Guided Interactive Machine Learning

Aaron J. Pace
Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Computer Sciences Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/477

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
GUIDED INTERACTIVE MACHINE LEARNING

by

Aaron Pace

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science
Brigham Young University
August 2006
This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date  
Dan R. Olsen, Chair

Date  
Eric K. Ringger

Date  
Tony R. Martinez
As chair of the candidate’s graduate committee, I have read the thesis of Aaron Pace in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Dan R. Olsen
Chair, Graduate Committee

Accepted for the Department

Date

Parris K. Egbert
Graduate Coordinator

Accepted for the College

Date

Thomas W. Sederberg
Associate Dean
College of Physical and Mathematical Sciences
ABSTRACT

GUIDED INTERACTIVE MACHINE LEARNING

Aaron Pace
Department of Computer Science
Master of Science

This thesis describes a combination of two current areas of research: the Crayons image classifier system and active learning. Currently Crayons provides no guidance to the user in what pixels should be labeled or when the task is complete. This work focuses on two main areas: 1) active learning for user guidance, and 2) accuracy estimation as a measure of completion. First, I provide a study through simulation and user experiments of seven active learning techniques as they relate to Crayons. Three of these techniques were specifically designed for use in Crayons. These three perform comparably to the others and are much less computationally intensive. A new widget is proposed for use in the Crayons environment giving an overview of the system “confusion”. Second, I give a comparison of four accuracy estimation techniques relating to true accuracy and for use as a completion estimate. I show how three traditional accuracy estimation techniques are ineffective when placed in the Crayons environment. The fourth technique uses the same computation as the three new active learning techniques proposed in this work and thus requires little extra computation and outstrips the other three as a completion estimate both in simulation and user experiments.
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Crayons</td>
<td>7</td>
</tr>
<tr>
<td>2.1</td>
<td>Traditional vs Interactive Machine Learning</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Iterative Correction Process</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Active Learning</td>
<td>11</td>
</tr>
<tr>
<td>3.1</td>
<td>Prior Work</td>
<td>12</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Algorithm-specific Active Learning</td>
<td>12</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Example Generation Active Learning</td>
<td>14</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Pool-based Active Learning</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>Methods</td>
<td>18</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Distribution-Based Active Learning</td>
<td>18</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Confusion-Based Active Learning</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>Active Learning Simulations</td>
<td>32</td>
</tr>
<tr>
<td>4.1</td>
<td>Simulation Implementation</td>
<td>33</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Oracles</td>
<td>33</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Object Generators</td>
<td>35</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Simulation Methodology</td>
<td>37</td>
</tr>
<tr>
<td>4.2</td>
<td>Simulation Results</td>
<td>38</td>
</tr>
<tr>
<td>4.3</td>
<td>Simulation Analysis</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>Active Learning Experiments</td>
<td>44</td>
</tr>
<tr>
<td>5.1</td>
<td>Experiment Implementation</td>
<td>45</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Freeform Labeling</td>
<td>45</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Iterative Correction</td>
<td>46</td>
</tr>
<tr>
<td>5.1.3</td>
<td>ICC</td>
<td>46</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Comparison Metrics and Experimental Tasks</td>
<td>50</td>
</tr>
<tr>
<td>5.2</td>
<td>Experimental Results</td>
<td>53</td>
</tr>
<tr>
<td>6</td>
<td>Accuracy Estimation</td>
<td>55</td>
</tr>
<tr>
<td>6.1</td>
<td>Accuracy Estimation Techniques</td>
<td>55</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Training Set Accuracy</td>
<td>55</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Hold-Out Set Accuracy</td>
<td>56</td>
</tr>
<tr>
<td>6.1.3</td>
<td>K-Fold Cross-Validation Accuracy</td>
<td>56</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Incremental Consistency Accuracy</td>
<td>57</td>
</tr>
<tr>
<td>6.2</td>
<td>Accuracy Estimation Simulations</td>
<td>57</td>
</tr>
<tr>
<td>6.3</td>
<td>Accuracy Estimation Experiments</td>
<td>61</td>
</tr>
<tr>
<td>6.4</td>
<td>Accuracy Estimation Analysis</td>
<td>63</td>
</tr>
<tr>
<td>7</td>
<td>Completion Estimation</td>
<td>66</td>
</tr>
<tr>
<td>7.1</td>
<td>Estimation Techniques Comparison</td>
<td>66</td>
</tr>
<tr>
<td>8</td>
<td>Conclusion &amp; Future Work</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>69</td>
</tr>
</tbody>
</table>
Chapter 1 - Introduction

The use of sensor-based inputs such as cameras and microphones has brought exciting new possibilities to user interfaces, but presents interface designers with some unique difficulties. To use a camera, for example, as an interactive input, would require the interface designer to have extensive knowledge of image processing and machine learning. Further complicating the process is the massive amount of time required to create a decent image classifier using traditional methods, typically in the range of days, weeks, or even months to sift through the features and fine-tune the learning algorithm’s parameters for a given learning task.

The key elements limiting the use of machine learning in user interfaces are: 1) the technical knowledge requirements, and 2) the time. As we mentioned earlier, to create a user interface for image classification, the designer would need extensive knowledge of both image processing and machine learning. Most interface designers don’t have expertise in either field. The Crayons interface [2] shown in Figure 1 provides an abstraction of the complexities of image processing and machine learning by providing the means for labeling training examples visually and using a pre-selected set
of features. Thus, Crayons significantly reduces the knowledge required to use machine learning in a user interface, thus handling the first key issue.

To reduce the time required to create a good image classifier, we need to determine the time-limiting characteristics of traditional machine learning. Four important areas merit consideration: 1) picking features which are relevant to the task, 2) choosing an appropriate learning algorithm and tweaking the algorithm’s parameters for a particular task, 3) finding appropriate examples for the particular task, and 4) training the learning algorithm. The Crayons interface addresses issues 1, 2, and 4 but finding the best examples to use in training the classifier is not addressed. The key problem with selecting the best examples is that the user must select among the thousands or even millions of examples those that will be most beneficial in training the classifier. This is a verbut user delay (problem number 5) remains the biggest bottleneck of the system.

To understand the example selection problem, let’s examine the task shown in Figure . This task is to teach the computer the difference between skin and not skin. In one extreme case the user could color every single pixel either skin or not skin. For an 800x600 image, this process would generate 480,000 training examples (one for each pixel). If the task had 30 800x600 images associated with it, coloring every single pixel would generate 14,400,000 training examples. Coloring every pixel of every image would likely take several days to complete and most training algorithms do not handle more than a few thousand training examples in interactive time1, let alone 14 million. Another extreme case is one which the user only colors a single pixel skin and a single pixel not skin. The time invested by the user is extremely small, but the learning

---

1 For our purposes we will define interactive time to be less than a few seconds. If the user must wait longer than 2-3 seconds for the image classification, he quickly becomes frustrated.
algorithm would likely not classify the 14,399,998 uncolored pixels with any degree of accuracy. The problem is, then, between these two extremes, how the user will know which pixels to color and how many pixels to color. Traditional Crayons [2] assumes the user will make intelligent decisions regarding which pixels to color as well as how many pixels to color, but we would like to explore just how much faster we can achieve an accurate classifier by directing the user to specific pixels in any given image.

Consider again the skin classification task shown in Figure 1. This task is composed of a set of 20 images. Our observations of users of the Crayons interface shows that as he begins his classification task, the user spends too much time on a single image, coloring a good portion of the image before switching to the next image. After the user has seen all 20 images, the classification task has easily gathered around 100,000 training examples for the algorithm to sift through and the user has spent around 30-40 minutes coloring these images. Clearly 100,000 training examples won’t be too difficult for many training algorithms to deal with in interactive time and 40 minutes isn’t a big deal, but if we extend the task to include 100 images, the algorithm is now faced with around 500,000 training examples (which is too many for most algorithms to handle in interactive time) and the user has spent around four hours coloring.

We pose an important question: how many of that half million training examples were actually necessary to get the same end result? Suppose that only 20,000 of those 500,000 examples were necessary for the skin classification problem. The user wasted most of his time coloring 25 times the number of pixels needed when he could have used that time to do something more productive. We need to consider two different approaches to figuring out which examples are the most important to the learning
algorithm: 1) pruning, and 2) prescreening. What we mean by pruning is that after all 500,000 examples have been labeled we prune the list to include only those that are necessary for correct classification of the unknown pixels. There are many techniques for pruning. Support Vector Machines [8], for example, are essentially just a pruning algorithm figuring out which examples are the most important in separating the different classes. While pruning may be a good technique from a hindsight perspective, we want to find a mechanism to not only distinguish the most important examples, but to do so without labeling more than we need because labeling from a user’s perspective is very time consuming.

Prescreening, like pruning, attempts to rate examples based on their relative importance to the classification problem at hand. Unlike pruning, however, prescreening attempts to decide training example importance during the coloring process rather than after the coloring has been completed. The machine learning community refers to this as active learning [19] because we are actively selecting training examples rather than passively accepting any example the user decides. Our goal is to find an active learning

![Typical Learning Curve](image)

**Figure 2**
Typical Crayons Learning Curve
technique that will be appropriate for use in the Crayons environment.

Let’s return to our problem where the user colors 500,000 training examples. Suppose that only 20,000 were actually important to the learner. Using active learning, we aim to direct the user to the best 20,000 pixels first so that the user doesn’t need to label the other 480,000. Labeling these “best” examples first, we are able to save the user valuable time.

The next important issue we face is knowing when the user has colored enough. Perhaps the user will simply continue to color after the best 20,000 until he has colored 500,000 pixels again even though only the first 20,000 were needed! We’ll refer to the point where the user’s task is complete as the learning plateau since further training will be less effective or sometimes even detrimental. The concept of the learning plateau is illustrated in the learning curve shown in Figure 2. As more training examples are given, the classifier becomes more accurate until we see the curve flatten out, after which the accuracy doesn’t increase substantially. The idea is that the user can stop coloring after seeing this plateau in accuracy. Unfortunately we generally don’t have a way to compute a classifier’s true accuracy but we can estimate the accuracy with techniques we will discuss later on. If we give the user an idea of when he has completed we will be able to save both time and effort on the user’s part.

