Interactive Optimization for Steering Machine Classification

Ashish Kapoor, Bongshin Lee, Desney Tan, and Eric Horvitz
Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA
{akapoor, bongshin, desney, horvitz}@microsoft.com

ABSTRACT

Interest has been growing within HCI on the use of machine learning and reasoning in applications to classify such hidden states as user intentions, based on observations. HCI researchers with these interests typically have little expertise in machine learning and often employ toolkits as relatively fixed "black boxes" for generating statistical classifiers. However, attempts to tailor the performance of classifiers to specific application requirements may require a more sophisticated understanding and custom-tailoring of methods. We present ManiMatrix, a system that provides controls and visualizations that enable system builders to refine the behavior of classification systems in an intuitive manner. With ManiMatrix, users directly refine parameters of a confusion matrix via an interactive cycle of reclassification and visualization. We present the core methods and evaluate the effectiveness of the approach in a user study. Results show that users are able to quickly and effectively modify decision boundaries of classifiers to tailor the behavior of classifiers to problems at hand.

Author Keywords
Interactive Machine Learning, Visualization, Decision Theory, Interactive Optimization.

ACM Classification Keywords
H.5.2 [User Interfaces]: Graphical User Interface; I.2.6 [Learning].

General Terms
Algorithms, Human Factors.

INTRODUCTION

Machine learning (ML) methods are becoming increasingly popular amongst HCI researchers who seek to build predictive models from a set of representative training data. In use, the models provide classifications of states of the world or of users that cannot be inspected directly, based on sets of available observations. Researchers using off-the-shelf ML techniques in a "black-box" manner can find that they need to iteratively change various parameters and settings in order to obtain good results. Refining the behavior of the predictive models and the classifications they generate often requires significant understanding of the learning and classification procedures. Currently, the parameters of classification algorithms are often set manually in a process that can be tedious even for ML experts. The task requires enough understanding of the parameter space, and the problem of shaping the behavior of a classifier becomes harder as the number of classes increases. Furthermore, there are often dependencies among parameters, which lead to a complex mapping between the parameter values and the behavior of the system. Such dependencies make it difficult to estimate parameters that might provide desirable results, leading to a trial and error approach to optimizing the behavior of a classifier.

We have been pursuing the creation of interactive techniques that enable users to better understand and to tailor the performance of classifiers via intuitive controls and visualizations. The challenges in such techniques lie in identifying the intuitive controls the user wishes to have, effectively providing those controls, computationally coupling them to the numerical procedures that are linked to model generation or usage, and providing relevant real-time feedback. The method relies on a tight coupling between human input and an embedded optimization procedure that shows how the overall behavior of the classifier changes with adjustments to parameters.

In this paper, we focus on the domain of *multiclass classification*, common in many HCI research problems. For such problems, the goal is to construct and use a statistical classifier that assigns unlabeled states of interest to one of the discrete number of classes, given a set of observations. It is common to evaluate such models using single goodness metrics such as overall accuracy. However, we assert that HCI researchers often need finer-grained control, and could benefit greatly from the ability to express preferences about how a classifier should work. For example, in a gesture recognition system it may be absolutely necessary to detect a small subset of common gestures correctly, while errors in the detection of less frequent gestures might be tolerated. By appropriately setting such parameters as the costs of misclassification of items, it is possible to modify the beha-

CHI 2010, April 10–15, 2010, Atlanta, Georgia, USA.
Copyright 2010 ACM 978-1-60558-929-9/10/04....$10.00.

vior of the algorithm such that it is best aligned with the desired performance of the system.

We introduce ManiMatrix, (for Manipulable Matrix), an interactive system that enables system designers to intuitively refine the behavior of classification systems. ManiMatrix centers on the manual refinement and consistent automated propagation of user refinement intentions on sets of thresholds that are used to translate the probabilistic output of classifiers into classification decisions. These settings can be viewed as assigning sets of costs of misclassification in a confusion matrix. ManiMatrix enables its users to directly interact with a confusion matrix and to view the implications of incremental changes to the matrix via a real-time interactive cycle of re-classification and visualization. The approach allows users to specify preferences about different types of misclassification via a display of densities and distances of items.

ManiMatrix manages the complexity for users by performing inferential "heavy lifting" under the hood to convert the intuitive human-centric inputs into complex settings of multiple parameters that are consistent with users' local refinements. The system employs a fast optimization routine to compute the implications of a user's sequence of refinements on all of the parameters of the confusion matrix, continuing to update parameters in response to a user's inputs. The coupling between interactions and the resulting classification system provides users with an intuition for the structure of the problem and helps them find a solution that is better aligned with their goals. The methodology can be applied to any multiclass classification system (e.g., models generated by Weka [14]).

Specifically, the main contributions of this paper are:

1. Presentation of ManiMatrix, a system that provides intuitive interactive visualization primitives in order to allow users to specify preferences about classifiers.
2. Methods for coupling simple, intuitive interactions in ManiMatrix with efficient optimization routines that allow instant modification of classification boundaries, thus providing real-time feedback to users.
3. Report on a user study conducted with two types of classification tasks. Results showed that users were not only able to use the system, but that they were able to quickly create classifiers aligned with their preferences.

RELATED WORK

A recent survey of 112 HCI professionals reported that about one third have used machine learning in their HCI work [18]. Some of this work seeks to develop novel input modalities by building models that disambiguate and interpret noisy streams of data. Examples include handwriting and speech recognition, vision-based sensing techniques, and muscle-computer interfaces [23]. Machine learning has also been used to infer such hidden user states as affect, intentions, and workload and cost of interruption from user activity, context signals, and/or physiological signals [20, 15, 16, 1216, 17]. Furthermore, researchers have been designing machine learning techniques to support adaptive or mixed-initiative interfaces. These interfaces aim to provide rich automated assistance so that the human-computer system can become more effective than the sum of its parts.

