MIT Media Laboratory Technical Report 365
Also appears as MIT thesis for the degree of
Master of Engineering in Electrical Engineering and Computer Science
Supervised by Rosalind W. Picard

An Image Database Browser that Learns From User Interaction

Thomas Minka
Vision and Modeling Group
MIT Media Laboratory
20 Ames Street; Cambridge, MA 02139
tpminka@media.mit.edu

Abstract

Digital libraries of images and video are rapidly growing in size and availability. To avoid the expense and limitations of text, there is considerable interest in navigation by perceptual and other automatically extractable attributes. Unfortunately, the relevance of an attribute for a query is not always obvious. Queries which go beyond explicit color, shape, and positional cues must incorporate multiple features in complex ways. This dissertation uses machine learning to automatically select and combine features to satisfy a query, based on positive and negative examples from the user. The learning algorithm does not just learn during the course of one session: it learns continuously, across sessions. The learner improves its learning ability by dynamically modifying its inductive bias, based on experience over multiple sessions. Experiments demonstrate the ability to assist image classification, segmentation, and annotation (labeling of image regions). The common theme of this work, applied to computer vision, database retrieval, and machine learning, is building in enough flexibility to allow adaptation to changing goals.

This work was supported in part by BT, HP, and IBM.

Contents

Chapter 1

Introduction

1.1 Introduction

Digital libraries of images and video are rapidly growing in size and availability. Though visualization tools are advancing, most database systems still use human-generated text annotations (or worse, flip-books) for basic navigation. Such annotations are costly to produce and are ineffective for many kinds of search. This ineffectiveness can be due to ambiguous wording (such as "a busy scene"), underspecifying the data (a picture, after all, is worth 10^3 words), or a lack of vocabulary (e.g. what is the word for the visual texture of fast moving cars at night?). Because of these inadequacies, searching digital image data is emerging as a major cost factor for multimedia applications.

This section examines why this is so and looks at some alternatives being pursued by researchers and practitioners, using the discussion to motivate the design philosophy behind FourEyes, an extensible, power-assisted, and self-improving database browser and annotator.

1.1.1 Visual indexing is hard

When the data is visual, it is important to be able to navigate directly on the perceptual features of that data. For example, finding a particular texture. It is also useful to be able to automatically extract annotations from the data. But both of these tasks are plagued with difficulties.

Research does not yet have the capability to mimic human perception in general contexts. Computational models which isolate one perceptual component, e.g. texture, have succeeded in benchmarks like the Brodatz [5] collection [43] [24], but have yet to prove themselves on naturally-occurring imagery in which textures may be occluded, in perspective, warped around a surface, at different scales, in different colors and contrast, or have different lighting. Consequently, perceptual models are highly data-dependent; see figure 1.1 for an example. Unfortunately, the nature of the specialization is rarely clear from the definition of the model. (But see [24] for one attempt at this.)

There is no clear separation between perception and semantics in image similarity. The right notion of similarity changes depending on context: the particular image, the particular database, the particular user, or their particular task; see figure 1.2 for an example.

On top of this subjectivity, common database requests require complex interactions between multiple image attributes. For example, a stock photography agency may get a request for "a dry, open shot, at a weird angle," "a threatening forest, with a repeating pattern," or "a group of people reflecting in rippled water" [49]. It is an unsolved problem to derive such annotations automatically, or even to offer database navigation with these kinds of abstractions. Fielding such requests seems to require a close coupling between perceptual modeling, semantic modeling, and domain knowledge, gleaned from sources as diverse as computed attributes, text annotations, usage patterns, and subjective opinions.

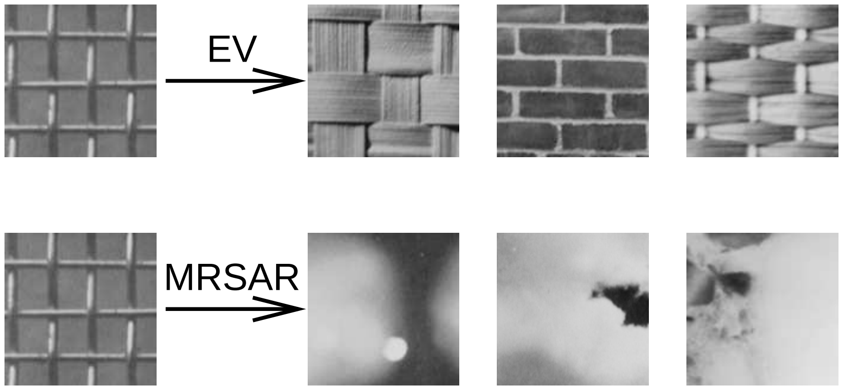


Figure 1.1: Data-dependent performance of texture models. The three patterns on the right were retrieved based on their similarity to the pattern on the left, given the particular model space EV [43] or MRSAR [27]. The multi-resolution simultaneous auto-regressive (MRSAR) model, because it attempts to model fixed-size neighborhoods, misses the high-level structure that the shift-invariant eigenvector (EV) model does not. However, on other images the MRSAR does substantially better than EV [43].

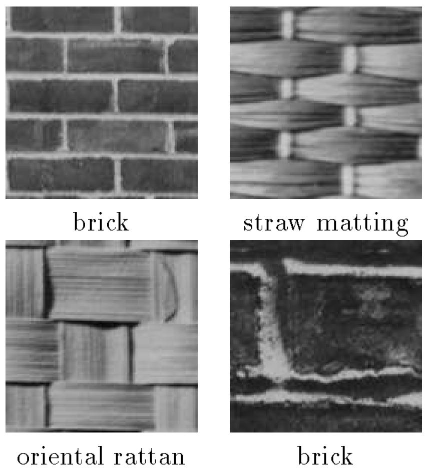


Figure 1.2: Image similarity can be perceptual or semantic. It can also be influenced by culture, visual context, or user preference. Most computational models notice that all of these have similar horizontal/vertical structure. However, one model may decide that the two brick images are most similar, and another model may decide that the top two images are most similar. People, too, will arrive at different decisions, depending on their goals.

1.1.2 Browsing via similarity measures

The most common approach undertaken today is to attach real-valued attributes to the visual data, each attribute being the parameter of some perceptual or semantic model. Along with a distance function such as the Euclidean or the Mahalanobis, these attributes define a measure of image similarity. In the Euclidean case, each image, patch, or sequence maps onto a point in a possibly high-dimensional space, such that nearby points are similar in some way. The user chooses a similarity measure and navigates this space, via a one, two, or three dimensional display which represents distance to a some prototype (one dimension being the most common). The prototype datum may be selected from the database or provided by the user, e.g. with a sketchpad. Some examples of this approach are QBIC [34], SWIM [62], CORE [61], and Photobook [39] (figure 1.3).

Unfortunately, this approach does not scale to arbitrary content, because it relies crucially on the similarity measure and the user's ability to choose it. Designing the measures is extremely hard, since each of the difficulties identified above apply. Furthermore, the user is asked to select a data-dependent model for a context-dependent phenomenon, without knowing the extent of those dependencies. Knowing how to *combine* such models to achieve a complex task is thus completely out of the question. In short, this approach to navigation expects users to project their intent, or information need, onto a limited set of opaque algorithms.

1.1.3 Browsing via relevance feedback

FourEyes attempts to automate the transformation of intent to algorithm, via a rich example-based interaction with the user. Recognizing that user desires cannot be modeled beforehand, it chooses to resolve ambiguity as late as possible. By representing similarity with *groupings*, not high-dimensional spaces, FourEyes provides a lingua franca for diverse information sources, allowing them to readily combine in complex but controllable ways (chapter 2). The selection and combination process is furthermore controlled by learning from positive and negative examples from the user. For example, "these objects are cars, those are telephones," or "these are the kind of images I'm looking for; those aren't." The latter kind of interaction, known as *relevance feedback* [52], is common in document retrieval.

FourEyes puts much more emphasis on learning how to select and combine similarity measures than designing and understanding them. Besides making user interaction easier, this approach eases development of browsers, since newly discovered measures can be added invisibly to the user. It has the potential to completely take on the burden of modeling complex phenomena like "powerful scenes," given a powerful enough learning mechanism. Fortunately, such mechanisms are general-purpose, and so can be constructed from well-known parts. Three algorithms from the machine learning literature, adapted to use sets (FourEyes' grouping representation), are demonstrated in chapter 3.

The FourEyes scenario has a number of advantages over conventional learning scenarios. First, examples tend to be more directed, and hence more useful to the learner, than in a typical pattern recognition problem, where examples are chosen randomly. Second, the same data may be utilized several times, for often similar purposes, allowing the learner to usefully integrate training across multiple users and uses. That is, FourEyes can *self-improve*. This is the idea behind *continuous learning*, which as demonstrated in chapter 4 can significantly improve learning performance, hence FourEyes' usefulness for retrieval and annotation.

1.2 The FourEyes extension to Photobook

The FourEyes interface is intended to allow selection of image regions without requiring the user to carefully outline the region of interest. The paradigm is similar to that of the perceptually organized editing program PerSketch [53]. Under this paradigm, the single object hierarchy of conventional paint programs is traded for multiple, possibly conflicting organizations. The amount of structure imposed by the system is mediated by an example-based interaction with the user. This makes image organization more like a process of discovery, for both the system and user. In PerSketch,

Figure 1.3: Example of a search in Photobook. The user queried for images similar to the one in the upper left, using an eigenpicture space [57]. The ordering is one-dimensional, but displayed in raster scans.

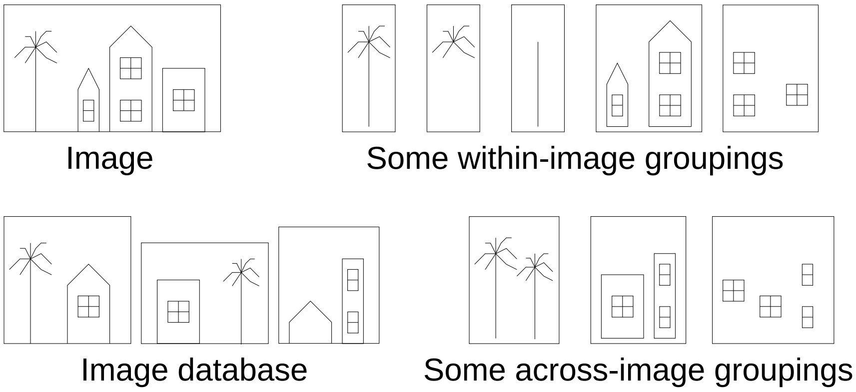


Figure 1.4: A basic task for image database retrieval and annotation tools, addressed by FourEyes: recovering useful within-image or across-image groupings. Note that these generally cannot be captured by a single model, or even a single partition or hierarchy, and the similarity measure required to induce these groupings may be quite complex.

the user can indicate the region of interest in a line drawing by making a gesture similar to its shape. In FourEyes, the user indicates the region of interest in an arbitrary color image by tapping points (mouse-clicking on them) or sweeping a path through the region. The touched pixels become positive examples from which the system immediately attempts to extrapolate, using precomputed within-image and across-image groupings provided by image similarity measures (the computation is described in section 2.3). Negative examples, i.e. pixels which should not be part of the region can be entered in the same fashion, but with a different mouse button.

The components of FourEyes are shown in figure 1.6. FourEyes extrapolates by forming a *compound grouping* from known groupings using either a union (set cover) or arbitrary set operations (decision list or decision tree). This the third stage, the one closest to the user, described in chapter 3. When the user is satisfied with the system's selection, FourEyes updates its bias, e.g. the weights of the groupings or the groupings themselves, as described in chapter 4. These are the backwards "learning" arcs. They cause the groupings that were used to form the selection to be favored the next time a selection is made, so that in many cases only a single tap is needed to reselect a region or part of a region which has been operated on before. Thus FourEyes automatically chooses, combines, or constructs the most appropriate similarity measures for segmentation.

FourEyes allows the attachment of a label to the selected region. This attachment is part of another example-based interaction: annotation of image regions throughout the database. As with segmentation, precomputed groupings of image regions, this time across images instead of only within them, are selected and combined to extrapolate new annotations. Attachment of a label adds a positive example for that label and also a negative example for all other labels. This exclusivity assumption, when correct, greatly reduces the total number of user examples needed to get a satisfactory labeling of a database. The attached labels could later be used to generate context-dependent semantic keys for querying and retrieving database contents.

Other conceivable operations on the selection region, besides labeling it, include paint tools like modifying the color, moving, or export, as well as database tools like retrieving similar regions (a special case of labeling) and pasting them into the image.

The task for FourEyes is not to determine the single "correct" similarity measure, the "correct" grouping of database regions, or the "correct" segmentation of an image but rather

1. (Chapter 2) to contribute to a rich repertoire of reasonable groupings
2. (Chapter 3) to select from and/or combine these groupings to match an example set, with groupings from multiple measures if necessary
3. (Chapter 4) to learn a weighting on groupings, and alter groupings, so that useful ones are recovered from few examples.

Figure 1.5: The FourEyes computer-assisted annotation tool. The user has mouse-clicked some patches of sky in the two right images, and assigned them the label "sky." Within-image groupings allowed FourEyes to grow those labeled patches into larger "sky" regions (indicated by cross-hatching). Across-image groupings allowed FourEyes to also place tentative labels on the two left images. The menu buttons allow the user to control which sets of groupings are available to the learner.

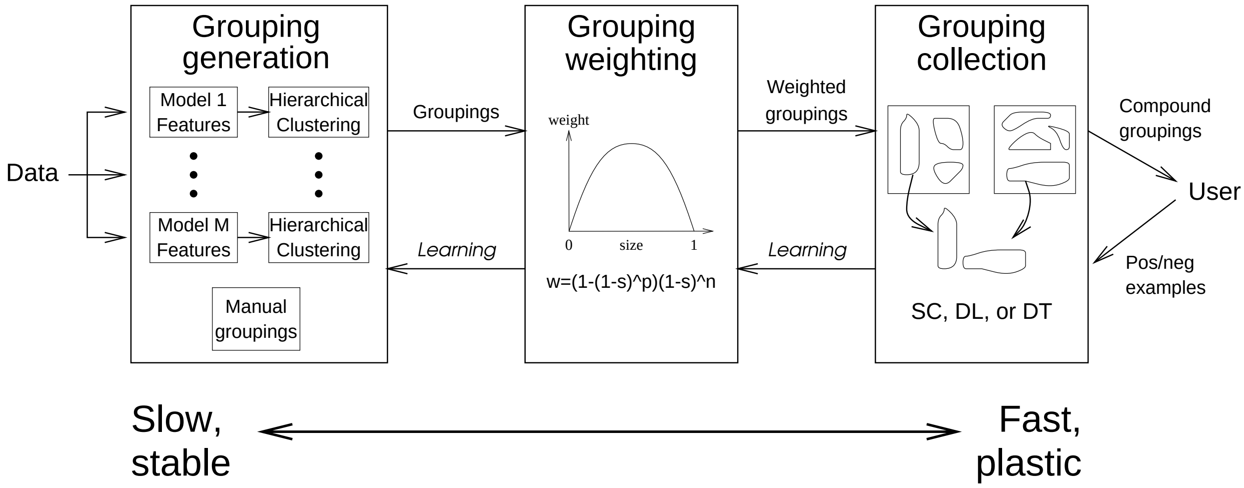


Figure 1.6: Multi-layer learning structure of FourEyes. The arrow at the bottom describes the rate at which the three stages learn. The stages are detailed in chapters 2, 4, and 3, respectively.

An appropriate performance metric is *the number of examples required before the user is satisfied with the response.* This assumes, of course, that not all possible generalizations from the user's examples are equally likely to be valid; otherwise no assistance could be provided. The challenge of FourEyes is to determine what the likelihood function actually is and to submit responses in accordance with it. An additional constraint is that this should all occur in interactive time. Since saving wall-clock time for database access is the objective, a system which processes 4 examples per second and requires 100 examples can be an order of magnitude better than a system which processes 4 examples per *minute* and requires 17 examples.

FourEyes' three-stage method, in meeting these goals, differs from conventional attribute extraction followed by object classification in three crucial ways. First, the feedback arc between the classifier and the attributes is performed by the computer, not the designer. This avoids the usual human cycle of trying lots of classification rules with lots of attributes, and trying to find the one combination that is best for the problem at hand. Second, each stage develops at different times and different rates, with the stages closest to the user changing fastest. This allows the computations to be distributed in time and space, facilitating interactive use and the incorporation of more complex models. Third, training is accumulated across sessions with the user, so that the system improves over time and can solve similar problems better, i.e. learn faster, the next time.

1.3 Applications

1.3.1 Example-based User Interfaces

FourEyes can be viewed as a user interface agent [8]. Agents automate repetitive tasks and fill a niche missed by macros and scripting, by learning from examples. In FourEyes, the repetitive task is grouping pixels into objects and objects into classes. Conventional programs support this but with considerable user effort. Grouping cannot be done with scripts because it is too complex, even with all of the image processing tools in the world, mostly because we do not even know how we perceive. Macros, such as the grouping commands provided by a paint program, are on the other hand too brittle, since they only capture one level of abstraction and one point of view.

While text manipulation and indexing are not solved, it is generally sought that images be as usable as text. For example, in a text editor the user can switch levels of abstraction (character, word, sentence, paragraph) with a keystroke. One can search for a word in a database, including misspellings and synonyms. However, there are numerous obstacles to achieving this kind of efficiency with multimedia. First, there are far more levels of abstraction at which the data can be perceived.

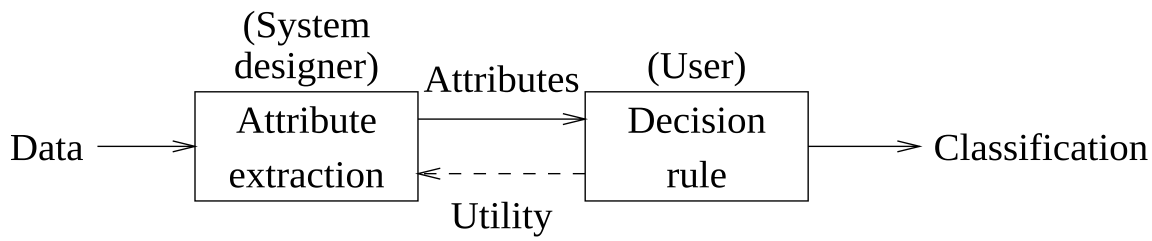


Figure 1.7: The pattern recognition approach and how FourEyes extends it, with a utility feedback arc.

This makes the notion of "misspelling," "synonym," or "same" much harder to quantify. (See [40] for an example.) Second, automatic modeling of human perception is unsolved, requiring some degree of user guidance. (If we raise our standards for text manipulation, for example to the domain of meaning, we can find the same difficulties.)

FourEyes presents an approach to seemless interaction between multiple levels of abstraction and multiple media, based on a purely example-based metaphor. Positive and negative examples automatically choose combinations of the known groupings induced by similarity measures or given by previous users. Since the grouping representation is a lingua franca, groupings can come from diverse information sources (even from the interaction itself: section 4.4). The example-based learning, supported by continuous self-improvement, allows rapid and natural specification of the level of abstraction.

1.3.2 Attribute Selection for Pattern Recognition

Solutions to pattern recognition problems can generally be described by a two-stage process (figure 1.7, see also [56], section 1.3). First the raw input data is projected into a suitable attribute space, then decision rules are applied to the attribute values. For example, a rule might be "anything which is red is an apple," where color is the representation of the input. A more realistic application is distinguishing various letters of the alphabet, based on the locations of edges, using the same paradigm.

This model is useful because given suitable attributes, there are well-developed methods for finding optimal decision rules. However, determination of the attributes remains. For well-posed problems, it is possible, and common, for the designer to experiment with several different set of attributes, pick those which work best on a test set, and then fix the first stage to use these. When the system is put into practice and applied to specific problems, only the decision stage is adapted automatically.

Unfortunately, there are many conditions under which this approach is insufficient. First, the available test data may be insufficient to determine the correct attributes. Second, the designer may never find a single set of attributes which gives acceptable performance over all problems of interest. Third, the problem may be ill-posed, i.e. the set of acceptable solutions is dynamic and dependent on context, where little is known about the context into which the system will be placed. Under these conditions, it is important to have a system which can adapt its attributes in addition to its decision rule.

While the decision rule should be readily adapted from problem to problem, the attributes should adapt more slowly. The reason is that the attributes constitute an important *bias* (chapter 3) of the system: what it assumes about the world even before it sees the problem. When chosen properly, bias allows a learner to quickly and accurately find a decision rule. For example, we usually do not use the date at which we see an object in judging whether or not it is an apple. (This may change if we are in the fruit-of-the-month club.) Only after witnessing a number of tasks and drawing analogies between them can a system have reason to alter its bias. Therefore the attribute extraction stage should only change in response to the evolving needs of the decision stage. This can be accomplished by adding the missing feedback signal from the decision stage to the attribute extraction stage.

FourEyes demonstrates a way to add such a feedback signal, via its hierarchy improvement algorithm (section 4.4). This is one implementation of *continuous learning*, which tries to push the complexity of the decision rule back into the attributes.

1.3.3 User-Adaptive Image Segmentation

FourEyes is part of the next step in a progression of image segmentation tools. This progression, moving from optimization to blackboards to user-guidance, is marked by an increasing utilization of domain knowledge or context and retaining of the ambiguity in the image.

A segmentation approach which capstones much of the existing research is region competition [63], which attempts to optimize a general-purpose fitness function over region interiors and region boundaries. The algorithm strives for the "simplest" segmentation, i.e. the one with smoothest interiors and smoothest boundaries. This intractable optimization problem must always be solved by an approximate method; in [63] this is region growing. The algorithm thus scatters seed regions randomly across the image and then iteratively grows them in a way that incrementally produces the greatest increase in the fitness function. The optimization approach to segmentation is powerful because various kinds of image attributes can be inserted into the optimization function. In principle, any segmentation can be realized by the proper choice of optimization function, aside from the limitations of the approximation method, e.g. region growing.