Imagine the skin classification task we had previously with 100 images, except now we use active learning methods and task completion estimation. The user begins coloring on whatever image he would like. After coloring for some short amount of time, the active learning portion of the process begins to suggest sections of an image to direct the user’s focus (see Figure 3). Throughout the whole training period, the accuracy
estimator provides the user with feedback as to how well the computer is learning its task. Once the completion estimator gives the signal that the task is finished, the user stops coloring. The learning task has then been completed in a fraction of the time required previously.

The rest of the paper will be organized as follows. We will discuss the Crayons interface in Chapter 2 which will form the basis for our study. In Chapter 3 we will discuss active learning in detail and give a few methods for providing active learning in Crayons. We discuss the active learning experiments and results in Chapters 4 and 5. Chapter 6 will focus on accuracy estimation, an important part of completion estimation. Completion estimation will be covered in Chapter 7. We conclude in Chapter 8.
Chapter 2 - Crayons

The platform we will be working with is called Crayons and was developed by Fails & Olsen [2]. In this chapter we will give the reader an insight into why Crayons is beneficial. For a better explanation of Crayons and its background, we refer the interested reader to the Crayons paper [2]. We begin our discussion of Crayons with a comparison of the Crayons machine learning sequence we call Interactive Machine Learning to traditional machine learning. We then shift our focus to an observation of user behavior in Crayons because it will give us a key insight into possible active learning techniques.

2.1 Traditional vs Interactive Machine Learning

Machine learning involves two pieces of important information: 1) the training examples\(^2\), and 2) the learning algorithm. There are many learning algorithms and combinations of learning algorithms today such as Neural Networks [3], Nearest Neighbor [4], Boosting [5], Decision Trees [6], Naïve Bayes [7], Support Vector Machines [8], and many more. The Crayons interface makes use of the Decision Tree learning algorithm used by Fails [1] with a few slight modifications. We will simply accept the default Crayons implementation of Decision Trees in our study because it has been shown to perform well in the interactive environment that Crayons

\[ \text{Feature Engineering} \rightarrow \text{Train} \rightarrow \text{Classify} \]

**Figure 4**

Traditional Machine Learning Sequence

Note: Feature Engineering refers to a person manually selecting features that will be pertinent to the task. For example, for an image, perhaps the red value of a pixel is important to the pixel’s classification, so the red value will be one of the values in the feature vector.

\(^2\) For our work we define a training example as a set of feature values (a feature vector to the machine learning community) and a label (the classification of the feature vector).
provides.

Traditional Machine Learning algorithms and techniques generally follow the sequence depicted in Figure 4. In analyzing this sequence for use in an interactive environment, we find that there are two major drawbacks. The first problem is that this sequence requires the user to have an extensive knowledge of both image processing and machine learning to be able to choose features intelligently. The second problem is that training and classification can be (and often is too slow to be interactive).

The Interactive Machine Learning (IML) sequence explained in [1] is shown in Figure 5. Notice that the Feature Engineering step of the sequence has been abstracted away (represented by the grayed-out Feature Engineering bubble) and a new bubble has been added creating what we call the user-interactive loop. We will discuss this interactive loop in more detail in the next section. In order to abstract away the Feature Engineering step, Crayons has pre-selected a set of around 250 features\(^3\). Only a subset of these features will be useful in any given learning task, but Crayons relies on the ability of the training algorithm to select only those features that are most important for the particular task (known as automatic feature selection to the machine learning community). The extremely slow training and classification times exhibited by many traditional machine learning algorithms in an environment with 250 continuous-valued features and thousands of

\[^3\] The original Crayons interface had around 150 features [1] but that number has increased since Fails first presented it in ‘03. The features Fails used were continuous-valued and typically ranged from 0 to 255. An alternative would be to have discrete-valued features in that same range. For our experiments we will use the continuous-valued features already implemented in Crayons.
training examples prohibit their use in such a sequence. Fails, however, shows that a variant of the Decision Tree (DT) algorithm [1] selects important features, trains, and classifies all in a matter of seconds. Because Fails used DT in his Crayons studies, we will embrace it in our work as well.

The second problem that we are faced with in the Traditional Machine Learning sequence is the slow process of training the learning algorithm since most of the algorithms presented earlier are too slow to be interactive. Fails [1] presents two versions of the Decision Tree algorithm that perform their task of feature selection and training in a relatively short amount of time. By adding the user interactive loop, Fails has shown that accurate image classifiers can be created in a matter of seconds for single images. The problem with this simple sequence is that on tasks with hundreds of images, the time required to create an adequate classifier is significant (on the order of hours or days depending on the task). Our goal is to test the Interactive Machine Learning sequence on projects of 20-30 images, and to tighten the user-interactive loop by guiding the user to specific portions of the images to further improve performance and minimize user time.

Figure 6
Interactive Machine Learning
The user colors on an image (b), gets feedback (c), corrects (d), and repeats the process until the image is classified well (g).
2.2 Iterative Correction Process

Let’s consider the user-interactive loop we introduced in Figure 5. To illustrate this process, imagine the task of classifying sky shown in Figure 6(a). As the user colors (b) the system gives feedback periodically. This feedback is in the form of a semi-transparent layer over the top of the image with each pixel classified either sky or not sky (c). This loop continues (color pixels, wait for feedback, color more) until the user is satisfied that the system correctly classifies the entire image (Figures 6(d-g)).

At this point we observe an interesting trend. Take a look at Figure 7 showing an enlarged version of Figures 6(c) and (d). In this figure we notice that the user has colored the upper portion of the image (the stripe circled in red) which was previously classified incorrectly. We call this kind of behavior corrective. Had the user colored more pixels in the correctly classified regions, this would be a reinforcing behavior. For example, had the user colored more pixels on the mountain (which are all classified correctly right now), this behavior would be reinforcing rather than corrective. The corrective behavior we observe in Figure 7 is typical of most Crayons users [1]. Whether this behavior is beneficial or not to the Crayons system is not our focus, but this corrective style of input will provide a basis for some of the active learning techniques we present in this paper. For more details on corrective input, see Fails’ paper on the Crayons environment [1].
Chapter 3 - Active Learning

The field of active learning has had a growing audience in the machine learning community during the last several years. The idea behind active learning is that out of all the training examples that could possibly be presented to the learning algorithm, we would like to select only those that would be the most beneficial. By beneficial we mean those examples that provide the best classifier with the least number of training examples. By choosing these examples, we reduce complexity, training time, and classification time. Shohn and Cohn [10] observe that in certain cases a classifier trained on a well-chosen subset of the training examples can actually perform better than one which has been trained on all possible examples. Training and classification times are critical bottlenecks in the Crayons environment.

Suppose we are faced with the challenge of training a robot to recognize sidewalk so that it can direct itself and not wander into the street. We first gather a set of, say, 100 images which may be typical for the robot to view with its onboard camera. Next we import those images into Crayons and begin to label pixels as either sidewalk or not sidewalk (Figure 8). We are faced with a very important issue during this process: what subset of the millions of pixels gives the best classifier in the least amount of time? Labeling every pixel of all images is impractical since it would take around an hour per image to label and as we pointed out earlier, most training algorithms won’t be able to train or classify in a reasonable amount of time for

![Figure 8](image_url)

Sidewalk Classification Task sample image.
these millions of training examples. So the issue is: which pixels should be labeled? This question is the driving force behind active learning.

In this chapter we will discuss a few of the current active learning techniques, followed by the techniques we will consider in the Crayons environment.

### 3.1 Prior Work

Machine learning is composed of two separate entities: the learning algorithm and the examples from which the algorithm generates a learned model. Active learning uses one or both of these entities to select the “best” unlabeled examples for labeling. We separate active learning techniques into three categories based on how the technique uses the two entities mentioned above: algorithm-specific, example generation, and pool-based learning.

#### 3.1.1 Algorithm-specific Active Learning

The first of these active learning categories, algorithm-specific, uses information about the specific learning algorithm to determine beneficial examples. For example, suppose we use linear regression in the support vector machine (SVM) model as the learning algorithm for the problem in Figure 9(a). This figure shows the world of possible training examples

---

4 The figures we use to illustrate the various active learning techniques show a simple dividing line as the decision surface. We give these illustrations for simplicity. Decision trees are the focus of our study and generally produce decision surfaces with much more complexity than a simple dividing line or hyperplane.
with a clear separation of the two classes. Suppose that the user labels the set of red highlighted examples. A linear SVM will attempt to separate the x’s from the o’s by placing the decision surface highlighted in red of Figure 9(b) while the correct surface is the dashed line. Notice how different the two lines are. One active learning technique would be to request the label of those examples closest to the estimated division, those highlighted in blue in Figure 9(c). With these examples labeled, the learner comes very close to the true decision surface (Figure 9(d)). This algorithm-specific form of active learning has perhaps received the most attention from the machine learning community. We will discuss specifically two examples of active learning dealing with support vector machines below.

Shohn and Cohn used the SVM dividing hyperplane to select examples. Their hypothesis was that selecting examples near the hyperplane would maximize the difference between the current dividing hyperplane and the hyperplane of the new SVM (the SVM trained on the old set of examples plus the newly selected examples). In their experiments, this technique proved quite promising.

Tong and Koller presented a slightly different idea for active learning using SVMs [15]. Their idea was to reduce the version space (the set of all hypotheses consistent with the already labeled data) as quickly as possible. One way of maximizing the reduction of the version space is to literally compute the size of the version space for each potential example. Computationally, this isn’t practical, so Tong and Koller present three techniques for approximating the size of the version space if an example is chosen for labeling. These three are simple margin, MaxMin margin, and ratio margin. Simple margin selects the example whose dividing hyperplane is closest to the SVM unit vector.
MaxMin margin creates two SVM classifiers for each potential training example (one assuming a positive label and the other assuming a negative label) and selects the example whose classifiers are most similar in size. Ratio margin is similar to MaxMin margin in that it creates two new SVM classifiers, but chooses the example whose ratio of hypersphere radii (which correspond to the SVM margins) is the largest.

