# Interactive Machine Learning

Jerry Alan Fails, Dan R. Olsen, Jr.
Computer Science Department
Brigham Young University
Provo, Utah 84602
{failsj, olsen}@cs.byu.edu

**ABSTRACT**

Perceptual user interfaces (PUIs) are an important part of ubiquitous computing. Creating such interfaces is difficult because of the image and signal processing knowledge required for creating classifiers. We propose an interactive machine-learning (IML) model that allows users to train, classify/view and correct the classifications. The concept and implementation details of IML are discussed and contrasted with classical machine learning models. Evaluations of two algorithms are also presented. We also briefly describe Image Processing with Crayons (Crayons), which is a tool for creating new camera-based interfaces using a simple painting metaphor. The Crayons tool embodies our notions of interactive machine learning.

**Categories:** H.5.2, D.2.2

**General Terms:** Design, Experimentation

**Keywords:** Machine learning, perceptive user interfaces, interaction, image processing, classification

**INTRODUCTION**

Perceptual user interfaces (PUIs) are establishing the need for machine learning in interactive settings. PUIs like VideoPlace [8], Light Widgets [3], and Light Table [15,16] all use cameras as their perceptive medium. Other systems use sensors other than cameras such as depth scanners and infrared sensors [13,14,15]. All of these PUIs require machine learning and computer vision techniques to create some sort of a classifier. This classification component of the UI often demands great effort and expense. Because most developers have little knowledge on how to implement recognition in their UIs this becomes problematic. Even those who do have this knowledge would benefit if the classifier building expense were lessened. We suggest the way to decrease this expense is through the use of a visual image classifier generator, which would allow developers to add intelligence to interfaces without forcing additional programming. Similar to how Visual Basic allows simple and fast development, this tool would allow for fast integration of recognition or perception into a UI. Implementation of such a tool, however, poses many problems. First and foremost is the problem of rapidly creating a satisfactory classifier. The simple solution is to using behind-the-scenes machine learning and image processing.


*IUI'03,* January 12–15, 2003, Miami, Florida, USA.


Machine learning allows automatic creation of classifiers, however, the classical models are generally slow to train, and not interactive. The classical machine-learning (CML) model is summarized in Figure 1. Prior to the training of the classifier, features need to be selected. Training is then performed "off-line" so that classification can be done quickly and efficiently. In this model classification is optimized at the expense of longer training time. Generally, the classifier will run quickly so it can be done real-time. The assumption is that training will be performed only once and need not be interactive. Many machine-learning algorithms are very sensitive to feature selection and suffer greatly if there are very many features.

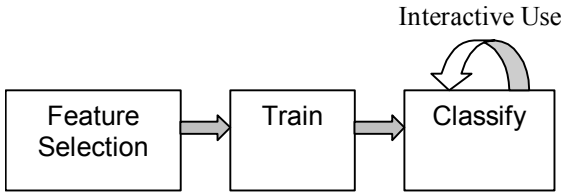


**Figure 1** – Classical machine learning model

With CML, it is infeasible to create an interactive tool to create classifiers. CML requires the user to choose the features and wait an extended amount of time for the algorithm to train. The selection of features is very problematic for most interface designers. If one is designing an interactive technique involving laser spot tracking, most designers understand that the spot is generally red. They are not prepared to deal with how to sort out this spot from red clothing, camera noise or a variety of other problems. There are well-known image processing features for handling these problems, but very few interface designers would know how to carefully select them in a way that the machine learning algorithms could handle.

The current approach requires too much technical knowledge on the part of the interface designer. What we would like to do is replace the classical machine-learning model with the interactive model shown in Figure 2. This interactive training allows the classifier to be coached along until the desired results are met. In this model the designer is correcting and

teaching the classifier and the classifier must perform the appropriate feature selection.