However, general-purpose image segmentation remains unsolved. Why? Even though the optimization model provides the capability of encoding the user's desires, it does not make it straightforward to do so. This is the same problem with most database browsers. Since segmentation is ill-posed, there can be no fixed algorithm which will always satisfy. Instead, a true solution must provide an infinite set of algorithms as well as a mapping from tasks to algorithms. The quality of the solution is dependent on the quality of this mapping. Unfortunately, optimization-based algorithms like region competition bury their parameters in split/merge/smoothing thresholds, which together conspire to alter the resulting segmentation. Even the designer of such an algorithm would be hard pressed to transform a segmentation task into a suitable set of parameters. For example, [2] applies a sophisticated optimization algorithm simply to find the right parameters of the Phoenix segmentation algorithm for images taken under different conditions. Phoenix uses histogram splitting to do top-down segmentation, where each split is chosen to optimize a fitness function. Interestingly, the most important parameters of the fitness function were found to be the size of the histogram smoothing window and the min/max threshold for histogram peak determination. These seem quite far removed from the goals of the segmentation application. Consequently, as the application goals change, it is hard to know how to adapt the segmentation parameters.

A concise statement of this complaint is that segmentation fitness functions have a weak connection to domain knowledge. Domain knowledge is the information that we have at hand to express the kinds of segmentations we want, e.g. that we are interested in segmenting people, and that people usually have four limbs and a head.

Another problem with optimization algorithms is that they strive to destroy the ambiguity inherent in the image. For automated tasks like robot tanks driving at 60 mph, this disambiguation is necessary for decision-making. The power of information tools, though, is the user's ability to view the same data in different ways. Thus it is important in a database application that the machine decompose an image in multiple ways, ordered according to their probable utility to a user.

An approach which addresses both of these complaints is context-based vision, as practiced by CONDOR [54]. CONDOR works by updating a blackboard of segmentation/classification hypotheses, using explicit rules for creating and removing hypotheses. The rules specify which of a variety of specialized algorithms, or agents, should be consulted to produce a hypothesis. This makes CONDOR robust, since it will only apply an agent in the proper context and is therefore not contingent on any one of them. Note that the agents themselves may be arbitrarily complex, e.g. one might be region competition. The rules also specify which conditions exclude a hypothesis from further consideration. This provides a direct representation of domain knowledge, e.g. the sky should not be below the ground. The specialized agents can also be used to encode domain knowledge. This kind of control is not available in the optimization-based techniques. Upon termination, the blackboard contains all segmentations suggested by the agents which are consistent with the domain knowledge. This provides the desired retainment of ambiguity.

A program employing a similar approach, but geared toward interactive applications, is PerSketch [53], a perceptually-organized graphical editor. Here agents also propose segmentations, but instead of using rules to eliminate hypotheses, the hypotheses are all made available to the user. Instead of navigating menus or otherwise being forced to cope with imposed structure, the user indicates the desired segmentation just by giving gestural examples. Hence the image is manipulated as objects, not pixels, but also in its full ambiguity. The domain knowledge of PerSketch is provided by the agents.

FourEyes continues this trend in the direction of an *adaptive* segmentation tool, which strives to improve its own segmentation performance. FourEyes, like PerSketch, employs agents to build hypotheses which are mediated by the user, but differs in that the agents *learn* from the experience. FourEyes can also *synthesize* new segmentations at the user's request, by combining hypotheses suggested by the agents, or through manual outlining. Such synthetic hypotheses then become part of the tool, as though suggested by an agent. In this way, the user does not simply observe the blackboard, as in CONDOR, but may also contribute to it.

1.3.4 Bias Selection for Machine Learning

Learning and induction is not black magic. Learners work by having a bias which supplies the information missing from the training data (chapter 3). Hence the success of a learner hinges on its bias. Unfortunately, not all learning algorithms have an obvious bias, nor is it always clear which bias is appropriate. For example, the parameters of neural network and genetic learning constitute a bias, but have a weak connection with the domain knowledge so that it is seldom clear how to change them to match a particular problem domain. Thus learners are like similarity measures for indexing, image segmentation algorithms, and many other solutions to ill-posed problems. A robust learner should make its bias as flexible as possible so that it can be changed dynamically, as the objective is determined (as in the way FourEyes indexes with multiple similarity measures, by learning from the user). This is the idea behind the *continuous learning* component of FourEyes, described in chapter 4, which allows it to integrate training across multiple uses to determine which bias to use.

1.3.5 Malleable Solutions for Ill-Posed Problems

A common theme of this thesis—from database retrieval to computer vision to pattern recognition to machine learning—is that solutions to ill-posed problems must be malleable, i.e. easy to adapt to changing requirements. Anything else is solving a well-posed problem, like adding two and two. This requirement parallels that of software engineering: well-written programs are easy to adapt. In both cases, the correct solution is not a fixed program, but rather a set of powerful tools and a language for combining them.

In order for adaptation to be "easy" this language must be straightforward, predictable, and sufficient. The easier the adaptation, the more useable the solution. For example, the learning algorithm used in FourEyes was designed so that its parameters (the groupings) have a direct correspondence with the user's desires. Thus it was easy to automatically change these parameters to track the user (section 4.4).

Chapter 2

Retrieval and Annotation with Multiple Similarity Measures

Retrieval and annotation is complicated by the abundance of attributes which must be used to fulfill a query and the irrelevance of available attributes. These are the attribute combination and attribute selection problems, respectively. This chapter describes how a dual view of retrieval, using examples from document retrieval and image retrieval, is helpful in attacking these problems. The dual view comes from the fact that the items to retrieve can be represented in two complementary ways: as points arranged in a metric space or as members of fuzzy sets (figure 2).

Relevance feedback is also crucial for attacking these problems. FourEyes incorporates relevance feedback for the purpose of annotation, the labeling of image regions. In the presence of relevance feedback, the dual view is instrumental in allowing attribute selection and combination to be handled by rule-induction methods used in machine learning.

2.1 A dual view of retrieval

2.1.1 Document retrieval

In the generalized retrieval problem discussed in [3], the user wants to find a document containing certain terms. A term might be a single word like "rocket" or multiple words like "rocket fuel" or "National Aeronautics and Space Administration." The terms to use for indexing are decided upon when the database is populated. To fulfill a request, the retrieval system gives each document a *relevance score,* and presents them to the user in order of relevance.

One approach for achieving this is to make a table which maps documents into term vectors. Vector component i is intended to encode the strength of the connection between the document and term i, i.e. the relevance of the document to a user searching for term i. This might be 1 if the document has the term, 0 otherwise, or perhaps the normalized frequency of occurrence of the term in the document. These vectors can be interpreted as rows of a document-term connection matrix (figure 2.2; see also [52], section 2-5). The system then assigns each document a relevance score according to the similarity of its terms with the query and the connection strength of those terms to the document as a whole. If the query is also represented as a term vector, each component quantifying the user's desire for that term, then a simple dot product per document (corresponding to a matrix-vector multiply for the whole database) can accomplish this task. Alternatively, the query could be specified as a Boolean formula on terms, in which case a more complicated similarity relation would be used. But in either case, documents are represented as points in a metric space. For example, we may cluster documents to accelerate search.

Another approach is to make a table which maps terms into documents. This corresponds to storing the document-term connection matrix by columns instead of rows (figure 2.3). Each column is interpreted as a fuzzy set of documents, where the connection strengths indicate degree

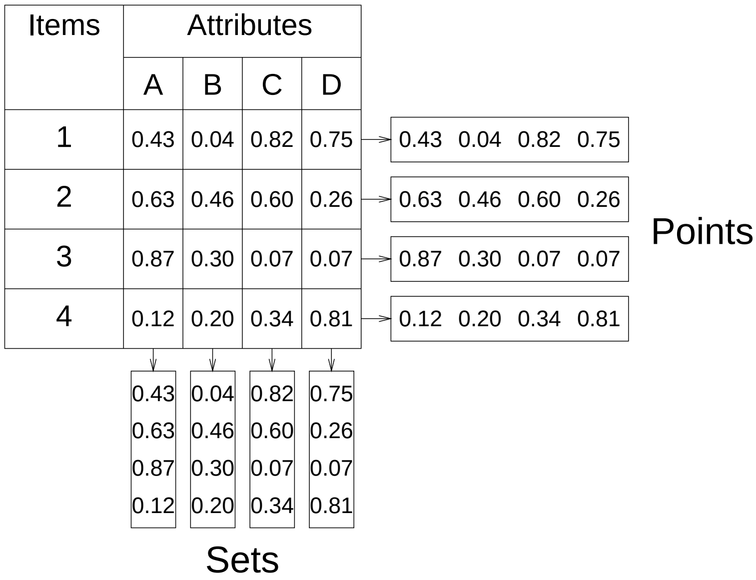


Figure 2.1: A dual view of database items: points in a metric space or members of fuzzy sets.

Document	Term		
	rocket	fuel	NASA
1	1	1	0
2	0	1	1
3	0	0	1
4	1	0	1

Query

0.5	0.5	0

Scores
1
0.5
0
0.5

Figure 2.2: Retrieval with a document-term matrix

Term	Documents
rocket	{1,4}
fuel	{1,2}
NASA	{2,3,4}

Query	Output
rocket & fuel	{1} (2 and 4 have half membership)

Figure 2.3: Retrieval with document sets

of membership. The output of the system, the relevance scores, can also be interpreted as a fuzzy set of documents. Boolean queries now correspond to fuzzy set operations on columns and do not require scanning every document's entire term vector, as in the first approach. Instead of clustering documents, this representation allows us to cluster terms, e.g. for automatic thesaurus generation [52].

The membership values in each term column define a similarity measure over documents. For example, according to "rocket", document 1 is similar to 4 but not to 2. By placing particular terms in a query, the user is selecting which of these measures to use and how they should be combined to get relevance scores. In this sense, the document retrieval interaction is identical to the Photobook interaction (section 1.1.2).

2.1.2 Image retrieval

The basic problem in computer vision is extracting useful information from visual data. Unlike document retrieval, desired attributes are seldom raw components of the data but rather functions of local regions or an entire image. Unfortunately, the kind of information that can be robustly extracted is a tiny subset of the information that we would like to have. More often we compute attributes which correlate with what we want, over a restricted domain of interest.

In the computer vision approach to retrieval, the user wants to find an image with certain attributes. These attributes are computed from pixel values and may correspond to local color, texture, or matched filter response, the shape or arrangement of regions defined by those attributes, or still further iterations of analysis (though this is rare). To fulfill a request, the system gives each image a relevance score, and presents them to the user in order of relevance. This is the approach used by Photobook.

The difference between this problem and the document retrieval problem is the attributes. While document attributes are scalar, image attributes are typically vectors. For example, color may be described by a histogram of pixel values. The analogous image-attribute connection matrix would have vectors in its cells. If the query is also represented as a row of vectors, then the row-based scanning used above will work, though vectors will have to be compared instead of scalars. This comparison can be done with vector product, Euclidean/Mahalanobis/Bayes distance, etc. Unfortunately, this requires touching the entire database, as in the document retrieval example.

A more efficient column-based approach can be used here, too, after some modification. The fuzzy set interpretation of attributes requires scalar membership values, not vectors. This transformation can be achieved by vector quantization, where retrieval attributes are now the quantization bins of the original image attributes. Let a *grouping* be one of these new scalar attributes, viewed as a fuzzy set. (We use this word instead of "group" to distinguish from group theory.) Since vector quantization is not unique without an optimization metric, many different quantizations and thus many different groupings are possible from a single image attribute. Which of these quantizations is "best" is unknown until retrieval time, although there are some heuristic preferences. One solution is to compute several quantizations of the same attribute and store all of the resulting groupings. Another solution is to periodically re-quantize on the basis of usage patterns.

2.1.3 Vector quantization in FourEyes

FourEyes allows either a row-based or column-based approach, by using hierarchical clustering to quantize visual attributes. For example, figure 2.4 shows how some textures are clustered by FourEyes on the basis of the MRSAR texture model [27]. Like the document-term matrix, the output of hierarchical clustering can be interpreted in two ways:

1. A new metric over the images, where distance between two images corresponds to the height of the lowest common ancestor ([10], section 6.10.4).

2. A collection of sets, where each node of the tree is the set of images under that node.

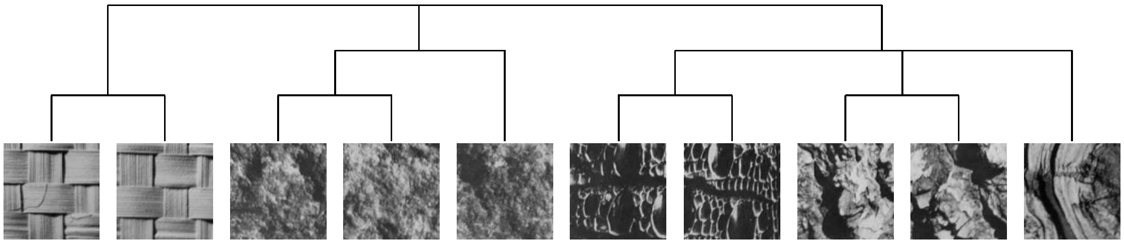


Figure 2.4: Hierarchical clustering with MRSAR

Photobook, with its raw similarity measures, uses only the first representation. However, the second representation has the benefit of speeding database access, raising recall rates, automatically introducing a "visual vocabulary," and providing a basis for the fusion of diverse similarity measures.

Hierarchical clustering is obtained in FourEyes by the shared-neighbor algorithm in [17]. Define x to be a k-nearest neighbor of y when x is one of the k nearest data points to y in the vector space. Define $P(k, c)$ as the partition induced by the equivalence relation:

> $x = y$ when x and y are k-nearest neighbors of each other and have c of their k nearest neighbors in common.

A hierarchical clustering is obtained by the sequence $P(k = 0..\infty, c)$. This always defines a hierarchy because (1) all points are distinct when $k = 0$, (2) all points merge for sufficiently large k, and (3) if $x = y$ in $P(k, c)$ then $x = y$ in $P(k + 1, c)$.

This algorithm is a *single-link* method [16] because two clusters merge in the hierarchy as soon as two of their members get close enough. This is opposed to *complete-link* methods which merge two clusters only when *all* of their members are close enough. Complete-link clustering is similar to minimizing mean squared error, such as in the popular k-means algorithm [16]. Hence it would be expected to give the best preservation of the distances in the original vector space. However, there is no reason to believe that the desired perceptual groupings occur as equal-sized spheres in attribute space. Single-link clustering is preferred for this application because it can capture a broader range of structure. For example, the shared-neighbor algorithm can separate concentric rings, parallel surfaces, and spheres of differing sizes. Single-link clustering also tends to group areas of similar density, as opposed to complete-link methods which will make seemingly arbitrary cuts through regions of constant density. These advantages of single-link clustering, which seem most appropriate for perceptual problems, have been demonstrated in the literature; see e.g. [16], [10] (section 6.10.2). If this heuristic is in doubt, one can always add additional quantizations, as mentioned earlier.

The metric interpretation of the resulting hierarchy yields a distance function similar to that provided by the original vector space. However, many of the unnecessary distinctions between images have been removed. A consequence is that retrieval using a hierarchy is much faster than row-based scanning through the whole database. Furthermore, the single-link algorithm can extract nonlinear structure from the linear vector space, improving performance in some cases (figure 2.5). A drawback is that the single-link method has so much flexibility in defining clusters that it can be easily distracted by noise. We will see this phenomenon again in a different context in section 3.3.1, as a result of a weak *inductive bias*.

The set interpretation of the hierarchy is also useful. If we view each grouping (hierarchy node) as a term, then basic set operations can be used to answer queries concerning terms, as in the column-based approach to document retrieval. The groupings thus form FourEyes' vocabulary about images. This is similar to the process of automatic thesaurus construction for document databases [52]. However, it is unclear how this new "visual vocabulary" could be used for manually formulating queries. It seems that a more automatic approach to formulation is needed. We will return to this issue later.

(a)

(b)

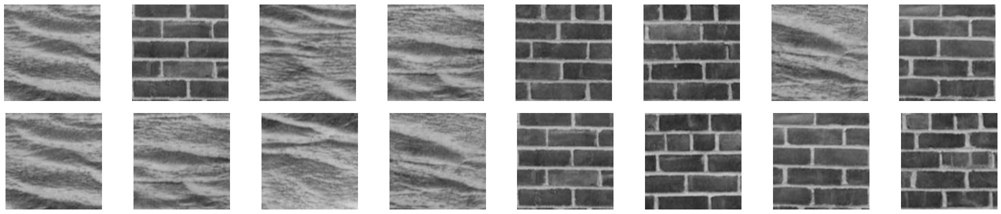


Figure 2.5: While a top performer overall, the MRSAR places bricks next to water in feature space. These are retrieval results in Photobook using the left image as the query; the images are ordered by the MRSAR similarity measure. (a) Using vector distance, we get a mixing of the two textures. (b) After clustering, the texture groups separate. The system is also allowed to arrange images in the same group however it pleases.

Document	Measure 1		Measure 2		Fusion		
	Relevance	Rank	Relevance	Rank	Min	Max	Avg
A	0.99	1	0.7	4	1	4	2.5
B	0.7	2	0.85	2	2	2	2
C	0.3	3	0.9	1	1	3	2
D	0.2	4	0.8	3	3	4	3.5

Figure 2.6: Data fusion using ranks

2.1.4 Combining multiple similarity measures

It is unreasonable to expect, as Photobook does, that database browsing should only involve a single attribute at a time. In the visual domain, many low-level attributes of the picture conspire to generate the attributes we call "grass," "buildings," or "a threatening scene." In order to navigate at the user's desired level of abstraction, many attributes need to be considered simultaneously. Furthermore, no existing visual attribute mimics human perception over general imagery. Multiple similarity measures, which individually specialize on an image domain, must cooperate to simply provide good color-based or texture-based navigation. In short, human desires rarely map onto a single extractable visual attribute.

The basic issue in combining multiple measures (also known as data fusion [12]) is the comparison of incommensurates. When two measures have a linear tradeoff, they can simply be combined into one composite vector. In fact, this constitutes one definition of a feature vector: a collection of dimensions with known exchange rates. Unfortunately, color and texture, or even two texture attributes, do not generally exhibit a linear exchange. Mere concatenation can just as easily degrade performance as enhance it.

An important step in combining measures is to subject each to a nonlinear transformation which makes them more commensurate. One such transformation, in terms of document retrieval, is the rank of a document when ordered by relevance score. For example, in figure 2.6 four documents have been assigned relevance scores by two different measures. Rather than combine scores (which differ in range, mean, and variance), we can combine ranks (minimum rank, average rank, etc.) to get new measures along which the documents may be ordered. In document retrieval, this has been shown to give substantially better results than using either measure alone. A similar, but more adaptive, method has achieved synergy between texture models [24]. In that work, the ranks are combined with unequal weight, determined from a measure of the periodicity of the texture. The weights can be determined a-priori because of careful selection of the models to be combined.

Hierarchical clustering allows FourEyes to take a different approach. Here quantization is the transformation which makes measures commensurate. The metric interpretation of the hierarchy, like ranks, gives a neutral notion of distance. A benefit is that emergent clusters, as in figure 2.5, are less likely to be shattered, since a hierarchy metric, unlike ranks, does not discriminate between points in the same cluster. The set interpretation of the hierarchy allows us to hide the fact that we are doing fusion at all. We can consider all sets in all hierarchies together, as though they were part of one big amorphous collection of sets, and then form normal queries using this expanded vocabulary. This is the approach used in FourEyes. However, it places even more burden on query formulation, to be discussed next.

Another benefit of using hierarchies is that fusion can include measures which are not defined by a vector space. For example, it can include clusterings directly input by users or derived from usage patterns (chapter 4). It is important to note that some, perhaps most, of the attributes that people want to use for navigation cannot be computed from the image data alone. This is because either they involve unsolved problems like modeling perception, or they are subjective (based solely on the current user), or they are collaborative (based on the judgements of other users). One technique for collecting such non-computable attributes is to allow users of the database to attach their own annotations. General-purpose retrieval systems which attempt fusion must therefore be prepared for a large number of irrelevant attributes which may not necessarily be represented from the outset by a vector space.

2.1.5 The query formulation problem

A remaining problem with all of these solutions is that they require users to translate their information need into a query. Specifically, it means that they must know the relevance of terms or visual attributes to their need. For example, a user who wants to know about NASA's research on rocketry does not just want to find documents containing the exact word "rocket," but probably also "propulsion" and "spacecraft," with different weights. There are several methods explored in document retrieval to alleviate this burden. First, a thesaurus can be used to add new terms to the query (query expansion). Second, terms which rarely occur in documents can be automatically increased in importance [18]. Third, the user can indicate which documents are relevant and which are not (relevance feedback: [52], section 7-4), allowing term weights to be automatically modified. Such methods are conspicuously rare in visual retrieval, for no apparent reason. Perhaps the belief remains that computed visual attributes like the MRSAR or EV are sufficiently intuitive.

Another problem with these retrieval solutions is that they only describe documents and images with the literal terms or pixel values that they contain. The document-term matrix should be generalized to a document-attribute matrix, where attributes might be "about NASA rocketry," "something Joe liked," or "well-written, with lots of pictures." The same applies to images; users may want "dry, open shots" or "powerful lines" [49]. While arbitrary attributes like these can significantly improve perceived performance on some queries and for some users, they may just as likely be irrelevant. In fact, this addition significantly worsens query formulation, since (1) the meaning of attributes becomes more obscure and subjective and (2) a virtually infinite number of attributes are possible.

