_i, lab_i) + \phi)$$

Because the test-time performance was so slow, cross-validation for values of ϕ and τ was not feasible in this study. Instead, an attempt was made to estimate reasonable values by hand, based on a small sample of page images. Proper cross-validation for these values presents an excellent opportunity for future work.

Chapter 7

Applications

Several applications exist for document segmentation and labeling systems. This chapter introduces several such applications, and presents examples of performance.

7.1 Optical Character Recognition Pre-Processor

Optical character recognition pre-processors are such a popular application of document understanding systems that the two terms have nearly become synonymous. This application consists of several sub-domains for which different measures of performance are appropriate. Three such sub-domains are discussed in this chapter: searchable image-documents, compression and storage, and preparation for display on the web.

7.1.1 Searchable Image-Documents

The primary target application kept in mind during the course of this study was that of creating searchable image-documents. The goal was twofold: to create a system that would allow an automated searching system for academic journals, such as CiteSeer, to be reliably extended to journals for which only images are available, and second to allow such systems to make use of the labeling information to improve results.

The most significant aspects of such as system are whether the text of the document is accurately reproduced by OCR, and whether regions have been correctly classified among the 25 precise labels. Many of the most common segmentation "errors" made by the algorithms tested, such as incorrectly separating paragraphs into separate regions, or joining multiple references into a single region, would not in any way degrade performance for this application. The third segmentation performance metric used in Chapter 4.2.2 is a good measure of segmentation performance for this application. The labeling performance on the 25 precise labels in Chapter 5 is a good measure of labeling performance.

7.1.2 Compression and Storage

Storing an image of each entire page is a remarkably inefficient method of storing journals. A single standard 8.5 x 11 page, sampled at 170 DPI, contains over 2.7 million pixels that need to be represented. Since most of this page is actually text, it would be far more efficient to store the text regions as text, and only store the raw pixels for the images. The DjVu document compression system [1] from ATT Labs achieves extremely high compression rates using such techniques.

Performance on this task is best measured by how accurately text is preserved, and by how high a degree of compression is achieved. Labeling performance should be evaluated in terms of the 4 generalized labels, since these are the categories that would require different treatment in this application.

7.1.3 Preparation For HTML

Another popular reason to apply OCR to a document is to prepare it for display on the World Wide Web in HTML format. Having accurate segmentation and labeling information allows for the creation of an OCR pre-processor designed specifically for preparation for

[1]http://www.djvuzone.org

HTML.

Evaluating the segmentation and labeling results for this application depends greatly on the details of the target HTML preparation system. A simple such system would include just the post-OCR text from all text regions, and images of non-text regions. In this case, segmentation performance would be in the same manner as for searchable image documents, whereas labeling performance would be measured in terms of the four generalized categories. More complex systems that attempted to create (or recreate) complex formatting may require different performance metrics.

7.1.4 Prototype System

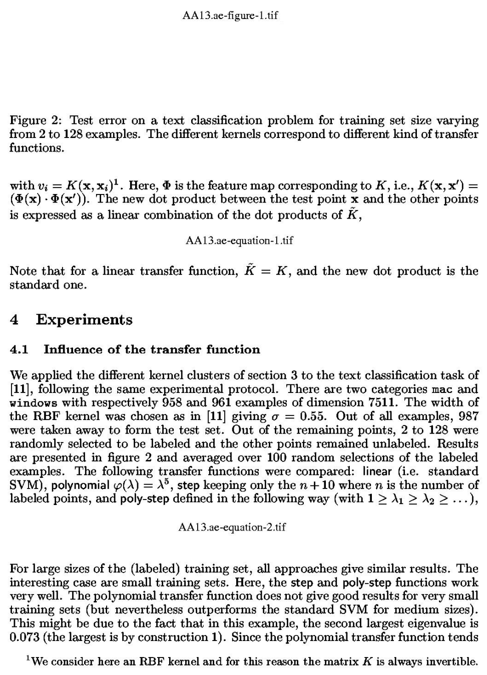
A simple pre-processing system was built to prepare a page for OCR. The intent of this system is to replace non-text regions with a single line of text that indicates where to find an image of the non-text region. If the non text regions were put directly through the OCR system, they would at best be ignored, and at worst be translated into a collection of garbled text.

The system begins by running one of the implemented segmentation algorithms on the page. The system currently used Concurrent Learn To Cut, as this algorithm has achieved the best results.

The results of the segmentation are then passed to a labeling algorithm. The current system uses the logistic regression algorithm for labeling. While the MEMM tested does provide somewhat better results, it is by far the most computationally intensive at test time, whereas logistic regression is extremely fast.

Once the segmentation and labeling steps are complete, the non-text regions must be removed from the page image. Each non-text region is saved as an image file. The region on the page is then blanked out. A single line of text referencing the newly created image file is placed in the center of the blanked out region. The resulting page image is then re-saved to a new image file. Examples of pages that have been put through the system are shown in

Figure 7.1.



AA13.ae-figure-1.tif

Figure 2: Test error on a text classification problem for training set size varying from 2 to 128 examples. The different kernels correspond to different kind of transfer functions.

with $v_i = K(\mathbf{x}, \mathbf{x}_i)^1$. Here, Φ is the feature map corresponding to K, i.e., $K(\mathbf{x}, \mathbf{x}') = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}'))$. The new dot product between the test point $\mathbf{x}$ and the other points is expressed as a linear combination of the dot products of $\tilde{K}$,

AA13.ae-equation-1.tif

Note that for a linear transfer function, $\tilde{K} = K$, and the new dot product is the standard one.

4 Experiments

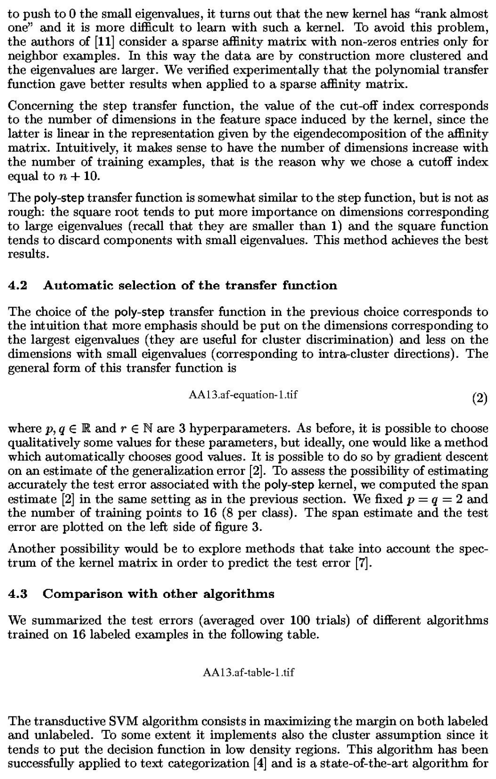
4.1 Influence of the transfer function

We applied the different kernel clusters of section 3 to the text classification task of [11], following the same experimental protocol. There are two categories `mac` and `windows` with respectively 958 and 961 examples of dimension 7511. The width of the RBF kernel was chosen as in [11] giving $\sigma = 0.55$. Out of all examples, 987 were taken away to form the test set. Out of the remaining points, 2 to 128 were randomly selected to be labeled and the other points remained unlabeled. Results are presented in figure 2 and averaged over 100 random selections of the labeled examples. The following transfer functions were compared: linear (i.e. standard SVM), polynomial $\varphi(\lambda) = \lambda^5$, step keeping only the $n + 10$ where n is the number of labeled points, and poly-step defined in the following way (with $1 \geq \lambda_1 \geq \lambda_2 \geq \ldots$),

AA13.ae-equation-2.tif

For large sizes of the (labeled) training set, all approaches give similar results. The interesting case are small training sets. Here, the step and poly-step functions work very well. The polynomial transfer function does not give good results for very small training sets (but nevertheless outperforms the standard SVM for medium sizes). This might be due to the fact that in this example, the second largest eigenvalue is 0.073 (the largest is by construction 1). Since the polynomial transfer function tends

[1]We consider here an RBF kernel and for this reason the matrix K is always invertible.



to push to 0 the small eigenvalues, it turns out that the new kernel has "rank almost one" and it is more difficult to learn with such a kernel. To avoid this problem, the authors of [11] consider a sparse affinity matrix with non-zeros entries only for neighbor examples. In this way the data are by construction more clustered and the eigenvalues are larger. We verified experimentally that the polynomial transfer function gave better results when applied to a sparse affinity matrix.

Concerning the step transfer function, the value of the cut-off index corresponds to the number of dimensions in the feature space induced by the kernel, since the latter is linear in the representation given by the eigendecomposition of the affinity matrix. Intuitively, it makes sense to have the number of dimensions increase with the number of training examples, that is the reason why we chose a cutoff index equal to $n + 10$.

The poly-step transfer function is somewhat similar to the step function, but is not as rough: the square root tends to put more importance on dimensions corresponding to large eigenvalues (recall that they are smaller than 1) and the square function tends to discard components with small eigenvalues. This method achieves the best results.