For our study, we could use one of these approaches or any other algorithm-specific technique. On the other hand, Crayons was developed for use with any learning algorithm, so we would like to introduce an active learning technique into Crayons which will be more independent of the algorithm. Thus we will not explore this category of active learning further.

### 3.1.2 Example Generation Active Learning

The second category we investigate is example generation. The basis for example generation is that the active learning algorithm can arbitrarily generate an example for the

![Figure 10](image)

The user has labeled some initial data (a), the estimated decision surface is shown in red, the true decision surface is represented by the dashed line. In (b) the active learner queries for the label of some specific locations, which the user labels in (c). The new estimated decision surface (the blue line) in (d) is much closer to the true decision surface than the original estimate (the red line).

---

5 Size being the volume of the hypersphere defined by the SVM. This hypersphere is not to be confused with the dividing hyperplane that Tong and Koller use as the SVM. The hypersphere used in their paper is the means for comparing different SVMs. Taking from their paper: “The center of the [hypersphere] corresponds to the SVM, its radius is proportional to the margin of the SVM in [the feature space].” For more detail, see Tong and Koller’s paper [15].
user to label. For example, in a 2D plane, the example-generation active learner can generate any x-y pair for labeling. For example, Figure 10 shows a process of example generation and labeling for several rounds along with the true decision surface. As the active learning algorithm generates examples for labeling, the decision surface approaches the true surface.

Perhaps the most common example-generation active learning technique is query-by-committee [5]. The query-by-committee algorithm begins with a set of classifiers. The active learning step randomly picks two of the classifiers and generates examples until the two classifiers disagree on the example’s classification. The user then labels this example and all classifiers that don’t agree with the label are discarded.

In Crayons, using example generation would mean that the algorithm automatically generates pixels for the user to label rather than selecting pixels from an image. Two of the problems with this technique are: 1) that single pixels generally don’t mean much to a user, and 2) labeling a single pixel is highly inefficient when thousands of pixels are readily available and rapidly labeled in Crayons. For example, Figure 11(a) shows a single randomly generated pixel. Is this pixel skin, or not skin? Without the context of the surrounding pixels, it is

(a) A single pixel for classification. Is this pixel skin or is it not? It’s hard to tell. The pixels from the image in (b), however, are much easier to see which are skin pixels and which are not.
practically impossible to know if this pixel is supposed to be skin or not. The pixels from the image in (b) are much easier for a user to classify since we have the pixel context in addition to the pixel. Because of this problem of lack of context in example generation active learning, this is another active learning technique we will not explore.

### 3.1.3 Pool-based Active Learning

The final category of active learning we will discuss is pool-based learning. Pool-based learning is used in situations where the world of data the algorithm can use is limited to a fixed set. In this case generating random examples isn’t practical or even possible. For example, the picture of Figure 11(b) is comprised of 480x360 pixels, all of which may be used as training examples. As we discussed earlier, a single pixel randomly generated will be quite difficult for a person to label. Figure 11 illustrates this point.

One example of pool-based active learning is to use a committee. A committee is a group of classifiers that work together to produce a single classification. The idea behind a committee, presented by Freund [5], is that a committee of classifiers can often outperform a single classifier. For example, imagine a patient walking into a doctor’s office for a medical diagnosis. Once in the exam room, the patient is greeted by five doctors who begin taking measurements and asking him questions. Once they have collected all the data about the condition, the doctors confer about the possible diagnoses. Each doctor votes on what he feels is the most probable diagnosis and the diagnosis with the most votes is the conclusion presented to the patient. In this example, the doctors are acting as a committee to combine their knowledge and experience for the benefit of the patient.
Now imagine that instead of doctors, we have computers diagnosing a patient’s condition. After the computers place their vote as to what the diagnosis should be, a real doctor observing notices that not all the computers agree. In this case, the doctor diagnoses the condition and gives the computers the real diagnosis to add to their body of knowledge. The steps we performed in this simple example are 1) have several models vote on the classification of some example; 2) if the classifiers all agree, the classification is likely correct; 3) if the classifiers do not agree, then providing the label for the example will be beneficial in synchronizing the classifiers for better classification. Much work has been done in the area of committee-based learning. Sarawagi [14] as well as Freund [5] give techniques for generating effective classifier committees, such as randomizing model parameters, partitioning training data, and partitioning attributes. We will discuss committee classifiers further in the next section.

Another pool-based active learning technique is pre-clustering [15]. Pre-clustering is the term Tong, et al. defined as grouping similar examples from the pool into clusters before beginning the training. While training, only a representative sample from each cluster is used to train the learner. They showed that pre-clustering is beneficial in text classification with an initial computation overhead to compute the cluster representatives. Unfortunately this overhead can be quite large and thus limits its usability in an interactive environment such as Crayons and thus we will not discuss it further.
3.2 Methods

In our work we will focus on two groups of active learning techniques: distribution-based, and confusion-based. We will discuss each of these in detail in the sections that follow.

3.2.1 Distribution-Based Active Learning

A technique which hasn’t received attention in the machine learning community is a method we’ll call distribution-based active learning. In this technique the algorithm selects examples for which the resulting distribution, the \textit{training distribution}, of labeled examples resembles some predetermined distribution, the \textit{desired distribution}. Suppose the user labels a set of examples which all fall into the same area of the feature space. Because the user is so focused on a particular type of example, we would like to nudge him into another area of the feature space to get a better sampling of the world.

Imagine that we are trying to match the uniform distribution with a set of training examples (which currently give a Gaussian-shaped distribution). We now need a mechanism for ordering the possible examples so we know which examples will push our Gaussian distribution to the uniform distribution. We could add each example individually to the set of training examples and give the resulting distribution a score as to how well it fits the desired uniform distribution. This would be extremely time-consuming because we would need to calculate a new distribution for each of the thousands of training examples.

Rather than actually adding each example to the training set individually and calculating the resulting distribution, we propose a new technique to provide the same information in a much less computationally intensive way. This is possible by using
what we call difference histograms. A difference histogram is a representation of the difference\(^6\) between the training distribution (the distribution of the labeled examples) and the desired distribution (the distribution that the labeled examples’ distribution should resemble when finished labeling). Figure 12 provides a visual representation of the difference histogram concept. As is shown in the figure, we first discretize the training and desired distributions into a number of “bins”, thereby creating a histogram of each distribution. The desired distribution, shown in red at the back, is the uniform distribution in this case, and the training distribution, shown in yellow, is a Gaussian curve. Shown in the front is the difference histogram giving the difference between the two other distributions. Intuitively, this histogram shows us which examples will push the training distribution to the desired distribution. In this case, the examples with values close to the extremes are the examples that are the most desirable.

With the difference histogram we can then “score” each example based on where it fits into the histogram. If we have an example that falls into the center bin from Figure 12, it should

\[\text{Figure 12}
\]
Ten-bucket difference histogram with the training and desired distributions.

\(^6\) Another measure of the difference between two distributions is called the Kullback-Leibler divergence (discussed by Baker and McCallum [20]). If our goal was to quantify the difference between distributions, K-L divergence would be the correct method. However, since our goal is to evaluate examples with respect to the distributions, we instead use difference histograms.
get a very low score because we have so many examples that fall into that bin already, while the examples which fall into the outer bins should get very high scores since those examples are quite rare in our training data. If we then take the examples with the highest score we will push the training distribution towards the desired distribution.

We now define exactly how a set of examples generate this difference histogram. In a $d$-dimensional feature space, we generate $d$ difference histograms from the desired distribution (which is given to us) and the $d$ training distributions (one for each dimension of the feature space). For example, if we use the pixel’s red value (from 0-1), green value, and blue value as three features for the pixel, we generate three difference histograms, one for each of red, green, and blue. Suppose we are told that the desired distribution of examples is uniform. In this situation the algorithm would choose those pixels whose color is not represented as much as the other colors.

The basic algorithm for calculating the training distribution and the difference histograms is shown in Figure 13.

```
computeHistograms( labeled example set s, desired distribution histogram D )
    initialize training distribution histograms R to 0’s
    foreach example e in s
        for all feature values f in e
            b = bucket f falls into
            R[b]++
        foreach bucket b in R
            R[b] /= example_count  // normalize the buckets
            H[b] = D[b] - R[b]
    return H
```

**Figure 13**
The algorithm for computing the training distribution and difference histograms. $H$ represents the set of all difference histograms $H_i$, one for each feature.

Once the difference histograms have been calculated, we score each example based on the following formula.

\[
\sum_i H_i[\text{bucket}(f_i)]
\]
In this equation, \( f_i \) is the value of the example at feature \( i \). The \textit{bucket} function maps the feature value to a bucket of the difference histogram.

With this algorithm in mind, we shift our attention to two of the possible desired distributions: uniform, and example. The uniform distribution will, as we discussed earlier, provide a good sampling of the entire feature space. The example distribution is the distribution of examples we find in the world. For example, in a set of pictures where we are trying to classify sky such as in Figure 14, the distribution will be weighted heavily to the blue side of the spectrum. Thus, we would like to get examples that resemble this distribution. Theoretically, if we train on a set of examples whose training distribution resembles the example distribution, the classifier should then be able to classify the remaining unlabeled examples correctly.

Let’s look at an interesting example to compare the uniform distribution and a specific example distribution. The image in Figure 15 shows a picture of a laser spot on a white board. Suppose our task is to train Crayons to recognize the laser spot. Using active learning, the algorithm asks the user for the label of several of the pixels in the image (how the algorithm asks for pixel labels is a
topic we will discuss in Chapter 5). If the algorithm asked for a uniform sample of all pixel colors (red, blue, green, yellow, orange, etc) then most of the user’s time is wasted because only red, blue, and black pixels are pertinent to our task. This type of querying uses the uniform distribution as its underlying desired distribution.

Suppose now that the algorithm consistently asks for the pixel color which appears most in the image (blue in our case since the whiteboard appears blue in the image). In this case, the algorithm consistently asks for the label of blue pixels since blue pixels comprise 99% of the pixels in our task. It appears that neither method will produce a very good classifier.

![Figure 16](image)

**Figure 16**
Mixed Distribution is a Combination of Uniform and Example Distributions. The smiley operator is a min function.