Relevance feedback has the potential to alleviate this problem as well, since both attributes and their weights can be deduced from examples. The initial query formulation by the user can be skipped altogether in favor of examples, e.g. "give me more documents like these" (not "this"). The advantage is that the user need not give a formula for what they want, so they need not know which terms have been extracted from the documents or what other attributes are available. Instead, the system can select among the often-irrelevant, possibly subjective attributes and use only the best ones for assigning relevance scores. This realizes the performance advantage of specialized attributes without burdening the user. When documents have very many attributes attached to them, attribute selection itself becomes a kind of information retrieval problem. Therefore it is natural for the computer to also assist the user in that task.

2.2 The dual view applied to annotation

Semi-automatic annotation of image regions ("patches") is closely related to retrieving images. However, annotation merges seamlessly with relevance feedback: the tool constantly suggests and the user corrects. Interaction has the pure example-based flavor recommended above. Hence annotation was chosen as the vehicle for this research, though the results also apply to retrieval.

Given a vector space and a set of labeled patches, several heuristics can be applied to propagate the label to new patches. For example, the label can spread to all patches inside of the convex hull defined by the labeled patches. Alternatively, the label can spread to all patches closer to a labeled patch than any other labeled patch (or perhaps closer than the *farthest* labeled patch). Various other classification algorithms apply. FourEyes could use any of these, using the metric interpretation of its groupings (hierarchy nodes).

Under the set interpretation of the groupings, a different set of heuristics emerge. The set of patches which receive a label is determined by set operations on the groupings. One possible rule is to take the union of all sets containing a labeled patch. Another takes the minimal union needed to include all labeled patches. These methods are directly analogous to the vector space methods; the difference is only conceptual.

FourEyes focuses on the latter approach. It employs methods for rule-induction used in machine learning, but based on sets. Note that this automatically allows annotation with multiple similarity measures, using the hierarchy-combination method. The transformation of the annotation problem into a rule-induction problem is detailed in the next chapter.

2.3 Constructing groupings from images

Groupings are the building blocks from which labelings are constructed. A grouping is a set of one or more image regions ("patches") which are associated in some way. (This is the Webster definition of grouping: a set of objects combined in a group.) The elements of a grouping may not necessarily come from the same image. This representation is useful since it admits different kinds of associations without adding complexity. For example, one set may represent "regions containing between 15% and 25% blue pixels" while another may represent "regions containing waterfalls" while yet another may represent "regions which were browsed very often this week." It also allows specific associations between patches to be weighted independently, since each set may have its own weight. This is important because, for example, lettering may be best grouped by shape whereas sky may be best grouped by brightness and location in the image. The notion of grouping, including the generation methods to be described, apply to all kinds of data, not just images.

FourEyes computes within-image groupings for each similarity measure, e.g. color or texture, in three steps as illustrated in figure 2.7. First, a dense feature image is computed from the source image. Each point in the feature image is a feature vector (e.g. a color histogram) computed from a neighborhood around the corresponding point in the source. Normally the source image is a color still, but for images in a sequence the source image could be optical flow data. The feature image should ideally be at the same resolution as the source but may be coarser depending on computational constraints. Second, a coarse feature image is computed from the first one by computing a neighborhood average and covariance. This is the first step of segmentation, which performs local smoothing and obtains feature covariances for use of Mahalanobis similarity in the next step. Third, the coarse feature image is hierarchically clustered via the shared neighbor algorithm to produce within-image groupings. The typical image size in this work is 512×512, with a coarse feature image of size 16×16. This size reduction significantly reduces the number of possible groupings, but still leaves 2^{256} to choose from (all subsets of 256 elements; the patches in a grouping need not be connected in the image). The result at this stage is a hierarchical set of image regions for each image, for each measure. These may be used directly for segmentation, as well as for the next step: computation of across-image groupings.

Across-image groupings are computed from a hierarchical clustering of an attribute measured over the within-image groupings. The within-image groupings need not correspond to the same attribute used for across-image grouping; they may have come from optical flow or even manual outlining. The within-image groupings simply provide information about which image regions should be usefully taken as a whole. For example, if a within-image grouping utilizes face detection to produce segments containing faces, the across-image grouping can use a face classifier.

The advantage of incorporating within-image relationships for across-image annotation is described in [51]. For color-based annotation of image regions, that work demonstrated a clear qual-

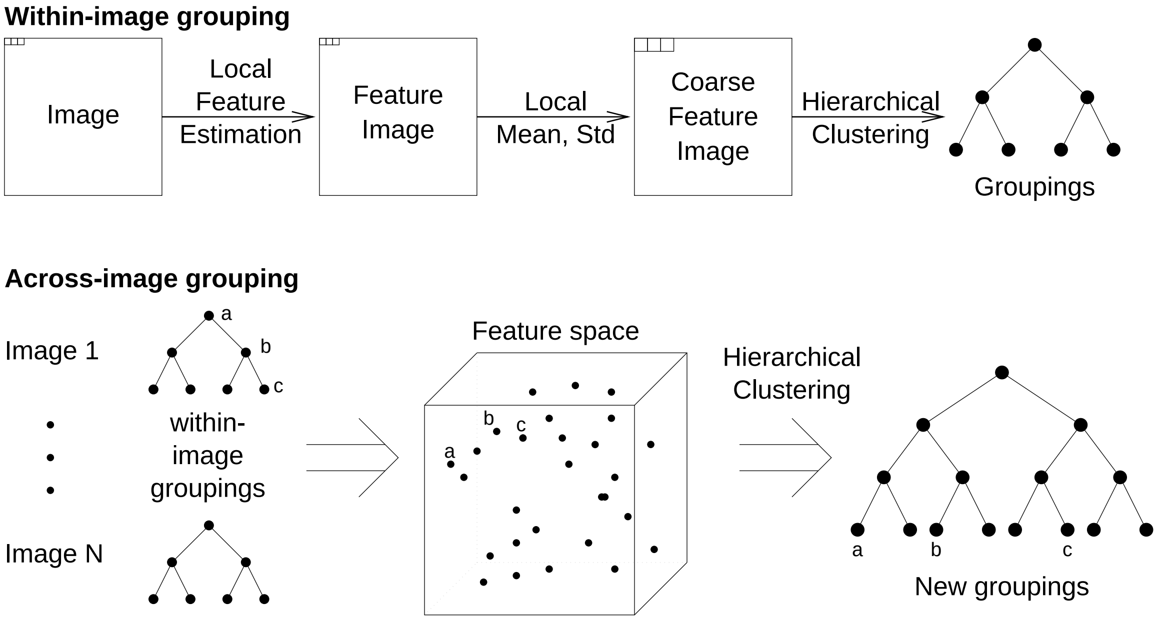


Figure 2.7: Computing within-image and across-image groupings. In image 1, grouping a contains b, which contains c; e.g. they might be house, door of the house, window on the door. When projected into feature space, they are considered individually, and look different. The resulting clustering says that a looks more like b than c.

ity improvement when scene-adaptive class thresholds, based on preserving the continuity of the within-image class-likelihood histogram, were used instead of fixed, universally optimized thresholds. FourEyes approximates this behavior by forming its across-image groupings from within-image groupings. Moreover, the shared neighbor clustering algorithm used by FourEyes behaves similarly to the histogram splitting used in [51], so the within-image groupings generated by both methods similarly preserve class-likelihood continuity.

All of these within-image and across-image groupings are computed off-line, before the user begins interaction with the system. This slow-learning stage, separate from the fast-learning stage with the user, is important for practical implementation in a real image database retrieval system. For example, when clustering happens off-line, it can perform extensive cross-validation, noise sensitivity, and stability checks, possibly utilizing several different algorithms. This level of evaluation is currently infeasible for on-line use, but the off-line use allows state-of-the art results from pattern recognition to be incorporated, improving the overall system performance. Feature extraction routines, since they run off-line, can likewise use larger neighborhoods, more accurate estimators, and have more diversity. New feature extraction or clustering methods can be developed independently of work on the other components. Such engineering concerns are important to those who would construct real systems.

A disadvantage of precomputing groupings is that these must be recomputed when a novel image is added to the data set. In FourEyes, this is done with a full reclustering for all of the similarity measures, though a faster or gradual incremental algorithm could be used. Since queries occur orders of magnitude more often than additions to the database, the interactive speedup can offset the recomputation cost.

Since the later stages of the system only see groupings, not feature values, it is not necessary for continuous similarity spaces to be used. For example, this is advantageous for incorporating subjective associations among content [41]. For humans, it is often easier to specify groupings of image regions than to attach meaningful and consistent attributes to them.

Chapter 3

Learning image annotations

Chapter 2 showed how different similarity measures can be placed into a common vocabulary of groupings. This chapter describes how user input, in the form of positive and negative examples, can be transformed into a query or labeling rule in terms of groupings. By viewing this problem as concept induction in the AI sense, we can apply standard machine learning algorithms, but in terms of set-based attributes. Three adapted algorithms: set covering [28] [14], decision list [47], and decision tree [15] [45] [4], are described here.

This chapter drives home the point that learning performance hinges on the bias of the learner. Hence a robust solution is to make the choice as flexible as possible and then adapt it at runtime. This extension is explored in chapter 4.

	Training set								Test set							
Attribute 1	1	1	0	1	0	0	1	0	0	0	0	1	1	1	0	1
2	1	1	1	0	0	0	0	1	0	0	1	0	1	1	1	0
3	1	0	1	0	1	0	1	0	0	1	0	1	0	1	0	1
4	1	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1
Category	A	A	A	A	B	B	B	B	?	?	?	?	?	?	?	?

Figure 3.1: Pattern classification task

Take a moment to consider the classification problem in figure 3.1 (taken from [37]). Can you determine the categories of the test patterns? How do you even know when you have the right answer? A common assumption is that the "simplest" rule is best; in this case, that is the exclusive-or of attributes 3 and 4, making all of the test patterns category A. However, there are many other rules consistent with the training set and if we assume that the training patterns are noisy, i.e. some of the attribute values have been randomly flipped, then even more rules are possible. In this case, the only explanation for choosing XOR is a bias toward simpler rules. The next section formalizes this learning process.

3.1 Machine learning formalism

Let *instances* be the members of the universe of interest. Each instance is defined by the *attributes* which describe it. Given a set of instances in a class (positive examples) and instances not in the class (negative examples), our task is to determine the *target concept*, or full set of instances in the class. A *concept* is just a set of instances, which may or may not be the target concept. This problem is ill-posed, since there is not a unique answer. Any set of instances which is *consistent*, i.e. contains all of the positive examples and none of the negative examples is a *candidate* for the target concept (figure 3.2). If the set of instances is infinite (or nearly infinite), then so will be the set of candidate concepts, and no amount of examples (or practical amount of examples) will pin down the target

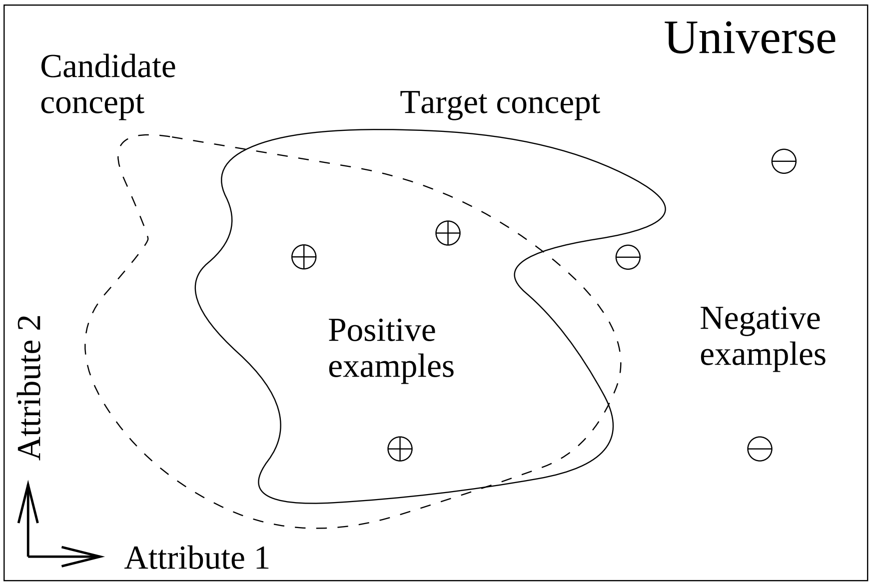


Figure 3.2: Illustration of machine learning terms.

concept on the basis of consistency alone. The possibility of noise in the examples or attributes may prevent any concepts from being excluded.

Fortunately, in learning problems of interest it is not the case that all candidate concepts are equally likely. This is demonstrated by the fact that humans regularly solve ostensibly ill-posed learning problems. Such problems only become ill-posed when taken out of context and presented in the abstract, as in the introduction to this chapter. Unfortunately, this is exactly what happens when we present our learning problems to a computer. So the issue is not the ill-posedness; it is the lack of context. Context is the provider of *inductive bias,* which is the reason we choose one candidate concept over another [33].

The classical algorithm for concept learning is this: choose the candidate concept most favored by the bias. This methodology is analogous to Bayesian inference [25], where the likelihood function serves to isolate candidates and the prior provides the bias. Two important axes along which biases vary are *strength* (or *precision*) and *accuracy* (or *correctness*) [58]. Strong biases focus the learner on a relatively small number of concepts. Assigning equal probability to all candidate concepts is the weakest possible bias. (Hence the weakness of a bias is analogous to the entropy of the prior probability distribution. Intuitively speaking, they both express the amount of a priori uncertainty about the target concept.) Strong biases are accurate when the target concept is in their focus; strong and accurate biases are the best biases for a learner to have. Assigning nonzero probability to only one concept is the strongest possible bias, with a slim chance of being accurate. When a bias assigns nonzero probability to all concepts, it is always accurate. (Hence this definition of accuracy is weak.)

Learning algorithms have a variety of means by which they can specify their bias. These include (1) restricting the space of concepts and (2) employing a particular search algorithm for finding the most favored candidate. These are "declarative" and "procedural" biases, respectively. A common bias is to only allow concepts expressible in some *concept grammar*, e.g. disjunctions of attributes (the set of instances with attribute A or attribute B). This is hazardous, because e.g. if we exclude XOR (or equivalent) from our grammar we would never get the "right" answer to the problem which introduced this chapter. Another approach is to reduce the number of attributes, thereby reducing the number of instances and sets of instances (concepts). This is a restriction on *concept vocabulary.* In the example problem, this would work well, since the first two attributes are irrelevant, but in general this is equally hazardous since we do not know a priori which attributes to remove. The objective in designing learning algorithms is to provide a useful bias without giving up generality.

3.2 Three classical algorithms

In the context of annotation, we will assume the absence of noise and that there are a finite number of distinguishable candidate concepts. In this situation a learner can always isolate the target concept, given enough examples, regardless of its bias. Many so-called "nonparametric," "unbiased," or "robust" methods exploit this situation by requiring little knowledge of the problem but many training examples. However, in many applications, including this one, examples are costly. Therefore proper bias selection is still needed, where performance is measured not only in terms of the accuracy of the concepts but also in the number of examples required to achieve the concept, i.e. learning *efficiency*.

All three algorithms use a concept grammar bias based on Occam's razor. They attempt to choose the candidate concept which has the shortest representation in a particular grammar. The choice of grammar is not guided by a belief that the teacher actually has the grammar in mind; the grammar only serves as a convenient means of expressing a preference ordering over a large set of concepts. This methodology is analogous to expressing Bayesian inference in terms of minimum description length [25], by inventing a suitable notion of "description." The concept vocabulary available to the algorithms is the same: groupings derived from hierarchical clustering (chapter 2).

With a suitable vocabulary (described below), each of the three grammars is sufficient to encode all concepts. Thus the number of candidate concepts, for any set of examples, is the same for all three grammars. Traditional analysis methods are unable to distinguish these grammars, since their VC dimension [14] is infinite and they are all accurate [58]. However, it will be shown that they have significant performance differences. How can this be explained?

One observation is that the grammars differ in how many ways they can express any particular concept (a concept is defined as a set of instances, even though the same set may be constructed or expressed in more than one way). To capture this difference, let a *rule* be a particular expression of a concept, e.g. a predicate in first-order logic. There may be many different rules which denote the same concept. There is some confusion in the literature about the difference between a rule and a concept, but here we see the importance of the distinction. Using this terminology, we can say that each algorithm differs in how many rules it has for a particular concept. This has a direct effect on their precision, as we shall see, because the rules denoting the same concept may have different sizes. The effect on accuracy is more problematic. Since the target concept is always a candidate, some authors would say that all three biases are accurate [58], however this is not very informative. A better definition would include the preference assigned to the target concept, e.g. the average size of the rules which can represent it. This is left as a research exercise.

3.2.1 Set covering

The set covering algorithm represents concepts via set union. Given an assortment C of sets to choose from, the algorithm's job is to find the smallest union of sets which covers (includes) all of the positive examples and none of the negative examples. This problem is NP-complete [11], but there are a variety of greedy methods which get close to the optimum covering [28] [14]. For example, algorithm SC:

1. Choose the set c from C which covers the most positive examples and no negative examples. Resolve ties with a coin toss.
2. Remove all positive examples covered by c from consideration.
3. If positive examples remain, go back to step one.

This is a greedy algorithm since at each iteration we take the action which gets us closest to the goal. The resulting concept is the union of the chosen covers. This algorithm will succeed as long as there is a way to make progress at every iteration, i.e. there is always a set which covers at least one positive example and no negative examples. To guarantee this, we can make sure that C contains all possible sets containing one instance (singleton sets).

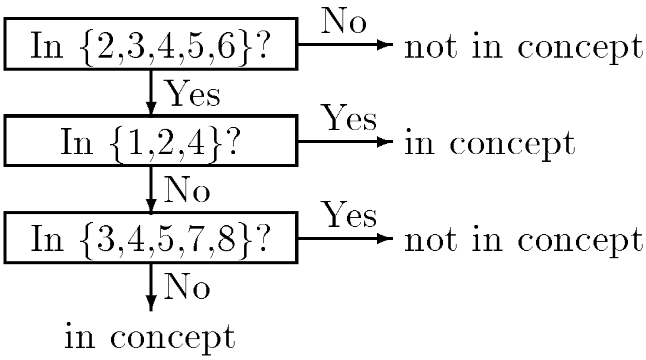


Figure 3.3: A decision list to test if an instance is contained in the concept {2,4,6}. We do this by funnelling the instance in the direction determined by a decision. Each decision checks for membership in a particular set, which either contains no positive examples or no negative examples at that point in the list. Consequently, each decision finalizes the classification of some set of instances; this can happen in four possible ways (see table below), three of which are used here.

We have only guaranteed that the algorithm produces a candidate concept, not the target concept. Of the candidates, it chooses the one which is smallest in terms of the number of sets to be merged. Hence this is its inductive bias. Whether this bias is appropriate as usual depends on the learning task. The fact that the algorithm is an approximation also adds bias, although in this case we know the approximation error is not great.

3.2.2 Decision list

The requirement of SC that at each step we cover no negative examples seems somewhat harsh and asymmetric. The decision list [47] algorithm instead chooses sets which either (1) include positive but no negative examples or (2) include negative but no positive examples. The complements of sets from C may also be used, even though they are not explicitly contained in C. The resulting rule is in the form of a list of decisions; the outcome of each decision determines the classification (in or out) of some set of instances (see figure 3.3). Each decision queries for membership in a particular set in C. The outcome classifies the instance or passes the instance on to the next decision.

A decision list is completely described by a list of pairs (c, t): the first element of the pair indicates which set c in C is used for a membership test and the second element, t, indicates whether instances inside vs. outside c are to be classified and how they are to be classified (in or out of the concept). Thus there are four possibilities for t, corresponding to the rows in this table:

t	Is the instance in c?	Therefore, is it in the class?
1	Yes	Yes
2	Yes	No
3	No	Yes
4	No	No

If the outcome doesn't match the first column for t, then the instance is passed on to the next decision in the list. Cases three and four cover the complementation of c.

A decision list is constructed incrementally, via the greedy algorithm DL:

1. If the set of examples is all positive or all negative, then say "in concept" or "not in concept," respectively.

2. Choose the (c, t) which assigns a correct classification to the most examples and add it to the decision list. Resolve ties with a coin toss.

3. Remove all examples classified by (c, t) from consideration, and go back to step one.

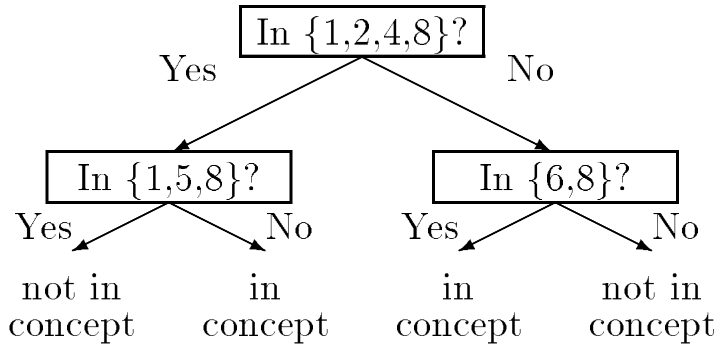


Figure 3.4: A decision tree to test if an instance is contained in the concept {2,4,6}. Like a decision list, we do this by funnelling the instance in the direction determined by a decision. Each decision again checks for membership in a particular set, except it may be any set which divides the instances into two parts.

As before, we require that all singleton sets be included in C to guarantee a solution.

SC is the special case of DL where only $t = 1$ is used. Therefore DL can always produce a rule found by SC, as well as others. We know that SC can already output any target concept, so these extra rules must be redundant. Since both learners use a bias toward shorter rules, the redundancy causes DL to have a weaker bias, because for any fixed rule size there are more candidate concepts expressible by a decision list than a set cover. This difference will be demonstrated by the experiments in section 3.3.