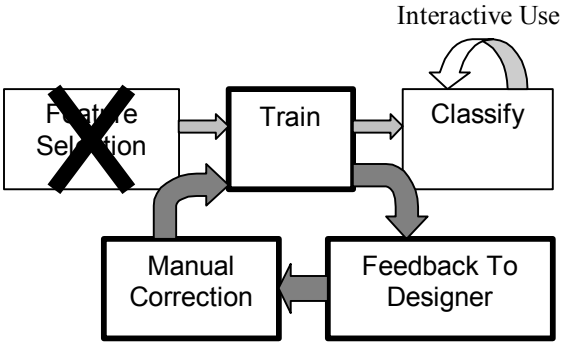


**Figure 2** – Interactive machine learning (IML) model

The pre-selection of features can be eliminated and transferred to the learning part of the IML if the learning algorithm used performs feature selection. This means that a large repository of features are initially calculated and fed to the learning algorithm so it can learn the best features for the classification problem at hand. The idea is to feed a very large number of features into the classifier training and let the classifier do the filtering rather than the human. The human designer then is focused on rapidly creating training data that will correct the errors of the classifier.

In classical machine learning, algorithms are evaluated on their inductive power. That is, how well the algorithm will perform on new data based on the extrapolations made on the training data. Good inductive power requires careful analysis and a great deal of computing time. This time is frequently exponential in the number of features to be considered. We believe that using the IML model a simple visual tool can be designed to build classifiers quickly. We hypothesize that when using the IML, having a very fast training algorithm is more important than strong induction. In place of careful analysis of many feature combinations we provide much more human input to correct errors as they appear. This allows the interactive cycle to be iterated quickly so it can be done more frequently.

The remainder of the paper is as follows. The next section briefly discusses the visual tool we created using the IML model, called Image Processing with Crayons (Crayons). This is done to show one application of the IML model's power and versatility. Following the explanation of Crayons, we explore the details of the IML model by examining its distinction from CML, the problems it must overcome, and its implementation details. Finally we present some results from some tests between two of the implemented machine learning algorithms. From these results we base some preliminary conclusions of IML as it relates to Crayons.

## IMAGE PROCESSING WITH CRAYONS

Crayons is a system we created that uses IML to create image classifiers. Crayons is intended to aid UI designers who do not have detailed knowledge of image processing and machine learning. It is also intended to accelerate the efforts of more knowledgeable programmers.

There are two primary goals for the Crayons tool: 1) to allow the user to create an image/pixel classifier quickly, and 2) to allow the user to focus on the classification problem rather than image processing or algorithms. Crayons is successful if it takes minutes rather than weeks or months to create an effective classifier. For simplicity sake, we will refer to this as the UI principle of *fast and focused*. This principle refers to enabling the designer to quickly accomplish his/her task while remaining focused solely on that task.

Figure 3 shows the Crayons design process. Images are input into the Crayons system, which can then export the generated classifier. It is assumed the user has already taken digital pictures and saved them as files to import into the system, or that a camera is set up on the machine running Crayons, so it can capture images from it. Exporting the classifier is equally trivial, since our implementation is written in Java. The classifier object is simply serialized and output to a file using the standard Java mechanisms.

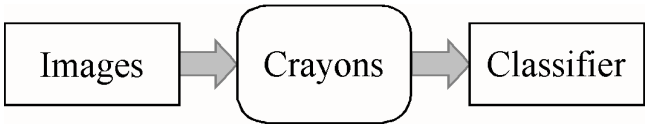


**Figure 3** – Classifier Design Process

An overview of the internal architecture of Crayons is shown in Figure 4. Crayons receives images upon which the user does some manual classification, a classifier is created, then feedback is displayed. The user can then refine the classifier by adding more manual classification or, if the classifier is satisfactory, the user can export the classifier. The internal loop shown in Figure 4 directly correlates to the aforementioned train, feedback, correct cycle of the IML (see Figure 2). To accomplish the *fast and focused* UI principle, this loop must be easy and quick to cycle through. To be interactive the training part of the loop must take less than five seconds and generally much faster. The cycle can be broken down into two components: the UI and the Classifier. The UI component needs to be simple so the user can remain focused on the classification problem at hand. The classifier creation needs to be fast and efficient so the user gets feedback as quickly as possible, so they are not distracted from the classification problem.