4.2 Automatic selection of the transfer function

The choice of the poly-step transfer function in the previous choice corresponds to the intuition that more emphasis should be put on the dimensions corresponding to the largest eigenvalues (they are useful for cluster discrimination) and less on the dimensions with small eigenvalues (corresponding to intra-cluster directions). The general form of this transfer function is

AA13.af-equation-1.tif (2)

where $p, q \in \mathbb{R}$ and $r \in \mathbb{N}$ are 3 hyperparameters. As before, it is possible to choose qualitatively some values for these parameters, but ideally, one would like a method which automatically chooses good values. It is possible to do so by gradient descent on an estimate of the generalization error [2]. To assess the possibility of estimating accurately the test error associated with the poly-step kernel, we computed the span estimate [2] in the same setting as in the previous section. We fixed $p = q = 2$ and the number of training points to 16 (8 per class). The span estimate and the test error are plotted on the left side of figure 3.

Another possibility would be to explore methods that take into account the spectrum of the kernel matrix in order to predict the test error [7].

4.3 Comparison with other algorithms

We summarized the test errors (averaged over 100 trials) of different algorithms trained on 16 labeled examples in the following table.

AA13.af-table-1.tif

The transductive SVM algorithm consists in maximizing the margin on both labeled and unlabeled. To some extent it implements also the cluster assumption since it tends to put the decision function in low density regions. This algorithm has been successfully applied to text categorization [4] and is a state-of-the-art algorithm for

Figure 7.1: Two examples of pages that have been pre-processed for OCR.

7.2 Document Re-Flowing

In addition to making it text-searchable, one possible reason to run a document through optical character recognition is to include it in a new publication. The new publication is unlikely to use the same formatting scheme, so the original document will need to be re-flowed to fit the new scheme. While this generally involves running OCR on the document, and then manually extracting the images, with accurate segmentation and labeling information the re-flowing can be done without ever using OCR.

We have implemented a rudimentary reflowing system for converting single-column documents into multi-column documents. It can be used either autonomously, making use of the segmentation and labeling algorithms developed, or on document images which have been segmented and labeled by hand. The steps of the system are as follows:

1. Region segmentation
2. Region labeling
3. Region ordering
4. Text line identification in text regions
5. Word identification in text lines
6. Paragraph reconstruction
7. Region re-flowing
8. Column re-flowing

The first two steps, segmentation and labeling, have already been dealt with in depth. This chapter will provide brief descriptions of each of the steps above, followed by several examples of reflowed pages.

7.2.1 Steps of Reflowing System

The reflowing system developed was meant to be a proof-of-concept. Descriptions of the simplistic systems used to implement each of the steps are given below, along with suggestions for how a more robust system could be built.

Region ordering

Reflowing a document means laying down the regions with the same reading order, but a different page arrangement. In order to do this, the original reading order of the document must be determined.

Our prototype system assumes that incoming pages are single-column, and uses a simple top-to-bottom, left-to-right procedure for ordering the regions. This algorithm is described in detail in Chapter 5.5.3, where it was used to order the regions on a page for the application of Maximum Entropy Markov models for the labeling problem. A more general ordering system, such as the Soft Ordering procedure [17], would likely improve results.

Text line identification in text regions

Text regions of all types were broken down into individual lines of text. A variant of the simple smearing algorithm described in Chapter 4.3.3 was used for this purpose.

Lines of text are often crowed very close together. In many cases, the projection profiles do not have even a single pixel of separation between the lines. Different lines of text almost never, however, contain "touching" marks, allowing us to use the simple smearing algorithm.

This process begins by smearing the whitespace by one pixel both up and down. This ensures that lines of text in which are very close together will have at least one pixel of separation in the horizontal projection profile. Black pixels are then smeared 20 pixels in the horizontal direction, connecting consecutive letters and words together. The value of 20 pixels was estimated by hand as an appropriate value.

Individual marks of ink are then identified, and bounding boxes placed around them. These same bounding boxes are then placed on the original page image, and "snapped" to the edges of the ink they contain, as demonstrated in Figure 3.3. Any overlapping or contained segments (such as those sometimes created by the dots above lowercase i and j) are then merged. These text lines are sorted top-to-bottom, left-to-right, as described in Chapter 5.5.3, with a slope of $\alpha = 10$.

Word identification

When a document is re-flowed, lines of text are likely to be either shorter or longer than in the original document. To accommodate this, individual words must be identified, so that they can be kept together in the new document.

The XY Cuts algorithm described in Chapter 4.3.2 was applied to each line of text in order to identify individual words. Note that since the lines of text are presumed to be separated, only vertical cuts should end up being made. A vertical cut threshold of 6 pixels was used, and a horizontal cut threshold of 10 (as no horizontal cuts are wanted, any prohibitively large horizontal cut threshold is appropriate). The resulting individual word images were sorted left-to-right.

The results of this process could have been improved by identifying words that had been split between two lines of text in the original document, and re-merging them.

Paragraph Reconstruction

An ideal reflowing system would identify individual paragraphs. Long paragraphs are often divided by page breaks or figures. Reconstructing these paragraphs would allow for better layout of the new document. Likewise, identifying the separations between paragraphs in the original document would allow these separations to be maintained in the new document.

Our system assumes that any two text blocks separated by a page break and any number of page number, footnote, and decoration regions may actually be a single paragraph. In particular, if the first of the two regions contains a character in the bottom-right corner (indicating that the text reached the end of the last line), the two regions are assumed to represent a single paragraphs. The word images for the two regions (found in the previous step) are then joined together, effectively creating a single text region.

This system could be improved significantly by detecting the indentations that generally begin paragraphs. These could be used to separate paragraphs into different regions, preventing them from being incorrectly merged into a single paragraph in the new document. In addition, these could be used to determine whether text regions on either side of a page break or figure should be merged into a single region.

Region re-flowing

Each text region can now be re-flowed to a new width. A simple two step process is used to accomplish this.

First, individual words are strung together horizontally until, with a set amount of whitespace between them, adding another word would exceed the target width for the region. This creates the lines of text for the new region.

Lines of text are then concatenated vertically, creating a block of ink for placement in the new document. The individual lines of text are, however, retained, in case the region can be best laid out by splitting it between two pages.

Column re-flowing

Column re-flowing is perhaps the most involved step in this process. The goal is to produce attractive, compact page distributions that (generally) maintain the reading order of the original document.

Certain formatting rules should be followed when reflowing a document. For example, some pairs of regions, such as a figure and its label, should be kept together in the same column. The rules enforced by our system are:

- Figures and their caption must be kept together.
- Tables and their caption must be kept together.
- Equations and their number must be kept together.
- Footnotes should be placed at the bottom of a page. Note that the correct page cannot easily be determined without applying OCR to find the location in the text of the footnote.
- Decoration regions above footnotes should remain above the footnotes.
- Page numbers and footers should remain at the bottom of the page.

- Headers should remain on the bottom of the page.
- Centering, and right-alignment of regions should be maintained.
- Text regions should be fully justified, except for the last line.
- Certain regions may not be placed side-by-side with another region in the same column.
- Only plain text regions may be divided between multiple columns.

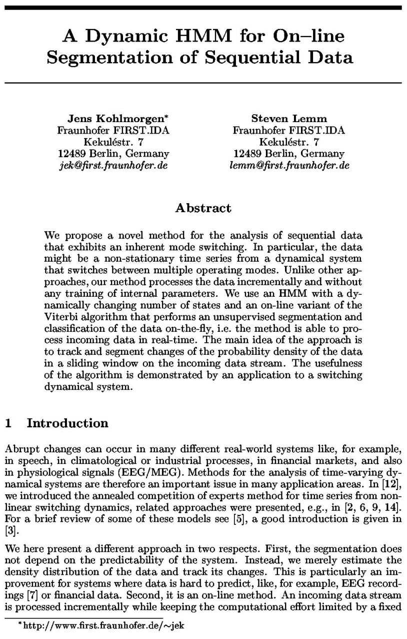
This list represents a minimal number of formatting rules that can be used to perform reasonable two-column reflowing. An improved, more exhaustive set of rules could be created.

The column arrangement algorithm is described by the pseudo-code below.

```
Start a new page

While any blocks remain:
  If the next block does not fit vertically within this page:
    If it is a text region:
      Add as many lines to this page as possible, then start a new page.
    Else:
      Start a new page.
  If the next block does fit vertically within this page:
    While any blocks remain:
      If the next block can be placed beside this one without going
      outside the column bounds:
        Create a new block by placing the two blocks beside each other.
    If the current block is NOT required to go at the page bottom:
      Place it at the top of the page
    Otherwise:
      If are already regions on the bottom of this page:
        Shift these upward.
      Place the block on the bottom of the page.
```

While this is likely not an ideal column layout algorithm, the results and run-time efficiency were sufficient for it to act as a proof of concept. The results of the column reflowing can simply be placed side-by-side to provide a multi-column page layout.