3.2.3 Decision tree

The decision list algorithm requires that some examples are removed from consideration at each step of the loop. This means that every decision must be sufficient to determine the classification of some set of instances. A further relaxation of constraints allows the learner to choose a decision which simply makes the best division of the examples into two parts, each of which may require further division. This yields a greedy algorithm which builds decision trees [15] [45] [4] (see figure 3.4). Unlike the methods in [45] and [4], we're going to use no stopping or pruning criterion to halt tree growth, because in an interactive setting the learner is expected to classify all training points correctly. Another difference is that DT uses set membership to make decisions, not raw attribute values.

The criterion for selecting a decision is a softer version of that for the decision list. Each set c divides the examples (E) into those which are members ($I_c = E \cap c$) and those which are not ($O_c = E - c$). Intuitively, we want the decision which minimizes the impurity of I_c and O_c (ideally, all positive examples are in I_c and all negative examples are in O_c, or vice versa). Consider the examples as data points of value one or zero, for positive and negative, respectively. Two useful definitions of impurity are then the sample variance and the sample entropy under an independence assumption (these are the splitting rules suggested in [4]):

- p_I = the fraction of I_c which are positive examples
- p_O = the fraction of O_c which are positive examples
- $V(x) = 4x(1-x)$
- sample variance $= V(p_I) * |I_c| + V(p_O) * |O_c|$
- $H(x) = x log_2(x) + (1-x) log_2(1-x)$
- sample entropy $= H(p_I) * |I_c| + H(p_O) * |O_c|$
- impurity = sample variance *or* entropy

For each example e, let $V_c(e)$ be the value, or the negative of the impurity, of the set I_c or O_c in which e is contained. The value V_c of a decision is the average value per example. For example, if e is in I_c (i.e. covered by c), and I_c contains 3 positive and 4 negative examples, then the negative sample variance of I_c is $(-4)(\frac{3}{7})(\frac{4}{7}) = -\frac{48}{49}$; this is e's contribution to the value of c.

The DT algorithm:

1. If the set of examples is all positive or all negative, then return a leaf for "in concept" or "not in concept," respectively.
2. Choose the set c from C which has the largest value V_c defined above. Resolve ties with a coin toss. Create a tree node T_c for c.
3. Recurse separately on the examples in the sets I_c and O_c. This will yield the two subtrees of T_c.

No difference was found in the following experiments between the use of variance or entropy, so the simpler variance rule was chosen (as in [4]).

DL is the special case of DT where impurity is either zero, meaning pure, or one, meaning some mixture of positive and negative. Therefore DT is the most general of the three learning schemes. However, its performance on actual problems still depends on how closely its bias, which is weaker than DL or SC, suits the domain.

3.3 Experiments

These three algorithms were tested independently in a simulated annotation environment on three different data sets. In each case, the learner (either the SC, DL, or DT algorithm) was given a number of hierarchies of groupings (see chapter 2) from which to draw sets. (The hierarchies correspond to C in the above algorithms.) The learner began with all instances (images or patches) unclassified. The learner was trained by repeatedly querying it for the classes of all instances, tallying the errors, choosing an erroneously labeled instance at random, and then informing the learner of the proper class of that instance. (This is similar to the way training would occur with a user, but quite different from conventional pattern recognition, where classification is done by comparing to heavily pretrained prototypes or feature distributions, without on-line feedback. Here the learner only gets examples which will be relevant, because it has erred on them, instead of getting an arbitrary off-line selection of examples.)

At each step, an instance which was unclassified by the learner was scored as one error; an instance which was misclassified by the learner was scored as two errors, to make blind guessing disadvantageous. Thus the formula for error count is (False negatives) + 2 * (False positives). The progression of error counts forms a learning curve by which the algorithms can be judged.

The learner remembered all examples it had received and reconstructed the candidate concept from scratch after every new example. The learner also assumed that classes were disjoint, i.e. a positive example for one class was a negative example for all other classes. These two facts imply that the learner will always converge to zero error in at most N steps, where N is the number of instances. The minimum number of examples required is equal to the number of classes, since a learner must see at least one member of a class in order to speculate what is in it.

In these experiments, the disjoint-class assumption was always valid, and it significantly reduced the number of examples required to learn the classifications. Several modifications could allow non-disjoint classes:

1. The assumption could be abandoned from the outset. However, an unnecessarily large number of examples will be required when classes really are disjoint.
2. The assumption could be abandoned only when a contradiction is detected.
3. When the classes have meaningful names like "leaves" and "foliage," domain knowledge can provide clues of overlap.

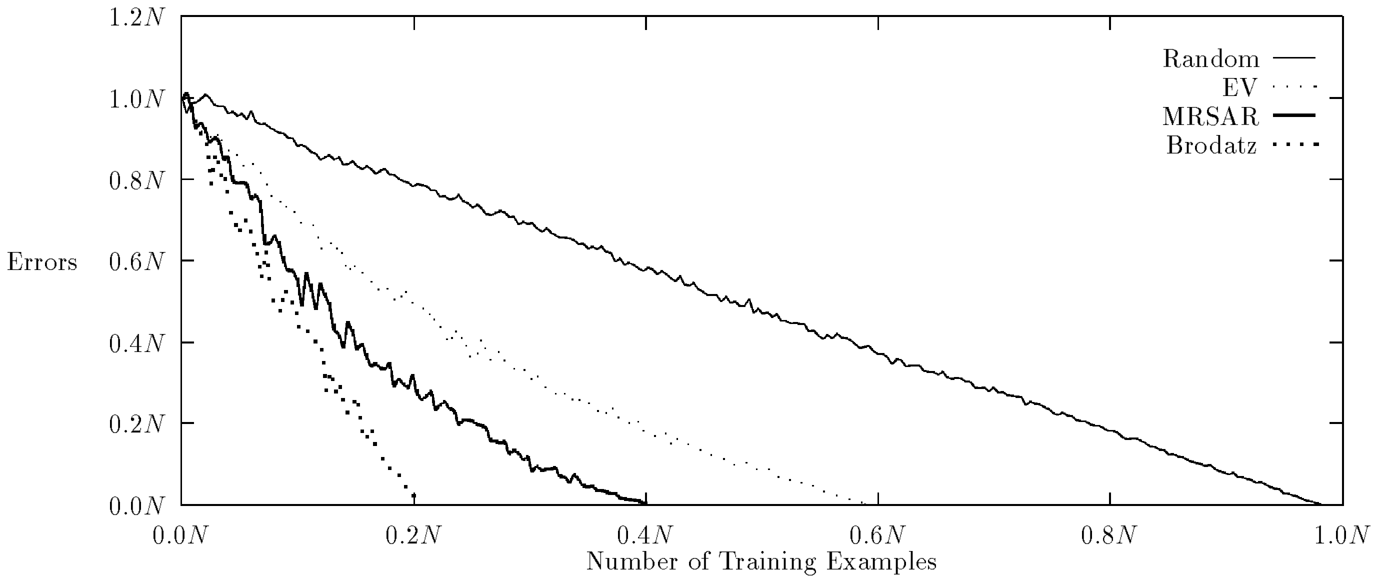


Figure 3.5: Learning performance of SC for different sets of groupings. The faster the curve drops, the better the performance.

There is some randomness both in the teacher's selection of examples and in the learner's selection of concept, both described above, but the shape and final point of the curves varies little (never more than 2%) across multiple runs. This stability is largely due to the on-line example selection. Hence the results for only one run is shown, rather than an average which would suppress the variation in error throughout a run.

3.3.1 Brodatz collection

The first data set consisted of images digitized from the monochrome Brodatz [5] texture album. Each of the 112 textures in the album was equally divided into 9 128×128 non-overlapping images. The desired classification corresponded to two disjoint groups each containing 56 of the 112 original texture classes. The members of each class were chosen at random; consequently the learning difficulty is very similar to that of learning the 112 textures individually. The fact that 504 of the $N = 1008$ images belong to the same class means that there is a large number of candidate concepts for most example sets, making the learner's bias important.

Four experiments with the SC algorithm, using a single hierarchy each time, are shown in figure 3.5. The first experiment provides a baseline; the learner only had available two randomly-generated hierarchies. These had a total of 700 groupings (excluding singleton groupings). Given this feeble knowledge, SC required $0.98N$ examples to reach zero error (call this the *learning time*). The second experiment used a hierarchy generated by clustering the images by EV features [42] (the hierarchy contained 427 groupings). This knowledge reduced the learning time to $0.59N$ examples. The third experiment used a hierarchy generated by clustering the images with MRSAR (the hierarchy contained 255 groupings). The MRSAR has demonstrated excellent matching performance on this database in earlier experiments [43], so we would expect learning to proceed even faster. This was indeed the case; the learning time was $0.4N$ examples. The fourth experiment used a hierarchy ("Brodatz") which explicitly contained the 112 textures as groupings (148 groupings total). The learner quickly exploited this knowledge, obtaining a time of $0.2N$ examples. To actually get zero error with the minimum number of examples (two), the learner would have to have available the correct two groupings and have a bias which favors these groupings over the others. This case is explored in chapter 4.

SC is an efficient learning algorithm. When using all 1530 groupings in these four hierarchies

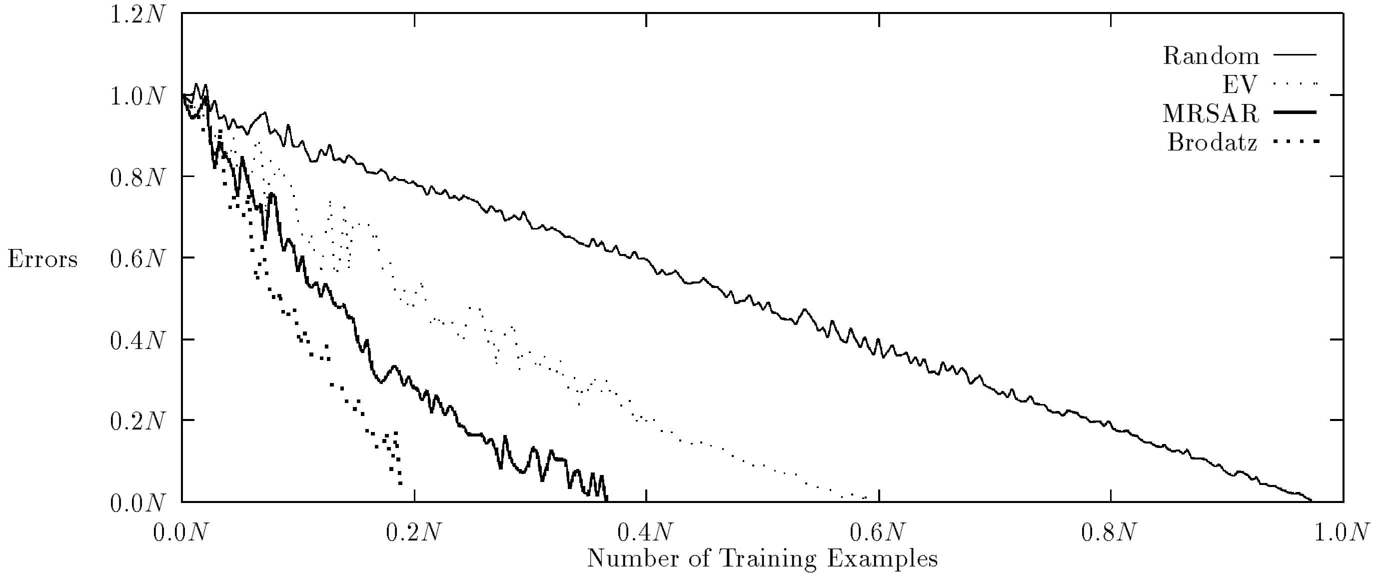


Figure 3.6: Learning performance of DL for different sets of groupings.

simultaneously, the learner processed over 20 examples per CPU second; it has been benchmarked with up to 120,000 groupings on 6400 patches, where it could still process 5 examples per CPU second on an HP 735/99 workstation. The time complexity for constructing a candidate concept from an example set is within a constant times the product of the number of examples, the number of trees, and the *height* of each tree; it is not directly dependent on the total number of groupings or the total number of instances. The time complexity for retrieving all of the instances in a candidate concept is linear in the size of that concept. All tree heights in these experiments were under 20.

The same four experiments were repeated with the DL and DT algorithms (figures 3.6 and 3.7). Note the significant variation in performance between examples (the "noise" in the curves). This is because the learner resolves ties with a coin toss; there are many more ties using DL and DT than with SC, because of the weaker bias. In an interactive application, this noise might be quite disturbing, requiring special modifications. For example, the learner could be repeatedly queried for a candidate concept, leaving the example set the same, and the different candidates could be averaged. Alternatively, the learner could be modified to directly output its estimate of the probability of each instance's membership in the target concept. Either of these may result in a rule which is no longer in the assumed concept grammar. The DT algorithm also had a large proportion of false positives, making its performance sensitive to the false positive penalty factor (2, in this case).

The DL and DT algorithms took one to two orders of magnitude longer than SC to run. The difference continues throughout all of the experiments in this thesis. This deficiency could be alleviated somewhat by using incremental or lazy concept modification instead of rebuilding from scratch on every new example, however SC can also benefit from such enhancements. The suitability of these algorithms for an interactive annotation system depends on the granularity: with a 16×16 tessellation of images the speed difference may not be noticeable, but at the pixel level SC is the only choice. Much complexity is added by the need to handle set complementation, which produces new sets that escape the hierarchical pattern. The time complexity for both of these algorithms multiplies SC by the number of decisions in the list or tree, which can be at worst twice the number of examples. The reason is that all decisions are reevaluated for every position in the list or tree. Since the number of examples here can reach 1000, it is clear how an order of magnitude difference can arise.

These four hierarchies do not exhibit synergy on this task; combining them yields performance slightly worse than the best contributor. Performance is worse because the other hierarchies add "grouping noise" to confuse the algorithm with spurious candidate rules. Figure 3.8 plots the effect

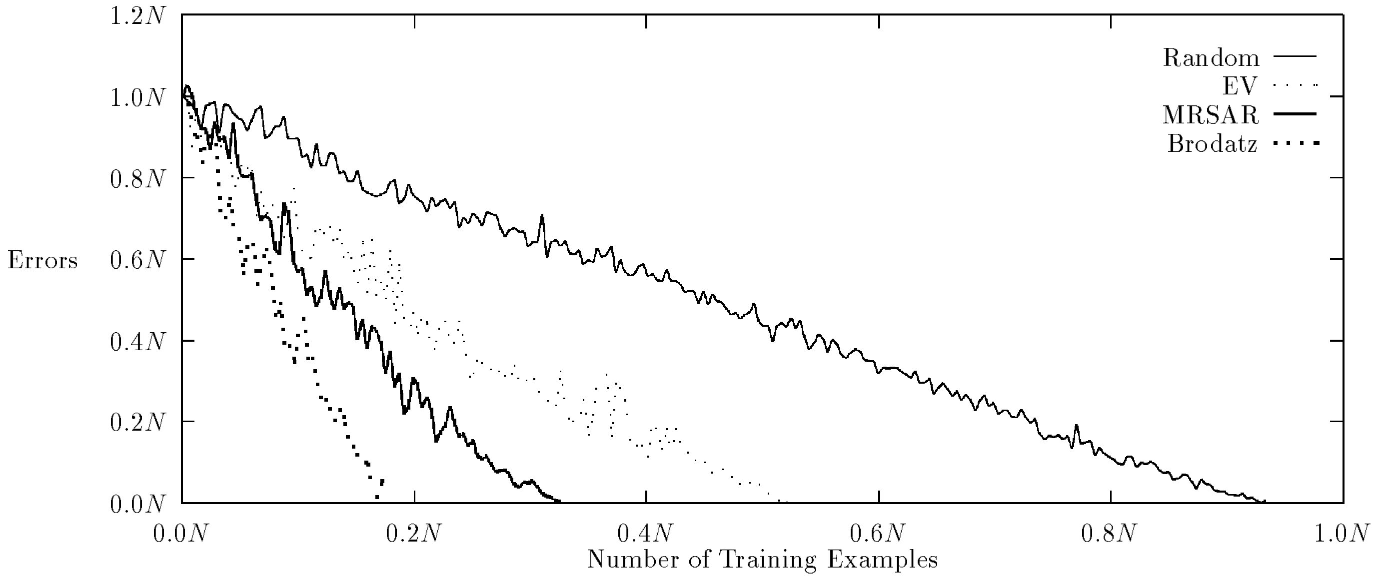


Figure 3.7: Learning performance of DT for different sets of groupings.

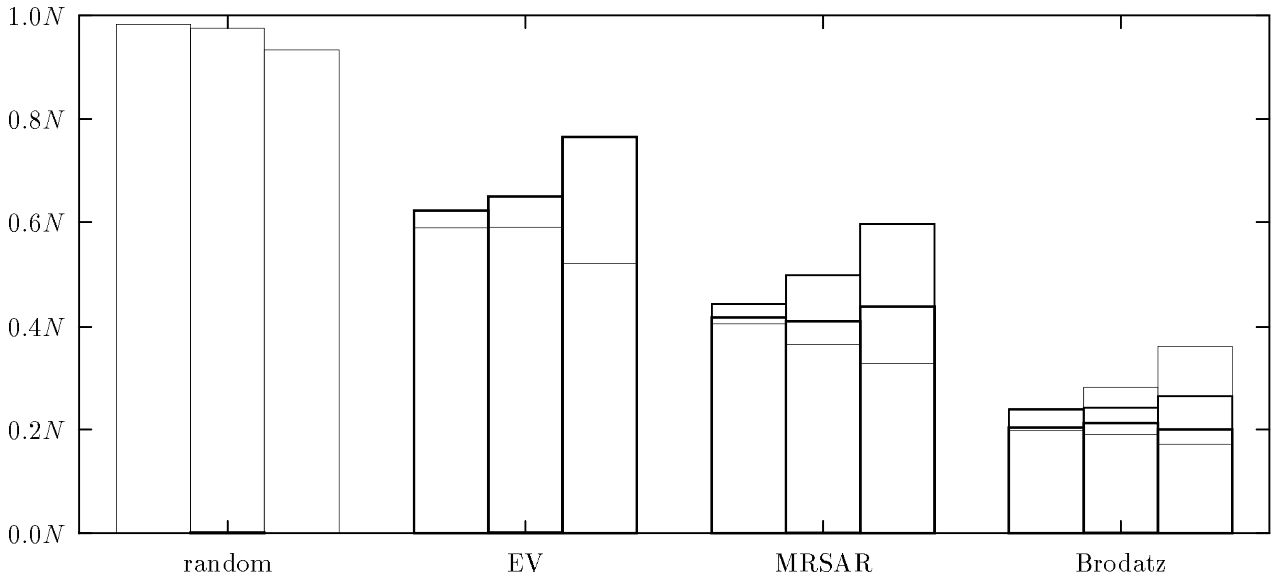


Figure 3.8: Learning time for varying amounts of spurious groupings. Learning time is the number of examples until zero error, not wall-clock time (which differs dramatically between these algorithms). Each set of three bars is SC, DL, and DT, respectively. Note that SC, which is worst with no grouping noise, gradually becomes best. (It is also fastest by wall-clock time.)

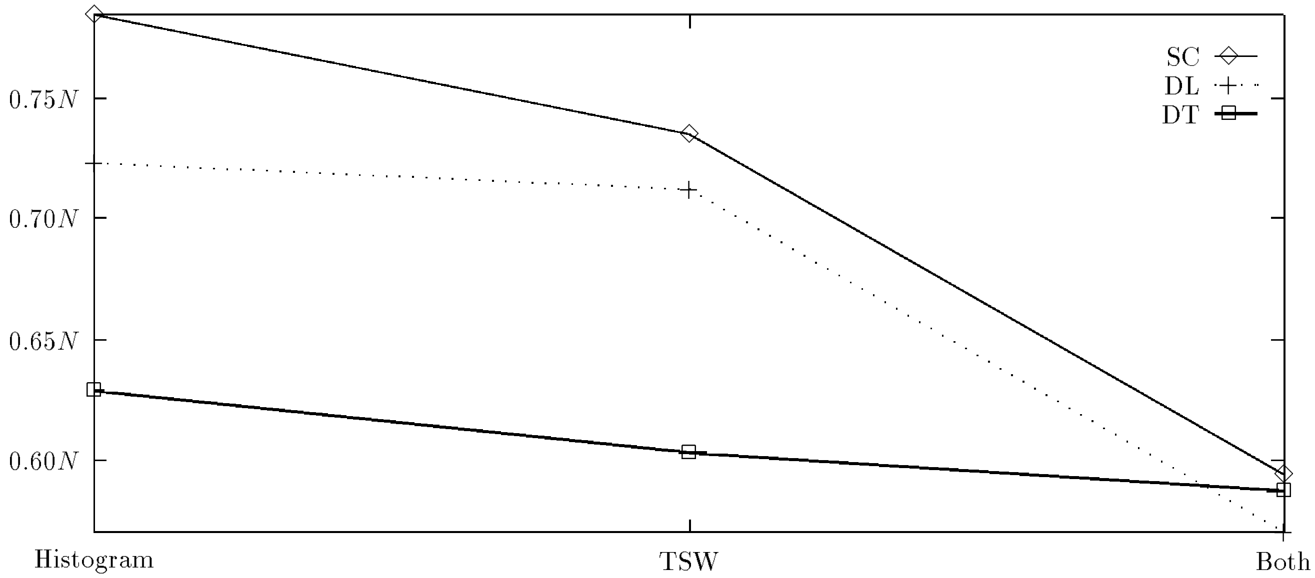


Figure 3.9: Learning time for a combination of equally good similarity measures. Notice how DL and SC catch up with DT.

on learning time for all three algorithms when the inferior hierarchies are added to each of these four. One run's information is another run's noise. For example, the "Brodatz" column shows the time for "Brodatz," (least noise) "MRSAR, Brodatz," "EV, MRSAR, Brodatz," and finally "Random, EV, MRSAR, Brodatz" (most noise). (Some of the SC bars overlap.)