To remedy this problem we propose a modified distribution we call the *mixed distribution*. The mixed distribution is given by the following formula: $mixed = \min(uniform, example)$. Figure 16 gives a visual representation of the mixed distribution. We combine the uniform distribution with the example distribution to give the mixed distribution. This has the effect of not focusing too much on examples that just aren’t important (like the yellow, green, and orange pixels for the whiteboard problem), which is the drawback of the uniform distribution, while not giving too much attention to the areas where we already have ample representation (like the blue pixels from the
whiteboard), which is the drawback of the example distribution. Using the mixed distribution the algorithm asks the user for the labels of several blue pixels as well as those for red and black pixels, giving a good classifier tweaked specifically for the current task.

### 3.2.2 Confusion-Based Active Learning

We now turn our attention to a different group of active learning techniques based on a measure of confusion between two classifiers. We begin our discussion on confusion-based active learning by defining a few terms to aid our understanding of the concepts involved in this group of active learning techniques. Confusion is the amount of disagreement between two or more classifiers. Suppose we have a set of classifiers $C$. We then classify each example from the world with all the classifiers in $C$. If all the classifiers agree on the example’s classification, we say that there is zero confusion among the classifiers. If the classifiers are equally divided on the example’s classification, we say that there is 100% confusion. A set of classifiers that give zero confusion for all possible examples is what we’ll call a stable committee. Our goal is to achieve a stable committee because theoretically this is an indication of a perfect committee classifier. This concept is illustrated in Figure 17 which presents several examples of two-classifier (and two-dimensional) confusion. The gray portion of the figures represents the areas where the two classifiers disagree, or the confusion areas. Figure 17(a) gives a representation of a zero-confusion set of classifiers (a stable committee), (b) represents 100% confusion, and (c) represents a typical confusion value somewhere between 0 and 100%.
Figure 17
Two-classifier confusion areas. (a) shows a zero-confusion committee (both classifiers classify exactly the same way). (b) shows 100% confusion (the classifiers classify exactly opposite). (c) shows typical confusion areas (the classifiers classify the same in the white areas but different in the gray areas).

With this idea of confusion, we score each example from the world based on the amount of confusion it causes among the classifiers in $C$. We then take the examples that give the highest score as the set of examples that should be added to the training set. This has the effect of minimizing the overall confusion among the classifiers, thereby approaching a stable committee. The algorithm for computing the confusion score is

```c
// Two-class (positive & negative) Confusion Score Algorithm
calculateConfusionScore( committee of classifiers $C$, example $E$ )
    initialize count := 0
    initialize members := number committee members in $C$
    foreach classifier $c$ in $C$
        classification = $c$.classify($E$)
        if classification is positive
            count++
        if count > members/2
            count = members – count
    return count/(members/2)
```

Figure 18
The two-class confusion score algorithm. The algorithm returns a number between 0 and 1 with 1 being maximum confusion and 0 being minimum confusion.
shown in Figure 18. In this algorithm, maximum confusion (1.0) will be when \( \text{count} = \frac{\text{members}}{2} \). This represents the case when half the committee members classify the example as one class and the other half classify the example as the other class. This is of course the situation where the committee is completely confused about the example’s classification. Minimum confusion is when \( \text{count} = 0 \), or in other words when all the committee members classify the example as a single class. This represents the situation where the committee is completely unified in its decision, so confusion is zero.

With this algorithm in mind, we present four confusion-based active learning techniques: k-fold committee, iterative correction, iterative committee, and iterative committee correction. In the next sections we will discuss each of these methods.

**K-fold Committee**

The k-fold committee has received quite a bit of attention in the machine learning community for its ability to produce a robust committee classifier. To create a k-fold committee, we first divide the training examples into \( k \) disjoint subsets. We then train each committee member with a different set of \( k-1 \) of these subsets. This training data splitting technique is represented in Figure 19. In this example we split the data into three subsets, and then train three classifiers, each on two of the three subsets. The confusion value for each example is computed according to the algorithm of Figure 18.

While the k-fold committee may be effective as a classifier for many situations, it has two major drawbacks when placed in the Crayons environment: speed of training and

![Figure 19](k-fold-classifier-data-partition.png)
speed of classification. Unfortunately, the k-fold committee requires much more training
time than a single classifier and in an interactive user interface time is a precious
commodity. In a k-fold committee, we must train \( k \) classifiers requiring nearly \( k \) times
the amount of time spent training a single classifier. In addition, when classifying, we
query all committee members instead of a single classifier so classification time is also
multiplied by \( k \). Considering these impairments, using the k-fold committee as our active
learning technique will likely not be efficacious. However, since the k-fold committee is
so widely used in the machine learning community as a classifier, we will test its
effectiveness in the Crayons environment as well.

**Iterative Correction**

Let’s consider the effect of having only
two classifiers in a committee, one which is the
true classification, and the second as the current
classification. The confusion area represents the
difference between the true classification and the
classification we have obtained from the user’s
labeling. In this case the confusion value
represents an accurate measure of the distance
between the current classifier and the true
classifier. In addition, we know the exact changes
that need to take place in order to achieve an
accurate classifier: give more examples in the areas of confusion between the true
classifier and the generated classifier.

![Figure 20](image)

Airplane classification task. (a) shows
the initial coloring of the airplane. (b)
shows the initial classification. In (c) the
user sees the errors and adds two more
strokes, one for airplane and one for
background. The user continues to add
strokes in (d) and (e) correcting errors in
training. Finally in (f) the user is
satisfied after adding many corrective
strokes.
With this ideal situation in mind, let’s recall the iterative process we discussed in Chapter 2. We observed the corrective behavior of the users, represented in Figure 20. First the user labels a few pixels as part of the airplane and a few that are not. The resulting classifier is then displayed. The user then labels more examples, following our observation, correcting errors in classification. Several iterations later we get a much better classifier. Let’s examine how this relates to the ideal situation we talked about earlier. The user knows the true classification while the computer knows the generated classification. The user gives the computer hints of the difference between the true classifier and the generated classifier. When the user stops correcting errors the generated classifier has arrived at the true classification (at least visibly to the user).

In this case we don’t guide the user to color specific pixels. As we discussed earlier, if we had the true classification, we would guide the user to pixels in the confusion area (or in other words, the errors in classification). Our observation that the users tend to correct errors shows that the user will manually provide the pixels in the confusion area without computer assistance.

Let’s look at the Iterative Correction technique in the light of the drawbacks we saw in the k-fold committee. The issue we had with the k-fold committee is that since we were using more than a single classifier, training and classification took longer. Iterative Correction requires us to train only a single classifier making both its training and classification much quicker than that of the k-fold committee. While Iterative Correction does not share the same defects as the k-fold committee it relies heavily on user judgment. For example if we have a 100-image task, Iterative Correction relies on the idea that the user will make a good choice as to which image to focus their attention.
Suppose that we have a classifier that is only 50% accurate on that task meaning essentially that 50 of the 100 images are classified completely incorrectly. Would we be naïve in thinking that the user can pick the best one or two images out of the 50 for labeling? While classification and training times are quite fast when using the Iterative Committee, we believe that there is likely a better solution to guide the user more effectively.

**Iterative Committee**

In the previous section we discussed the idea that users tend to correct errors in training. We took advantage of the fact that users know the true classification visually and can give the computer hints to where the true classification differs from the generated classifier. In this section we will discuss a similar idea we’ll call Iterative Committee.

We’ll begin our discussion on Iterative Committee with a small example. Let’s suppose we have 40 images in our classification task of distinguishing airplanes from the sky. The user labels a lot of examples on the first image until that particular image is classified nearly perfect as shown in Figure 21. Unfortunately, however, the user is unaware that the second image is almost completely incorrect and the overall classifier is quite far from the true classification. While the user knows the correct classification, his focus is too narrow to be effective.

![Figure 21](image-url) Airplane classification task. The user spent lots of time on the first image until it classifies well. Unfortunately the user didn’t know that the second image was classified almost completely incorrect.
In this situation, the user’s misdirected focus hinders our progress in obtaining a good image classifier quickly. With Iterative Committee we attempt to overcome this impediment. The Iterative Committee is represented in Figure 22. There are three classifiers in this figure: the true classification shown as the dotted red line, the current classification shown as the bold blue line, and the previous classification shown as the thin green line. The area where the current and the previous classifications disagree is what we refer to as the *confusion area*. The *confusion area* is grayed out in Figure 22. The important thing to note about the confusion area is that by minimizing this area, we converge on the true classification. We would like to direct users to examples that lie within the confusion area. We do this by giving those examples within the confusion area a score of 1 and those that are outside a score of 0.

**Iterative Committee Correction**

The final method of active learning we will discuss is called Iterative Committee Correction (ICC). The ICC method combines the two previous methods, Iterative Correction and Iterative Committee, into a single active learning technique. We discussed an interesting idea in the previous section that two classifiers generated during the Crayons iterative loop provide insight into where the computer and user should focus their attention. We also talked about the corrective behavior of users in the section on Iterative Correction. We would like to harness both of these phenomena to the user’s benefit. In the last section on Iterative Committee we made no assumptions about the user’s behavior when labeling examples. We assumed that all examples in the confusion area...
area were just as good as the next example in the confusion area. In Iterative Committee Correction, we allow the user to correct errors in the confusion area. While this may not seem to be a significant change, we will see in the next chapter that the user’s labeling behavior when it comes to correcting errors actually makes a difference.