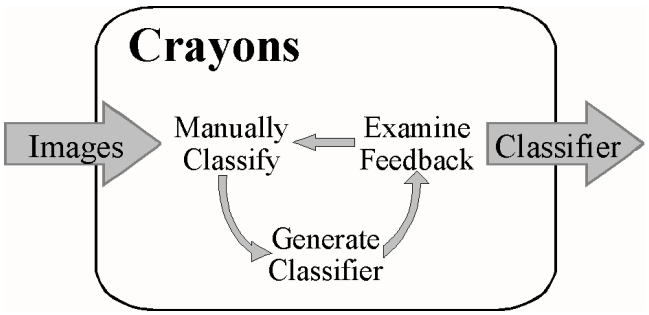


**Figure 4** – The classification design loop

Although the IML and the machine-learning component of Crayons are the primary discussion of this paper it is notable to mention that Crayons has profited from work done by Viola and Jones [19] and Jaimes and Chang [5,6,7]. Also a brief example of how Crayons can be used is illustrative. The sequence of images in Figure 5 shows the process of creating a classifier using Crayons.

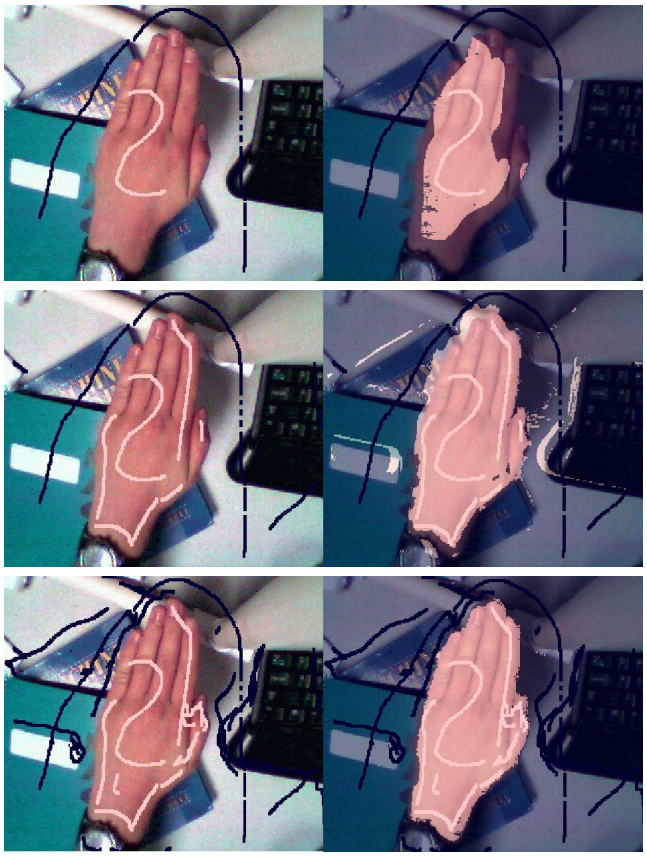

**Figure 5** – Crayons interaction process

Figure 5 illustrates how the user initially paints very little data, views the feedback provided by the resulting classifier, corrects by painting additional class pixels and then iterates through the cycle. As seen in the first image pair in Figure 5, only a little data can generate a classifier that roughly learns skin and background. The classifier, however, over-generalizes in favor of background; therefore, in the second image pair you can see skin has been painted where the classifier previously did poorly at classifying skin. The resulting classifier shown on the right of the second image pair shows the new classifier classifying most of the skin on the hand, but also classifying some of the background as skin. The classifier is corrected again, and the resulting classifier is shown as the third image pair in the sequence. Thus, in only a few iterations, a skin classifier is created.

The simplicity of the example above shows the power that Crayons has due to the effectiveness of the IML model. The key issue in the creation of such a tool lies in quickly generating effective classifiers so the interactive design loop can be utilized.