A Dynamic HMM for On–line Segmentation of Sequential Data

Jens Kohlmorgen*
Fraunhofer FIRST.IDA
Kekuléstr. 7
12489 Berlin, Germany
jek@first.fraunhofer.de

Steven Lemm
Fraunhofer FIRST.IDA
Kekuléstr. 7
12489 Berlin, Germany
lemm@first.fraunhofer.de

Abstract

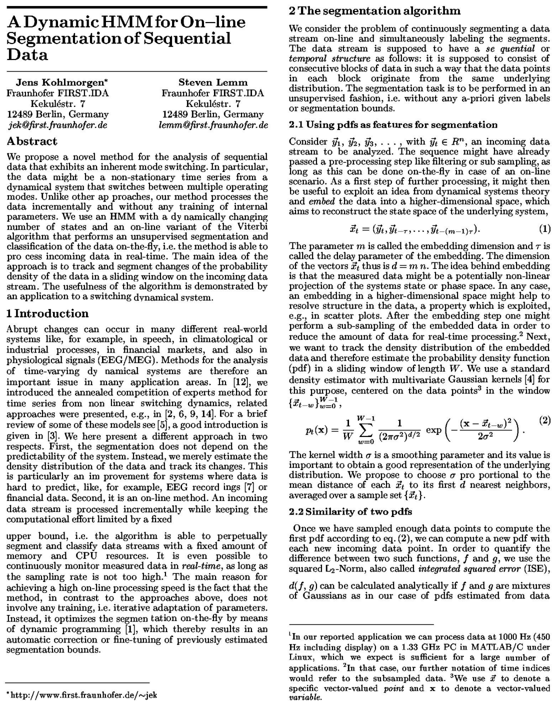
We propose a novel method for the analysis of sequential data that exhibits an inherent mode switching. In particular, the data might be a non-stationary time series from a dynamical system that switches between multiple operating modes. Unlike other approaches, our method processes the data incrementally and without any training of internal parameters. We use an HMM with a dynamically changing number of states and an on-line variant of the Viterbi algorithm that performs an unsupervised segmentation and classification of the data on-the-fly, i.e. the method is able to process incoming data in real-time. The main idea of the approach is to track and segment changes of the probability density of the data in a sliding window on the incoming data stream. The usefulness of the algorithm is demonstrated by an application to a switching dynamical system.

1 Introduction

Abrupt changes can occur in many different real-world systems like, for example, in speech, in climatological or industrial processes, in financial markets, and also in physiological signals (EEG/MEG). Methods for the analysis of time-varying dynamical systems are therefore an important issue in many application areas. In [12], we introduced the annealed competition of experts method for time series from non-linear switching dynamics, related approaches were presented, e.g., in [2, 6, 9, 14]. For a brief review of some of these models see [5], a good introduction is given in [3].

We here present a different approach in two respects. First, the segmentation does not depend on the predictability of the system. Instead, we merely estimate the density distribution of the data and track its changes. This is particularly an improvement for systems where data is hard to predict, like, for example, EEG recordings [7] or financial data. Second, it is an on-line method. An incoming data stream is processed incrementally while keeping the computational effort limited by a fixed

*http://www.first.fraunhofer.de/~jek



A Dynamic HMM for On–line Segmentation of Sequential Data

Jens Kohlmorgen*
Fraunhofer FIRST.IDA
Kekuléstr. 7
12489 Berlin, Germany
jek@first.fraunhofer.de

Steven Lemm
Fraunhofer FIRST.IDA
Kekuléstr. 7
12489 Berlin, Germany
lemm@first.fraunhofer.de

Abstract

We propose a novel method for the analysis of sequential data that exhibits an inherent mode switching. In particular, the data might be a non-stationary time series from a dynamical system that switches between multiple operating modes. Unlike other ap proaches, our method processes the data incrementally and without any training of internal parameters. We use an HMM with a dy namically changing number of states and an on-line variant of the Viterbi algorithm that performs an unsupervised segmentation and classification of the data on-the-fly, i.e. the method is able to pro cess incoming data in real-time. The main idea of the approach is to track and segment changes of the probability density of the data in a sliding window on the incoming data stream. The usefulness of the algorithm is demonstrated by an application to a switching dynamical system.

1 Introduction

Abrupt changes can occur in many different real-world systems like, for example, in speech, in climatological or industrial processes, in financial markets, and also in physiological signals (EEG/MEG). Methods for the analysis of time-varying dy namical systems are therefore an important issue in many application areas. In [12], we introduced the annealed competition of experts method for time series from non linear switching dynamics, related approaches were presented, e.g., in [2, 6, 9, 14]. For a brief review of some of these models see [5], a good introduction is given in [3]. We here present a different approach in two respects. First, the segmentation does not depend on the predictability of the system. Instead, we merely estimate the density distribution of the data and track its changes. This is particularly an im provement for systems where data is hard to predict, like, for example, EEG record ings [7] or financial data. Second, it is an on-line method. An incoming data stream is processed incrementally while keeping the computational effort limited by a fixed

upper bound, i.e. the algorithm is able to perpetually segment and classify data streams with a fixed amount of memory and CPU resources. It is even possible to continuously monitor measured data in *real-time*, as long as the sampling rate is not too high.[1] The main reason for achieving a high on-line processing speed is the fact that the method, in contrast to the approaches above, does not involve any training, i.e. iterative adaptation of parameters. Instead, it optimizes the segmen tation on-the-fly by means of dynamic programming [1], which thereby results in an automatic correction or fine-tuning of previously estimated segmentation bounds.

*http://www.first.fraunhofer.de/~jek

2 The segmentation algorithm

We consider the problem of continuously segmenting a data stream on-line and simultaneously labeling the segments. The data stream is supposed to have a *se quential* or *temporal structure* as follows: it is supposed to consist of consecutive blocks of data in such a way that the data points in each block originate from the same underlying distribution. The segmentation task is to be performed in an unsupervised fashion, i.e. without any a-priori given labels or segmentation bounds.

2.1 Using pdfs as features for segmentation

Consider $\vec{y}_1$, $\vec{y}_2$, $\vec{y}_3$, . . . , with $\vec{y}_t \in R^n$, an incoming data stream to be analyzed. The sequence might have already passed a pre-processing step like filtering or sub sampling, as long as this can be done on-the-fly in case of an on-line scenario. As a first step of further processing, it might then be useful to exploit an idea from dynamical systems theory and *embed* the data into a higher-dimensional space, which aims to reconstruct the state space of the underlying system,

$$\vec{x}_t = (\vec{y}_t, \vec{y}_{t-\tau}, \ldots, \vec{y}_{t-(m-1)\tau}). \tag{1}$$

The parameter m is called the embedding dimension and τ is called the delay parameter of the embedding. The dimension of the vectors $\vec{x}_t$ thus is $d = m\,n$. The idea behind embedding is that the measured data might be a potentially non-linear projection of the systems state or phase space. In any case, an embedding in a higher-dimensional space might help to resolve structure in the data, a property which is exploited, e.g., in scatter plots. After the embedding step one might perform a sub-sampling of the embedded data in order to reduce the amount of data for real-time processing.[2] Next, we want to track the density distribution of the embedded data and therefore estimate the probability density function (pdf) in a sliding window of length W. We use a standard density estimator with multivariate Gaussian kernels [4] for this purpose, centered on the data points[3] in the window $\{\vec{x}_{t-w}\}_{w=0}^{W-1}$,

$$p_t(\mathbf{x}) = \frac{1}{W} \sum_{w=0}^{W-1} \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(-\frac{(\mathbf{x} - \vec{x}_{t-w})^2}{2\sigma^2}\right). \tag{2}$$

The kernel width σ is a smoothing parameter and its value is important to obtain a good representation of the underlying distribution. We propose to choose σ pro portional to the mean distance of each $\vec{x}_t$ to its first d nearest neighbors, averaged over a sample set $\{\vec{x}_t\}$.

2.2 Similarity of two pdfs

Once we have sampled enough data points to compute the first pdf according to eq. (2), we can compute a new pdf with each new incoming data point. In order to quantify the difference between two such functions, f and g, we use the squared L_2-Norm, also called *integrated squared error* (ISE),