To illustrate the differences among four techniques for pixel labeling, reinforcing, Iterative Correction, Iterative Committee, and Iterative Committee Correction we turn to Figure 23. The image shown represents a skin classification task. Figure 23 (a) shows the user’s reinforcement labeling. Notice that the user labels pixels which have already been classified correctly rather than correcting what the classifier hasn’t classified correctly. Figure 23 (b) shows the typical corrective behavior exhibited by Crayons users. This behavior is known as Iterative Correction. Figure 23 (c) shows the user’s labeling using Iterative Committee active learning. The darker area shown in (c) is the confusion region (the region that changed classification between the last two classifiers). Notice that the user labels pixels in this dark region independent of corrective or reinforcing behavior by both correcting and reinforcing alike. Figure 23 (d) represents the user’s behavior when using Iterative Committee Correction. Notice that the user still only labels pixels in the dark region, but limits the labeling to the pixels that are classified incorrectly.
Figure 23
Crayons user behavior. (a) shows reinforcing behavior. (b) shows typical corrective behavior. (c) shows iterative committee reinforcing/correcting behavior (the darker region is the region that changed between iterations). (d) shows iterative committee correction behavior (only corrects errors in training inside the dark region.)
Chapter 4 - Active Learning Simulations

Now that we have defined the various active learning techniques we would like to compare, we need to discuss the metrics we will use to decide which technique is the best. Simply implementing all seven techniques, performing various user studies with each, and comparing the results is one possibility. We would want to perform more than one test per active learning technique so that we can minimize the noise associated with a single test case. Also, we would like to perform experiments with more than a single set of images so that our results are not tied to a specific task. Combining these factors, we need to perform between 100 and 200 user tests minimum and several thousand ideally.

One problem with the above evaluation technique is the amount of time users spend performing tests. What if we could simulate the user? Computer simulation in this case has two clear advantages: 1) we can perform thousands of tests in a reasonable amount of time, and 2) we can see how different behaviors with respect to labeling affect the active learning techniques. Thus we turn to simulation to provide an initial comparison. What we hope to accomplish with this simulation study is to rule out the least beneficial of the active learning techniques. With a smaller set of active learning techniques to compare we can then implement these few into Crayons and performing user tests on them is much more feasible. We will discuss these user tests further in Chapter 5.

The simulation environment we create needs two things in order to effectively mimic reality: 1) an artificial user, which we’ll call an oracle; 2) the training examples, which we’ll call object generators. We realize that actually simulating a user is somewhat difficult because of the unpredictable manner in which the user chooses to
label examples. We attempt to overcome this difficulty by providing three different oracles. While this clearly won’t test all possible user behaviors, we believe these three oracles will provide a sufficient comparison among the active learning techniques.

The same argument applies to object generators: the world of possible objects and tasks we can generate is infinite and we can clearly not simulate all possibilities. We provide four models for generating objects and like the artificial oracles we believe that these models, while clearly not representative of the wealth of possible tasks, are sufficient to compare the proposed active learning techniques. We will discuss the different oracles and object generators we designed in the next section.

4.1 Simulation Implementation

We will begin our discussion of the simulation environment with the three oracles followed by the object generators and finally talk about the methodology for testing the active learning techniques.

4.1.1 Oracles

The three types of oracle we implemented are a decision tree oracle, a linear oracle, and an ellipsoid oracle. Each of these simulates a different class of problems: those that decision trees can learn easily with relatively few examples, those that are harder for decision trees to learn but can still learn quite well with lots of examples, and those where decision tree will struggle.

- First is an artificially generated decision tree that is created by randomly selecting a feature and then randomly selecting a split point on that feature. We then recursively generate sub-trees for both the less-than and greater-than
sides of the split point. When a maximum recursion depth is reached, specified as either 2 or 4 levels, the node is randomly assigned to be a true or false leaf. These oracles are quite straightforward for decision trees to learn since they are just decision trees themselves. We use this oracle as a representation of those problems on which decision trees perform well.

- Second is a linear oracle generated by randomly selecting \( N + 1 \) coefficients between -1.0 and 1.0. These coefficients are used to create a linear function in \( N \)-dimensional feature space which defines the oracle’s decision. This oracle represents simple problems on which decision trees can approximate well, but can’t represent perfectly.

- Last is an ellipsoid oracle generated by randomly choosing a radius and a center point for each dimension. The oracle determines the label of a given example by using the following inequality: 
\[
1 > \sum_{i=1}^{N} \frac{(f_i - C_i)^2}{R_i^2},
\]
where \( f_i \) is the \( i^{th} \) feature, \( C_i \) is a coordinate of the ellipsoid center, and \( R_i \) is an ellipsoid radius in that dimension. The foreground is defined to be inside the ellipsoid. With this definition if the inequality holds for a specific example, the example lies in the ellipsoid and is therefore a foreground example. We will run oracles with one or two ellipsoids. These oracles simulate problems where desired examples lie in one or two blobby regions of a problem space, and can be difficult for decision trees to learn.
4.1.2 Object Generators

We use a variety of object generators to model the objects of the real world. Each object we use in simulation has a set of features, all with values in the range \([-1,1]\). For simulation we decided to keep the number of features to a minimum. This allowed us to run over a thousand simulations in just a few days. We decided also to vary the number of features to see if the number of features made a difference in the most effective active learning technique. With that in mind we chose to simulate objects with 2, 4, 8, 16, and 32 features. In order to simulate the values that each of these features would take on, we used four different models for object generation: 1) Uniform distribution, 2) Gaussian distribution, 3) Bimodal distribution, 4) Mean-split. The first three models generate features based on their respective distributions. For example, the uniform object generator will generate feature values uniformly between -1 and 1 while the Gaussian and bimodal object generators will be biased to feature values closer to the peaks of the distribution and spread out according to their standard deviations (which were randomly generated so that the peaks

![Figure 24](image)

A mean-split tree. The leaf nodes represent the decision points. Each leaf node has equal probability of being selected. This means that the ranges for each node are represented equally but that the overall range from -1 to 1 is represented very unequally.
were between -1 and 1). Of course the Gaussian distribution has a single peak whereas the bimodal distribution has two.

We created the mean-split model because many problems don’t follow the assumptions of typical distributions such as Uniform, Gaussian, and Bimodal. The mean-split object generator produces feature values based on a tree of decision points. This tree is created by starting at the top and randomly picking a split point between -1 and 1. We then recursively split the tree node’s range until a pre-selected tree depth is reached. To generate an object, we traverse the tree in a random fashion and take the value of the leaf node. For example, Figure 24 shows a simplified version of a mean-split tree. We start at the top of the tree randomly picking left or right until we reach a leaf node. We then randomly choose a value in the range of that leaf node. The idea is that each leaf node has an equal probability of being selected (in our case, 25% for each node). Let’s consider the effect this will have on the tree of Figure 24. If each leaf node has an equal probability of being selected, the range between 0.3 and 0.4 (5% of the overall range) will be represented 25% of the time rather than 5% like it would be in a uniform distribution. A histogram of the mean-split distribution will show many peaks and valleys according to the range of each of the leaf nodes. In the example of Figure 24, there will be a peak in the ranges [-1,0.8] and [0.3,0.4], and valleys in the other two ranges. A typical example of a 4-level mean-split distribution (16 peaks/valleys) is shown in Figure 25. This method was selected because it will provide an odd distribution violating all

Figure 25
Mean-split distribution sampling – 4 levels (16 peaks/valleys in the distribution).
statistical assumptions of the more typical distributions such as the Uniform and Gaussian distributions.

4.1.3 Simulation Methodology

Once the oracles and object generators are in place, we are ready to run the simulations. Each simulation consists of a single oracle and a single object generator (which generates objects with a specific number of features – 2, 4, 8, 16, or 32). We run all possible combinations of oracle/object generator several times thereby performing several thousand simulations. A single simulation was performed in the following manner:

1. Generate 300,000 examples using an object generator and label them using an oracle.
2. Choose 1,500 random examples from the set of 300,000 with which to begin and train the decision tree classifier based on these examples.
3. Classify the entire body of examples (the 300,000 from step 1) with the trained decision tree and record the accuracy achieved.
4. Allow each guidance technique to select its own set of 1,500 examples and train independently of any other guidance technique.
5. Classify the entire body of examples with each classifier and record the accuracies achieved with each type of guidance.
6. Continue adding 1,500 examples until one of the classifiers achieves an accuracy of 99% or we have reached the level of 60,000 training examples (max: 40 iterations).

Figure 26
Active Learning Simulation Algorithm.
4.2 Simulation Results

In this section we will look at two factors in comparing the seven active learning techniques: 1) the time to convergence (how many examples did the active learning technique require to achieve accuracy above 99%), and 2) the rate of convergence (how quickly did the technique level off in its learning curve). In the next section we will discuss some implications of these results. We have purposefully omitted average accuracy because we believe that the learning curve and final accuracy metrics will be more valuable than a simple average accuracy. One reason for this is because the user is not going to be concerned with the average accuracy he has achieved over the life of his task but rather the accuracy of his task right now.

In several days we were able to run 1,225 simulations, each performing at most 40 iterations (see Figure 26 for the simulation algorithm). After adding 1,500 new examples to the learning algorithm (each iteration of the algorithm), we recorded the accuracy

![Figure 27](image1.png)

**Figure 27**
Active Learning Technique comparison in simulation. The numbers shown here are the number of times that each active learning technique reported the highest accuracy at the end of each simulation. At the end of simulation more than one technique could report the highest accuracy.

![Figure 28](image2.png)

**Figure 28**
Second Active Learning Technique comparison from simulation. This figure is similar to Figure 25 with the difference that there is only a “best” accuracy when one technique exclusively reports the highest accuracy (no ties allowed).
achieved by the various active learning techniques. The first way we will analyze the results of these simulations is to see which active learning technique had the highest accuracy at the end of the simulation. By counting the number of times a particular technique had the highest accuracy, we obtain the graphs shown in Figures 27 and 28. These graphs show how many times each algorithm reported the highest accuracy of all the techniques. Figure 27 allows more than one technique to report the same highest accuracy, while Figure 28 only allows a single technique to report the same highest accuracy. In other words, Figure 28 is the graph of how often did a single technique report the highest accuracy. Figure 29 shows the algorithms used in calculating these values.

```plaintext
overallHighestAccuracy( simulations[][], algorithm )
    total <= 0
    foreach algorithmAccuracies[] in simulations
        if algorithmAccuracies[algorithm] is >= all other algorithmAccuracies
            total++
    return total

singleHighestAccuracy( simulations[][], algorithm )
    total <= 0
    foreach algorithmAccuracies[] in simulations
        if algorithmAccuracies[algorithm] is > all other algorithmAccuracies
            total++
    return total
```

Figure 29
The highest accuracy comparison algorithms.