## MACHINE LEARNING

For the IML model to function, the classifier must be generated quickly and be able to generalize well. As such we will first discuss the distinctions between IML and CML, followed by the problems IML must overcome because of its interactive setting, and lastly its implementation details including specific algorithms.

### CML vs. IML

Classical machine learning generally has the following assumptions.

- There are relatively few carefully chosen features,
- There is limited training data,
- The classifier must amplify that limited training data into excellent performance on new training data,
- Time to train the classifier is relatively unimportant as long as it does not take too many days.

None of these assumptions hold in our interactive situation. Our UI designers have no idea what features will be appropriate. In fact, we are trying to insulate them from knowing such things. In our current Crayons prototype there are more than 150 features per pixel. To reach the breadth of application that we desire for Crayons we project over 1,000 features will be necessary. The additional features will handle texture, shape and motion over time. For any given problem somewhere between three and fifteen of those features will actually be used, but the classifier algorithm must automatically make this selection. The classifier we choose must therefore be able to accommodate such a large number of features, and/or select only the best features.

In Crayons, when a designer begins to paint classes on an image a very large number of training examples is quickly generated. With 77K pixels per image and 20 images one can rapidly generate over a million training examples. In practice, the number stays in the 100K examples range because designers only paint pixels that they need to correct rather than all pixels in the image. What this means, however, is that designers can generate a huge amount of training data very quickly. CML generally focuses on the ability of a classifier to predict correct behavior on new data. In IML, however, if the classifier's predictions for new data are wrong, the designer can rapidly make those corrections. By rapid feedback and correction the classifier is quickly (in a matter of minutes) focused onto the desired behavior. The goal of the classifier is not to *predict* the designer's intent into new situations but rapidly *reflect* intent as expressed in concrete examples.

Because additional training examples can be added so readily, IML's bias differs greatly from that of CML. Because it extrapolates a little data to create a classifier that will be frequently used in the future, CML is very concerned about overfit. Overfit is where the trained classifier adheres

too closely to the training data rather than deducing general principles. Cross-validation and other measures are generally taken to minimize overfit. These measures add substantially to the training time for CML algorithms. IML's bias is to include the human in the loop by facilitating rapid correction of mistakes. Overfit can easily occur, but it is also readily perceived by the designer and instantly corrected by the addition of new training data in exactly the areas that are most problematic. This is shown clearly in Figure 5 where a designer rapidly provides new data in the edges of the hand where the generalization failed.

Our interactive classification loop requires that the classifier training be very fast. To be effective, the classifier must be generated from the training examples in under five seconds. If the classifier takes minutes or hours, the process of 'train-feedback-correct' is no longer interactive, and much less effective as a design tool. Training on 100,000 examples with 150 features each in less than five seconds is a serious challenge for most CML algorithms.

Lastly, for this tool to be viable the final classifier will need to be able to classify 320 x 240 images in less than a fourth of a second. If the resulting classifier is much slower than this it becomes impossible to use it to track interactive behavior in a meaningful way.

#### IML Implementation

Throughout our discussion thus far, many requirements for the machine-learning algorithm in IML have been made. The machine-learning algorithm must:

- learn/train very quickly,
- accommodate 100s to 1000s of features,
- perform feature selection,
- allow for tens to hundreds of thousands of training examples.

These requirements put firm bounds on what kind of a learning algorithm can be used in IML. They invoke the fundamental question of which machine-learning algorithm fits all of these criteria. We discuss several options and the reason why they are not viable before we settle on our algorithm of choice: decision trees (DT).

Neural Networks [12] are a powerful and often used machine-learning algorithm. They can provably approximate any function in two layers. Their strength lies in their abilities to intelligently integrate a variety of features. Neural networks also produce relatively small and efficient classifiers, however, there are not feasible in IML. The number of features used in systems like Crayons along with the number of hidden nodes required to produce the kinds of classifications that are necessary completely overpowers this algorithm. Even more debilitating is the training time for neural networks. The time this algorithm takes to converge is far to long for interactive use. For 150 features this can take hours or days.