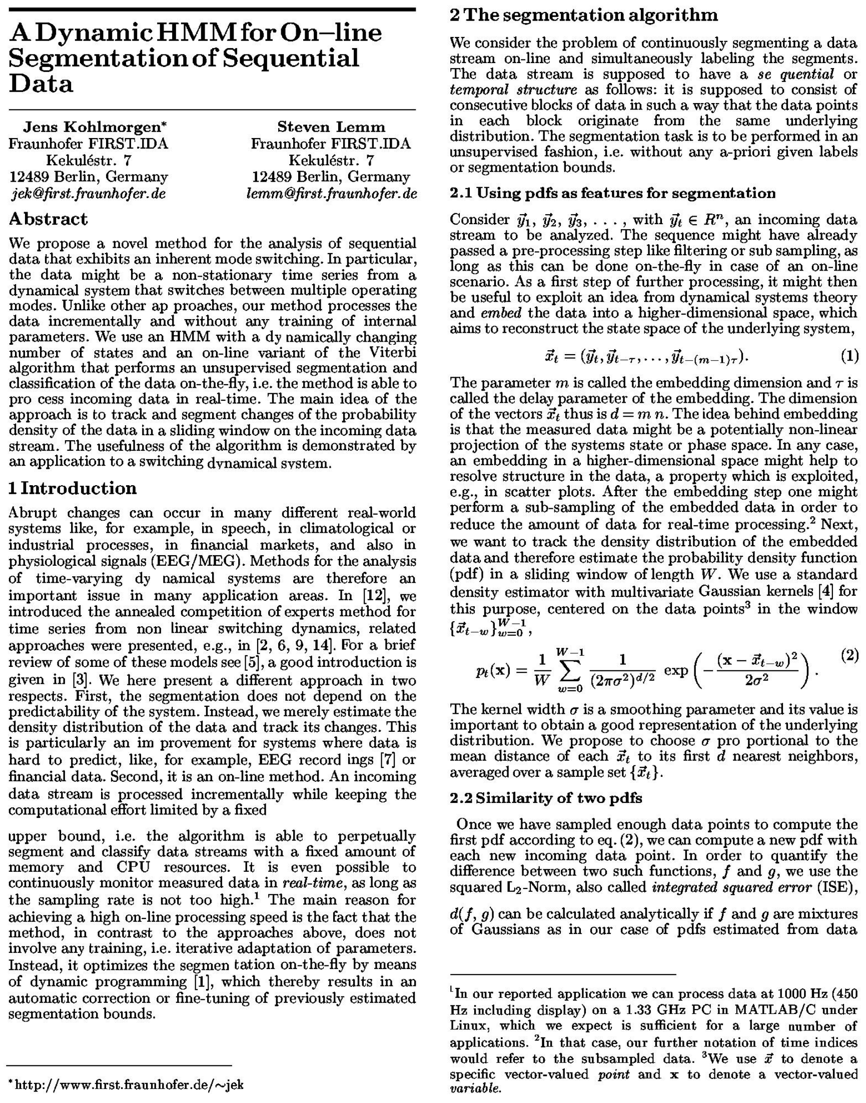
$d(f, g)$ can be calculated analytically if f and g are mixtures of Gaussians as in our case of pdfs estimated from data

[1]In our reported application we can process data at 1000 Hz (450 Hz including display) on a 1.33 GHz PC in MATLAB/C under Linux, which we expect is sufficient for a large number of applications. [2]In that case, our further notation of time indices would refer to the subsampled data. [3]We use $\vec{x}$ to denote a specific vector-valued *point* and $\mathbf{x}$ to denote a vector-valued *variable*.

Figure 7.2: The original first page of a document is shown on the left. On the right is the first page of the same document, re-flowed into a 2-column format.

7.2.2 Reflowing Results

Several examples of original 1-column pages, along with their corresponding reflowed 2-column pages, are provided in Figures 7.2 and 7.3.



A Dynamic HMM for On–line Segmentation of Sequential Data

Jens Kohlmorgen* Fraunhofer FIRST.IDA Kekuléstr. 7 12489 Berlin, Germany *jek@first.fraunhofer.de*

Steven Lemm Fraunhofer FIRST.IDA Kekuléstr. 7 12489 Berlin, Germany *lemm@first.fraunhofer.de*

Abstract

We propose a novel method for the analysis of sequential data that exhibits an inherent mode switching. In particular, the data might be a non-stationary time series from a dynamical system that switches between multiple operating modes. Unlike other ap proaches, our method processes the data incrementally and without any training of internal parameters. We use an HMM with a dy namically changing number of states and an on-line variant of the Viterbi algorithm that performs an unsupervised segmentation and classification of the data on-the-fly, i.e. the method is able to pro cess incoming data in real-time. The main idea of the approach is to track and segment changes of the probability density of the data in a sliding window on the incoming data stream. The usefulness of the algorithm is demonstrated by an application to a switching dynamical system.

1 Introduction

Abrupt changes can occur in many different real-world systems like, for example, in speech, in climatological or industrial processes, in financial markets, and also in physiological signals (EEG/MEG). Methods for the analysis of time-varying dy namical systems are therefore an important issue in many application areas. In [12], we introduced the annealed competition of experts method for time series from non linear switching dynamics, related approaches were presented, e.g., in [2, 6, 9, 14]. For a brief review of some of these models see [5], a good introduction is given in [3]. We here present a different approach in two respects. First, the segmentation does not depend on the predictability of the system. Instead, we merely estimate the density distribution of the data and track its changes. This is particularly an im provement for systems where data is hard to predict, like, for example, EEG record ings [7] or financial data. Second, it is an on-line method. An incoming data stream is processed incrementally while keeping the computational effort limited by a fixed

upper bound, i.e. the algorithm is able to perpetually segment and classify data streams with a fixed amount of memory and CPU resources. It is even possible to continuously monitor measured data in *real-time*, as long as the sampling rate is not too high.[1] The main reason for achieving a high on-line processing speed is the fact that the method, in contrast to the approaches above, does not involve any training, i.e. iterative adaptation of parameters. Instead, it optimizes the segmen tation on-the-fly by means of dynamic programming [1], which thereby results in an automatic correction or fine-tuning of previously estimated segmentation bounds.

*http://www.first.fraunhofer.de/~jek

2 The segmentation algorithm

We consider the problem of continuously segmenting a data stream on-line and simultaneously labeling the segments. The data stream is supposed to have a *se quential* or *temporal structure* as follows: it is supposed to consist of consecutive blocks of data in such a way that the data points in each block originate from the same underlying distribution. The segmentation task is to be performed in an unsupervised fashion, i.e. without any a-priori given labels or segmentation bounds.

2.1 Using pdfs as features for segmentation

Consider $\vec{y}_1$, $\vec{y}_2$, $\vec{y}_3$, . . . , with $\vec{y}_t \in R^n$, an incoming data stream to be analyzed. The sequence might have already passed a pre-processing step like filtering or sub sampling, as long as this can be done on-the-fly in case of an on-line scenario. As a first step of further processing, it might then be useful to exploit an idea from dynamical systems theory and *embed* the data into a higher-dimensional space, which aims to reconstruct the state space of the underlying system,

$$\vec{x}_t = (\vec{y}_t, \vec{y}_{t-\tau}, \ldots, \vec{y}_{t-(m-1)\tau}). \tag{1}$$

The parameter m is called the embedding dimension and τ is called the delay parameter of the embedding. The dimension of the vectors $\vec{x}_t$ thus is $d = m\,n$. The idea behind embedding is that the measured data might be a potentially non-linear projection of the systems state or phase space. In any case, an embedding in a higher-dimensional space might help to resolve structure in the data, a property which is exploited, e.g., in scatter plots. After the embedding step one might perform a sub-sampling of the embedded data in order to reduce the amount of data for real-time processing.[2] Next, we want to track the density distribution of the embedded data and therefore estimate the probability density function (pdf) in a sliding window of length W. We use a standard density estimator with multivariate Gaussian kernels [4] for this purpose, centered on the data points[3] in the window $\{\vec{x}_{t-w}\}_{w=0}^{W-1}$,

$$p_t(\mathbf{x}) = \frac{1}{W} \sum_{w=0}^{W-1} \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(-\frac{(\mathbf{x} - \vec{x}_{t-w})^2}{2\sigma^2}\right). \tag{2}$$

The kernel width σ is a smoothing parameter and its value is important to obtain a good representation of the underlying distribution. We propose to choose σ pro portional to the mean distance of each $\vec{x}_t$ to its first d nearest neighbors, averaged over a sample set $\{\vec{x}_t\}$.

2.2 Similarity of two pdfs

Once we have sampled enough data points to compute the first pdf according to eq. (2), we can compute a new pdf with each new incoming data point. In order to quantify the difference between two such functions, f and g, we use the squared L_2-Norm, also called *integrated squared error* (ISE),

$d(f, g)$ can be calculated analytically if f and g are mixtures of Gaussians as in our case of pdfs estimated from data

[1]In our reported application we can process data at 1000 Hz (450 Hz including display) on a 1.33 GHz PC in MATLAB/C under Linux, which we expect is sufficient for a large number of applications. [2]In that case, our further notation of time indices would refer to the subsampled data. [3]We use $\vec{x}$ to denote a specific vector-valued *point* and $\mathbf{x}$ to denote a vector-valued *variable*.

Figure 7.3: An example of a page that has been reflowed into double column format.

Chapter 8

Conclusions

The primary goal of this research program was to determine the applicability of statistical pattern recognition techniques to the problems of document labeling and segmentation. The research was largely successful, demonstrating that the field of document understanding lends itself well to the model-based approach used in machine learning. Strong results from the Concurrent Learn To Segment algorithm demonstrate that the segmentation problem can be treated effectively as a learning problem. The labeling problem has proven to be a natural fit to the supervised classification paradigm, with results from the application of logistic regression and maximum entropy Markov models coming in extremely strong.

A new model for how ink is laid out on a page has been created, and used to create a combined segmentation / labeling system, dubbed Recursive Generative Segmentation. While RGS did not produce strong results, it offers a starting point for models-based approaches for combined segmentation and labeling.

Several applications for this technology have been discussed, and rudimentary systems implemented to demonstrate their potential.

8.1 Future Work

As this research was largely exploratory, many opportunities for future work have been identified.

8.1.1 Segmentation

The Concurrent Learn-To-Cut algorithm demonstrates enormous potential for solving the segmentation problem for pages following a Manhattan layout scheme. Several possible extensions to the algorithm are proposed below.

It was noted that the logistic regression learner used in this study optimizes for the number of cutting errors, rather than the number of segmentation errors that result. If a manner could be found of determining, from a correctly segmented page, how many segmentation errors would be induced by each cutting error, this could be used to modify the logistic regression learning, likely improving both results and generalization. The Concurrent Learn-To-Segment algorithm could likewise be improved.