We see that DL and DT are more sensitive to grouping noise than SC. This is expected since they have a weaker, less focused, bias. Even though DT exploits a single hierarchy best, adding groupings allows SC to overtake everyone. Thus the relative application performance of these learners also depends on the amount of grouping noise, i.e. the number of irrelevant similarity measures, which for image and video libraries is expected to be high (chapter 1).

The dominance of some similarity measures (i.e. MRSAR) over others was obvious in the previous four experiments, but it need not be so in general. The next set of experiments attempted to achieve synergy with two roughly equally good measures, a Euclidean gray-level histogram distance and the tree-structured wavelet (TSW) transform [7]. The histogram distance is the squared error between all 256 pixel-value bins, accumulated over an entire image. The learning time for the three algorithms, using the hierarchies alone and together, is shown in figure 3.9. Using either hierarchy alone, the order from best to worst was: DT, DL, SC. Here we see a case where a less constraining bias has advantage. Using both hierarchies, all three algorithms were roughly equivalent, i.e. DL and SC recouped their position. One explanation for this is that DT's superior expressive power was counteracted by the ambiguity introduced by the second hierarchy.

3.3.2 VisTex collection

The next data set was more challenging. Each of the 167 textures in the color VisTex collection (http://www-white.media.mit.edu/vismod/imagery/VisionTexture) was equally divided into four 256×256 non-overlapping images. The desired classification corresponded to the 19 mutually exclusive categories "bark," "clouds," "water," etc that were attached by the creators of the distribution. Unlike the previous test, the classes ranged in size from 8 (clouds) to 80 (fabric). This database has considerably more variety and less image homogeneity than Brodatz, and the classes are more semantic. For example, some of the images classified "building" have a dark building at the top and green grass at the bottom, and some are a close-up of a different building at an oblique angle.

The learning times for the three algorithms are given in figure 3.10. The color histogram metric

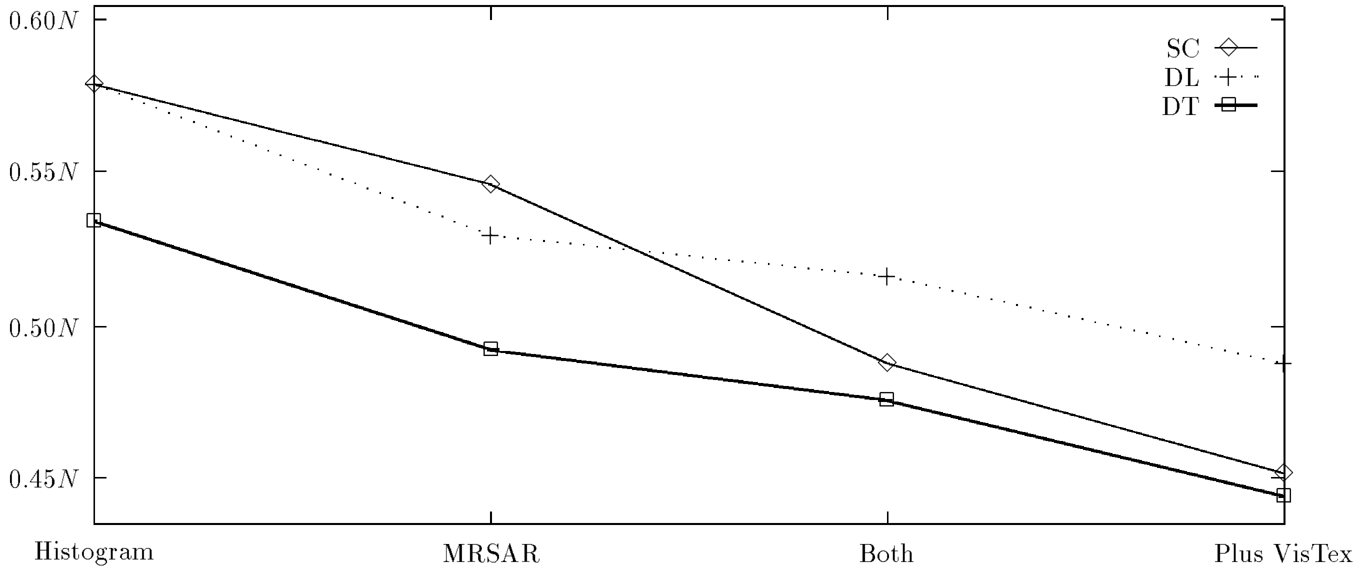


Figure 3.10: Learning time summary for the VisTex collection.

is the same as the gray-level histogram metric except that there are 3 color channels $\times 256 = 768$ bins. The color space was the Ohta principal components of the RGB cube [35]. The "VisTex" hierarchy contains the 167 four-image groupings defined by the cropping process. The DT algorithm does best, but as before it exhibits diminishing returns with more hierarchies, while the SC does not. The best time is $0.44N$, indicating that little perceptual consistency was observed for this categorization, or that the learner's bias was not properly focused. Chapter 4 will show how to deal with this situation, by changing the parameters of the bias and improving the groupings available to the learner.

3.3.3 BT collection

Finally, the performance of the algorithms was measured for labeling the color natural scenes in the "BT images" (http://www-white.media.mit.edu/vismod/imagery/Bt_photos). Three human subjects were asked to freehand outline regions in 25 of the natural scenes and assign the seven labels "building," "car," "grass," "leaves," "person," "sky," and "water" to them. They were not asked to make precise boundaries or make decisions on a strictly perceptual basis (both of which would have aided learning). Then a majority vote among the subjects was used to derive a single, approved ground-truth segmentation and labeling of those images. Since within-image groupings were computed using a 16×16 tessellation, the ground-truth segmentations were quantized to that resolution. This yielded $N = 4546$ labeled patches of 32×32 pixels. The resulting ground-truth is shown in appendix A. The learning benchmark was then used to simulate what the same labeling process would have been like with a computer assistant, if the user had tapped a few patches of each class instead of carefully outlining all occurrences of them.

The results using within-image groupings are shown in figure 3.11. All learners had available groupings from the MRSAR and Ohta histogram measures, computed from an 8×8 tessellation of the images (see section 2.3). This tessellation is coarser than the ground-truth, but as usual singleton groupings, at the finest scale, are always available to the learner.

Adding groupings computed from a 16×16 tessellation did not improve learning time. Surprisingly, across-image groupings computed via the same two measures also did not improve learning. This indicates that the across-image perceptual variations in the semantic classes escaped these measures. Hierarchy improvement (section 4.4) will be required to deal with this task.

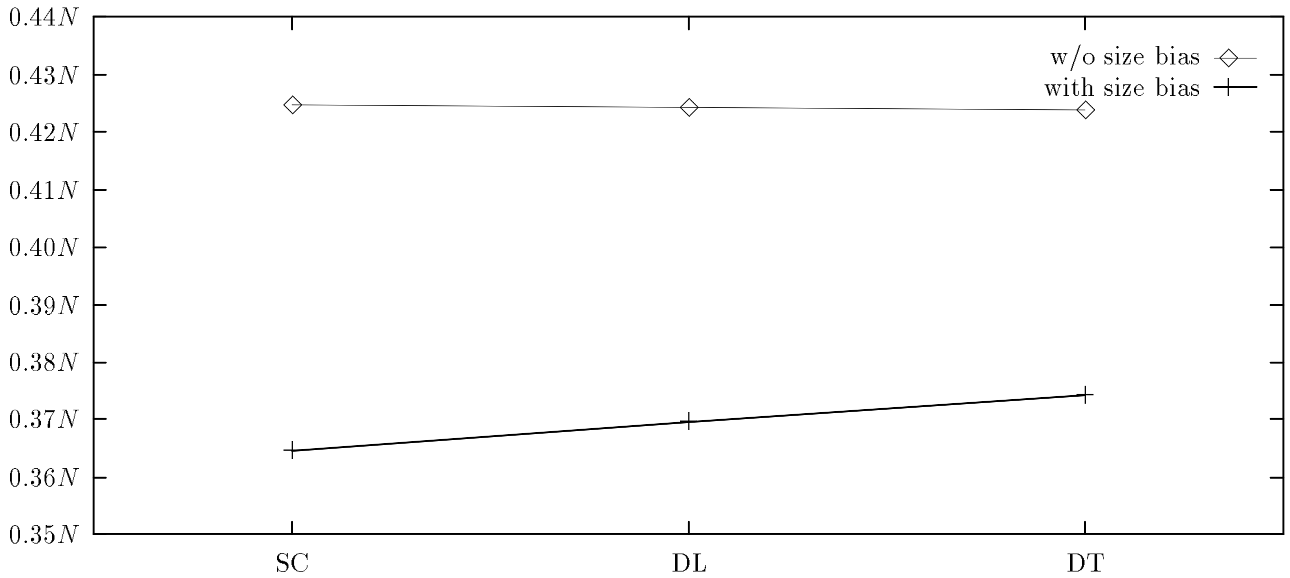


Figure 3.11: Learning time summary for the BT collection. Also shown are the times with the size bias to be explained in section 4.3; note the superiority of SC ($0.364N$) with the improved bias. The time for all grammars without the size bias is $0.424N$, hence this collection is slightly easier than VisTex ($0.44N$).

3.4 Conclusions

The experiments in this chapter illuminated the effect of bias on the learning process. The SC algorithm, by having the strongest bias, was the most stable as new examples arrived and was the least sensitive to spurious groupings. The DT algorithm, by having a weak bias, made the best use of available groupings but was highly sensitive to spurious groupings. The DL algorithm fell between these two extremes.

Because of its robustness and speed, SC emerges as the preferred learning algorithm, of these three, for FourEyes. However, it is not the best in all cases, and there is more to consider than concept grammar. A good learner has a strong and accurate bias. When a bias is strong but inaccurate, then a weak bias may do better. But while bias strength can be derived from the algorithm, bias accuracy cannot. Hence choosing the bias strength of a learner is somewhat of a gamble. As observed in [15] with respect to a decision tree algorithm (CLS):

> This faces the potential user of a CLS program with something of a paradox. Depending on the type of problem, he should use a particular program. He does not know what the type of problem is until he applies a program to solve the problem. What should he do?

Fortunately, we have faced a similar problem before. Because they are task-dependent, biases for learning resemble attributes for image retrieval. As with attributes, we can improve robustness by using a flexible architecture which does not require commitment to an ad hoc choice. Then we can let the computer make the choice, when more information is available. This is the topic of the next chapter.

Chapter 4

Bias selection

In machine learning, both the examples and the bias play crucial roles, as demonstrated in the previous chapter. If we want to design a system which learns well, we have to choose its bias carefully. It is tempting to think that this choice can be avoided by assigning equal weight to all answers, however that is not a position of ignorance but rather a strong statement about the world. No matter what we do, we are going to have a bias, so it makes sense to take the bull by the horns and make sure the bias is a good one. (This argument is similar to those for Bayesian priors and regularization.)

Interestingly, while bias selection is common in designing a learner, it is typically done manually by the designer, either arbitrarily or via trial and error. Without manual intervention, the bias is then fixed, making the learner inflexible to new contexts. Bias includes:

- The instance attributes, such as color, texture, or shape
- The learning algorithm, such as set-covering, decision tree, or multi-layer perceptron
- The parameters to the algorithm, such as pruning thresholds, initial weights, and weight decay factors.

This chapter investigates automation of the bias selection process. Not only does this relieve the designer's burden and make the resulting learner more accurate and systematic, but it makes it feasible for the learner to change its bias dynamically, from problem to problem. Thus the learner can *improve its own learning ability.*

4.1 Historical trends toward bias selection

The task of building a more efficient learner is generally known as "speedup learning," where the learner attempts to improve its own learning performance. The task of learning how to learn has cropped up repeatedly in the machine learning and artificial intelligence literature, culminating recently in the topic of multistrategy learning [30], where the learner is given multiple biases to choose from.

The fundamental adversary being fought is the generality/efficiency tradeoff: when a learner weakens its inductive bias and allows a greater variety of concepts to be learned, then it takes more examples, on average, to learn any particular concept. For example, in Markov modeling the number of examples required for training increases dramatically from discrete to mixture output densities, and it gets much worse for stochastic grammars. While more general learners can solve more problems, their low efficiency can make them worthless. At the other extreme, learners which perform well tend to be the ones with the strongest and most accurate inductive bias. Since accuracy can only be determined empirically, it is important to be able to change it to track the problem domain.

Two provocative early AI experiments, which had good biases but did not modify them, were BACON [21] and AM [22]. BACON's task was to discover simple mathematical equations to explain a data set, such as a table of planetary revolution times versus their orbital distance. The program was able to discover Kepler's law and several others, by exploring and refining hypotheses which were arithmetic combinations of attributes. Its power came from a constrained search through the concept space, which trimmed down the choices available to the learner and thus reduced the apparent complexity of the problem, as described in [46]. AM also used heuristics for mathematical discovery, this time to optimize a notion of "interestingness." Starting with set theory and some general heuristics, the program could synthesize concepts ranging from natural numbers to the fundamental theorem of arithmetic. Like BACON, it could build new concepts via heuristic refinements on old ones and thus rapidly reach a respectable level of sophistication without getting lost in a combinatorial explosion. Indeed, it was this notion of heuristics which prune the search through concept space that gave rise to the term *inductive bias*: "any basis for choosing one generalization over another, other than strict consistency with the observed training examples" [33]. Bias is crucial to any problem for which many possible answers may be valid, i.e. problems which are ill-defined such as representation, extrapolation, segmentation and clustering.

Recognizing this, subsequent learners added control knobs for their bias. For example, the clustering program in [31] provided the user detailed control of its cluster evaluation function (its bias), in terms of simplicity, commonality, fit, discrimination, and so on. However, if the underlying attributes are nonintuitive, such controls will not make the clustering process intuitive again. Furthermore, ten or twenty control knobs are insufficient for representing all of the possible clusterings that the learner could produce. Note the similarity between this situation and the database query situation described in chapter 1. As we noticed there, it would seem more appropriate to leave the clustering process ambiguous to the user in order to allow automatic construction of clustering rules sophisticated enough to make the measured attributes useful.

Thus we see attempts to reduce the burden of setting these control knobs. One approach is transfer of training from old to new problems. For example, backpropagation training of artificial neural networks can be biased by the initial connection weights. In [44] and [26] this property was exploited by incorporating pretrained networks as parts of a new, larger network which was then retrained on a new problem and required fewer examples and/or was more accurate. Unfortunately, this technique seems to require careful selection of the pretrained networks, without some form of deliberate noise suppression. DT suffered from this problem in section 3.3.1.

A more automatic approach is for the learner itself to modify its bias at runtime, e.g. by changing the set of available concepts. Utgoff [58] showed a learner which could automatically enlarge its concept language when the current one was unable to express the target (i.e. all concepts were inconsistent with the examples). However, it was limited to only weakening its bias, which will eventually degrade overall performance, without a corresponding strengthening operation. Multistrategy or task-adaptive learners [29] choose from a fixed set of biases in a problem-dependent manner. For example, [38] used a chain of learners, from explanation-based (strongly biased) to theory-driven (less biased) to similarity-based (weakly biased), which were applied in turn until one produced a consistent concept. Problems which benefited from the domain knowledge of the explanation-based learner were solved using the first component, while problems which could only be solved by raw similarity considerations (i.e. in terms of attribute clusters) were handled by the last component.

4.2 Continuous learning

Continuous learning attempts speedup by learning continuously, across problems, not just within them. As opposed to multistrategy learning, which selects among several fixed biases, continuous learning adds the ability to construct new biases on the fly. The continuous learner collects knowledge over the problems and problem domains it has experienced. As it acquires experience it develops a new bias. Continuous learning generalizes multistrategy learning, transfer of training to new problems, learning by analogy to previous problem solutions, and concept language modification.

While the individual tasks assigned to the learner may be supervised, continuous learning is an unsupervised process. The learner is not explicitly told which aspects of a problem will carry over to the next. Therefore a continuous learner must *self-organize*. This gives a hint about how to construct one, by following what we will call the **self-organization principle:**

> After solving a problem, make a change that will allow the system to perform better on the same problem again.

For example, these "neural" algorithms all follow the self-organization principle (though they are not continuous learners), where each "problem" is a single input point:

Kohonen map [20] maintains a set of cluster centers. Given a point to approximate, the map outputs the nearest cluster center. Then it moves that center (and possibly others) slightly closer to the input point. The idea is that the approximation will be better for that point again. The result of this simple algorithm is that cluster centers will arrange themselves to match the distribution of the input, so that the mean squared error of the approximation is minimized.

Backpropagation [50] modifies connection weights to minimize the difference between the last output and the desired output. Network weights will arrange themselves to approximate the desired input-output mapping.

Perceptron modifies the normal of the linear discriminant to point closer to a positive example or farther from a negative example. The effect is to find a boundary between the classes.

Sparse distributed memory [19] maintains a randomly distributed set of fixed location memory cells. To write a value into an arbitrary location, average the value with the contents of the D nearest cells. To read a value from an arbitrary location, take the average of the D nearest cells.

The self-organization principle is also used in searchable data structures, to dynamically optimize them according to the distribution of queries. For example, the "move to front rule" is used for sequential lists [48] and the "move to root rule" is used for binary trees [1]. Of course, if the query distribution is fixed and known beforehand, a static Huffman tree is optimal; this is the usual tradeoff between off-line and on-line solutions. Unfortunately, a continuous learner has neither of these two luxuries for determining its bias.

Now we can derive the **self-organization principle for continuous learning:**

> Modify the parameters of the learner's bias to place more favor on the target concept just learned.

In the algorithms described in chapter 3, bias is determined by the concept grammar (set cover, decision list, or decision tree) and the vocabulary C. So one way to make a continuous learner would be to learn an algorithm for multistrategy combination of output from the different grammars, e.g. by increasing the weight of the grammar which gave the correct answer last time. However, this may produce a very slow system. A second way is to modify the vocabulary to include groupings which would simplify the expression of the last target concept. This is explored in section 4.4. A third way is to first extend the bias of the algorithm, beyond just its concept language, to include more parameters and then modify those. This option is explored next.

4.3 Vocabulary weighting

A straightforward bias extension for the three described algorithms is to give each grouping a weight which can vary depending on the kind of problem being solved. The bias toward a concept is a function of its description length, as before, plus the total weight of the groupings in it. This allows the learner to encode the plausibility of a grouping. An experiment was made to learn the best weight function over groupings, using a kind of Sparse Distributed Memory; this yielded a significant decrease in the learning time of SC [32]. That work used a lookup table to store the

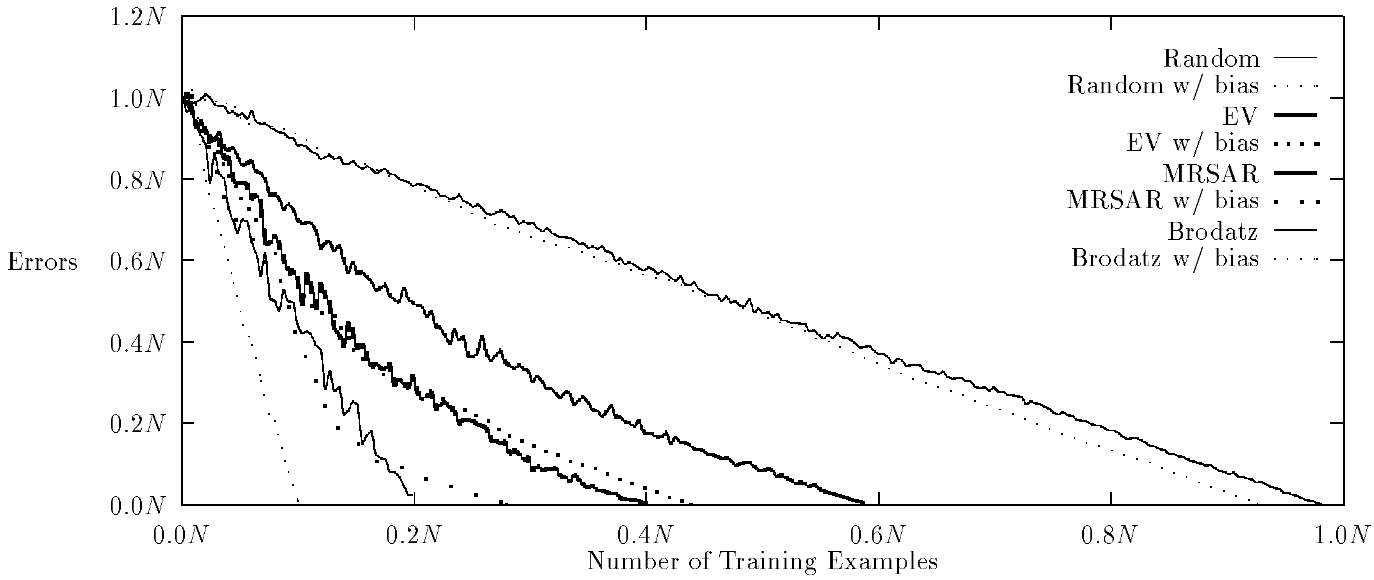


Figure 4.1: SC with a grouping size bias: Brodatz collection. Includes figure 3.5. Using size information produces a faster drop.

weights, however more generalization could be achieved by computing the grouping weight from some characteristic of the grouping, not just its name. One informative characteristic of a grouping is its size; inspection of the weight lookup tables revealed that under certain conditions there is a simple relation between the size of a grouping and its weight.

The relation between size and weight can be computed for SC in a straightforward manner. Only those groupings which contain at least one positive example and no negative examples can contribute to a set cover; thus grouping weight should be proportional to the probability of this occurrence over a randomly chosen set of examples. Let's assume, as in the benchmark program of section 3.3, that instances are chosen with uniform probability. A convenient visualization is throwing darts blindfolded at figure 3.2: if the dart lands in the target it is a positive example, otherwise a negative one. The positive and negative examples thus randomly produced will number in proportions $p : n$, depending on the size of the target concept. For example, if the target covers one third of the space then the proportions are $1 : 2$.