The nearest-neighbor algorithm [1] is easy to train but not very effective. Besides not being able to discriminate amongst features, nearest-neighbor has serious problems in high dimensional feature spaces of the kind needed in IML and Crayons. Nearest-neighbor generally has a classification time that is linear in the number of training examples which also makes it unacceptably slow.

There are yet other algorithms such as boosting that do well with feature selection, which is a desirable characteristic. While boosting has shown itself to be very effective on tasks such as face tracing [18], its lengthy training time is prohibitive for interactive use in Crayons.

There are many more machine-learning algorithms, however, this discussion is sufficient to preface to our decision of the use of decision trees. All the algorithms discussed above suffer from the curse of dimensionality. When many features are used (100s to 1000s), their creation and execution times dramatically increase. In addition, the number of training examples required to adequately cover such high dimensional feature spaces would far exceed what designers can produce. With just one decision per feature the size of the example set must approach $2^{100}$, which is completely unacceptable. We need a classifier that rapidly discards features and focuses on the 1-10 features that characterize a particular problem.

Decision trees [10] have many appealing properties that coincide with the requirements of IML. First and foremost is that the DT algorithm is fundamentally a process of feature selection. The algorithm operates by examining each feature and selecting a decision point for dividing the range of that feature. It then computes the "impurity" of the result of dividing the training examples at that decision point. One can think of impurity as measuring the amount of confusion in a given set. A set of examples that all belong to one class would be pure (zero impurity). There are a variety of possible impurity measures [2]. The feature whose partition yields the least impurity is the one chosen, the set is divided and the algorithm applied recursively to the divided subsets. Features that do not provide discrimination between classes are quickly discarded. The simplicity of DTs also provides many implementation advantages in terms of speed and space of the resulting classifier.

Quinlan's original DT algorithm [10] worked only on features that were discrete (a small number of choices). Our image features do not have that property. Most of our features are continuous real values. Many extensions of the original DT algorithm, ID3, have been made to allow use of real–valued data [4,11]. All of these algorithms either discretize the data or by selecting a threshold $T$ for a given feature $F$ divide the training examples into two sets where $F<T$ and $F>=T$. The trick is for each feature to select a value $T$ that gives the lowest impurity (best classification improvement). The selection of $T$ from a large number of features and a large number of training examples is very slow to do correctly.

We have implemented two algorithms, which employ different division techniques. These two algorithms also represent the two approaches of longer training time with better generalization vs. shorter training time with poorer generalization. The first strategy slightly reduces interactivity and relies more on learning performance. The second relies on speed and interactivity. The two strategies are Center Weighted (CW) and Mean Split (MS).

Our first DT attempt was to order all of the training examples for each feature and step through all of the examples calculating the impurity as if the division was between each of the examples. This yielded a minimum impurity split, however, this generally provided a best split close to the beginning or end of the list of examples, still leaving a large number of examples in one of the divisions. Divisions of this nature yield deeper and more unbalanced trees, which correlate to slower classification times. To improve this algorithm, we developed Center Weighted (CW), which does the same as above, except that it more heavily weights central splits (more equal divisions). By insuring that the split threshold is generally in the middle of the feature range, the resulting tree tends to be more balanced and the sizes of the training sets to be examined at each level of the tree drops exponentially.

CW DTs do, however, suffer from an initial sort of all training examples for each feature, resulting in a O($f$ * N log N) cost up front, where $f$ is the number of features and N the number of training examples. Since in IML, we assume that both $f$ and N are large, this can be extremely costly.

Because of the extreme initial cost of sorting all N training examples $f$ times, we have extended Center Weighted with CWSS. The 'SS' stand for sub-sampled. Since the iteration through training examples is purely to find a good split, we can sample the examples to find a statistically sound split. For example, say N is 100,000, if we sample 1,000 of the original N, sort those and calculate the best split then our initial sort is 100 times faster. It is obvious that a better threshold could be computed using all of the training data, but this is mitigated by the fact that those data items will still be considered in lower levels of the tree. When a split decision is made, all of the training examples are split, not just the sub-sample. The sub-sampling means that each node's split decision is never greater than O($f$*1000*5), but that eventually all training data will be considered.