During these simulations, a single technique reported the highest accuracy 960 times out of the 1,225 simulations. This shows that at the end of the simulation there was a clear “best” technique a majority of the time. Notice that the iterative techniques (iterative committee, iterative correction, and iterative committee correction) are second
only to K-Fold Committee in both overall highest accuracy and single technique highest accuracy. We will discuss this further in the next section.

The second approach we will take to analyzing these simulation runs is to take into consideration the entire learning curve to see which technique maintained the best accuracy throughout the simulation. As soon as we record the accuracy during a single iteration we compare each technique to see who had the highest accuracy. The difference here is that we are comparing within a simulation rather than just at the end of a simulation. This comparison produced the graphs shown in Figures 30 and 31. These two graphs were constructed in the same manner as those of Figures 27 and 28.

We recorded accuracy a total of 24,673 times in the 1,225 simulations. 21,449 times, a single technique reported the highest accuracy, again indicating that there was a clear best most of the time. In these graphs we notice that instead of K-Fold Committee, the best techniques seem to be Iterative Committee and Iterative Committee Correction. A possible explanation for this phenomenon is that these methods achieve a higher accuracy early on and make slower progress thereafter while the K-Fold Committee takes

Figure 30
Third Active Learning Technique comparison in simulation. The numbers shown here are the number of times that each active learning technique reported the highest accuracy immediately after recording accuracy (at most 40 times per simulation). More than one technique can report the highest accuracy.

Figure 31
Fourth Active Learning Technique comparison from simulation. This figure is similar to Figure 28 with the difference that there is only a “best” accuracy when one technique exclusively reports the highest accuracy (no ties allowed).
longer to get to its final accuracy level as is shown in the graph of Figure 32. This suggests that the iterative methods (Iterative Committee and Iterative Committee Correction) find the examples early on that will improve accuracy very quickly. This was our goal, to get to the best examples as quickly as possible. We include only Uniform Distribution, K-Fold Committee, and Iterative Committee Correction in the graph of Figure 32 because they are the most relevant. Of these three, Uniform Distribution clearly performs the worst. K-Fold Committee is not only a slow-starting algorithm, but gives an inconsistent, oscillatory accuracy. This indicates that the examples that the K-Fold Committee chooses to add to the set of training examples will be beneficial sometimes and just confuse the algorithm other times. ICC, on the other hand, converges fairly quickly to its final accuracy giving a nearly monotonically increasing accuracy.

4.3 Simulation Analysis

From the graphs in Figures 27 and 28, we conclude that there are four active learning techniques we could choose to implement: K-Fold Committee, Iterative
Correction, Iterative Committee, or Iterative Committee Correction. We will clearly leave out Uniform Distribution, Equal Distribution, and Mixed Distribution because their performance in simulation was pitiful compared to the other techniques. Iterative Correction is already implemented since it is the basis of the Crayons environment [1], so we don’t need to do anything special to implement it. As for the other three, we turn to an analysis of their characteristics to choose between them.

As we discussed in the previous section, K-Fold Committee seems to dominate the end result, but not the path toward the end result. If we are only concerned with the end result, then perhaps K-Fold Committee would be the active learning technique to choose. The iterative methods, on the other hand, seem to converge more quickly on their final accuracy and have a more consistent learning curve (as compared to the K-Fold Committee’s oscillatory curve). In addition, K-Fold Committee methods are very expensive computationally. Obtaining a k-fold committee requires us to split the training data into k independent sets and train k classifiers independently. Because this is an interactive environment, training k classifiers will likely be too slow. In our experience, training a single classifier and classifying an image takes around 2-3 seconds. If we use a 5-fold classifier, this training/classification time would be closer to 10-15 seconds, pushing the limits of interactivity as well as trying the patience of the users. Adding an active learning component to the training/classification time would further increase this delay and render K-Fold Committee unusable in the Crayons environment.

The iterative methods, however, are completely native to the Crayons environment. Because we already have the classifiers used in the committees, the only time delays we add to the 2-3 second training/classification time are 2-3 more seconds of
comparisons (which we will discuss more in Chapter 5). While this is still moving out of the truly interactive range, we will explore ways in Chapter 5 of decreasing this delay to get a better interactive experience.

Because of the speed benefits inherent in using the iterative methods and their performance in the simulations, we will implement Iterative Committee in Crayons for use in user experiments. We also note that by implementing Iterative Committee we also gain Iterative Committee Correction (ICC) since the “Correction” portion of ICC is only dependent upon the user, not the algorithm.
Chapter 5 - Active Learning Experiments

We now turn to user experiments associated with active learning. As we discussed earlier, implementing all seven active learning techniques would be unwise, so we resorted to artificial simulations. Simulation, however, is not sufficient to show the usability of an interactive technique. In this Chapter we will compare in a formative manner three techniques for providing active learning in Crayons: Freeform, Iterative Correction, and Iterative Committee.

Freeform learning is a very passive form of the Crayons environment. Fails [2] presented a form of Crayons learning which provides feedback to the user periodically for correction. Rather than presenting feedback to the user, freeform learning simply observes the users labeling and creates a classifier after the user has finished. This type of learning would be termed passive machine learning rather than active. We include this type of learning in our comparison because it provides our study with a baseline for the comparison of our active learning techniques.

Iterative Correction, while not the best active learning technique in our simulation studies, is the basis for the Crayons environment. Any active learning technique that we provide to the Crayons environment must compare favorably to Iterative Correction. The Iterative Correction technique is as follows: the user labels a set of pixels, the learner responds by giving feedback to the user, and the user corrects errors in training [1].

Iterative Committee was the technique we found in simulation to provide the best classifier accuracy while giving the best interactive time. We include this technique in our study to verify the findings of our simulations. We will refer to this technique as ICC.
for the rest of this work unless there is a need to distinguish Iterative Committee with Iterative Committee Correction, in which case we will refer back to their full names.

At this point we will modify the interactive machine learning sequence presented by Fails [1] and shown in Figure 33(a) with the one in Figure 33(b). In this new sequence we replace the user-interactive loop with the active learning loop. This loop replaces the “Correct” step with “Guidance” followed by “User Input”. This sequence will provide the basis for our implementation of our active learning techniques into Crayons.

5.1 Experiment Implementation

The three techniques we will implement are freeform labeling, feedback, and ICC. These techniques all follow the same pattern: 1) display an image for the user to “color”, 2) run the guidance mechanism, and 3) modify the view the user sees according to the active learning algorithm. We will discuss each active learning technique and how each uniquely implements this pattern followed by a discussion of the comparison metrics we will use to measure their behavior.

5.1.1 Freeform Labeling
The simplest technique we implement is clearly freeform labeling. The only necessary part of this technique is to have a way for users to label pixels. We will use the standard Crayons mechanism of using left and right mouse buttons for foreground and background respectively. For example, in the grass-classification task of Figure 34, the user clicks and drags with the left mouse button to label grass pixels and conversely with the right mouse button to label non-grass pixels. We also create a mechanism for automatically saving the users labeling to a file so that we can process the data at a later time.

5.1.2 Iterative Correction

Feedback is only slightly more complicated than freeform labeling. The user labels pixels the same way as freeform labeling, but now we add an extra step in the loop, feedback. When the user has labeled a specific number of pixels we create a classifier, classify all the pixels of the currently displayed image, and display feedback to the user as a semi-transparent layer over the image for the user’s correction. This method is the default Crayons environment, so all we need to add is the mechanism for saving the user’s labeling and note at what points we created a classifier.

5.1.3 ICC

ICC is by far the most complex mechanism we added to the Crayons environment. We again use the same mechanism for labeling pixels and displaying
feedback, but add an extra active learning step whenever a classifier is created. As we discussed earlier, ICC is based on the current and previous classifiers to form a committee. The pixels that cause the two classifiers to disagree are those which we suggest to the user. We implemented the simulation in exactly this manner. This method, however, has two fundamental flaws when moving to an interactive environment: 1) asking for the label of a single pixel is a highly inefficient use of the user’s time, and 2) classifying all pixels related to the task is too slow (anywhere from 30 seconds to several minutes) for an interactive environment such as Crayons.

**Single Pixel to Pixel Regions**

A solution to the problem of asking for the label of a single pixel was solved by using pixel regions with the most disagreement. In our final version of the interface, the pixel regions were whole images. Thus the image with the highest pixel disagreement from between the two classifiers was the image suggested to the user.

![Image of a thermometer widget](image)

**Figure 35**
The thermometer widget. This widget gives a visual representation of the amount of confusion in an image.

**The Thermometer Widget**

As an enhancement to the Crayons interface we present a new widget we’ll call the thermometer widget. The thermometer widget is shown in Figure 35. It is displayed
next to each image in the image selection area of the Crayons interface as is shown in Figure 36. The thermometer value represents the image confusion value as a percent of how many pixels changed between classifiers. For example, if the thermometer shows 100%, every pixel in the image changed labels from one classifier to the next. If the thermometer shows 0%, no pixel changed values between classifiers. Our active learning simulations suggest that labeling pixels in the images with the highest confusion values will be of most benefit to the learning algorithm. With the thermometer widget, the user
can keep track of how much change is happening with all images without the need to look at all the images.

An interesting side note about the thermometer widget is that it also gives the user a clue as to the task completion. Let’s suppose that all the thermometer values are high, around 70%, the task isn’t anywhere near complete because the two classifiers are far from converging on the true classification. When the thermometer values are down near 0, very little change is happening between classifications, indicating that the classifiers are finally converging on the true classification. We will talk more about completion estimation later on in Chapter 7.

Let’s take a look at the example task shown in Figure 37. We start the task with thermometer values at 100% (a). As we label more pixels (b), the thermometer values decrease and we notice that the classification has improved significantly (c).

**Image Sub-Sampling**
The second problem with trying to classify all pixels in all the images is that this will be prohibitively slow. Our solution to this problem is to sub-sample the images and base our active learning on this subset rather than all pixels (see Figure 38). In our studies we chose, somewhat arbitrarily, to sample one pixel in every 25x25 block of pixels. In this way we were able to reduce the

![Figure 38](image.png)  
*Sub-sampling used for ICA in user experiments.*
classification time from several minutes to a few seconds. An example of the image sampling we used is shown in Figure 38. Notice that the sampled pixels include all the major parts of the image, including even the ankle bracelet. This sampling reduces the number of pixels we classify by ~90%.