However, effective usage of such machine learning models require adeptness and understanding that usually comes with experience. A recent investigation on the current use of machine learning by non-expert researchers identified three major difficulties: (1) difficulty in applying an iterative exploration process, (2) difficulty in understanding the machine learning models, and (3) difficulty in evaluating performance [19]. We tackle some of these issues in the context of multiclass classification by enabling users to directly encode their detailed preferences about classification actions using an interactive system.

Our work is closely aligned with evolving research on *interactive machine learning*. Fails and Olsen assert the importance of human involvement to provide training data and propose an interactive system that allows users to train, classify, and correct classifications in a real-time iterative loop [10]. Talbot et al. apply the same philosophy to help users easily build an ensemble classifier by combining multiple component classifiers [25]. Their main goal is to achieve high classification accuracy across all classes rather than to refine the behavior of a classifier in accordance with specific preferences. In our work, we address the challenge of providing users with tools to efficiently manipulate the confusion matrix so as to specify preferences, to understand the implications of local refinements on performance, and ultimately to converge in an iterative manner on a classification system that better aligns with their design goals.

Several general approaches have been used for decades to probe the performance of learned classifiers. These include the use of summary statistics of classifiers such as log scores, classification accuracies, *F*-measures, and a variety of other statistics [24,26], ROC curves, area under ROC, and cost curves [7], which support evaluation of model performance as a function of misclassification costs. While these visualizations are commonly used in the machine learning community, they are restricted to binary classification tasks for practical purposes. More importantly, they do not directly support iterative improvement of a classifier. To facilitate the use of statistical classifiers within systems, researchers have tried to visualize data and mathematical representations of the classification model. For example, one approach is to plot the data instances or cases in some projection of feature space and to visualize the boundaries predicted among classes [13, 22]. This approach generalizes across algorithms but is limited in that it does not reveal insights about the internal operation of algorithms. Ware et al. showed that people can produce better classifiers than automatic techniques [27] when assisted by a tool that provides visualizations about the operation of specific machine learning algorithms such as decision trees [2], naïve-Bayes [3], SVMs [4], and HMMs [6].

SPECIFYING FINE-GRAINED PREFERENCES

We consider a general multiclass classification problem, in which the goal is to build a classifier that can accurately assign a label, out of c different choices, to a test data point. Such classification tasks can be performed by first training a classifier from some training corpus and then applying the classifier to unseen test data points. This is typical of the use of ML within HCI applications.

Many researchers rely heavily on the overall classification accuracy of predictive models, often coupled with a consideration of the lift in classification accuracy provided by the predictive models over the accuracy of predictions generated by a marginal model that classifies according to the background statistics of classes. This classification accuracy is calculated by taking the number of correctly labeled instances of test data divided by the total number of instances the model has tried to classify. Given a confusion matrix, classifier performance can be expressed as the sum of test instances correctly classified, and thus placed on the diagonal, divided by the sum of all values in the matrix.

For most real-world problems, achieving perfect classification accuracy is usually impossible. For a variety of reasons, some cases will usually be incorrectly classified. While the classification accuracy metric treats each misclassified instance equally, real-world scenarios often make different misclassifications more or less costly or desirable. For instance, in junk email filtering, misclassifying junk as regular email is typically less costly than misclassifying regular email as junk. There are many examples of HCI applications where the costs of misclassification are non-uniform across classes and these situations highlight the potential value of creating tools that could allow system designers—as well as end users—to specify their preferences about classification in a fine-grained manner. Refinements to performance achievable via the use of such tools may yield more value than efforts aimed at increasing a singular measure of overall classification accuracy.

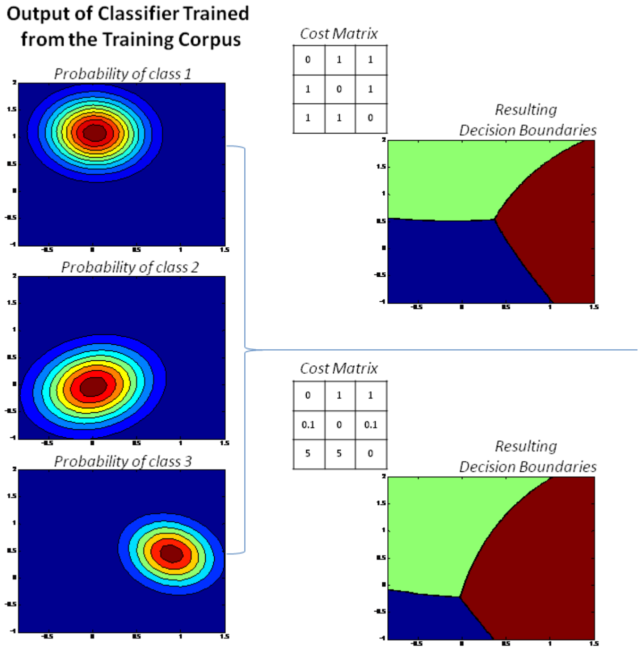
In the most general setting, a classifier generates c different likelihood scores (denoted $p_1, \dots, p_c$), for each test case which correspond to each of the c possible class labels. These scores reflect the confidence of assigning each particular label to the case and can be easily transformed into probabilities using appropriate normalizing techniques (see [21] for example). Here, we assume that these scores have been appropriately normalized. Hence, p_i is the probability that a test case belongs to class *i*. In the simplest form, each test point is assigned the label with the maximum score.

To enable users to specify finer-grained preferences about the process of assigning cases to classes, we leverage prior work on Bayesian decision theory [8]. This methodology depicts the preferences over the classifiers using values that denote the cost or penalty incurred when a data point is misclassified. Since users can specify a cost value for each cell in the confusion matrix generated by the model, we can define a symmetric $c \times c$ structure known as the *cost matrix*. More formally, we denote parameters $Cost_{ij}$ as cost of misclassifying a test case as class j, when its true class was *i*. We represent the costs for all i and j as a real-valued cost matrix $\boldsymbol{C} = [Cost_{ij}]$, where the entry in *i^th^* row and the *j^th^* column corresponds to $Cost_{ij}$.