Quinlan used a sampling technique called "windowing". Windowing initially used a small sample of training examples and increased the number of training examples used to create the DT, until all of the original examples were classified correctly [11]. Our technique, although similar, differs in that the number of samples is fixed. At each node in the DT a new sample of fixed size is drawn, allowing misclassified examples in a higher level of the DT to be considered at a lower level.

The use of sub-sampling in CWSS produced very slight differences in classification accuracy as compared to CW, but reduced training time by a factor of at least two (for training sets with N ≥ 5,000). This factor however will continue to grow as N increases. (For N = 40,000 CWSS is approximately 5 times faster than CW; 8 for N = 80,000.)

The CW and CWSS algorithms spend considerable computing resources in trying to choose a threshold value for each feature. The Mean Split (MS) algorithm spends very little time on such decisions and relies on large amounts of training data to correct decisions at lower levels of the tree. The MS algorithm uses $T$=mean($F$) as the threshold for dividing each feature $F$ and compares the impurities of the divisions of all features. This is very efficient and produces relatively shallow decision trees by generally dividing the training set in half at each decision point. Mean split, however, does not ensure that the division will necessarily divide the examples at points that are meaningful to correct classification. Successive splits at lower levels of the tree will eventually correctly classify the training data, but may not generalize as well.

The resulting MS decision trees are not as good as those produced by more careful means such as CW or CWSS. However, we hypothesized, that the speedup in classification would improve interactivity and thus reduce the time for designers to train a classifier. We believe designers make up for the lower quality of the decision tree with the ability to correct more rapidly. The key is in optimizing designer judgment rather than classifier predictions. MSSS is a sub-sampled version of MS in the same manner as CWSS. In MSSS, since we just evaluate the impurity at the mean, and since the mean is a simple statistical value, the resulting divisions are generally identical to those of straight MS.

As a parenthetical note, another important bottleneck that is common to all of the classifiers is the necessity to calculate all features initially to create the classifier. We made the assumption in IML that all features are pre-calculated and that the learning part will find the distinguishing features. Although, this can be optimized so it is faster, all algorithms will suffer from this bottleneck.

There are many differences between the performances of each of the algorithms. The most important is that the CW algorithms train slower than the MS algorithms, but tend to create better classifiers. Other differences are of note though. For example, the sub sampled versions, CWSS and MSSS, generally allowed the classifiers to be generated faster. More specifically, CWSS was usually twice as fast as CW, as was MSSS compared to MS.

Because of the gains in speed and lack of loss of classification power, only CWSS and MSSS will be used for comparisons. The critical comparison is to see which algorithm allows the user to create a satisfactory classifier the fastest. User tests comparing these algorithms are outlined and presented in the next section.

## EVALUATIONS

User tests were conducted to evaluate the differences between CWSS and MSSS. When creating a new perceptual interface it is not classification time that is the real issue. The important issue is designer time. As stated before, classification creation time for CWSS is longer than MSSS, but the center-weighted algorithms tend to generalize better than the mean split algorithms. The CWSS generally takes 1-10 seconds to train on training sets of 10,000-60,000 examples, while MSSS is approximately twice as fast on the same training sets. These differences are important; as our hypothesis was that faster classifier creation times can overcome poorer inductive strength and thus reduce overall designer time.

To test the difference between CWSS and MSSS we used three key measurements: wall clock time to create the classifier, number of classify/correct iterations, and structure of the resulting tree (depth and number of nodes). The latter of these three corresponds to the amount of time the classifier takes to classify an image in actual usage.

In order to test the amount of time a designer takes to create a good classifier, we need a standard to define "good classifier". A "gold standard" was created for four different classification problems: skin-detection, paper card tracking, robot car tracking and laser tracking. These gold standards were created by carefully classifying pixels until, in human judgment, the best possible classification was being performed on the test images for each problem. The resulting classifier was then saved as a standard.