Consider a grouping which covers a fraction s of the instance space: if $s = 0$ it covers no instances and if $s = 1$ it covers all instances. The probability that the grouping will not cover n negative examples is $(1 - s)^n$. Similarly, the probability that it will cover at least one of p positive examples is $1 - (1 - s)^p$. The probability that the grouping satisfies the set cover requirement is therefore $w = (1 - (1 - s)^p)(1 - s)^n$.

Each of the three algorithms works by iteratively evaluating sets for use as a cover or decision. To incorporate the size bias, they simply multiply the grouping weight w by the previous optimization criterion (number of covered examples or sample variance). This corresponds to the second stage of figure 1.6: it is independent of the learning algorithm used in the third stage. The parameters p and n were arbitrarily fixed at 1, reducing the formula to $w = s(1 - s)$. For the Brodatz test, this is correct; for the other tests, it is only an approximation, since the class sizes vary. The formula was also offset slightly to prevent it from reaching zero.

The three data sets of chapter 3 were attempted again using the grouping size bias. The SC curves for the Brodatz test are shown in figure 4.1 (dotted), along with the curves without the size bias (solid). The size bias often reduced learning time by $0.1N$ examples, corresponding to one less example per nine-texture group. These extra examples had apparently just served to isolate the right-sized candidate concept. Also note that the error variance between steps has decreased, as there are fewer ties between covers. Similar speedups occur for the other hierarchies and the other

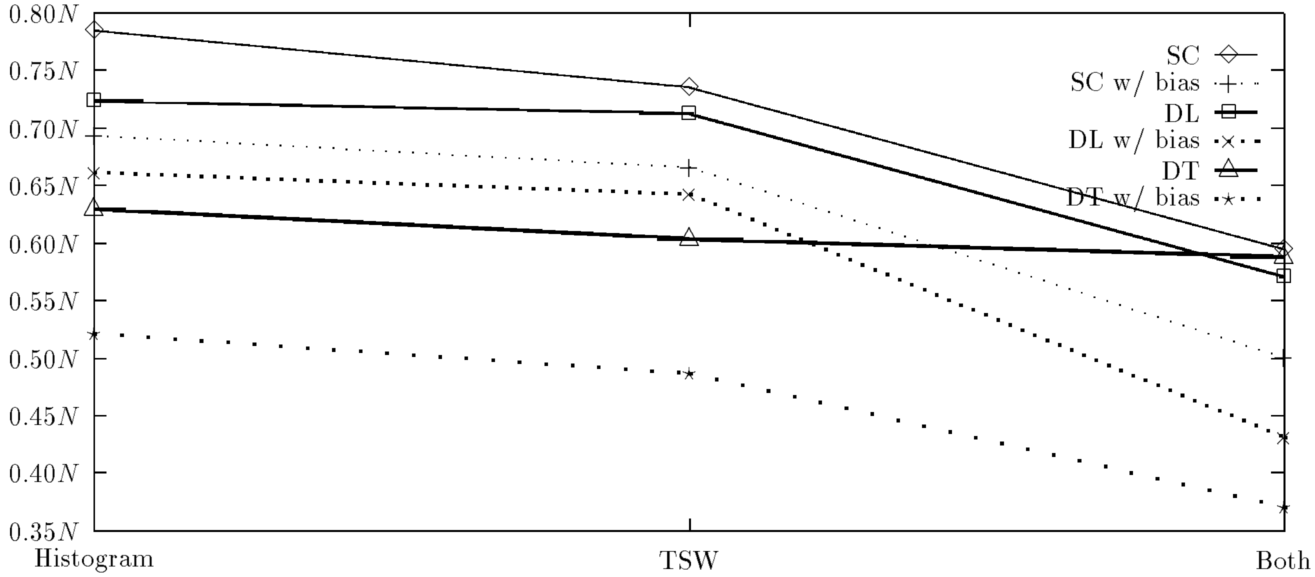


Figure 4.2: Learning time for a combination of equally good similarity measures, using a size bias. Includes figure 3.9.

two data sets (figures 4.2, 4.3, 4.4, and 3.11).

The size bias is also effective for DL and DT. This is surprising because, unlike in SC, the value of a grouping depends on previous decisions and so may change radically during the development of a decision structure. In particular, the relative proportions of positive and negative examples may differ down different paths from a decision node. Even so, the size bias improves performance on almost all problems. Unfortunately, it appears to increase sensitivity to grouping noise (figures 4.3 and 4.4).

These experiments demonstrate the effectiveness of the size bias for certain proportions p and n. To make sure that the learner has a good estimate of these, a standard self-organization algorithm, or simple averaging, can be used to generalize across problems. The DL and DT algorithms might also benefit from a principled way to propagate class proportions through decisions.

4.4 Vocabulary modification

Speedup learning can also be understood in terms of data compression. A successful learner must represent, in some form, all of the target concepts expected of it. The algorithms we have seen so far (set cover, decision list, and decision tree) require more examples to learn more complex concepts, because they all try to produce the simplest concept consistent with their examples. So to reduce the average number of examples required by the learner we should design its concept language so that more frequent targets have shorter expression. Consider a (possibly infinite) message consisting of all of the target concepts expected of the learner, occurring in proportion to their probability. Then a good concept language should achieve good compression of this message. (However, good compression may not be all that is required for good learning.) This suggests an algorithm for language construction, via analysis of the stream of target concepts requested of a learner, perhaps along the lines of LZW or another on-line compression algorithm. (Note that these also follow the self-organization principle.)

This section presents such an algorithm, which works by incremental modification of grouping hierarchies. The algorithm is executed by the learner after a target concept is identified. The hierarchy improvement algorithm closely follows the self-organization paradigm, since the modification is chosen to reduce the number of examples that the learner would require had it been given the same target again. Since it only addresses concept vocabulary, not grammar, this algorithm is equally

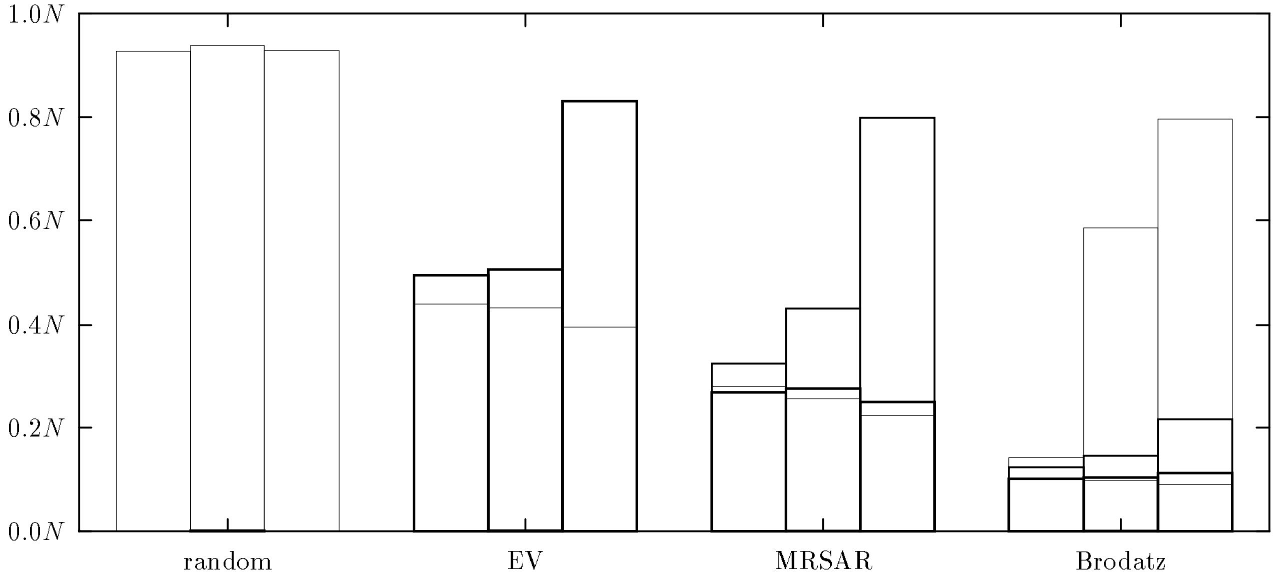


Figure 4.3: Learning time for varying amounts of spurious groupings, using a size bias. Each set of three bars is SC, DL, and DT, respectively. With little noise, the learning time has decreased for all three, compared to figure 3.8. However, DL and DT have become much more sensitive to noise.

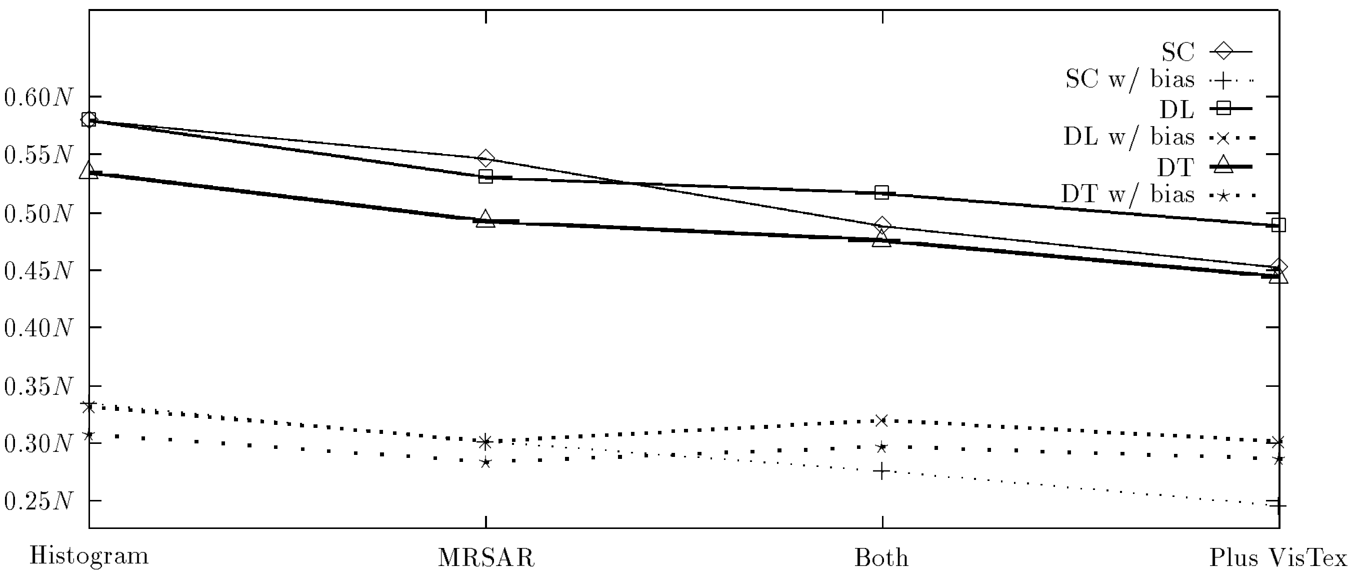


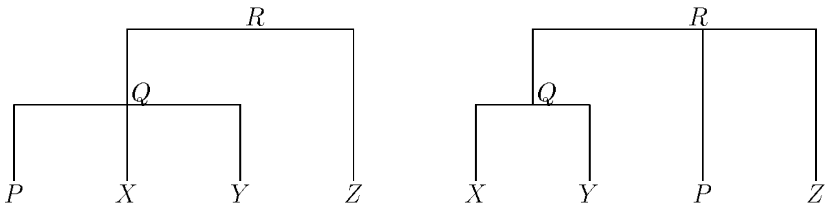
Figure 4.4: Grouping size bias: VisTex collection. Includes figure 3.10. Note the eventual superiority of SC with the size bias.

compatible with SC, DL, and DT: it corresponds to modifying the first stage of figure 1.6, which is independent of what happens in the second or third stages.

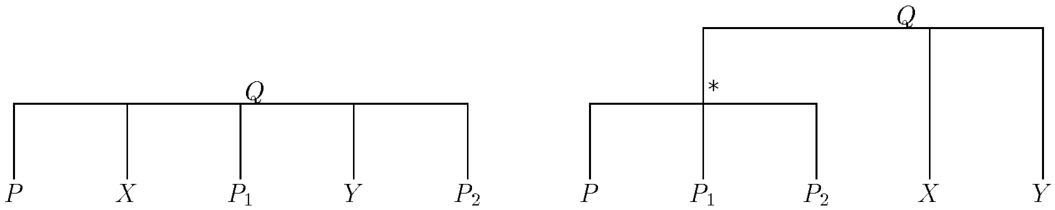
4.4.1 Hierarchy improvement algorithm

The hierarchy improvement algorithm is executed by the learner once it has succeeded in learning the target concept, i.e. all leaves are labeled either positive (included in the target) or negative (not included in the target). Let an interior node be "positive" if it parents only positive nodes. Then the algorithm iterates over all positive non-root nodes P (with parent Q) in the hierarchy, starting at the leaves, and tries to apply two modification rules:

Promotion If P has no positive siblings, make P a sibling of Q.



Union Otherwise construct a new positive node under Q to contain P and its positive siblings (P_i).



The running time of this algorithm is linear in the number of nodes in the tree. If nodes with a single child are omitted from the tree, then the number of nodes is linear in the number of leaves, making the total time linear in the number of leaves. In this thesis, the number of leaves is always the number of instances (database entries).

Figure 4.5 shows the effect of this algorithm on an entire hierarchy. Nodes with a single child are redundant and so are omitted. One more execution of the algorithm on this hierarchy will unify all positive nodes. Indeed, the algorithm will always eventually do this.

This algorithm satisfies the self-organization principle, i.e. it reduces the number of examples that the learner would require had it been given the same target again. Proof:

1. The algorithm never increases the size of the representation of the target concept for either the SC, DL, or DT grammars. This follows from the fact that it never increases the impurity of a node or alters the internal structure of positive nodes.

2. The algorithm will eventually contain the target concept as a grouping, when executed enough times. This follows from the fact that all positive nodes must eventually join under one positive node. If there are two positive nodes which are siblings, they will be merged by the union rule. If there are two positive nodes which are not siblings, they will be promoted until they become siblings, which must happen when they reach the root.

The first claim shows that learning time is nonincreasing; the second shows that it converges to the optimum. Learning time may remain the same after one hierarchy improvement, but not indefinitely.

The algorithm can also be judged by how well it satisfies two reasonable requirements for a successful self-organization algorithm:

Generalization The modification for one problem improves performance on subsequent problems.

Retention The modification for one problem does not clobber too many previous modifications.

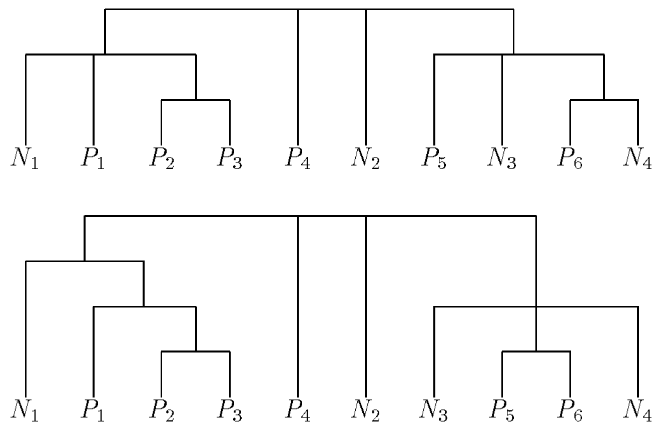


Figure 4.5: Hierarchy improvement. Groupings now contain less mixing of positive (P) and negative (N) nodes.

In neural networks, generalization is ensured by making changes which average together and retention is ensured by an annealing constant. The hierarchy improvement algorithm achieves generalization because it still allows the new groupings to be combined according to the concept grammar. Subsequent problems can utilize the new groupings even if target concepts are significantly different (see the transfer experiments below). For example, if two targets both agree that instances A and B should be grouped together, though one may say both positive and another both negative, then they will both strive for and benefit from the unification of A and B. If two targets disagree on the unification of A and B, then the hierarchy will oscillate. Points of agreement will be bolstered and points of disagreement made flexible (see figure 4.6).

The hierarchy improvement algorithm achieves retention by a curious automatic annealing process. Shorter hierarchies are more amenable to change, since the number of siblings, and thus the effect of the union rule, will be high. However, each application of the union rule deepens the hierarchy. So short trees rapidly become tall ones, where the number of siblings is low and the union rule has less effect. This is demonstrated in the experiments.

4.4.2 Experiments

The effect of hierarchy improvement on learning a single target is shown in figure 4.7. This is an ideal case for a continuous learner. The task was the two-class problem of section 3.3.1 ($N = 1008$) and the algorithm was SC starting with the MRSAR hierarchy. The graph shows the learning curve after successive hierarchy improvements, each of which cut the learning time in half. After five improvements the learning time fell to 11 examples. Figure 4.8 shows the evolution of learning time for all three algorithms. The learning time reduction between passes depends on the structure of the tree, as described above. Figure 4.9 shows the same experiments starting from a random, instead of the MRSAR, hierarchy. Improvement proceeds more slowly. Tree improvement and the grouping size bias are completely compatible with one another; when used together the learning times are further reduced.

Transfer of training was tested next. Ten classification problems were constructed which all had something in common, viz. the Brodatz nine-image texture groups. Each of the ten problems randomly merged these groups to form two equal-sized classes. Then the learner was asked to learn each of the ten problems in sequence. The hierarchy improvement algorithm, starting with the MRSAR hierarchy, was applied after solving each problem. As shown in figure 4.10, SC was able to achieve significant performance gains on the later problems, even before having seen them. The graph includes the "ideal" learning time, resulting from simply running the learner on the

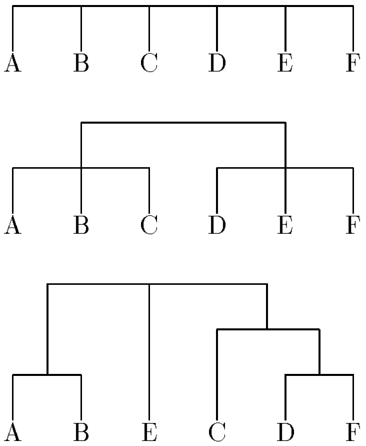


Figure 4.6: Improvements on a flat tree resulting from the target concepts $\{A, B, C\}$ and $\{D, E, F\}$, then $\{A, B, E\}$ and $\{C, D, F\}$. $\{A, B\}$ and $\{D, F\}$ have formed persistent groups, while C and E are loose. The asymmetry is due to the execution order (a sensitivity of most self-organization algorithms).

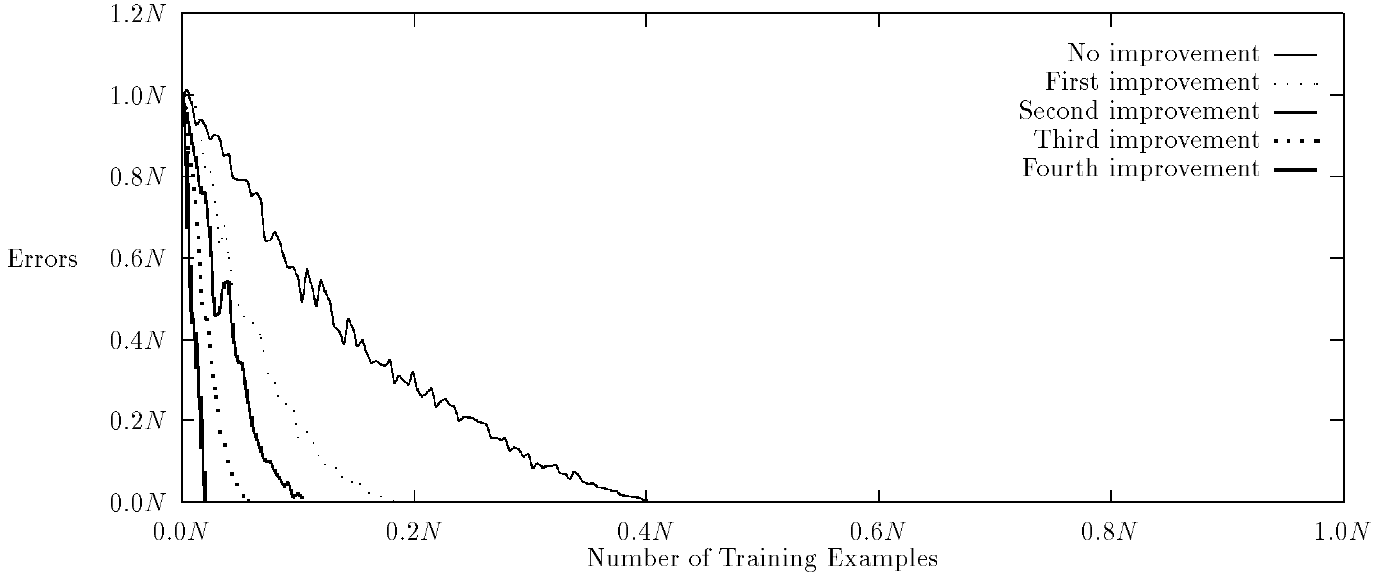


Figure 4.7: SC, with hierarchy improvement on MRSAR, iterated on the same problem.