Given these costs, a decision-theoretic analysis estimates the expected cost or risk of assigning a class label to the test data point by considering the probability distribution output from the classification model for each test case. In particular, the risk of assigning a label combines the classification output with the misclassification cost by considering the probability that the data point belonged to the other classes. The risk of assigning label j to each case can be written as:

$$Risk^j = \sum_{i=1}^{C} p_i \cdot Cost_{ij}$$

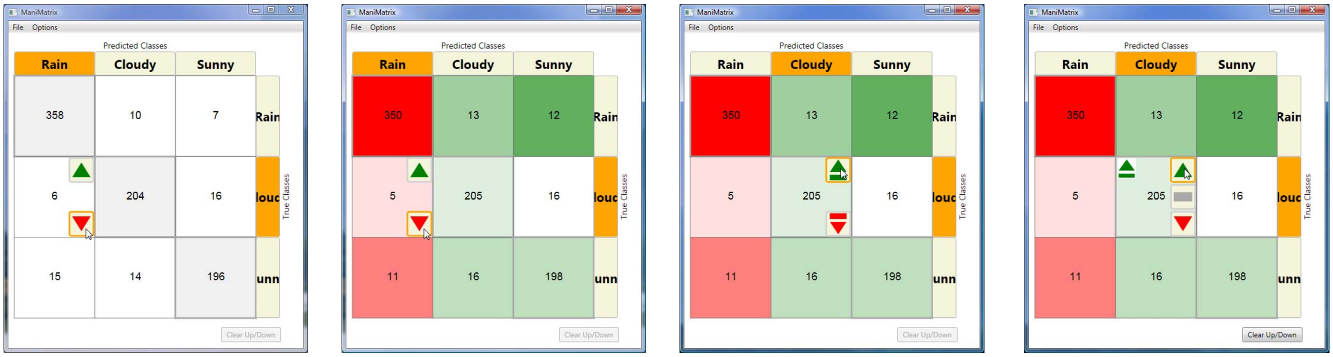
The test case is assigned a label that has the minimum risk. The labeling of the cases based on minimizing risk can be viewed as identifying decision boundaries for classification of cases. Such boundaries shift with changes in the values of $Cost_{ij}$. Users can specify their preferences over the classifier space by refining the values of the costs. By increasing the value of $Cost_{ij}$ we shift a decision boundary so as to make the classifier require a higher likelihood of label j being the true class before assigning label j to a test case when its true label is i. Similarly lowering $Cost_{ij}$ tends to favor labeling data points as class j, accepting a case as a class at a lower inferred probability.



0	1	1
1	0	1
1	1	0

0	1	1
0.1	0	0.1
5	5	0

Figure 1. From probabilities to classification boundaries via a cost matrix. Probabilities across each class (left) are combined with the cost matrix to generate decision boundaries (right) that capture a designer's preferences about classification.



(a)

Rain	Cloudy	Sunny	
358	10	7	Rain
6	204	16	louc
15	14	196	unn

(b)

Rain	Cloudy	Sunny	
350	13	12	Rain
5	205	16	louc
11	16	198	unn

(c)

Rain	Cloudy	Sunny	
350	13	12	Rain
5	205	16	louc
11	16	198	unn

(d)

Rain	Cloudy	Sunny	
350	13	12	Rain
5	205	16	louc
11	16	198	unn

Figure 2. Interacting with ManiMatrix.

Figure 1 shows how different cost matrices can result in different classification boundaries. The figures on the left depict the estimated probabilities over a 2-dimensional space for three different classes. These probabilities are derived from an underlying classification system that was built using a training corpus. We compute an expected cost of using the classifiers on the test sets by combining these probabilities with different cost matrices in order to produce different classification boundaries. We see that different settings of cost shift the boundary in order to minimize the classification risk.

Unfortunately, specifying such fine-grained preferences via the cost matrix can be tedious. A *c* class classification problem requires the user to specify c^2 parameters which becomes infeasible as *c* becomes even marginally large. Furthermore, setting these parameters by hand can be challenging as the classification model and the costs interact in complex, non-linear ways which is often unpredictable, even to expert users. In some scenarios, one might estimate such parameters using monetary considerations (such as direct profit or loss). However, such considerations are hard to make in various HCI settings where the cost of the misclassification can correspond to such outcomes as user annoyance, frustration, usability, and other subjective metrics.

MANIMATRIX

ManiMatrix is an interactive system that allows users to directly manipulate the confusion matrix in order to specify preferences and explore the classification space. The system consists of a visualization and control interface joined with an optimization algorithm that computes the global implications of a user's local refinements, enabling users to make changes and to understand how the predictive model interacts with their preferences (Figure 2).

Interacting with the Confusion Matrix

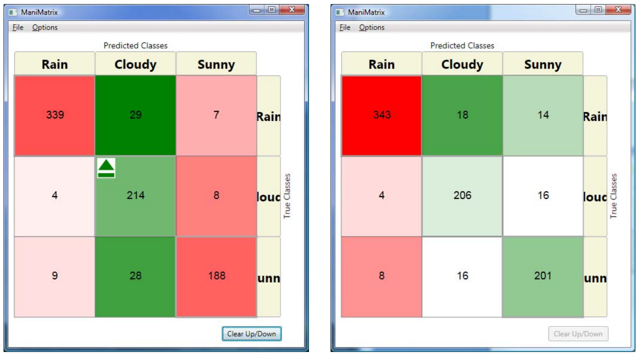
At the core of ManiMatrix is a confusion matrix, which represents classification results by aggregating instances within a grid. Each row in the matrix represents an instance's true class and each column an instance's predicted class. For example, Figure 2a (see left-most cell in the middle row of the matrix) shows that 6 cloudy days were misclassified as rainy within a party location planning problem. The confusion matrix is a common visualization because it is easy to interpret and can be used with any classification algorithm. Other visualizations may also serve as the basis for building insights and encoding preferences about classification. We leave exploration of such visualizations as future work.