Ten total test subjects were used and divided into two groups. The first five did each task using the CWSS followed by the MSSS and the remaining five MSSS followed by CWSS. The users were given each of the problems in turn and asked to build a classifier. Each time the subject requested a classifier to be built that classifier's performance was measured against the performance of the standard classifier for that task. When the subject's classifier agreed with the standard on more than 97.5% of the pixels, the test was declared complete.

Table 1, shows the average times and iterations for the first group, Table 2, the second group.

| | CWSS | | MSSS | |
|---|---|---|---|---|
| **Problem** | **Time** | **Iterations** | **Time** | **Iterations** |
| Skin | 03:06 | 4.4 | 10:35 | 12.6 |
| Paper Cards | 02:29 | 4.2 | 02:23 | 5.0 |
| Robot Car | 00:50 | 1.6 | 01:00 | 1.6 |
| Laser | 00:46 | 1.2 | 00:52 | 1.4 |

**Table 1** – CWSS followed by MSSS

| | MSSS | | CWSS | |
|---|---|---|---|---|
| **Problem** | **Time** | **Iterations** | **Time** | **Iterations** |
| Skin | 10:26 | 11.4 | 03:51 | 3.6 |
| Paper Cards | 04:02 | 5.0 | 02:37 | 2.6 |
| Robot Car | 01:48 | 1.2 | 01:37 | 1.2 |
| Laser | 01:29 | 1.0 | 01:16 | 1.0 |

**Table 2** – MSSS followed by CWSS

The laser tracker is a relatively simple classifier because of the uniqueness of bright red spots [9]. The robot car was contrasted with a uniform colored carpet and was similarly straightforward. Identifying colored paper cards against a cluttered background was more difficult because of the diversity of the background. The skin tracker is the hardest because of the diversity of skin color, camera over-saturation problems and cluttered background [20].

As can be seen in tables 1 and 2, MSSS takes substantially more designer effort on the hard problems than CWSS. All subjects specifically stated that CWSS was "faster" than MSSS especially in the Skin case. (Some did not notice a difference between the two algorithms while working on the other problems.) We did not test any of the slower algorithms such as neural nets or nearest-neighbor. Interactively these are so poor that the results are self-evident. We also did not test the full CW algorithm. Its classification times tend into minutes and clearly could not compete with the times shown in tables 1 and 2. It is clear from our evaluations that a classification algorithm must get under the 10-20 second barrier in producing a new classification, but that once under that barrier, the designer's time begins to dominate. Once the designer's time begins to dominate the total time, then the classifier with better generalization wins.

We also mentioned the importance of the tree structure as it relates to the classification time of an image. Table 3 shows the average tree structures (tree depth and number of nodes) as well as the average classification time (ACT) in milliseconds over the set of test images.

| | CWSS | | | MSSS | | |
|---|---|---|---|---|---|---|
| **Problem** | **Depth** | **Nodes** | **ACT** | **Depth** | **Nodes** | **ACT** |
| Skin | 16.20 | 577 | 243 | 25.60 | 12530 | 375 |
| Paper Cards | 15.10 | 1661 | 201 | 16.20 | 2389 | 329 |
| Car | 13.60 | 1689 | 235 | 15.70 | 2859 | 317 |
| Laser | 13.00 | 4860 | 110 | 8.20 | 513 | 171 |

**Table 3** – Tree structures and average classify time (ACT)

As seen in Table 3, depth, number of nodes and ACT, were all lower in CWSS than in MSSS. This was predicted as CWSS provides better divisions between the training examples.

While testing we observed that those who used the MSSS which is fast but less accurate, first, ended up using more training data, even when they used the CWSS, which usually generalizes better and needs less data. Those who used the CWSS first, were pleased with the interactivity of CWSS and became very frustrated when they used MSSS, even though it could cycle faster through the interactive loop. In actuality, because of the poor generalization of the mean split algorithm, even though the classifier generation time for MSSS was quicker than CWSS, the users felt it necessary to paint more using the MSSS, so the overall time increased using MSSS.