Both the learning algorithm and the feature set used for the Concurrent Learn-To-Cut algorithm were fairly simple. A richer feature set would likely improve results. Using a more complex learning algorithm that either allowed for non-linear discrimination, or that did not make the assumption of independent features, would likely improve results.

Another weakness of the Concurrent Learn-To-Cut algorithm lies in the fact that it considers cuts in isolation. Likewise, a weakness of the Concurrent Learn-To-Segment algorithm was the fact that its feature set considered only the space between the cuts (the segment), not the cuts themselves. A new algorithm could be developed using the structural algorithms presented for the Concurrent Learn-To-Segment algorithm could be employed to consider pairs of cuts. A feature set for a cut pair could be constructed by using the Learn-To-Cut features, augmented with features relating to the new region that would be created between the two cuts. Allowing an algorithm to use information about both cuts *and* the space between cuts would likely produce much stronger results.

The generalized bottom-up algorithm provides an opportunity for creating a learning algorithm that would be applicable to non-Manhattan layouts. A discriminant classifier could be trained to determine whether pairs of ink marks belong together, based on feature vectors extracted from the pair of marks. This could yield an improved version of the Voronoi Diagram technique, in much the same way that Concurrent Learn-To-Cut yielded an improved version of the XY Cuts technique. This would likely be a relatively simple, and extremely fruitful, direction of future research.

8.1.2 Labeling

One obvious area for improving performance on the labeling problem would be using more complex learning algorithms. Both the logistic regression and MEMM algorithms learn linear hyperplanes for separating classes. There is no reason to assume that the classes are linearly separable, hence an algorithm that does not make the linearity assumption may perform better.

Labeling performance could also be improved through the use of a richer feature set. For example, many labeling systems make use of the text provided by applying OCR to a region. While this was not done in our study, a variety of powerful features could likely be created from this source.

8.1.3 Combined Segmentation and Labeling

Combining the segmentation and labeling problems remains a prospect for improved performance. The current implementation of Recursive Generative Segmentation provides a starting point, with many potential extensions.

The use of a richer feature set may well yield improved results for the Recursive Generative Segmentation algorithm. Specifically, the feature set should be modified to include more information about the "cuts" of whitespace separating proposed regions from their neighbors (possibly the feature sets in use by the Concurrent Learn-To-Cut algorithm).

A more sophisticated learning mechanism for estimating $P(SEG|LAB)$ may yield improved results. It is clear that neither the Naive assumption, nor the assumption of Gaussian distributions, hold true for the features used. Discretizing all of the features into a small number of bins would make it possible to relax these assumptions. In addition, this method uses actual probability estimates, rather than probability density estimates. As noted in Chapter 6, the use of probability densities to estimate likelihood often leads to likelihood estimates greater than one, causing the system to create as many regions as possible to maximize the total likelihood, which results in a tendency towards over-segmentation.

A version of the algorithm should be developed that considers only a single cut at each level of recursion, rather than a full segmentation of the page. This may prove to be less computationally expensive, making optimization more feasible.

Once the speed of the algorithm has been improved, cross-validation should be used to optimize the parameters in place used to reduce the tendency toward over-segmentation.

A new "discriminative" model for Recursive Segmentation may prove extremely fruitful. Such a model would normalize over all possible segmentation and labeling schemes in an attempt to find the segmentation and label assignment that maximize $P(SEG, LAB|PIX)$. A finite number of segmentations could be provided for this normalization based on the segmentation "trees" provided by the structural algorithm used for Recursive Generative Segmentation.

Another opportunity for combining the segmentation and labeling problems would be to approach the problem from a bottom-up perspective, starting with an over-segmentation of the page, and trying different mergings of segments.

Appendix A

Classes Used For Labeling

This appendix contains a detailed listing of the 25 precise labels used. Each description includes a description of what the label represents, as well as the rules followed for creating these regions when hand-tagging.

header

Generalized label: *TEXT*.

A header at the top of a page. If the header is composed of more than one part (eg part of it is left-aligned, part is right-aligned), these are considered to be separate regions.

code block

Generalized label: *FIGURE*.

A block of code, or code-like text. These often come in the form of pseudo-code, which may contain bullets, equations, and small diagrams. A block of code is considered a single region unless it is clearly divided by a large block of whitespace, or a region of another type. When a block of code is surrounded by a border, this border is considered part of the region.

section heading

Generalized label: *TEXT*.

A top-level heading such as “1. Introduction”. The heading may be numbered, such as “1. Introduction”, or lettered, such as “Appendix A.”. The number of the heading and the associated text are considered a single region.

subsection heading

Generalized label: *TEXT*.

A second, third, fourth, etc. level heading, such as “1.2 Motivation for Research”, “1.2.1.4 Why They Did it That Way”, or “A.2 Detailed proofs”. There is some ambiguity as to what qualifies as a heading, and what is merely a single line of bold text. This data uses the rule that if it was numbered or used a different font, it is a heading.

decoration

Generalized label: *OTHER*.

Bits of ink detached from other regions, used to decorate or divide the space. Some decorations are not considered separate regions: a vertical line along the left edge of another region is considered part of that region, and a small square box used to denote the end of a proof may be considered part of a text region if it is not separated from the text by whitespace. Common decorations include horizontal lines above footnotes, and squares used to denote QED at the end of proofs.

figure caption

Generalized label: *TEXT*.

The caption associated with a figure. If the figure label is included with the figure caption in a single rectangular block, it will be considered part of the figure caption.

figure label

Generalized label: *TEXT*.

The label assigned to a figure, such as "Figure 3". This is only considered a figure label if it is separated in some way from the figure caption, such as:

```
Figure 3.  This is the caption that is associated with figure 3.  It
           describes the contents of the figure.  In this particular
           case, the figure label is separated from the caption, and
           would be considered a separate region.
```

references

Generalized label: *TEXT*.

References, usually found at the end of a paper. Each reference is considered to be its own region.

abstract

Generalized label: *TEXT*.

The abstract, usually found at the beginning of the paper. The word "Abstract", which generally appears at the top, is also part of this region.

text

Generalized label: *TEXT*.

Any block of text not classified as another region. This includes, but is not limited to, items such as paragraphs of body text and isolated lines of text between equations. If paragraphs have a full line of whitespace between them, they are separated into different regions; otherwise, they are grouped together. The size of the line of whitespace between paragraphs is somewhat subjective, and there may be inconsistencies in the ground truth data in terms of which paragraphs were separated, and which were not.

bullet item

Generalized label: *TEXT*.

A single item in a bulleted or numbered list. Even if there is no line of whitespace between lines, multiple bullet items are each given a separate region.

pg number

Generalized label: *TEXT*.

The page number. Usually found at the bottom or top of the page.

table

Generalized label: *FIGURE*.

A table. Cells may be separated by lines, or by whitespace. May or may not be associated with a label or caption.

main title

Generalized label: *TEXT*.

The main title of the paper. A main title that spans multiple lines is considered a single region.

footer

Generalized label: *TEXT*.

A footer is a small bit of text repeated at the bottom of each page. This is NOT the same as a footnote, which explains some detail about the contents of the paper.

table cation

Generalized label: *TEXT*.

The caption associated with a table. If the table label is included with the table caption in a single rectangular block, it will be considered part of the table caption.

table label

Generalized label: *TEXT*.

The label assigned to a table, such as "Table 3". This is only considered a table label if it is separated in some way from the table caption, such as:

```
Table 3.  This is the caption that is associated with table 3.  It
          describes the contents of the table.  In this particular
          case, the table label is separated from the caption, and
          would be considered a separate region.
```

equation

Generalized label: *EQUATION*.

An equation that is given its own line. Small equations placed in the middle of a paragraph are considered part of that text region; they are not labeled as equations. For numbered equations, the number is generally given its own region. An exception is made if the equation number would be contained by a box drawn around the equation - in this case, the equation number is considered part of the equation region.

eq number

Generalized label: *TEXT*.

The number (or letter) assigned to a particular equation. Always paired with an *equation* region.

author list

Generalized label: *TEXT.*

Information about the authors of the paper. Usually comes at the beginning of the paper. If some information is left-aligned, and some is right-aligned, and there is a significant space between them, the left and right aligned sections each get their own region.

footnote

Generalized label: *TEXT.*

One or more footnotes explaining some detail about the contents of the paper. This is NOT the same as a footer, which is a small bit of repeating text found at the bottom of each page.

image

Generalized label: *FIGURE.*

A single picture (such as a photograph) in the paper. The picture can be modified in any number of ways, but must be recognizable as a picture, rather than a diagram. If multiple pictures are placed too close together to be effectively separated, they are grouped together in a *figure* region.

graph

Generalized label: *FIGURE.*

A single graph or chart. Must contain a drawn or strongly implied set of axes. Multiple graphs that are too close together to be reasonably segmented are not labeled as graphs; they are instead grouped together in a single *figure* region.

figure

Generalized label: *FIGURE.*

Any figure that does not fit into any of the other classes (code block, image, graph, etc.). Most figures are diagrams used to illustrate something. Multiple graphs that are too close together to reasonably segment are classified as figures.

editor list

Generalized label: *FIGURE.*

List of editors for a paper. If multiple editors are listed, with significant whitespace separating them, each will be contained in a separate region.