Depending on their preferences, users can specify an increase or decrease in the tolerance for numbers of cases classified into each cell. For example, if users want to prevent the cloudy days from being classified as rainy, they want to have as small a number as possible in the middle left cell. ManiMatrix supports this by allowing them to specify this desire with a single click directly on the confusion matrix. When users move the mouse pointer over a cell, ManiMatrix shows a green up arrow and a red down arrow on the right side of the cell (Figure 2a). Each click corresponds to the desire to increment or decrement the value in that cell by 1. When users click on either button, ManiMatrix recomputes the decision boundaries for all cases, working to satisfy the confusion matrix that accommodates the user request. This is done at interactive rates and users receive immediate feedback.

If ManiMatrix successfully finds a feasible confusion matrix, it updates the visualization; otherwise it notifies users that the request is not feasible. For example, Figure 2b shows the new confusion matrix after the user clicked on the down button from the middle left cell. To facilitate large desired value changes, ManiMatrix repeats this click interaction if users press and hold the button.

Operations, even on a single cell, typically lead to changes in other parts of the matrix. To show changes in each cell, ManiMatrix provides feedback by highlighting the cells whose values have changed; green represents an increase and red a decrease. The magnitude of change is represented by the opacity; the bigger the difference is, the more opaque.

It is important to note that multiple solutions may be consistent with a user's preferences. In the current version of ManiMatrix, a solution of parameters is generated that maximizes the stability of the matrix, minimizing the overall change in value as much as possible. As a result, the up



	Rain	Cloudy	Sunny
Rain	339	29	7
louc	4	214	8
unn	9	28	188

	Rain	Cloudy	Sunny
Rain	343	18	14
louc	4	206	16
unn	8	16	201

Figure 3. Effects of the direction lock. The down arrow was pressed for the cell in the middle row and the leftmost column. The result with the directional lock (left) showed much bigger gain than case without direction lock (right).

and down operations are not necessarily reversible. For example, clicking on the up arrow from the middle left cell in Figure 2b does not necessarily return to the original matrix shown in Figure 2a. Hence, ManiMatrix supports undo (Ctrl-Z) and redo (Ctrl-Y) to enable users to roll back states.

A change in one cell affects one or more other cells. When a cell is changed in an indirect manner, users may want to encourage the cell value to be changed in a specific direction. For example, while preventing the cloudy days from being classified as rainy days, users may also want to specify that as many cloudy days as possible should be classified correctly. ManiMatrix supports this by allowing users to specify the desired direction of the value changes for each cell. As opposed to the strong constraint of directly incrementing or decrementing a cell, these hints are *biasing cues*.

When users press the Control key on the keyboard, icons of the up and down buttons change so that users can lock the cell with a desired direction (Figure 2c). Once users click on the direction button with the Control key pressed, ManiMatrix adds a direction lock icon at the top left corner of the cell to show that a lock is set for that cell (Figure 2d). Figure 3 illustrates the effects of the use of the direction lock; clicking on the down arrow from the middle left cell with the direction lock set in the center cell (Figure 3, left) gives a bigger value change in the center cell than the one without the direction lock (Figure 3, right).

When the lock is set, the clear lock button is shown between the up and down buttons (Figure 2d) so that users can clear the individual lock set in the cell. Users can also clear all of the direction locks using the Clear Up/Down button at the bottom of the interface.

Learning Preferences from User Interactions

Now that we have described the functionality and operation of the interface, we present the methodology we use to modify the cost matrix in response to user preferences so as to achieve desired classifier behavior.

We first derive an objective function parameterized by the cost matrix. This function represents how well a given cost matrix matches the user intention. Thus, we can aim to satisfy user intentions by modifying the current cost matrix such that the value of the objective function increases. We use a *gradient-ascent* methodology. Specifically, given an objective function, we compute gradients with respect to the parameters of the cost matrix and take a step in a direction that would increase the value of the objective function.

Objective Function

The objective function reflects how well a cost matrix satisfies the user's preferences, as expressed by sequences of refinements. Since users are directly manipulating the confusion matrix, and they focus on how individual cases are being classified, we express the objective in terms of a *fitness function* over every case.

When users click on the up or down arrow in the confusion matrix, they express a desire for the number of instances in that particular cell to change. In response to a user's input, we compute a desirable state ***t*** every case should be in. This state is a vector with dimensions equal to the number of classes. Similarly, given the current cost matrix, we also compute the current state of every case (represented as ***o***). This state vector is a direct function of the current cost parameters. We include a detailed mathematical description of these states in Appendix A. Given these original and target states, our objective function is the sum of individual fitness functions over all *N* data points:

$$objective = \sum_{n=1}^{N} fitness(\boldsymbol{t}^n, \boldsymbol{o}^n)$$

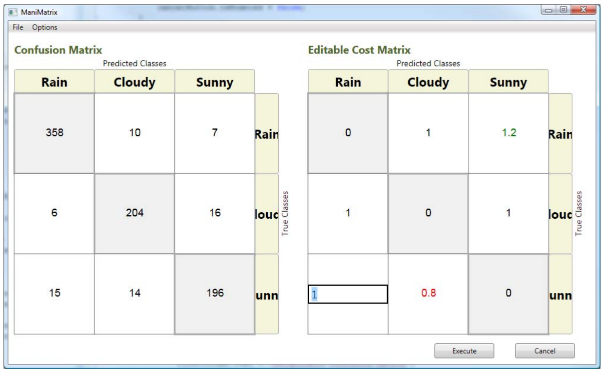
The fitness function applied to each point measures how well that point aligns with user preferences. In particular, the fitness function measures how 'near' the current state ***o*** is to the target state ***t***. This, 'nearness' property can be captured by the negative of KL-divergence, which is used often as a means of comparing probability distributions [5,8]. This can be formally represented as:

$$fitness(\boldsymbol{t}, \boldsymbol{o}) = \sum_{i=1}^{c} - t_i log \frac{t_i}{o_i}$$

Note that the maximum value attained by the fitness function is zero and is achieved when the current state ***o*** is same as the target state ***t***. In all other cases, the fitness function has a negative value.