## CONCLUSION

When using machine learning in an interactive design setting, feature selection must be automatic rather than manual and classifier training-time must be relatively fast. Decision Trees using a sub-sampling technique to improve training times are very effective for both of these purposes. Once interactive speeds are achieved, however, the quality of the classifier's generalization becomes important. Using tools like Crayons, demonstrates that machine learning can form an appropriate basis for the design tools needed to create new perceptual user interfaces.

## REFERENCES


1. Cover, T., and Hart, P. "Nearest Neighbor Pattern Classification." *IEEE Transactions on Information Theory*, 13, (1967) 21-27.
2. Duda, R. O., Hart, P. E., and Stork, D. G., *Pattern Classification*. (2001).
3. Fails, J.A., Olsen, D.R. "LightWidgets: Interacting in Everyday Spaces." *Proceedings of IUI '02* (San Francisco CA, January 2002).
4. Fayyad, U.M. and Irani, K. B. "On the Handling of Continuous-valued Attributes in Decision Tree Generation." *Machine Learning*, 8, 87-102,(1992).
5. Jaimes, A. and Chang, S.-F. "A Conceptual Framework for Indexing Visual Information at Multiple Levels." IS&T/SPIE Internet Imaging 2000, (San Jose CA, January 2000).
6. Jaimes, A. and Chang, S.-F. "Automatic Selection of Visual Features and Classifier." *Storage and Retrieval for Image and Video Databases VIII*, IS&T/SPIE (San Jose CA, January 2000).
7. Jaimes, A. and Chang, S.-F. "Integrating Multiple Classifiers in Visual Object Detectors Learned from User Input." Invited paper, session on Image and Video Databases, 4th Asian Conference on Computer Vision (ACCV 2000), Taipei, Taiwan, January 8-11, 2000.
8. Krueger, M. W., Gionfriddo. T., and Hinrichsen, K., "VIDEOPLACE -- an artificial reality". *Human Factors in Computing Systems, CHI '85 Conference Proceedings*, ACM Press, 1985, 35-40.
9. Olsen, D.R., Nielsen, T. "Laser Pointer Interaction." *Proceedings of CHI '01* (Seattle WA, March 2001).
10. Quinlan, J. R. "Induction of Decision Trees." *Machine Learning*, 1(1); 81-106, (1986).
11. Quinlan, J. R. "C4.5: Programs for machine learning." Morgan Kaufmann, San Mateo, CA, 1993.
12. Rumelhart, D., Widrow, B., and Lehr, M. "The Basic Ideas in Neural Networks." *Communications of the ACM*, 37(3), (1994), pp 87-92.
13. Schmidt, A. "Implicit Human Computer Interaction Through Context." Personal Technologies, Vol 4(2), June 2000.
14. Starner, T., Auxier, J. and Ashbrook, D. "The Gesture Pendant: A Self-illuminating, Wearable, Infrared Computer Vision System for Home Automation Control and Medical Monitoring." *International Symposium on Wearable Computing* (Atlanta GA, October 2000).
15. Triggs, B. "Model-based Sonar Localisation for Mobile Robots." Intelligent Robotic Systems '93, Zakopane, Poland, 1993.
16. Underkoffler, J. and Ishii H. "Illuminating Light: An Optical Design Tool with a Luminous-Tangible Interface." Proceedings of CHI '98 (Los Angeles CA, April 1998).
17. Underkoffler, J., Ullmer, B. and Ishii, H. "Emancipated Pixels: Real-World Graphics in the Luminous Room." Proceedings of SIGGRAPH '99 (Los Angeles CA, 1999), ACM Press, 385-392.
18. Vailaya, A., Zhong, Y., and Jain, A. K. "A hierarchical system for efficient image retrieval." In Proc. Int. Conf. on Patt. Recog. (August 1996).
19. Viola, P. and Jones, M. "Robust real-time object detection." Technical Report 2001/01, Compaq CRL, February 2001.
20. Yang, M.H. and Ahuja, N. "Gaussian Mixture Model for Human Skin Color and Its Application in Image and Video Databases." Proceedings of SPIE '99 (San Jose CA, Jan 1999), 458-466.