5.1.4 Comparison Metrics and Experimental Tasks

With these techniques defined and implemented we turn to a mechanism for comparing them in an effective manner. We would like to reduce the variability between users as much as possible while still allowing for a broad range of user abilities. The key point we have focused on in this work is a reduction in classifier creation time. The idea is to find an active learning technique that will minimize the amount of time to reach a desired accuracy level. By time we mean wall-clock time and not processor time. In this case, the best metric would be a measurement of time to reach a pre-determined accuracy level. Wall-clock time is somewhat difficult to compare among users, however, because of the different rates at which people perform work. Some people may rapidly label thousands of pixels in a few strokes while others may label a few pixels more carefully. Thus we will focus on a different metric that approximates wall-clock time: number of classifiers created before reaching a pre-determined accuracy level. With this metric we can easily compare the classifiers throughout our experiments.

With respect to the tasks we would like the users to perform, we need tasks that are difficult enough for the computer to not be able to learn for 30 or 40 iterations in order to discriminate between the learning techniques. On the other hand, we still want the classifier to converge to around 90-95% accuracy within 20 or 30 iterations. Since we don’t want our methods to be tied to any particular task we decided to have the users
perform two different tasks: sidewalk and sky classification. Each of these tasks is composed of 20 images. To determine true accuracy, we created “gold-standards” for each image, a completely-labeled version of the image with the exception of a few edge pixels (since edges are somewhat arbitrary as to how they are classified and will only add noise into our results). We had the users label pixels until they colored a total of 25,000 pixels.

Specifically, our user studies were performed in the following manner:

1) The user watched a brief instructional video on the Crayons environment and the user tests.

2) The user was allowed to practice as long as he would like on a skin classification task.

3) The first task was presented to the user with a popup description of what was to be classified (either sky or sidewalk) and the type of environment (freeform, iterative correction, iterative committee).

4) The user would then label pixels at his own pace. As soon as the user had labeled 750 pixels, he would be interrupted while the system classified the images and performed the active learning.

5) If the task is freeform, the user simply continues labeling pixels. If the task is iterative correction, we present a semi-transparent layer over the top of the current image with the image’s classification. If the task is iterative committee, we display the image’s classification and the thermometer widget with updated thermometer values according to the confusion value of each image.
6) When the user has labeled 25,000 pixels we signal that the task is complete.

7) Repeat steps 3-6 for all three techniques (freeform, iterative correction, and ICC).

We had twelve users perform the experiment, each taking about an hour to complete the given tasks. We altered the technique ordering for each user so that we could minimize the cross-task influence of one labeling technique on another. For example, if we give the iterative correction environment first, then freeform second, the user may remember the parts of the images that were difficult for the computer to learn and so will add more examples in those areas. The user’s experience with computers was quite varied as well as their exposure to the Crayons environment prior to the study.

Due to the inherent variation in user behavior, cross-user comparisons are a challenge. We decided to restrict user behavior in two areas to reduce as much variation among the users as possible. We realize that restricting the users will perhaps produce biased results, but our hope is that we will be able to make sound judgments based on the results of the experiments because of these restrictions. The first restraint we placed on the users was when they should switch images. In our tasks of 20 images we hoped to have the users eventually look at all 20 images rather than spending excessive amounts of time on a single image. To accomplish this, we added an alert to the test system telling the users to switch images. Most users followed the prompt and switched images at that time. By restricting the user in this way we were able to reduce the difference in time spent focused on a particular image among the users. The second restriction we made in our test system was to force the user to cycle through the images in a particular order.
For example, for ICC we forced the user to cycle through the images according to the confusion-based ranking mechanism we described earlier in Chapter 5. Unfortunately this restriction reduced the effectiveness of the thermometer widget we proposed as an addition to the Crayons environment. Thus we don’t provide any results or analysis of the thermometer widget and focus our attention in the results solely on the active learning techniques.

### 5.2 Experimental Results

The graph in Figure 39 shows the full learning curves for the three techniques we discussed earlier: freeform, iterative correction, and iterative committee (ICC). We see from this graph the benefit of the two iterative methods compared to freeform labeling. ICC performs better than freeform labeling 70% of the time performing much better after only a few iterations. We also see that both ICC and Iterative Correction converge to their final accuracies much more quickly than freeform. This shows the clear advantage that active learning has over the more traditional passive learning. In our study we used tasks with only twenty images, so we can only speculate on the time savings when used on a task composed of hundreds of images, but we assume it would be significant.
Another observation we make from this graph is that ICC and Iterative Correction perform almost identically. Several factors have likely contributed to this disturbing discovery. First, the stringent restrictions we placed on the environment to reduce variation among users could have created an uneven playing field for the two methods, giving an advantage to iterative correction. Perhaps this disconnect from simulation to Crayons is due to the fact that we don’t guide the user to specific pixels, making ICC nearly identical to Iterative Correction. Purifying the practical version of ICC so that it more closely resembles the simulation version is a topic worthy of future research.

Figure 39
Learning curves of ICC, Iterative Correction, and Freeform
Chapter 6 - Accuracy Estimation

In the previous chapters we have discussed frequently the idea of accuracy and how fast we can achieve a desired level of accuracy. Ideally we would like to use the true accuracy of the classifier. One problem, however, prevents us from using true accuracy: calculating true accuracy is generally not possible because we don’t have the correct labels of all the examples. Since we still need an idea of classifier accuracy, we turn to various techniques of accuracy estimation in place of calculating true accuracy. Traditional machine learning typically uses one of three techniques for estimating accuracy: training set accuracy, hold-out set accuracy, k-fold cross-validation accuracy [16]. We also propose a new method of estimating accuracy in Crayons called the Incremental Consistency Accuracy.

6.1 Accuracy Estimation Techniques

Training Set Accuracy

Training-set accuracy is defined as the accuracy obtained on the set of labeled examples with which the learning algorithm was trained [16]. The benefit of using this type of metric is that most algorithms calculate the training-set accuracy during training, so we get the estimate without any further calculation when the algorithm terminates. Considering our main goal is to reduce the time required to create a good image classifier while adding as little overhead as possible, this method is very compelling.
Hold-Out Set Accuracy

Hold-out set accuracy is defined as the accuracy on a set of labeled examples which were not used to train the classifier [16]. To obtain the accuracy estimate in this case, we first split the entire set of labeled examples into two subsets, one for training and one for testing accuracy of the trained classifier. In our experiments with the hold-out set accuracy estimate we used 70% of the examples for training and 30% for an accuracy estimate. While this may, and typically does, provide a better estimate than training-set accuracy, there is one major drawback to hold-out set accuracy: the user must label more examples than we can use for training.

K-Fold Cross-Validation Accuracy

K-fold cross-validation is defined by partitioning the labeled data into $k$ disjoint, equal-sized subsets [16]. These subsets are then used to train $k$ classifiers (each trained on $k-1$ of the subsets). With these classifiers, we can classify each labeled example using a simple vote. The estimate is reported as the percent of the labeled data correctly classified using this agglomerate classifier.

This is probably the most used accuracy estimation technique in machine learning because it generally provides a good estimate of the true accuracy. As we discussed in Chapter 5 K-fold Cross Validation is very computationally expensive and its usefulness is quite limited in an interactive environment. We include it in our study, however, because it is so widely used in the machine learning community.
Incremental Consistency Accuracy

In addition to the methods we described above, we propose a new accuracy estimation technique called the Incremental Consistency Accuracy (ICA) estimator. ICA is motivated by the iterative committee correction (ICC) active learning technique which utilizes successive classifier information for its confusion calculation. Just as ICC uses confusion for a metric in active learning, in ICA we use confusion for a measure of accuracy.

ICA is built around the idea that the entire body of examples is potentially beneficial to an accuracy estimate, not just those which have been labeled. In ICA we report the accuracy as how much change has occurred from one iteration of the active learning loop to the next. This “change” is the concept of confusion we discussed in Chapter 3. The idea is that when there is a lot of change between classifiers, the accuracy of either classifier must be poor, so the reported accuracy is low. When two classifiers are very similar, we assume the two are converging on the true classification, so we report a high accuracy. This works because when the task has just begun, the decision tree base changes the most between successive classifiers. As the task continues, the changes become isolated to further down the tree creating two nearly identical classifiers. Thus, we will take two similar classifiers as an indication that the task is nearing completion.

6.2 Accuracy Estimation Simulations

Once again we turn to a simulated environment to compare these accuracy estimation techniques. We will use the same environment we described in section 4.1. To select examples, we used the active learning techniques Iterative Correction, Iterative
Committee, and Iterative Committee Correction. Once the examples are selected, based on the learning technique, we create three different classifiers instead of just one: one trained on all the labeled examples we’ll call the 100-classifier (used for the ICA metric and the training-set accuracy metric), one trained on 70% of the examples we’ll call the 70-classifier (used for the hold-out set accuracy metric), and a 5-fold committee classifier we’ll call the 5-fold classifier (used for the k-fold cross-validation accuracy metric).

In this section we will first describe how each of the accuracy estimates was obtained followed by the results of the simulations.

For the ICA estimate, we compare the previous 100-classifier with the current 100-classifier by recording the number of examples that changed labels between the two classifiers (the confusion value). The confusion algorithm we presented in Chapter 3 along with the ICA estimate algorithm is shown in Figure 40.

```plaintext
// Two-class (positive & negative) Confusion Score Algorithm
// Returns a number between 0 (no confusion) and 1 (maximum confusion)
confusionScore( committee of classifiers C, example E )
    initialize count := 0
    initialize members := number committee members in C
    foreach classifier c in C
        classification = c.classify( E )
        if classification is positive
            count++
        if count > members/2
            count = members – count
    return count/(members/2)

// Returns the average confusion of all the examples in E
ICAEstimate( oldClassifier, curClassifier, examples E )
    initialize aggregate := 0
    foreach example e in E
        aggregate += confusionScore( {oldClassifier,curClassifier}, e )
    return aggregate / E.size()
```

Figure 40

The ICA estimate algorithm. The algorithm returns the average confusion of all the examples in E.
We also used the 100-classifier for the training-set accuracy estimate. We report the accuracy achieved on all the labeled data using the 100-classifier. The algorithm for this estimate is shown in Figure 41. We pass the 100-classifier and the entire labeled set of examples to the exampleSetAccuracy function.