Given this objective function, we compute gradients and modify the cost matrix such that we increase the value of the objective function. Appendix B shows how the gradients of the objective function with respect to the costs can be computed. These gradients provide directions which promise to increase the value of the objective function.

Aside from the direction of movement, we also consider another parameter: the step-size, or how much we should move in the ascending direction. In our work, we employ binary search in order to determine the step-size. Specifically, the system determines a step size such that the number



Confusion Matrix

Predicted Classes

Rain	Cloudy	Sunny	True Classes
358	10	7	Rain
6	204	16	louc
15	14	196	unn

Editable Cost Matrix

Predicted Classes

Rain	Cloudy	Sunny	True Classes
0	1	1.2	Rain
1	0	1	louc
1	0.8	0	unn

Figure 4. CostMatrix interface with the associated confusion matrix view on the left.

in the particular cell where the user pressed the arrow key increments or decrements exactly by one.

This search for the step size only considers the strong constraints and not the biasing cues, as the idea is that the strong constraints need to be satisfied while both the biasing cues and strong constraints are used to compute the ascent direction. Finally, in case the binary search does not find a step size that results in the required change, the algorithm flags an "infeasible" condition. If a feasible step size α is found, then the cost parameters are updated as follows:

$$Cost_{ij}^{new} = Cost_{ij} + \alpha \cdot \frac{\partial \; objective}{\partial \; Cost_{ij}}$$

We can think of the whole procedure as a type of human-in-the-loop optimization. The user interacts with the system and that interaction determines the ascent direction, while the system carries out optimization and determines the correct step size to take. This numerical procedure in conjunction with the interactive visualization results in a system that users can interact and modify classifiers according to their own preference.

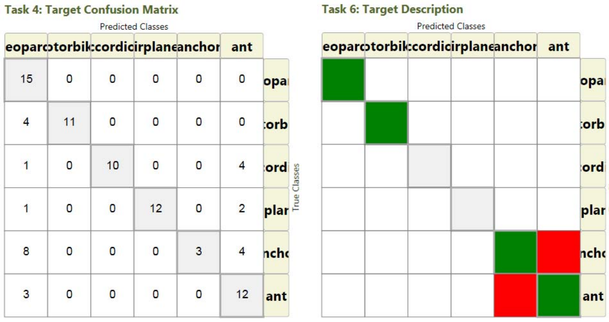
Avoiding Overfitting and Generalization on Unseen Data

One of the concerns in building and using statistical classifiers is overfitting. Carrying out too many operations and fine-tuning of classifiers just based on the training set can lead to poor performance on previously unseen test cases.

In order to avoid overfitting and to guarantee good generalization, we employ a classic *leave-one-out* technique to generate the confusion matrix that users interact with. Every case in the training set is classified using a classifier trained using the rest of the cases, and the confusion matrix summarizes these leave-one-out results. Operations carried out on the leave-one-out matrix are statistically robust. Thus, we expect that all of the operations performed via ManiMatrix will generalize to the other unseen points [9].

USER STUDY

We conducted a user study to explore how ManiMatrix influences the performance of users trying to adjust a model and attain specific configurations of the confusion matrix. We compared the use of the prototype with the more traditional method of directly specifying values within a cost matrix. We implemented the CostMatrix interface (Figure 4) that allows users to directly edit cell values in a separate pane from the confusion matrix. When users click on a cell, CostMatrix provides a text box for them to enter a value. After specifying one or more new values, users can execute their changes by pressing the Execute button, which updates the confusion matrix view on the left.



Task 4: Target Confusion Matrix

Predicted Classes

eoparc	otorbik	cordic	irplane	anchor	ant	True Classes
15	0	0	0	0	0	opa
4	11	0	0	0	0	:orb
1	0	10	0	0	4	:ord
1	0	0	12	0	2	plar
8	0	0	0	3	4	ncho
3	0	0	0	0	12	ant

Task 6: Target Description

Predicted Classes

eoparc	otorbik	cordic	irplane	anchor	ant	True Classes
🟩						opa
	🟩					:orb
						:ord
						plar
				🟩	🟥	ncho
				🟥	🟩	ant

Figure 5. Our two tasks: a fully specified target confusion matrix (left) and a more abstract, but more realistic desire to move the value of certain cells in certain directions (right).

We tested the interfaces with two different sized problems to investigate how well they scale. We also explored the effects of providing (color) feedback that shows how each cell within the confusion matrix changed in each iteration.

Methodology

We ran the study as a 2 (Interface: ManiMatrix vs. CostMatrix) × 2 (Feedback: Color vs. NoColor) × 2 (Size: Small vs. Large) within-subjects design. Small problems were 3-class problems (a 9-cell confusion matrix; e.g., Figure 2) and the Large ones were 6-class problems (a 36-cell confusion matrix; e.g., Figure 5). We independently counter-balanced the Interface and Feedback manipulations, and ran the Small problem before the Large one within each condition. We explained each interface to participants the first time they used it, and had them do a practice task with the Small problem before performing test tasks with the interface. To keep the study at a reasonable length, we imposed a time limit for each task: 2 minutes for the Small problem and 4 minutes for the Large one. If participants hit the time limit, our study application automatically stopped them and took the confusion matrix at that point as the final answer.