Appendix B

JTAG Technical Documentation

This chapter includes the technical documentation of the JTAG software. Both the software and the documentation were produced by Scott Leishman in the summer of 2003 at the University of Toronto. These documents (with minor editing done for formatting) are included here with the permission of the original author, Scott Leishman.

B.1 JTAG Requirements Specification

```
1. INTRODUCTION
---------------

1.1 Purpose:
    This document is intended to capture all specifications and
    functionality required of a software application used to identify and
    tag various components of scanned journal articles.

1.2 Overview:
    The jtag application software will be designed with the goal of allowing a
    user to load and display scanned image representations of journal articles
    so that different portions can be selected and classified by the user.
    These portions will include (but are not limited to) body text, equations,
    images, headers, tables, titles, and page numbers etc.  By keeping track
    of these classifications, we should then be able to build up a corpus of
    data that can be used to train a computer to automatically classify
    new journal articles into various components.

    The machine learning algorithms used to automatically classify the
    components of a journal image page will be rolled into a separate
    application which jtag will interact with.  The jtag application will
    send the journal currently being displayed to this application for
    automatic classification, then interpret the results from that application
    to display to the user the classifications it predicted.  The user can then
    look through the classified journal and make any necessary corrections
    manually.  The user can also choose to turn off classification suggestions
    and make their own classifications without aid of the external application.


2. MAJOR FUNCTIONALITY
----------------------

2.1 Display image representations of scanned journal articles
    * should handle various image formats (tiff, jpg, bmp, etc.)
      - Use Img package since it supports all of the above transparently
    * should have the ability to zoom in/out of the displayed image
    * provide scrollbars to navigate over portions of the image if it is
      too large to fit in the window
    * must be able to flip between pages in multiple page images (note that
      separate functionality will be provided to break up such images into
      multiple files -- see 2.7)

2.2 Select and "Crop" any portion of the displayed journal with the mouse
    * work with quadrilaterals only
    * 2 different modes:
      - typical "crop" to make a complete bounding box
      - simple horizontal selection.  Bound by page width to create bounding
        box
    * ability to redo or refine selection
    * ensure that selections are unique, and that no pixels are classified
      two or more times
    * store x,y co-ords of selection.  Since always square, need 4 #'s
      x_left, x_right, y_min, y_max.  Use UNIX conv. 0,0 = top left corner
    * store time to make each selection + time to classify selection (move it
      to a bucket) by timestamping each event
    * display cropped selection in default colour (black) until it has
      been classified (moved to a bucket)

2.3 Classify selected portion
    * "buckets" around document setup so that selections can be dragged over
       them (each bucket represents a different journal component)
    * bucket border width should change when a selection can be placed over
      it (selection itself will be a small icon)
    * must be able to configure (add/remove) buckets (through config file on
      startup only)
    * use of colour coding for each bucket/selection
    * ability to reclassify selections.  Make a classified selection active by
      clicking on it, then drag to a new bucket to reclassify.
    * store number of times selection was reclassified (user changed bucket)
    * Must save selections between iterations of opening and closing file
      (See section 5)

2.4 Display selections made
    * Should be able to display all selections made so far
    * Will use colour coding to show what each selection has been classified as

2.5 Read in Configuration Data from a file on startup
    * when loading selection data, look for config data in that same file
    * on program startup look in current directory, then $HOME for a file
      named .jconfig
    * if not found, load some default set of config options
    * Data in config file will include:
      - list of valid classification types and their associated colour
      - default selection mode (crop vs. simple)
      - default output directory / filename suffix
      - turn on/off look-ahead feature (see 2.8)

2.6 Store current classifications efficiently in memory, dump to file on
    save/exit
    * When user changes active selection, must be able to retrieve the
      selection based on where user clicks with mouse
    * Write to disk in order of top left corner pos, x value, then y value
    * See section 5 for output file format
    * See section 6.1 for data structure used while storing in memory
```

```
2.7 Break multiple page images (single file), into a series of single page
    images (multiple files)
    * Should be handled externally (in a separate application ex. tiffsplit)
    * Will be able to flip between pages in a split multiple page file,
      provided that they are named accordingly
    * If a file is loaded with a format of: <img_base>.aa.<img_format> where
      <img_base> denotes the file base and <img_format> denotes its type (ex.
      tiff or jpg), then the document is treated as a split multi-page
      document, and one can flip between pages, provided that they are named
      <img_base>.ab.<img_format> for page 2 and so on up to
      <img_base>.zz.<img_format> for page 26^2
    * If a non-split multipage file is loaded, only the first page will be
      displayed.  In the future, a prompt may be issed to the user informing
      them as such, but this will require extra programming and will be left
      out for now

2.8 Use selections made in previous pages, to look ahead and predict tags
    for future pages.
    * feature not required in initial version, but will be implemented in time
    * will make use of calls to helper application


3. CONSTRAINTS
--------------

3.1 Platform / Software / Libraries
    * Must run on CSLab machines (Red Hat Linux)
    * Write in Tcl/Tk (version 8.4 has been used and tested)
      - Will require Img package (for displaying image files) libimg1.2.so
      - Img in turn requires libtiff.so, libjpeg.so, libpng.so etc.
    * BLT used for drag and drop functionality (version 2.4 tested)
    * crc::cksum package 1.0.1 used to uniquely identify images.
    * ImageMagick's identify application used to aid in determining image
      type.  It will be distributed with the application in the bin dir.

3.2 Must save selections between invocations of program on given file
    * Make use of output file, written to disk on program termination.  See
      section 5
    * Look for such a file upon program startup.  If found, load selections
      on to image to be displayed
```

```
4. CONFIGURATION FILE FORMAT
----------------------------

4.1 File Name
    * .jconfig

4.2 Location
    * Will be searched for in current directory, and if not found, in $HOME
    * If not found at either place, a default configuration will be loaded

4.3 Contents
    * All leading blank characters (space/tab/newline) will be discarded
      when processing a line
    * Any line with a '#' character will be treated as a comment, and have
      all characters after it on the same line discarded

    4.3.1 Header Info
    *? Anything necessary??

    4.3.2 Body Info
    * First line will contain 'mode=' followed by a string representing
      the default mode.  Valid values are "crop" or "simple"
    * Second line will contain settings for default window resolution,
      default canvas size, and default bucket size @@@change this
    * Following line will contain a 'snap_threshold=' followed by a decimal
      number in the range of [0...100] that determines the percentage of the
      number of non-background pixels that must exist before we stop snapping
      that line.  So for example if it was set to 10, then when snapping a
      side of length 100, at least 10 pixels must be non-background colour to
      prohibit further snapping of the line.
    * Next lines will contain a 'classifiers=(' followed by a list of
      classification types, each of which has a name then some whitespace
      followed by a colour representing the colour selections of that type
      will be displayed as.  Each list item may span multiple
      lines.  End when closing ')' is reached.  Each classification type
      is a string representing what a portion of the journal may be.  e.x.
           classifiers=(  text blue
                          equation    red
```

```
                   title green
                       graph orange   )
    *? Other configuration attributes??


5. OUTPUT FILE FORMAT
---------------------

5.1 Name
    * <in_file_name_prefix>.jtag
    * <in_file_name_prefix>.jlog

5.2 Location
    * Same directory as input file

5.3 Contents
    * All leading blank characters (space/tab/newline) will be discarded
      when processing a line
    * Any line with a '#' character will be treated as a comment, and have
      all characters after it on the same line discarded

    5.3.1 Header Info
    * First line contains 'img=' followed by the associated image file name
    * Second line contains 'type=' followed by the image file type (tiff/bmp)
    * Third line contains 'resolution=' followed by the image resolution (in
      pixels e.x. 640x480)
    * Fourth line contains 'cksum=' followed by a checksum of the associated
      image file
    * Next lines are optional, and if present they contain configuration data
      used to make these selections, and are of the exact same format as seen
      in 4.3.2.  This data will override any information in any .jconfig files
      currently loaded.

    5.3.2 Body Info (.jtag file) Repeated 0 or more times -- each represents
      a single selection
    * First line contains a separator of '---'
    * Second line contains 'class=' followed by the classification type name
      e.x. text or title
    * Third line should contain 'pos=' followed by a series of 4 whitespace
      separated numbers representing selection co-ords (normalized by image
  resolution)
* Fourth line should contain 'mode=' followed by a string representing the
  mode used ("crop" or "simple")
* Fifth line will contain 'snapped=' followed by a 1 (representing true)
  or a 0 (representing false) depending on whether or not the rectangle
  was snapped into place, or if it was manually sized

5.3.3 Body Info (.jlog file) Repeated 0 or more times -- each represents
  a single selection
* First line contains a separator of '---'
* Second line contains 'pos=' followed by a series of 4 whitespace
  separated numbers representing selection co-ords (normalized by image
  resolution)
* Second line contains 'sel_time=' followed by a decimal number
  representing the time in seconds to make the final bounding box selection
  This includes edit and refinement time.
* Third line should contain class_time=' followed by the total time taken
  to classify the selection (i.e. once bounding box created, the time
  required to drag the selection over a bucket).  Must add time, each time
  a selection is reclassified -- moved from one bucket to another
* Fourth line should contain 'class_attempts=' followed by a positive
  integer representing the total number of classification attempts
  made (i.e. if placed in a bucket, then later changed to another).  Must
  be >= 1
* Fifth line will contain 'resize_attempts=' followed by a positive
  integer indictating the total number of times the rectangle has been
  manually resized
```

```
6. MAJOR DATA STRUCTURES
------------------------

6.1 Classification data representation (in memory)

    6.1.1 Diagram
    data              $data(class1)
    +-----------+     +--------+---------+--------+     +------+
    | class1 o--+---->| colour | num_sels | sel1 o | ... | seln |
    +-----------+     +--------+---------+------|-+     +------+
         .                                      |
         .       $data(class1,sel1)             v
         .      +----+----+----+----+----+---+----+-----+-----+-----+-----+
    +-----------+ | id | x1 | y1 | x2 | y2 | m | sn | sl_t | cl_t | cl_a | r_a |
    | classn    | +----+----+----+----+----+---+----+-----+-----+-----+-----+
    +-----------+

    6.1.2 Description
    * Top level component in the structure is an associative array called 'data'
      containing the entries class1 ... classn, each of which is labelled
      according to its classifier name (ex. body_text or pg_number).
    * Each of the classx entries is also an associative array containing a
      minimum of two elements.  One of which is the colour of the classifier,
      and the second is the number of selections currently existing for that
      classifier.  The remaining elements in the array represent the selections
      made thus far (there should be num_sels of these), and are indexed
      starting at 1.
    * Each of the selections is a list containing exactly 9 ordered
      elements.  These elements are (in the order in which they appear in the
      list): 'id' the unique id given to the rectangle when it is created on
      the canvas, 'x1' the pixel of the left edge of the selection (normalized
      to actual image resolution), 'y1' the pixel of the top edge, 'x2' the
      pixel of the right edge, 'y2' the pixel of the bottom edge, 'mode' the
      selection mode used (crop/simple) to create the rectangle, 'snapped'
      whether the bounding box was snapped, or manually resized to its current
      position, 'sel_time' the total time in seconds to create/refine the
      dimensions of the rectangle, 'class_time' the total time in seconds to
      drag the selection to a classification bucket, 'class_attempts' the
      number of times the selection has been classified/reclassified, and
      'resize_attempts' the total number of times the selection has been
      manually resized.

   6.1.3 Referencing
   * Tcl/Tk does not allow nested or multi-dimensional array structures, so a
     hack is used whereby multi-dimensional arrays are simulated by adding a
     comma between dimension element names i.e.  $data(dim1e,dim2e,dim3e) is
     used.  Not that this is in actual fact a single dimensional array element
     with an element name of "dim1e,dim2e,dim3e".  Therefore care must be
     taken to ensure that there is no whitespace anywhere in the name of the
     elements.
   * To reference the list containing selection data for the 4th selection of
     the body_text classifier, you would use the following:
     $data(body_text,4) .  You can then use lindex on the item returned to
     pick out particular components of the selection ex.  to get the mode
     (item 6) use [lindex $data(body_text,4) 5] since lists start numbering at 0
```