For the hold-out set accuracy estimate we used the 70-classifier. The set of labeled examples not used to train the 70-classifier is termed the “test set” and is composed of 30% of the labeled examples. We classified the test set with the 70-classifier and reported the accuracy achieved on this set as the hold-out set accuracy estimate. The algorithm for the hold-out set accuracy is identical to the training-set accuracy shown in Figure 41. In the hold-out set accuracy case we pass the 70-classifier and the test set to the exampleSetAccuracy function.

The final accuracy metric, the k-fold cross-validation accuracy, uses the 5-fold

```c
// Returns the percent of examples in E that the committee C classifies correctly
kFoldAccuracy( classifiers C[], examples E )
initialize correct := 0
foreach example e in E
    initialize counts[ C.size( ) ] := {0}
    foreach classifier c in C
        counts[ c.classify(e) ] ++
    classification = classification with most votes in counts[]
    if classification == e.classification
        correct++
return correct / E.size( )
```

Figure 42
The k-fold cross-validation accuracy algorithm.
classifier. We first split the training data into 5 subsets and train each of the 5 classifiers with 4 of these subsets. When classifying an example, the example is classified by all 5 classifiers, each one receiving a vote on the classification. The majority vote is the classification returned. The algorithm for the k-fold cross-validation accuracy is shown in Figure 42.

The results of these simulations are shown in Figure 43. This figure shows three graphs of the individual estimates along with the true classifier accuracies. We have purposefully left off training set accuracy because it is a very uninteresting graph maintaining an estimate of 100% accuracy throughout the simulation. In the three graphs shown here, “True Accuracy” refers to the accuracy the respective classifier achieved on the entire body of examples (rather than just the labeled examples). There are a few interesting observations we would like to make at this point. First is that the k-fold

---

**Figure 43**

Accuracy estimate comparison to true accuracy.
estimate maintains an estimate of 100% while the true accuracy is between 83% and 88%. We will discuss this phenomenon in the analysis section (section 6.3). Another observation we take from these graphs is that the ICA and hold-out set estimates are quite similar, hovering within about 5% of the true accuracy. Notice that the accuracy estimates are generally below the true accuracy. This phenomenon is actually very useful because being overly optimistic is not a good feature of an accuracy estimate (as we noticed with the k-fold estimate). Also, providing an estimate that is too pessimistic makes the estimate less useful as well because that estimate wouldn’t give us very much information about the true accuracy.

6.3 Accuracy Estimation Experiments

We use the same experimental setup we used in Chapter 5 for the accuracy estimates. We ran the same experiments with three of the users to test the accuracy estimates. After the users finished the three tests, we created the 100-classifier, the 70-classifier, and the 5-fold classifier as we did in simulation. We then computed the accuracy estimates according to their respective algorithms. We also computed one more accuracy estimate we’ll call SS-ICA which stands for sub-sampled ICA. You may recall from Chapter 3 that we discussed a sub-sampling technique to speed up ICC. We wanted to see if using this same sub-sampling
technique to quickly compute the accuracy metric would be detrimental to the ICA estimate. The graph of Figure 44 shows the comparison of SS-ICA (sub-sampled ICA) with ICA. As we can see, SS-ICA tracks the true accuracy just as well as ICA, indicating that we don’t lose information by sub-sampling. In fact, the sub-sampled version of ICA appears to be an even more conservative estimate (meaning that it is less likely to overestimate the true accuracy).

![Accuracy Estimation Comparison](image)

**Figure 45**
Hold-out set accuracy, K-fold Cross-validation accuracy, and SS-ICA results comparison.

The results of the experiments are shown in Figure 45. Again we have left off the training set accuracy estimate because it performed as predicted, maintaining a nearly constant 100% accuracy estimate. There are a few interesting points to note from this graph. First, just as in simulation, the k-fold estimate constantly reported an estimate of 100% accuracy. Second, the hold-out set estimate also reported a highly optimistic estimate. The final observation we make is that the SS-ICA estimate maintained a conservative, yet close estimate to the true accuracy.
6.4 Accuracy Estimation Analysis

In the last section we pointed out a few interesting phenomena from the results of our experiments. We offer in this section a few explanations for these results. Let’s begin with the simulations. We observed in simulation that the k-fold estimate always reported 100% accuracy no matter what the true accuracy was. To understand why this would be the case, let’s return to the argument we had about the decision tree algorithm. Because of how the decision tree is trained, it will always achieve 100% accuracy on its training data (unless there is noisy data). According to Freund, boosting will provide a classifier that performs better than any of the individual classifiers on the training data. The requirements for his claim are: there must be some difference in the generated classifiers, and the classifiers must perform better than 50% on the training data. In our case we satisfy both properties and so our boosted committee classifier will perform better than the already 100% accuracy provided by the decision tree. It is no wonder then that the k-fold estimate behaves exactly like the training set accuracy estimate.

In our practical experiments we observed that the k-fold estimate performed in the same manner, which would be expected as per our explanation given in this section. The interesting observation we didn’t expect in our study is to see that the hold-out set estimate performed like the training-set and k-fold estimates, giving a very optimistic estimate to true accuracy, maintaining an estimate of nearly 100% throughout the experiment. The question is: why did the hold-out set estimate fail in Crayons while it performed well in simulation? We offer one simple insight to explain this observation. The image in Figure 46 shows a typical stroke from the Crayons environment. We notice from this image that the colored pixels represent a highly redundant set of data. Suppose
we split the data into two sets, each example from one set will likely have an example that is extremely similar in the other data set. Since the hold-out set estimate relies on the two datasets being different, the hold-out set estimate fails in Crayons. In simulation, however, the datasets were likely quite different, so the hold-out set estimate was able to produce a better estimate than it did in Crayons.

Let’s take a look at the comparison graph again shown in Figure 47. The training set accuracy would be right along the top along with the k-fold and hold-out set estimates. We notice immediately that all three of these typical machine learning approaches to accuracy estimation fail miserably in the Crayons environment. The one estimation technique that produces a good estimate in Crayons is the ICA estimate.

Figure 46
Sky Classification Task
The yellow markings are where the user indicated “sky” while the red, “not sky.” Notice there is no change between pixels in the magnified box – all pixels are the same shade of blue.

Figure 47
Hold-out set accuracy, K-fold Cross-validation accuracy, and SS-ICA results comparison.
Since this technique requires less computation than either k-fold cross-validation accuracy or hold-out set accuracy, this result is very gratifying.
Chapter 7 - Completion Estimation

In an interactive environment it is important for the user to know when he has labeled enough examples. Traditionally the methods for determining completion are either based on an accuracy level or a threshold for number of iterations. In the accuracy level case, the task is complete when the task accuracy has risen above a certain threshold or has reached a plateau, where further training will likely not prove beneficial. By providing a good completion estimate to the user, we save time and effort when further training is either futile or even devastating.

As an example of the importance of completion estimation, imagine a task with 100 associated images. The user begins with the first image. After this image is classified well, the user moves to the next image, and so forth until the user has seen all images once. Suppose for a moment that the user needed only label pixels from the first 5 images and the other 95 would be labeled perfectly. In this case the user has wasted enormous amounts of time scrolling through the remaining 95 images only to discover that they are all classified well. This is one case where a completion estimate would prove useful. While this may seem like an extreme case, imagine taking images from a security camera in your office. Most of the images taken off the camera will be nearly identical. The point is, combined with the active learning and accuracy estimation techniques we described in this work completion estimation is a valuable part of a time-effective Crayons environment.

7.1 Estimation Techniques Comparison

We return to the graph given in the last chapter comparing the various accuracy estimation techniques. This graph is shown in Figure 48. Suppose that we choose a
completion estimate based on an accuracy threshold of 87%. Training-set accuracy, hold-out set accuracy, and k-fold cross-validation all terminate after the very first iteration when accuracy is at a low 75%. SS-ICA, on the other hand, terminates the task at iteration number 29 when the actual accuracy is very close to the highest accuracy achieved during the task – 90%. With an accuracy plateau threshold of +/-1%, we find that training-set accuracy, hold-out set accuracy, and k-fold cross-validation all terminate after the first few iterations when the actual learning curve is nowhere near its peak. SS-ICA on the other hand will terminate sometime after iteration 25, a few iterations after the true accuracy has already reached a plateau. In both cases, SS-ICA provides a good estimate of completion while the others fail miserably.

Figure 48
Comparison of all the accuracy estimation techniques.
Chapter 8 - Conclusion & Future Work

We have shown the results of simulation as well as user tests to show that the concept of comparing consecutive classifiers is a valuable alternative to traditional methods of active learning, accuracy estimation, and completion estimation. Using the iterative committee (ICC) methods of active learning we were able to achieve a good classifier in less time (fewer iterations) than simple coloring. Using the iterative consistency accuracy (ICA) estimate, we were able to give a better accuracy estimate than the traditional methods of training-set accuracy, hold-out set accuracy, and k-fold cross-validation.

Even though the benefit of using ICC over simple coloring is evident, the benefit of using ICC over the traditional iterative correct (IC) method used by Crayons is not as clear. The author is convinced, however, that the ICC active learning techniques will be more beneficial to many applications than IC. Unfortunately, the limited scope of this research did not provide ample evidence to support this claim. A topic of further research would be to simulate or find cases where ICC will significantly outperform IC and evaluate the plausibility of these cases. This is not to say that ICC did not provide other benefits to the Crayons environment over IC. For example, the sub-sampled percent changed metric used in ICC and SS-ICA provided a simple accuracy estimate which proved to be rather useful in completion estimation. An important direction for future research is to determine more areas where the iterative committee and iterative consistency concepts can be further utilized.