We tested the conditions with 2 different tasks. In one task, we provided the user with a specific target confusion matrix, which they were asked to match using the interface (Figure 5 Left). These target matrices were created by randomly adjusting values in the cells of the underlying cost matrix for each problem. In the other task, we instead specified that the participant should aim to increase or decrease the values as much as possible in certain cells of the matrix (Figure 5 Right). We created these by generating ecologically valid scenarios such as considering the identification of certain classes as highly important or maximizing separation between two groups of classes. While we could have probably created an arbitrary problem to test the interfaces,

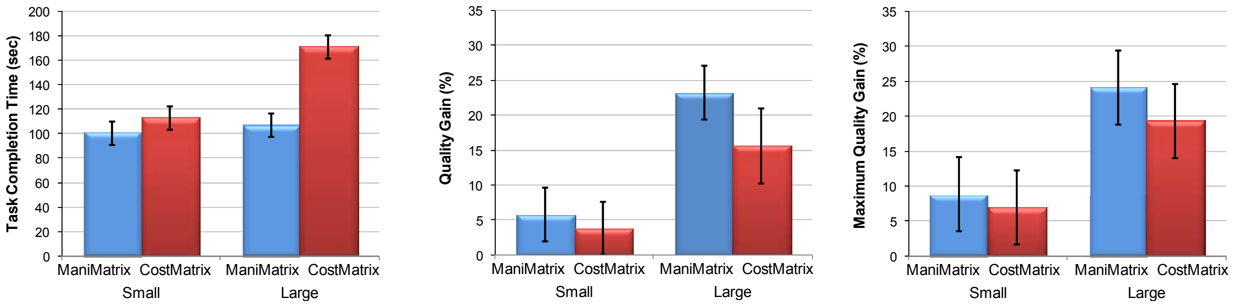


Figure 6. Means of performance metrics show ManiMatrix allows users to perform tasks faster (left) and more effectively (center and right) than with CostMatrix. Error bars represent standard error.

we found it easier to work with data from an existing problem. In our experiments, we used a subset of Newsgroup-20 [1] to generate the 3-class problem and a subset of Caltech-101 [11] dataset to generate the 6-class problem. The underlying classification was based on Gaussian Processes and we use leave-one-out confusion matrices in the study.

To counter-balance the order of Interface and Feedback, we prepared 4 sets of tasks; each set consisted of 4 tasks (2 for the Small problem and 2 for the Large one). Since we were not trying to compare efficacy across tasks, we always tested the first task before the second one.

Each task was presented as a matrix (Figure 5) at the left side of ManiMatrix or CostMatrix in our study application. When participants were ready to begin, they pressed the “Start" button. Upon the completion of each task, either when participants clicked the “Done” button or when the task timed out, the study proceeded onto the next task.

We logged all actions and collected several metrics while participants performed the tasks. These include task completion time as well as the number and type of operations performed in each interface. We also logged the cost and confusion matrices after each operation so that we could calculate the overall increase (or decrease) in quality of the confusion matrix at the end of the trial, as well as locate the maximum gain seen during the course of each trial. At the end of the study, we asked participants to complete a questionnaire to collect feedback about their experiences.

Participants and Apparatus

We recruited sixteen participants via an internal mailing list for those interested in machine learning at a local software company. The average age of participants was 33.4, ranging from 21 to 48 years. 8 of the participants were researchers, 4 were interns, and 4 were software developers. All had worked on at least one machine learning project.

We ran participants in pairs, with each working on a 3.16 GHz quad-core Dell T5400, with 8 GB RAM and 512 MB Video memory, running Windows XP, and using 24” Dell monitors at a resolution of 1920×1200. Since we had to put maximum of 3 matrices for the CostMatrix interface, the size of each matrix was 618×618. The study lasted 90 minutes, and participants were given a gratuity for their efforts.

Results: Learning Classification Preferences

We explore the results of our study in three parts. First, we analyze performance metrics to explore efficacy of each of the interfaces. Then, we inspect behavioral results to better understand usage within the various conditions. Finally, we look at subjective ratings and comments.

Performance Results

We first looked at core participant performance within our various conditions using three different dependent measures. For each of these, we used a mixed-model analysis of variance (ANOVA) with Interface, Feedback, and Size as fixed effects. We included Participant and Task as random effects. Modeling Participant accounts for variation in individual performance, and modeling Task accounts for any difference in difficulty of the two task formulations.

We first examine task completion time, which is measured as the time elapsed between the initial presentation of the problem to the moment the participant clicked on the “Done” button to commit their answer. When participants bumped up against the time limit for each of the tasks, we took that upper bound as their completion time. Since the times were positively skewed, we performed a log transform of the data prior to analysis. We found a significant effect of Interface ($F_{1,233}$=45.83, p<.001), with ManiMatrix leading to faster performance than CostMatrix (103.7 vs. 141.8 secs, respectively). Participants hit the deadline more when using CostMatrix than for ManiMatrix (3.25 vs. 1.625, respectively, on average across participants).

We also observed a significant effect of Size ($F_{1,233}$=13.31, p<.001), with the Small problems taking participants less time to complete than the Large ones (106.7 vs. 138.8 seconds, respectively). Furthermore, we saw an interaction between Interface and Size ($F_{1,233}$=12.55, p<.001). Post-hoc analyses with Bonferroni correction reveal that this was driven by a significant slow down when participants moved from the Small to the Large problem set in the CostMatrix condition, but not when they were using ManiMatrix. See Figure 6 (left) for a summary of the means. Interestingly, we did not find a significant effect for Feedback.