B.2 JTAG Usage Document

```
1. Starting Up
--------------

First ensure that you have installed the JTAG application and its dependent
software locally on your machine (see the top-level README file for
instructions on how to do this).

Once installed, you can start the JTAG application by entering the top-level
directory and issuing:

    ./main.tcl

from the command line.  This will start up and display the GUI, and from there
you can begin loading journal images for tagging.

Alternately, you can specify a journal image to load upon startup by passing
its path and name as a parameter.  If I had a file in my home directory called
```

```
img1.tiff I would load it on startup as follows (after cd'ing to the top-level
JTAG directory):

    ./main.tcl ~/img1.tiff

1.1 Supported Image Formats
---------------------------

One of the main purposes of the JTAG application is to render and display
raster image representations of journal articles.  All of this image rendering
and manipulation is carried out by an external Tcl/Tk library called TkImg.
It transparently supports all of the following formats (although only TIFF,
and JPG have been tested thus far):

  - tiff
  - jpeg
  - gif
  - png
  - bmp
  - xbm
  - xpm
  - partial postscript support (flakey and possibly broken)

Since most journal articles will come in multiple-page postscript or Adobe pdf
format, I have written a script to automatically convert such files, into
individual tiff files (1 page per file) for input into the JTAG application.
See the README file in the scripts directory as well as the file
scripts/pdf_to_tiff.pl for more information on converting to an appropriate
image format.

1.2 Configuration and Customization
-----------------------------------

A large portion of the default behaviour of the JTAG application can be
customized by the user before startup.  This is accomplished through the use
of a config file called .jconfig.  The application searches for this file upon
initial startup, first in the current directory, then secondly in the current
users home directory.  It then proceeds to read and load the config variable
values from the first .jconfig file it finds.  If a valid .jconfig file can
not be found, it loads a series of default values.  To see what these values
```

```
are you can have a look at the beginning of the config.tcl file.  For further
information on each config variable see the sample .jconfig file in the
top-level directory, as well as section 3. below.


2. Basic Controls and Actions
-----------------------------

Upon first loading the JTAG application you will see a menu and various
buttons along the top, 2 vertical columns of buckets containing various class
names & colours, and the main image viewing window.  Interaction with the
application is accomplished through mouse movements, mouse button presses, and
various keyboard combinations.

2.1 Image Display
-----------------

A journal image file can be opened for display by either selecting "Open" from
the File menu using the mouse, or with the keyboard shortcut <Ctrl><o> (i.e.
pressing the Ctrl key while also pressing the o key).  If an image is already
being displayed when a new image is opened, the previous image's selections
(if any) are saved to disk first before the new image is opened.

In order to get a better view of the image one can zoom in and out to magnify
and shrink what is shown.  Zooming in increases the magnification (allowing
one to focus on a portion of the document in detail) and zooming out does the
opposite.  This can be accomplished with the mouse by pressing the "Zoom In"
or "Zoom out" button, or by clicking on the desired zoom magnification from
the Zoom menu.  Alternately, one can incrementally zoom in (out) by pressing
the <+> key (<-> key).

If the image is magnified such that it doesn't entirely fit in the display
window a vertical and/or horizontal scrollbar will appear around the
right/bottom edge of the image window.  Using the mouse, one can slide the
scrollbar to view other regions of the document.

2.2 Creating Selections
-----------------------

A selection is a rectangular region of ink that belongs to a single class
```

(Ex. the ink making up an article title, or equation, or page number etc.)
The main purpose of the JTAG application is to allow one to hand create such selections and view them on screen.

With an image displayed on screen, a selection can be created by hand using the mouse. Simply move the mouse over where you want the selection to begin (one corner of the region), then press and hold down the left mouse button. With the left mouse button held, begin to drag the mouse towards the opposite corner, as you do so you will see a rectangular box being drawn. Once the rectangular box encloses your region, let go of the left mouse button (the black rectangular box should remain). Note that this box is not yet classified (see section 2.6 on how to associate it with a class).

By default the selection you draw out is "snapped" inward on each side until it touches a significant portion of ink (this ensures accurate results when creating training data etc.). Sometimes however, snapping the selection chops off too much, or you would like to manually set the size of the box. To resize a selection once it has initially been drawn out can be accomplished by moving the mouse into the selection, then towards the edge you wish to resize. At that point you will see the mouse cursor change and left clicking and holding you can now move the mouse and that edge will move too. When you are happy with the resize, let go of the left mouse button to finish the change. You can also shrink 2 adjacent edges at the same time by moving the mouse to the corner (where the edges meet) and repeating the procedure outlined above.

If you have resized a selection manually and would like to snap it to ink, this can be accomplished by selecting "Snap Selection" from the Edit menu, then left clicking inside the selection you wish to snap (left-click outside of all selections to cancel this). Alternately, you can place the mouse inside the selection you wish to snap, press <Ctrl><n> on the keyboard, and left click inside of it once the cursor icon changes to a crosshair.

Selections can also be created automatically (if MATLAB is installed, and you have set configuration settings to include training data -- see section 1.2). If there are no selections currently on the page displayed, and you click the "Auto Predict Selections" button with the mouse, the application will attempt to segment the document into regions and classify them using the learner you specify.