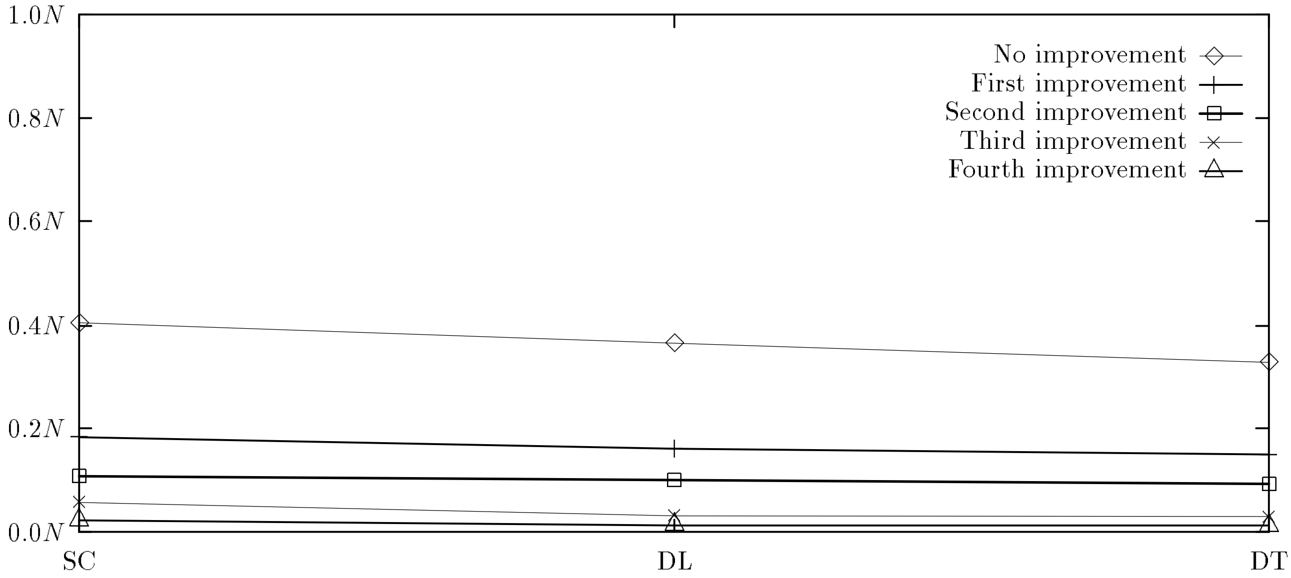


Figure 4.8: Learning time with hierarchy improvement on MRSAR, iterated on the same problem.

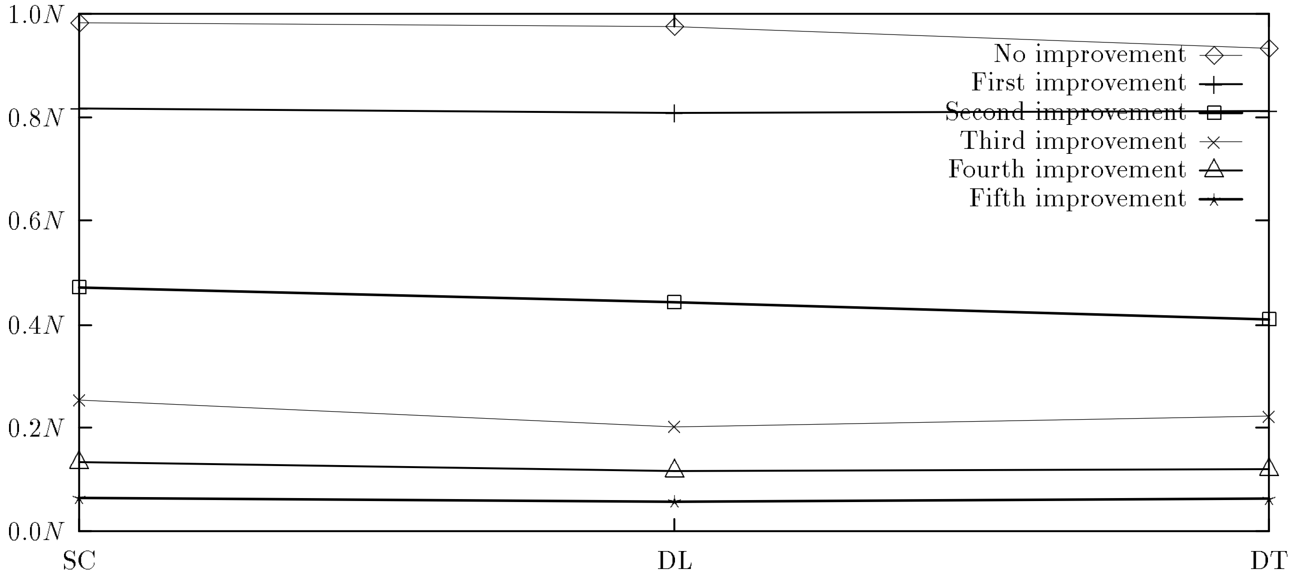


Figure 4.9: Learning time with hierarchy improvement on Random, iterated on the same problem.

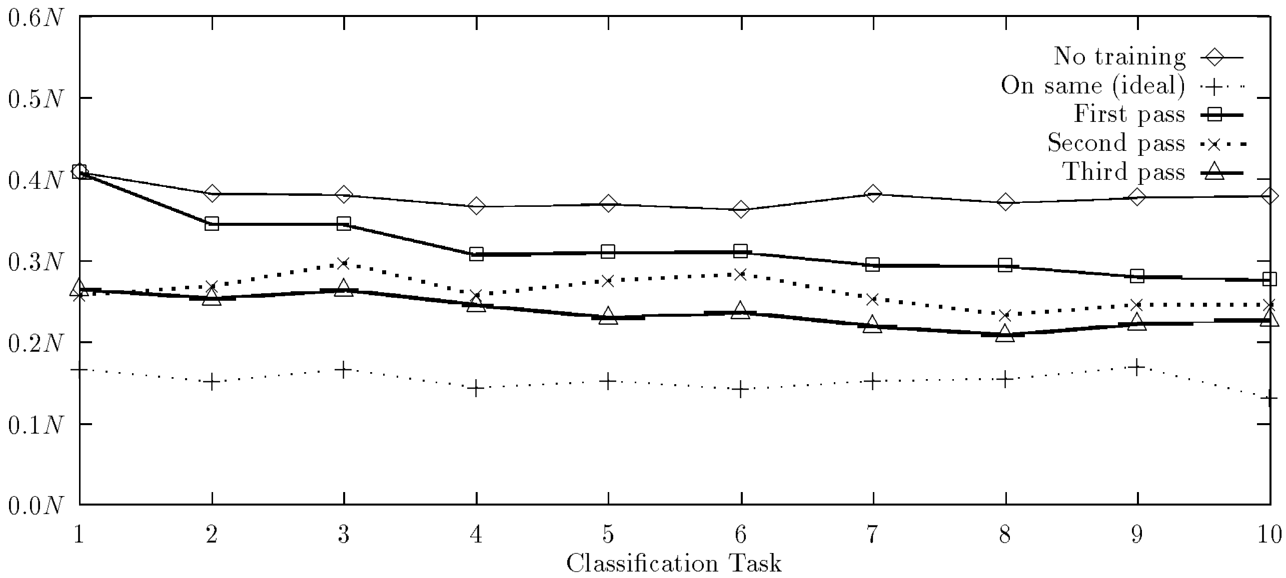


Figure 4.10: Learning time with hierarchy improvement on MRSAR, running on ten similar problems. The "no training" and "ideal" curves are the results of running the learner twice on the same problem. The other curves are the result of sequential training; they tilt slightly to the right.

problem twice (as in the previous experiment). The performance on problem ten during the first pass indicates that training on the first nine is half as good as training on number ten itself. The explanation is that the hierarchy develops most of the groups of nine textures, even though the learner never sees any of these groups in isolation—only mixed together as part of a $N/2 = 504$-texture class. Figure 4.6 demonstrates how this group extraction happens. When given a second and third run across all problems, the hierarchy develops all of the groups of nine, effectively becoming the "Brodatz" hierarchy of section 3.3.1. Thus the best time achieved in subsequent passes is approximately $0.2N$, the time for the "Brodatz" hierarchy (cf. figure 3.5). The retention required for "ideal" times is not achieved with this hierarchy, as opposed to the next experiment, with a flat hierarchy.

Figure 4.11 reports on the same experiment starting from a flat hierarchy (nothing but singleton groupings). Without improvement, such a hierarchy yields no generalization—all problems require N examples. However, this tree is so malleable that training on the same problem gives learning times of 2–4 examples (bottom of the graph). Sequential problem-solving, with improvement, eventually yields results comparable to starting with the MRSAR hierarchy. On the second pass, however, we see the effect of annealing. The hierarchy adapted to the first few problems better than to the rest, though the performance on all has greatly improved. This demonstrates an ability for both retention and generalization.

As a check, this experiment was repeated with ten random (nothing in common) two-class problems. Figure 4.12 shows that SC with MRSAR reaps little transfer benefit, especially compared to its ideal performance. Even after four passes, there is not much retention. Figure 4.13 shows SC starting from a flat hierarchy. Now the annealing, and the retention, is dramatic, as seen in the skew in the second pass. (This is not because of an idiosyncrasy in the latter problems; the initial problem ordering doesn't matter.) The hierarchy has adapted to the first problem very well, requiring only 8 examples. The retention has also allowed the performance on problem ten to improve slightly when it is seen again, ten problems later. (Specifically, it takes $0.83N$ examples after two passes, compared to $0.88N$ for four passes with MRSAR.)

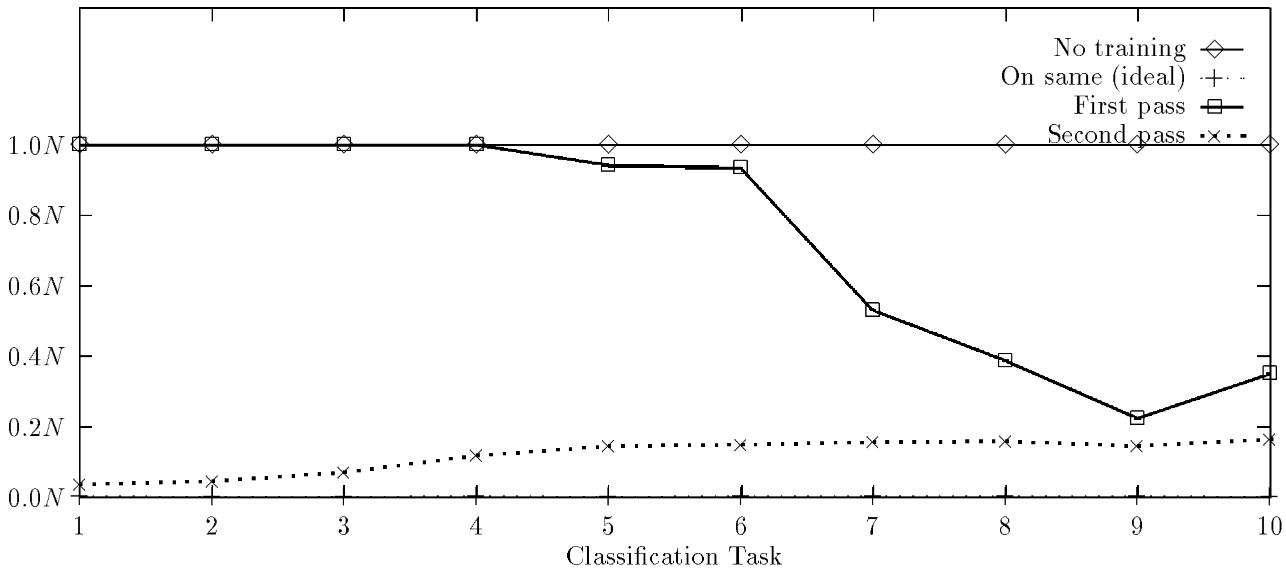


Figure 4.11: Learning time with hierarchy improvement on Flat, running on ten similar problems. (Cf. figure 4.10.)

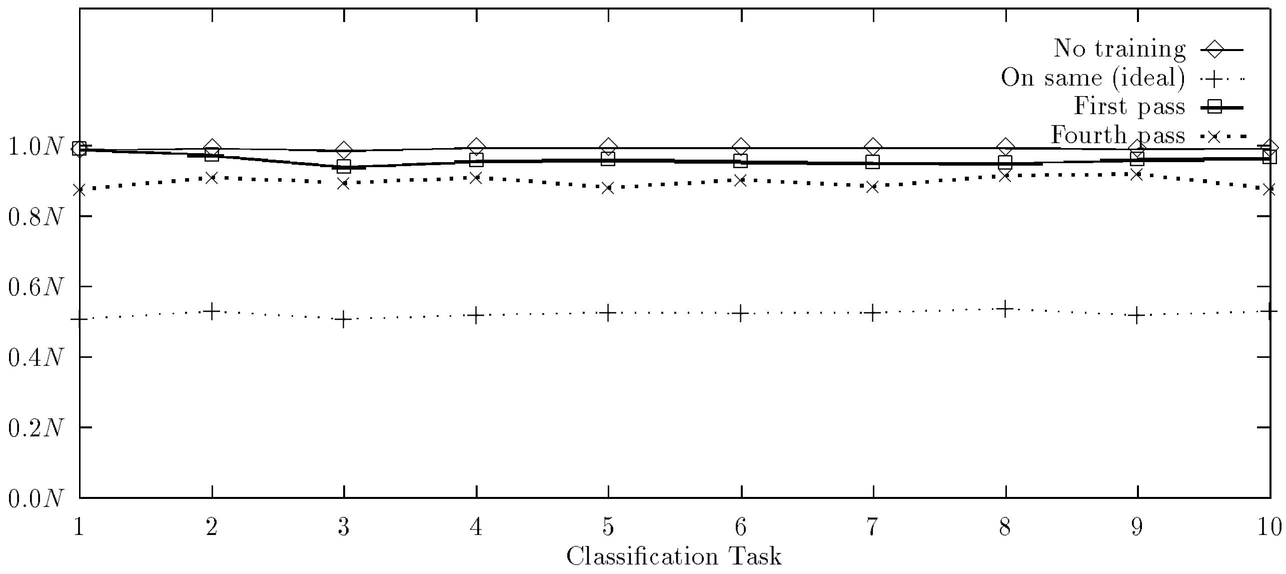


Figure 4.12: Learning time with hierarchy improvement on MRSAR, running on ten random problems. In contrast with figure 4.10, there is little transfer of training.

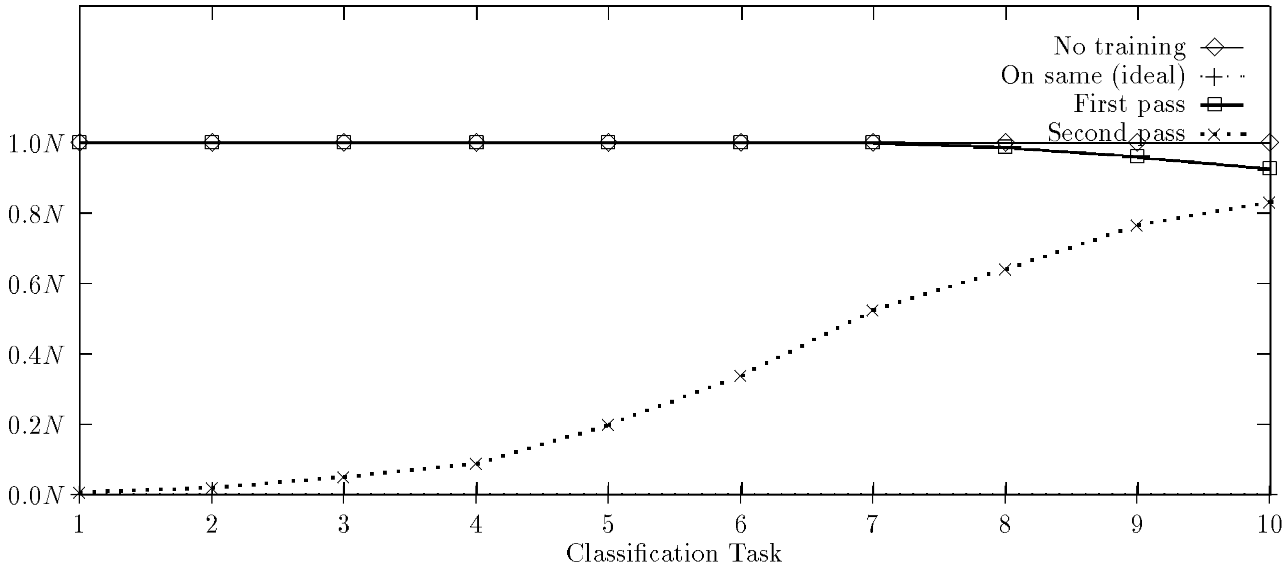


Figure 4.13: Learning time with hierarchy improvement on Flat, running on ten random problems. (Cf. figure 4.11.) The ideal times are again on the bottom on the graph.

4.4.3 Related work

Improving the vocabulary available to a learner is also investigated in [46]. That work was inspired by the observation that large decision trees are produced by concepts having many separate peaks in attribute space with many attributes defining each peak. Therefore the performance of a decision tree learner can be improved by applying various transformation operators to the attributes, in an attempt to reduce a notion of complexity called *dispersion*. Hierarchy improvement accomplishes the same goals as this scheme, using a different notion of complexity (e.g. the number of positive nodes with non-positive siblings, or the impurity measures of section 3.2.3). This general scheme carries over to learning problems with linear or continuous attributes, given suitable transformation operators and definition of dispersion. More research needs to be done in finding and validating instantiations of this scheme.

An attribute construction heuristic for decision trees was proposed in [36]. It was inspired by the observation that decision trees cannot explicitly represent branching on conjunctions of decisions. When a concept requires a conjunction of decisions, two identical subtrees are created; they call this the *replication* problem. The heuristic, then, is to construct new candidate decisions from the conjunctions of consecutive decisions which appear in the decision tree. The once-consecutive decisions will then appear as one decision in the next decision tree. We might call this "tree compaction;" it is similar to the compression heuristics for self-adjusting data structures. Unfortunately, the paper does not give a proof of convergence nor demonstrate the transfer of this training to new problems. The constructed attributes may not form a hierarchical structure, making the algorithm difficult to implement efficiently.

Continuous learning shares qualities with reinforcement learning [55], because it involves simultaneous modeling of the world and acting upon that model. But while reinforcement learning, e.g. Q-learning [60], learns observation-action rules for direct interaction with the world, continuous learning is indirect: it provides a bias for a general-purpose learner which then does the interaction. However, continuous learning can be applied to any learning problem, including reinforcement learning. For example, most reinforcement learners do not use a look-up table to store Q-values, as Watkins suggests [60], but rather a generalizer such as a neural network. Such generalizers can benefit from continuous learning to adapt their bias.

Neural networks and genetic algorithms have a number of parameters to their bias, yet these are notoriously difficult to adjust without brute-force search. Indeed, the learning problem required

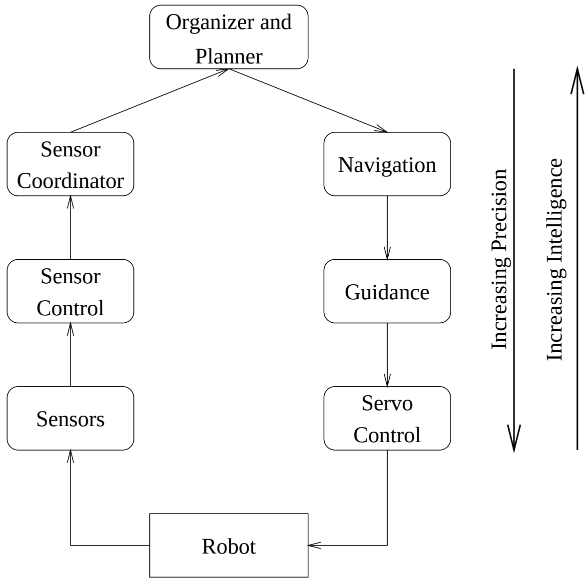


Figure 4.14: Multi-scale structure for intelligent control (taken from [13], Editor's Introduction).

to determine the best learning parameters may exceed in difficulty the original learning task. In this sense, they resemble optimization schemes for image segmentation (section 1.3.3). Hence, these learners, unlike SC, DL, and DT, are not suitable for continuous learning because they are not *malleable* enough, given our present knowledge about them.

Intelligent control is a discipline in which continuous learning has great potential. Solutions to complex control problems generally have a multi-scale structure, as shown in figure 4.14. Abstract, slow, and imprecise layers feed information down to the dumb, fast, precise layers. If each of these layers can learn, then the higher layers effectively provide the bias for the lower layers, which report on their performance so that the higher layers can adapt. This "multi-scale learning" architecture, used in FourEyes, has numerous parallels:

- The "data reductions" of [23]: induction, deduction, calculation, and access. Each reduction produces a simpler computational task. They give the example of inducing a quadratic equation from examples vs. deducing the formula for the roots vs. calculating the roots given the formula vs. memorizing the roots.
- Program compilation: refining a task down to its essentials, for performance.
- Conscious and unconscious thought: acting vs. dreaming.
- Reasoning and perception. Our eyes may "learn" the right focus or light setting, while our minds learn occlusion and grouping.
- Evolution. Each creature perceives and reasons while also participating in a common learning process.

An advantage of a multi-scale learning architecture is that it avoids the stability-plasticity tradeoff [6] faced by using a single layer. That is, a layer which learns fast also forgets fast, because it cannot take into account all that it has seen. Adding slower, long-term memories gets the best of both worlds, because the system can balance computational effort to make fast decisions which are also informed by experience. Since speed is coupled with scale, we can also call this method "multi-speed learning."

Multi-speed learning can be viewed as an on-line version of *structural risk minimization* [59]. Structural risk minimization arises from the observation that a learner's bias makes it suitable only for certain numbers of examples. For example, the number of parameters in a function approximator determines both how many examples it needs and how many it can understand. Structural risk minimization instead chooses the number of parameters based on the number of examples. For example, a different function approximator is used when 1000 examples are received than when 10 examples are received: one with more degrees of freedom so that it can build a more accurate model, without overfitting. An alternative description is that we have an infinite supply of parameters, each of which has an associated learning rate which specifies how many examples must be received before the learner can change that parameter. But this is just another way of describing multi-speed/multi-scale learning, which can deal appropriately with both very small and very large numbers of examples.