Second, we looked at the magnitude of quality increase in each confusion matrix. For the task in which we presented the target matrix, we calculated this by taking the difference

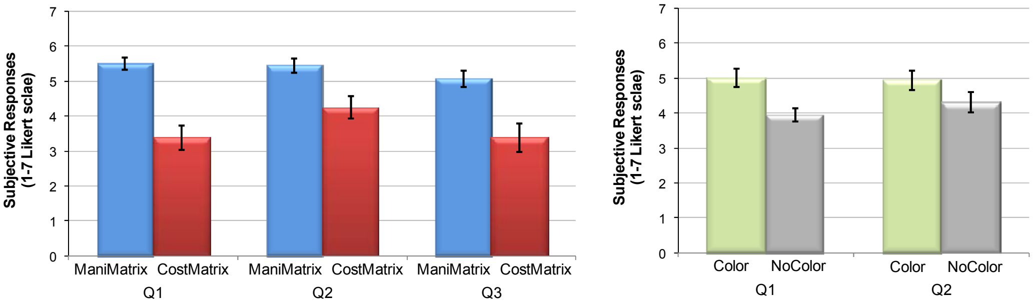


Figure 7. Means of subjective ratings show that users found ManiMatrix easier to use (left Q1), felt that it helped them get to better answers (left Q2), and liked the system better (left Q3). Users also found the system easier to use when Color Feedback was provided (right Q1) and felt that the Feedback helped them get to better answers (right Q2). Error bars represent standard error.

in distance between the initial and target matrix and the distance between the final and the target matrix. For the task in which we presented desires as values on specific cells, we summed the differences between the final and the initial matrix, in the cells of interest. To be able to compare the Small and Large problems more easily, we further normalized these values by the total number of instances within each matrix, resulting in a percentage value. Larger values represent better performance for both tasks.

Again, our ANOVA revealed a significant effect of Interface ($F_{1,233}$=7.35, p=.007), with ManiMatrix seeing a larger gain than CostMatrix (14.4% vs. 9.7%, respectively). The analysis also revealed a significant effect of Size ($F_{1,233}$=70.62, p<.001), with the Large problems seeing a larger gain than the Small ones (19.4% vs. 7.9%, respectively). See Figure 6 (center) for a summary of the means.

Finally, as participants did not always know when to stop trying to optimize their solution, we observed that they sometimes ended the task with a confusion matrix that was of lower quality than another state that they had passed through during the course of the session. Hence, we also looked at the maximum quality increase observed during the course of a trial. We again found a main effect of Interface ($F_{1,248}$=.035, p=.035) as well as of Size ($F_{1,248}$=75.74, p<.001), with valences of the effects in the same direction as before. See Figure 6 (right) for the mean values.

In summary, we have evidence suggesting that ManiMatrix is not only faster, but also leads to better performance than manipulation of a more traditional CostMatrix interface. Additionally, it was interesting to see the interaction between Interface and Size, at least with the task time metric. We were encouraged to find that ManiMatrix does not seem to impose significantly increased overhead with the growth of the problem, as opposed to the CostMatrix interface, which was associated with a significant slowing on larger tasks. While we had initially expected to see differences in performance caused by the Feedback manipulation, we saw no such effects, and will return to this finding below.

Behavioral Results

In order to better understand strategies that participants used to complete their tasks, we explored several behavioral metrics. First, we looked at the use of undo and redo, which were used both to correct errors and to explore the space of tradeoffs that exist when adjusting the various parameters. Performing analyses that were similar to those used for the performance metrics, we found a significant effect of Interface for both Undo ($F_{1,233}$=48.51, p<.001) and Redo ($F_{1,233}$=6.15, p=.014). ManiMatrix showed more Undo operations on average (7.04 vs. 2.92, respectively) but fewer Redos (0.09 vs. 0.41, respectively). This is consistent with our observations that participants tended to use Undo in order to correct errors and roll back in ManiMatrix, since the operations were not reversible (i.e., clicking on the down arrow and then the up arrow does not necessarily return one to the initial matrix). However, they seemed to use Undo to compare values in the CostMatrix condition, hence the increased use of Redo to return to states.

We also looked at the number of total operations used in the various conditions. This was counted as the number of times the user executed a set of constraints in either interface condition. This analysis showed a significant effect of Interface ($F_{1,233}$=117.23, p<.001) with ManiMatrix leading to far more operations than CostMatrix (87.9 vs. 13.6, respectively). This makes sense as operations in ManiMatrix were more lightweight and led to greater granularity in exploration. In general, it also implies that participants tended to both be a little slower with operations, but also set multiple simultaneous constraints when using CostMatrix, potentially leading to slightly lower exploration.

Further, we counted the number of infeasible operations (i.e., the user set constraints that could not be satisfied) and found that participants hit about 5 of these per task. While this may seem reasonable, given that participants would use the constraints to push the bounds of what the model could do until it could do no more, exploring the feasibility of better satisfying constraints is a challenge for future work. Analysis of the number of operations such as clicks or locking the values within the cells for each ManiMatrix condition yielded no significant effects or interactions.

Subjective Responses and Strategies

After they had performed all tasks, we asked participants several questions to acquire subjective ratings on the different Interface conditions. We conducted paired *t*-tests to compare responses across the conditions. We found significant differences on all three questions: *It was easy to use this system* (t_{15}=5.35, p<.001), *I was able to find answers pretty close to the target with this system* (t_{15}=3.05, p=.008), and *I liked to use this system* (t_{15}=3.51, p=.003). All subjective responses preferred ManiMatrix over CostMatrix and means for the responses are shown in Figure 7.

We also had participants rate the Feedback conditions regardless of the interface, and found a similar preference for having the Color Feedback based on the first two questions (t_{15}=4.25, p<.001 and t_{15}=2.30, p=.03, respectively), even though performance data does not seem to indicate that this helped users very much.

DISCUSSION

The results of the user study are encouraging and show that participants used ManiMatrix to effectively construct classifiers aligned with target preferences. This highlights the promise of interactive optimization for steering classification systems. We now reflect about observations and opportunities that arose over the course of this work.

Basic Strategy

When asked to describe the process they used, participants reported common strategies. Regardless of the interface, they first looked for the cells with biggest differences between the target and the current. Then, with ManiMatrix, they click on the up and down buttons to make that difference smaller. With CostMatrix, participants adjusted the cells they wanted to penalize or reward; increased the costs of cells whose values needed to be reduced, and decreased the costs of the cells whose values needed to be increased.