2.3 Deleting Selections

```
-----------------------

If you wish to delete a selection, you can either select "Delete Selection"
from the Edit menu, then left-click inside the selection you wish to delete
(left-click outside of all selections to cancel this).  Alternately, you can
press <Ctrl><x> on the keyboard then left-click inside of the selection you
want to delete.

2.4 Splitting Selections
------------------------

If you wish to split a region into two regions (for instance if you let the
application detect regions and it incorrectly grouped multiple ones under a
single region), you can do so by selecting "Split Selection" from the Edit
menu.  Then move the mouse inside the selection you wish to split.  You will
see a vertical (or horizontal) line stretching from edge to edge of the
selection.  Move the line where you would like the split to occur, then
left-click to carry out the split.  If you would like to change the line from
vertical to horizontal (or vice versa), simply right-click with the mouse
before you finish creating the split (i.e. before you left-click).
Alternately you can start the split process by pressing <Ctrl><p> from the
keyboard, then following the procedure above.  To cancel the split, simply
left-click outside of all selections.

2.5 Merging Selections
----------------------

If you have two or more adjacent selections that should be grouped as one
single selection, simply select "Merge Selections" from the Edit menu.  Then
using the mouse, move it inside each of the selections you wish to include in
the merge and left-click inside of it (when you move outside that selection it
should remain highlighted).  Once you have highlighted all the selections you
wish to include in the merge, simply right-click and a new single rectangle
will be created using the outermost co-ordinates of each of the selected
rectangles.  If you have highlighted a selection for inclusion in the merge
and you wish to change that, you can do so by left-clicking inside of it a
second time (now when you move outside of it, it should not remain
highlighted).  To cancel the merge, simply un-highlight each selection and
right-click the mouse.  You can also being the merging process by pressing
<Ctrl><m> with the keyboard, then proceed as explained above.
```

```
2.6 Classifying Selections
--------------------------

Once a selection has been created, it should be classified (as text, or a page
number etc.).  To accomplish this simply move the mouse inside the selection
you wish to classify, then click and hold the left mouse button.  Then with
the left-button held down, simply drag the mouse over the bucket you wish to
classify the selection as.  You will see that a floating button is created
called "selection" while you drag the mouse.  You can let go of the left-mouse
button once the bucket you wish to classify the selection is highlighted.
Once a selection is classified it is diplayed in the same colour as its
bucket.

To change the classification of a selection, simply drag it to the new bucket
following the procedure above.

You can also let the application guess at the classification for all
selections currently on the page by pressing the "Auto Predict Selections"
button (this button is displayed if you have MATLAB installed and you have set
the .jconfig file appropriately).

2.7 Saving Results
------------------

You can save the selections created out to disk at any time either by
selecting "Save" from the File menu, or by pressing <Ctrl><s> from the
keyboard.  This will result in files in the same directory as the image file
created, one with a .jtag extension, and one with a .jlog extension.  The jtag
file is in ASCII format, and stores selection information (co-ordinates,
class, etc.), configuration information, and image information.  The jlog file
is also in ASCII format, but it stores auxiliary data like the time taken to
create a selection, or classify it etc.

2.8 Switching Pages
-------------------

If image files happen to be multiple pages from the same journal, they can
easily be recognized as such and switch between while the application is
running.  Consecutive images from the same journal should have the same base
```

```
filename followed by .aa for the first page, .ab for the second and so on (up
to .zz).  Following the page should be the image format .tif or .jpg for
example.  So a valid page 3 of an image named foo.tif would be foo.ac.tif for
example.

If a file is recognized as being multiple pages, you can switch to the
previous or next page simply be clicking on the "Prev Page" button or "Next
Page" button with the mouse.  Alternately, you can do the same with the
keyboard by pressing the <Page Up> or <Page Down> keys.

When you switch pages, the contents of the first page are saved to disk before
the next page is loaded.


3. Configuration & Customization
--------------------------------

As discussed in section 1.2 above, the JTAG application makes extensive use of
the .jconfig file to set most configurable aspects.  For a discussion of what
each config item does, and how to set its value, see the sample .jconfig file
kept in the top-level directory.

Customization can also be made to the scripts in the matlab directory (for
things like segmentation thresholds etc.), since these are used with the auto-
predict feature to come up with selections and predict classifications.  For
more information on how to setup these variables, see the README file in the
matlab directory.


4. Limitations & Misc. Issues
-----------------------------

Tcl/Tk and the TkImg toolkit use a rather crude representation for images (at
least tiff files) whereby they use 32 bits per pixel and render the *entire*
image at any given time.  As a result, any large (in terms of number of
pixels) image will take up a large amount of memory while the application is
running.  Further compounding the problem occurs whenever you zoom in on the
page repeatedly.  Increasing the magnification simply doubles etc. the number
of pixels that have to be rendered.  At extremely high magnifications this
takes a large amount of cpu-time and largely increases the memory footprint of
```

the application. It is best not to zoom in too much!

When creating selections, you should classify them as soon as you finish creating one (as opposed to creating multiple selections then classifying each). The way the application was designed, it only stores auxiliary information for the most recently created selection. This information is only save internally, after it has been classified. As a result if a new selection is created before classification, the originals auxiliary data may be lost or become garbage.

If a selection is really narrow (in the vertical or horizontal sense) it may be automatically deleted. This feature was put in place to ensure that each rectangle created is larger than a minimum size (so random noise and "dirt" are not created as selections. A trick to creating "thin" selections that should stay as selections involves making them to large to begin with (by including another region), then classify it, then resize it manually so you are only left with the thin piece of ink and whitespace, then snap it to a region.

Bibliography

[1] A. Antonacopoulos, B. Gatos, and D. Karatzas. Icdar 2003 page segmentation competition. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, pages 688–692, August 2003.

[2] Robert D. Cameron. A universal citation database as a catalyst for reform in scholarly communication. *First Monday*, 2(4), 1997.

[3] R. Cattoni, T. Coianiz, S. Messelodi, and C. M. Modena. Geometric layout analysis techniques for document understanding: a review. Technical Report 9703-09, ITC-IRST, Via Sommarive, I-38050 Povo, Trento, Italy, January 1998.

[4] F. Cesarini, M. Gori, S. Marinai, and G. Soda. Structured document segmentation and representation by the modified x-y tree. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, pages 563–566, August 2003.

[5] H. Cheng, C. A. Bouman, and J. P. Allebach. Multiscale document segmentation. In *Proc. of IS&T's 50th Annual Conference: A Celebration of All Imaging*, pages 417–427, Cambridge, Massachusetts, May 1997.

[6] A. Dengel. Automatic visual classification of printed documents. In *Proceedings of the International Workshop on Industrial Applications of Machine Intelligence and Vision (MIV-89)*, Tokyo, April 1989.

[7] A. Dengel. Initial learning of document structure. In *Proceedings of the Second International Conference on Document Analysis and Recognition*, volume 20-22, pages 86–90, October 1993.

[8] A. Dengel and F. Dubiel. Clustering and classification of document structure-a machine learning approach. In *Proceedings of the Third International Conference on Document Analysis and Recognition*, volume 2, pages 587–591, August 1995.

[9] T. G. Dietterich. *Machine Learning for Sequential Data: A Review.* Springer-Verlag, 2002.

[10] S. Imade, S. Tatsuta, and T. Wada. Segmentation and classification for mixed text/image documents using neural network. In *Proceedings of the Second International Conference on Document Analysis and Recognition*, volume 20-22, pages 86–90, October 1993.

[11] K. Kise, A. Sato, , and M. Iwata. Segmentation of page images using the area voronoi diagram. *Computer Vision and Image Understanding*, 70(3):370–382, June 1998.

[12] M. Krishnamoorthy, G. Nagy, S. Seth, and M. Viswanathan. Syntactic segmentation and labelling of digitized pages from technical journals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(7):737–747, July 1993.

[13] Steve Lawrence. Online or invisible? *Nature*, 411(6837):521, 2001.

[14] J. Liang, I. T. Phillips, and R. M. Haralick. Performance evaluation of document layout analysis algorithms on the uw data set. In *Proceedings of SPIE Conference on Document Recognition*, pages 149–160, San Jose, California, 1997.

[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] A. Mccallum, D. Freitag, and F. Pereira. Maximum entropy markov models for information extractions and segmentation. In *Proc. 17th International Conf. on Machine Learning.*, 2000.

[17] P. E. Mitchell and H. Yan. Document page segmentation and layout analysis using soft ordering. In *Proc. of the International Conference on Pattern Recognition (ICPR00)*, volume 1, Barcelona, Spain, September 2000.

[18] G. Nagy and S. Seth. Hierarchical representation of optically scanned documents. In *Proceedings of International Conference on Pattern Recognition*, volume 1, pages 347–349, July 1984.

[19] A. Ng and M. Jordan. On discriminative vs. generative classifiers: A compairson of logistic regression and naive bayes. In *Proc. 15th Neural Information Processing Systems Conf.*, 2001.

[20] Andrew Odlyzko. On the road to electronic publishing. *Euromath Bulletin*, 2(1):49–60, 1996.

[21] Andrew Odlyzko. The economics of electronic journals. *The Journal of Electronic Publishing*, 4(1), 1998.

[22] Andrew Odlyzko. The rapid evolution of scholarly communication. In *Proc. of Picing Electronic Access to Knowledge Conf.*, Ann Arbor, Michigan, USA, March 2000.

[23] Z. Shi and V. Govindaraju. Dynamic local connectivity and its application to page segmentation. In *Proceedings of the 1st ACM workshop on Hardcopy Document Processing*, pages 47–52, Washington, DC, USA, November 2004.

[24] K. Summers. Near-wordless document structure classification. In *Proceedings of the Third International Conference on Document Analysis and Recognition*, pages 426–456, Montreal, Canada, August 1995.

[25] Kwan Y. Wong, Richard G. Casey, and Friedrich M. Wahl. Document analysis system. *IBM Journal of Research and Development*, 26(6):647–656, 1982.

[26] Berrin A. Yanikoglu and Luc Vincent. Pink panther: A complete environment for ground-truthing and behnchmarking document page segmentation. *Pattern Recognition*, 31(9):1191–1204, 1998.