4.5 Conclusions

This chapter presented two methods for bias selection via continuous learning: one tuned for SC (vocabulary weighting) and another for any set-combining algorithm (vocabulary modification). Both significantly improve learning performance by exploiting properties of the problem domain, i.e. average class size and common sub-problems. The general continuous learning approach, however, can extend to other properties.

Inspection of these two approaches to speedup learning suggest a general approach which we can call *meta-learning.* If we consider each target concept as a single example, then the goal of meta-learning is to recover a model for those examples (a meta-concept). For example, attributes, such as average class size (section 4.3) or what instances get grouped (section 4.4), can be extracted from each target and their values estimated for future targets. This chapter followed the self-organization approach, but the meta-learning problem can be approached like any learning problem, e.g. with SC, DT, or a neural network. (Thus meta-learning can also be done continuously, leading to "speedup meta-learning" or meta-meta-learning. But remember that this is just a conceptual distinction, not necessarily an implementational one.) The parameters of the learner can then be chosen based on the meta-concept, to give the strongest and most accurate bias.

However, meta-learning requires an accurate way to transform known attributes of the domain into a bias for learning. Learning algorithms differ in how straightforward it is to do this. We can call this their *malleability.* The algorithms used here were quite malleable, but we would not expect this of a neural or genetic algorithm. Thus the discovery of not just new learning algorithms, but malleable learning algorithms, is an important research topic.

An extreme form of malleability would be a learner capable of *reflecting* on itself, i.e. a learner with complete access to, and control over, its own learning mechanisms [9], e.g. rewriting its own program. In this sense, a reflective learner would be self-aware: it is not only conscious of its situation, but also conscious of itself in that situation. These correspond to understanding the attributes of the target concepts and how inductive bias exploits those, respectively. A reflective learner could use this information to choose a better bias for itself by modifying its design. This subsumes modifying parameters or hierarchies. No such system has yet been built, though it would have profound influence on how we design machine learning algorithms.

Chapter 5

Summary

FourEyes is an extensible and self-improving learning system for assisting users in digital library segmentation, retrieval, and annotation. Digital library access requires the use of many context-dependent or noisy features, whose relevance is not always obvious. FourEyes addresses this problem on multiple fronts:

1. It first makes tentative organizations of the data, in the form of groupings. The grouping representation provides a lingua franca for different measures of similarity. Groupings can be manually provided, induced by color/texture models, derived from optical flow information, audio, etc. FourEyes uses both within-image groupings and across-image groupings composed of these (chapter 2).

2. The user no longer has to choose features or set feature control knobs. Instead, the user provides positive and negative examples which allow FourEyes to choose groupings (hence, similarity measures) automatically. The interaction is more like a conversation where both parties give each other prompt and relevant feedback in order to resolve ambiguities. The implementation of this feedback in FourEyes is cast as a concept learning problem and thus approached by an adaptation of machine learning methods. The three learning algorithms available in FourEyes: set-cover, decision list, and decision tree, adapted to the grouping representation, are analyzed in chapter 3.

3. With many groupings to choose from, the number of examples required to isolate good groupings can get large. FourEyes circumvents this via continuous learning, which is shown to significantly improve learning performance for all three learning algorithms (chapter 4). Two compatible approaches are supported: (1) placing weights on groupings according to their size and (2) changing the available groupings to reduce the complexity of learned rules. In this way, FourEyes learns at multiple scales: on a small scale for each request and on a longer scale across requests.

This dissertation introduced *malleability* as a general criterion for evaluating concept learners. This is more practical than Mitchell's proposition that *inductive bias* be used as the criterion for comparing and evaluating concept learners [33], because the quality of an inductive bias varies with the problem domain. Malleability, instead, is the ability of a learner to adapt its bias to the problem domain. A malleable learner allows direct translation of domain characteristics into learning parameters. This translation is most useful when it can be done automatically, e.g. by inspection of sample problems in the domain. Inference of learning parameters is also a concept learning problem (meta-learning: section 4.5) and can be handled in the same way. The combination of a malleable learner and a meta-learner gives a continuous learner (chapter 4), capable of modifying its concept language (as is done by choosing between strategies) and transferring training from old problems to new (as is done by making analogies).

One may ask whether highly interactive systems in the spirit of FourEyes are practical for more autonomous systems, such as robot cars or satellites, where there is no obvious source of feedback.

Furthermore, there may be cases where an operator is present but is incapable of giving good advice. However, there is always some source of feedback on system design, if only when the satellite after two years crashes into a moon. At this point it will remain better to have a design which can incorporate feedback directly, rather than one which must be re-engineered. Designing a solution to a difficult or ill-posed problem is a kind of reinforcement learning undertaken by humans: make a bunch of choices (feature space, clustering algorithm, learning architecture, etc.), get a reward (either the system works in practice or it does not), and iterate, given this new experience. Such a costly and error-prone procedure is ripe for streamlining and automation.

Appendix A

Segmentation ground-truth

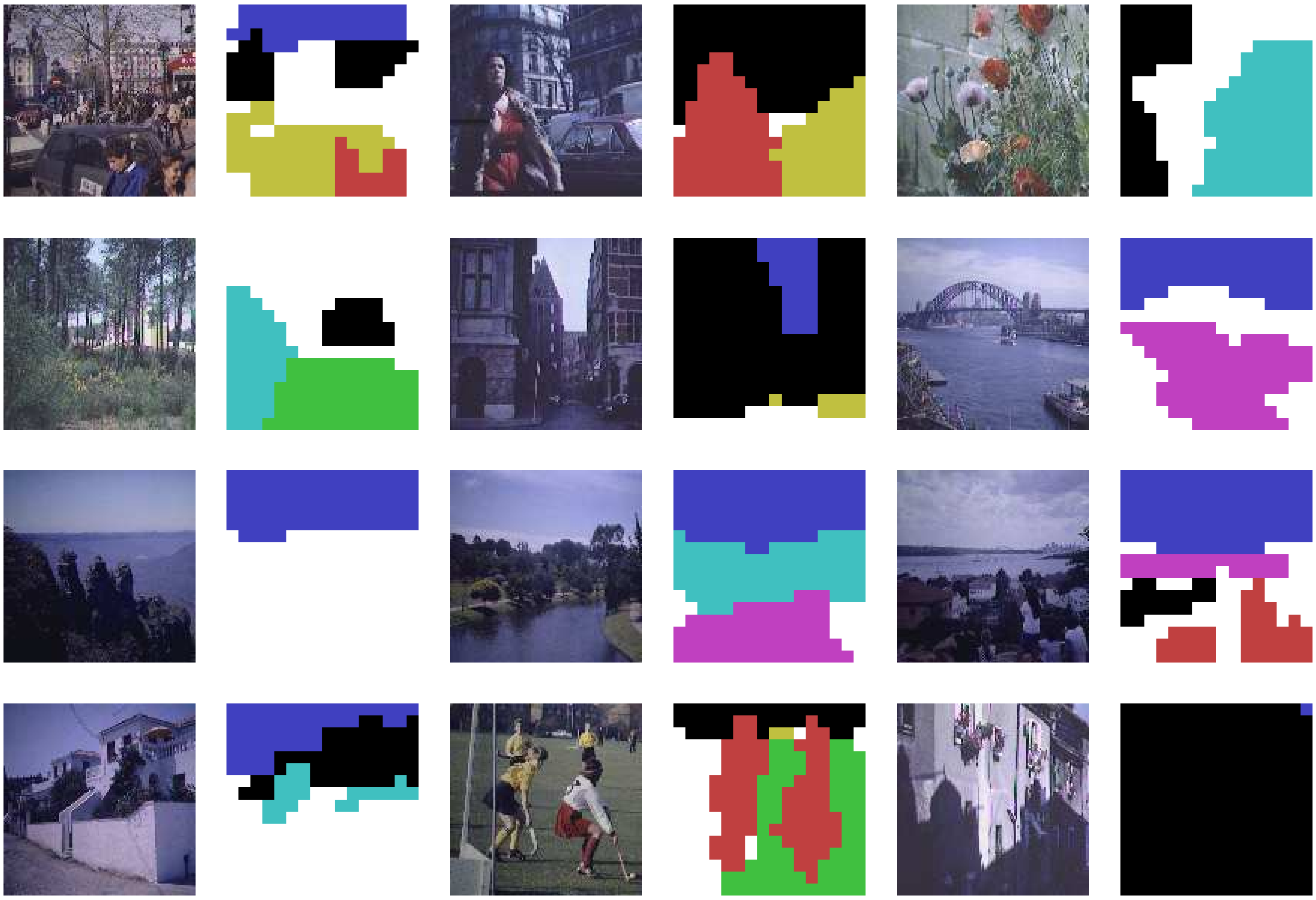


Figure A.1: The first twelve natural scenes and their ground-truth labelings. Regions labeled "building" are colored black; "car" is yellow, "grass" is green, "leaves" is cyan, "person" is red, "sky" is blue, and "water" is purple. Unlabeled regions are white.

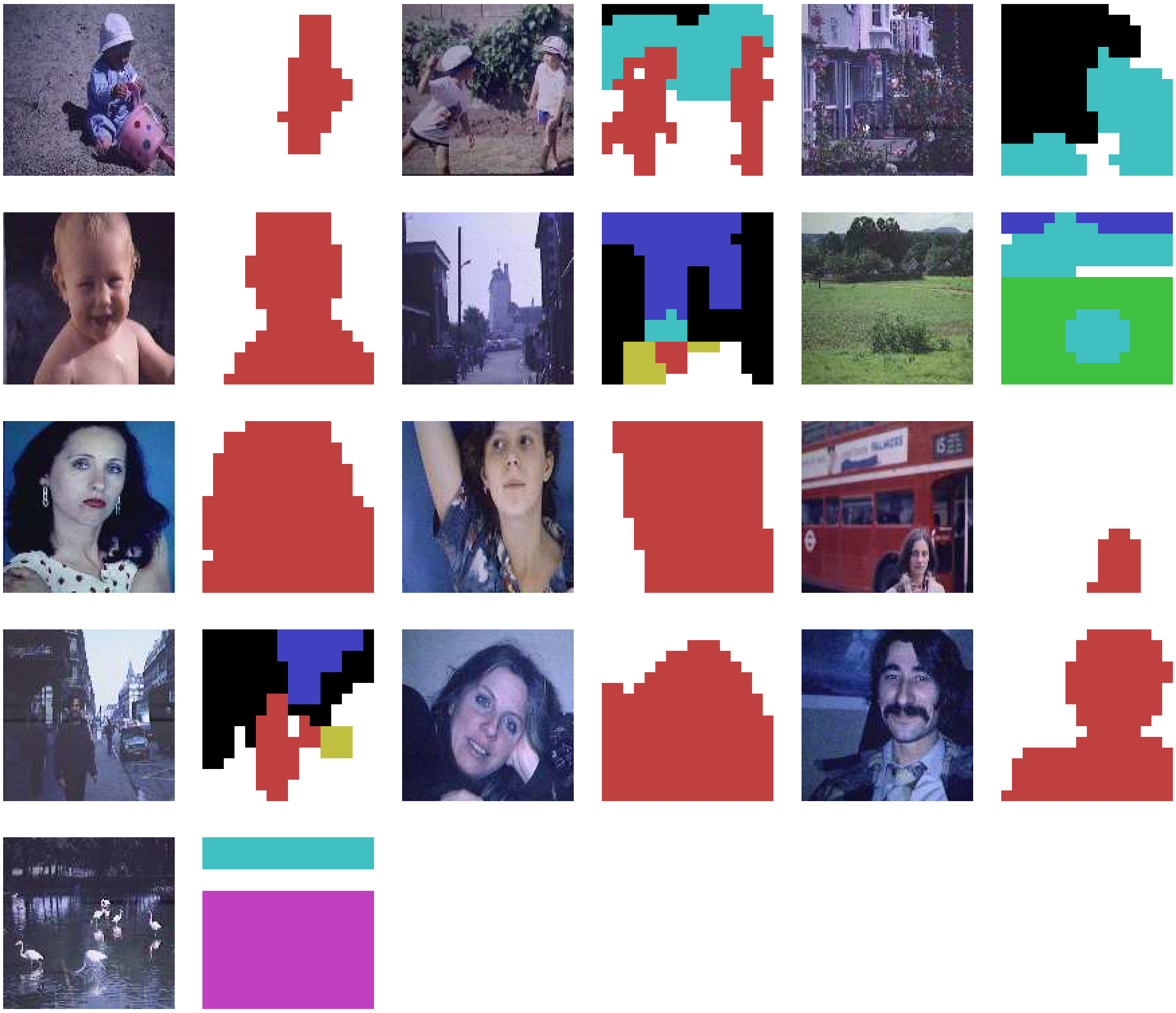


Figure A.2: The remaining thirteen natural scenes and their ground-truth labelings.

Bibliography

[1] B. Allen and I. Munro. Self-organizing search trees. *Journal ACM*, 25:526–535, 1978.

[2] Bir Bhanu and Sungkee Lee. *Genetic learning for adaptive image segmentation.* Kluwer Academic Publishers, 1994.

[3] A. Bookstein. The generalized retrieval problem. In C. J. Crouch, editor, *Proc. 4th Int'l Conf on Info. Stor. and Retr.*, volume 16, pages 4–14. ACM-SIGIR, Summer 1981.

[4] L. Breiman, J. H. Freidman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

[5] P. Brodatz. *Textures: A Photographic Album for Artists and Designers.* Dover, New York, 1966.

[6] G. Carpenter and S. Grossberg. ART2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26(23):4919–4946, 1987.

[7] T. Chang and C. C. J. Kuo. Texture analysis and classification with tree-structured wavelet transform. Technical Report USC-SIPI198, University of Southern California, Los Angeles, CA, February 1992.

[8] Allen Cypher, editor. *Watch what I do: programming by demonstration.* MIT Press, Cambridge, MA, 1993.

[9] Jon Doyle. *A model for deliberation, action, and introspection.* PhD thesis, Massachusetts Institute of Technology, 1980. Also MIT Artificial Intelligence Laboratory Memo AIM-TR-581.

[10] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley-Interscience, 1973.

[11] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[12] D. L. Hall. *Mathematical Techniques in Multi-Sensor Data Fusion.* Artech House, 1992.

[13] C. J. Harris, editor. *Advances in Intelligent Control.* Taylor & Francis, 1994.

[14] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

[15] Earl B. Hunt, Janet Marin, and Philip J. Stone. *Experiments in Induction.* Academic Press, 1966.

[16] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, NJ, 1988.

[17] R. A. Jarvis and E. A. Patrick. Clustering using a similarity measure based on shared near neighbors. *IEEE T. Comp.*, pages 1025–1034, Nov. 1973.

[18] K. Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[19] Pentti Kanerva. *Sparse distributed memory.* MIT Press, Cambridge, MA, 1988.

[20] T. Kohonen. *Self-Organization and Associative Memory.* Springer, Berlin, Heidelberg, 1984. 3rd ed. 1989.

[21] P. Langley. Rediscovering physics with Bacon. In *Proc. Fifth International Joint Conference on Artificial Intelligence*, pages 505–507, 1977.

[22] D. B. Lenat. AM: Discovery in mathematics as heuristic search. In R. Davis and D. B. Lenat, editors, *Knowledge-Based Systems in Artificial Intelligence*, pages 3–228. McGraw-Hill, 1980.

[23] D. B. Lenat, F. Hayes-Roth, and P. Klahr. Cognitive economy in artificial intelligence systems. *IJCAI*, 6:531–536, 1979.

[24] F. Liu and R. W. Picard. Periodicity, directionality, and randomness: Wold features for image modeling and retrieval. *IEEE T. Patt. Analy. and Mach. Intell.*, To appear. Also MIT Media Laboratory Perceptual Computing TR#320.

[25] D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992.

[26] R. Maclin and J. W. Shavlik. Refining alogrithms with knowledge-based neural networks: Improving the Chou-Fasman algorithm for protein folding. *Machine Learning*, 11:195–215, 1993.

[27] J. Mao and A. K. Jain. Texture classification and segmentation using multiresolution simultaneous autoregressive models. *Patt. Rec.*, 25(2):173–188, 1992.

[28] R. S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20(2):111–161, 1983.

[29] R. S. Michalski. Inferential theory of learning as a conceptual basis for multistrategy learning. *Machine Learning*, 11:111–151, 1993.

[30] R. S. Michalski, editor. *Multistrategy Learning.* Kluwer Academic Publishers, 1993.

[31] R. S. Michalski and R. E. Stepp. Automated construction of classifications: Conceptual clustering versus numerical taxonomy. *IEEE T. Patt. Analy. and Mach. Intell.*, PAMI-5(4):396–409, July 1983.

[32] T. P. Minka and R. W. Picard. Interactive learning using a 'society of models'. *Submitted for Publication*, 1995. Also appears as MIT Media Lab Perceptual Computing TR#349.

[33] T. M. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, Rutgers University, May 1980.

[34] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, and C. Faloutsos. The QBIC project: Querying images by content using color, texture, and shape. In W. Niblack, editor, *IS & T Storage and Retrieval for Image and Video Databases*, volume 1908, Feb. 1993. IBM TechReport RJ9203 (815111), Feb. 1993.

[35] Y-I Ohta, T. Kanade, and T. Sakai. Color information for region segmentation. *Comp. Graph. and Img. Proc.*, 13:222–241, 1980.

[36] G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990.

[37] M. Pavel, M. A. Gluck, and V. Henkle. Constraints on adaptive networks for modeling human generalization. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1, pages 2–10, 1988.

[38] M. J. Pazzani. Learning causal patterns: Making a transition from data-driven to theory-driven learning. *Machine Learning*, 11:173–194, 1993.

[39] A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Tools for content-based manipulation of image databases. In *SPIE Storage and Retrieval of Image & Video Databases II*, pages 34–47, San Jose, CA, Feb. 1994.

[40] R. W. Picard. Toward a visual thesaurus. In *Springer-Verlag Workshops in Computing*, 1995. To appear; Also appears as MIT Media Lab Perceptual Computing TR #358.

[41] R. W. Picard. Computer learning of subjectivity. *ACM Computing Surveys*, 1996. To appear; Also appears as MIT Media Lab Perceptual Computing TR #359.

[42] R. W. Picard and T. Kabir. Finding similar patterns in large image databases. In *Proc. IEEE Conf. on Acoustics, Speech, and Signal Proc.*, pages V–161–V–164, Minneapolis, MN, 1993.

[43] R. W. Picard, T. Kabir, and F. Liu. Real-time recognition with the entire Brodatz texture database. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 638–639, New York, June 1993.

[44] L. Y. Pratt. Experiments on the transfer of knowledge between neural networks. In S. J. Hanson, G. A. Drastal, and R. L. Rivest, editors, *Computational Learning Theory and Natural Learning Systems*, volume 1, pages 523–560. MIT Press, Cambridge, MA, 1994.

[45] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[46] L. Rendell and R. Seshu. Learning hard concepts through constructive induction: Framework and rationale. In S. J. Hanson, G. A. Drastal, and R. L. Rivest, editors, *Computational Learning Theory and Natural Learning Systems*, volume 1, pages 83–142. MIT Press, Cambridge, MA, 1994.

[47] R. Rivest. Learning decision lists. *Machine Learning*, 2:229–246, 1987.

[48] R. L. Rivest. On self-organizing sequential search heuristics. *Comm. ACM*, 19:63–67, 1976.

[49] D. Romer. The Kodak picture exchange, April 1995. seminar at MIT Media Lab.

[50] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(9):533–536, October 1986.

[51] E. Saber, A. M. Tekalp, R. Eschbach, and K. Knox. Annotation of natural scenes using adaptive color segmentation. *IS&T/SPIE Electronic Imaging*, Feb. 1995. San Jose, CA.

[52] G. Salton. *Automatic Information Organization and Retrieval.* McGraw-Hill, New York, 1968.

[53] E. Saund and T. P. Moran. Perceptual organization in an interactive sketch editing application. In *Proc. Fifth International Conference on Computer Vision*, pages 597–604, Cambridge, MA, June 1995.

[54] T. M. Strat and M. A. Fischler. Context-based vision: Recognizing objects using information from both 2-d and 3-d imagery. *IEEE T. Patt. Analy. and Mach. Intell.*, 13(10):1050–1065, Oct. 1991.

[55] Richard S. Sutton, editor. *Reinforcement learning.* Kluwer Academic Publishers, 1992. Reprinted from Machine Learning vol. 8, nos. 3–4.

[56] C. W. Therrien. *Decision Estimation and Classification.* John Wiley and Sons, Inc., New York, 1989.

[57] M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, Maui, HI, June 1991.

[58] P. Utgoff. *Machine Learning of Inductive Bias.* Kluwer Academic Publishers, Boston, MA, 1986.

[59] V. Vapnik. Principles of risk minimization for learning theory. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 831–838, 1992.

[60] C. J. C. H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3–4):279–292, 1991.

[61] J. K. Wu, A. D. Narasimhalu, B. M. Mehtre, C. P. Lam, and Y. J. Gao. CORE: a content-based retrieval engine for multimedia information systems. *Multimedia Systems*, 2:25–41, Feb. 1995.

[62] H.-J. Zhang and S. W. Smoliar. Developing power tools for video indexing and retrieval. In W. Niblack and R. C. Jain, editors, *Proceedings SPIE Storage and Retrieval for Image and Video Databases II*, pages 140–149, San Jose, CA, Feb. 1994. SPIE. Vol. 2185.

[63] S. C. Zhu, T. S. Lee, and A. L. Yuille. Region competition: Unifying snakes, region growing, energy/Bayes/MDL for multi-band image segmentation. In *Int. Conf. on Computer Vision*, pages 416–423, Boston, MA, 1995.