We observed that lock usage varied between participants. Some used much more aggressively than others. While a few participants abandoned it after a couple of trials, most participants frequently used at least one lock especially for the Large problem. A participant who slightly favored CostMatrix in their feedback liked the lock and wanted to have the lock on the output confusion matrix in the CostMatrix interface so that he could not commit changes that would break the lock.

Interactive Optimization

We observed that participants were surprised and frustrated more often by the result of CostMatrix. We assert that this is because participants had to manipulate the cost matrix to control the confusion matrix. While it is easy to decide which cells to penalize or reward, it is non-trivial to choose the right cost to appropriately penalize or reward cells. Even though we provided participants with the reasonable cost value range (between 0 and 2), participants had to spend time understanding the effect of each cost change on the entire matrix. For example, several participants gradually changed the cost of one cell to check the effect. While a high cost value works reasonably well to reduce the value of the target cell, participants had a hard time figuring out how to change the costs to increase the value, and an even harder time with achieving simultaneous improvements.

Compared to ManiMatrix, CostMatrix has two inherent burdens. First, participants had to manage two matrices rather than one, and retain the mapping between the two. For example, several participants used their fingers to map cells between the matrices. Second, it is considerably challenging for participants to manipulate multiple cost parameters simultaneously. ManiMatrix on the other hand allows participants to directly work on the confusion matrix and the embedded optimization routine automatically estimates change across all the cost parameters.

Color Feedback

Participants reported that the basic strategy was the same with or without (Color) Feedback. While we found no significant effect of Feedback in terms of task completion time, 12 out of 16 participants indicated that color feedback was helpful because, without color feedback, they had invest more time and effort to identify drastic, adverse value changes. Several participants reported that they resorted more to undo and redo operations to manually compare old values with new ones in the conditions with no feedback. Some of the participants who did not use the color feedback mentioned that it was distracting. One participant wanted to see the feedback only for cells that had locked directions because those cells were important. For simplicity, we did not provide the aggregated changes for each row or column, or the entire matrix. However, it may be useful to provide additional information summarizing changes. Overall, we believe that it is important to convey changes at every iteration. Exploring such visualizations remains future work.

CONCLUSION

We presented ManiMatrix, an interactive system that allows interactive refinement of classification boundaries in a multiclass setting. The system weaves together visualization, interaction, and fast optimization routines to enable users to steer classification behavior according to their preferences. A user study indicates that methodology can be used to identify numerical parameter settings far better and more quickly than manual tuning.

ManiMatrix is the result of an initial attempt to design a system that enables system designers to directly encode their preferences about the performance of a classification system as well as to provide insights about the structure of the classification problem. In future research, we aim to identify richer forms of interaction and embed them into other aspects of ML, such as the allocation of discriminatory effort during the construction of predictive models.

REFERENCES

1. 20 Newsgroups. http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html.
2. Ankerst, M., Elsen, C., Ester, M. and Kriegel, H.-P. Visual classification: an interactive approach to decision tree construction. *KDD 1999.*
3. Becker, B., Kohavi, R. and Sommerfield, D. Visualizing the Simple Bayesian Classifier. Information Visualiza-

tion in Data Mining and Knowledge Discovery, [ed.] Fayyad, U., Grinstein, G. and Wierse, A. (2001).

4. Caragea, D., Cook, D. and Honavar, V.G. Gaining insights into support vector machine pattern classifiers using projection-based tour methods. *KDD 2001*.
5. Cover, T.M. and Thomas, A.J. Elements of Information Theory. Wiley-Interscience, Hoboken, NJ, USA, 2006.
6. Dai, J. and Cheng, J. HMM Editor: a visual editing tool for profile hidden Markov models. *BMC Genomics 9*, Suppl1 (2008).
7. Drummond, C. and Holte, R.C. Cost curves: An improved method for visualizing classifier performance. *Machine Learning 65*, 1 (2006).
8. Duda, R.O, Hart, P.E. and Stock, D.G. Pattern Classification. Wiley-Interscience, Hoboken, NJ, USA, 2004.
9. Evgeniou, T., Pontil, M. and Elisseeff, A. Leave One Out Error, Stability, and Generalization of Voting Combinations of Classifiers. *Machine Learning 55*, 1 (2004).
10. Fails, J.A. and Olsen, D.R.J. Interactive machine learning. *IUI 2003*.
11. Fei-Fei, L., Fergus, R. and Perona, P. One-Shot Learning of Object Categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 4 (2006).
12. Fogarty, J., Ko, A., Aung, H., Golden, E., Tang, K. and Hudson, S. Examining task engagement in sensor-based statistical models of human interruptibility. *CHI 2005*.
13. Frank, E. and Hall, M. Visualizing class probability estimators. *LNCS 2838*, Springer (2003).
14. Garner, S.R. WEKA: The Waikato Environment for Knowledge Analysis. *New Zealand Computer Science Research Students Conference*, (1995).
15. Grimes, D., Tan, D., Hudson, S., Shenoy, P. and Rao, R. Feasibility and pragmatics of classifying working memory load with an electroencephalograph. *CHI 2008*.
16. Horvitz, E. and Apacible, J. Learning and Reasoning about Interruption. *ICMI 2003*.
17. Kapoor, A. and Horvitz, E. Experience sampling for building predictive user models: a comparative study. *CHI 2008*.
18. Moustakis, V. Do People in HCI Use Machine Learning? *HCI 2*, (1997).
19. Patel, K., Fogarty, J., Landay, J. and Harrison, B. Examining Difficulties Software Developers Encounter in the Adoption of Statistical Machine Learning. *AAAI 2008*.
20. Picard, R.W. Affective computing. MIT Press, Cambridge, MA, USA, 1997.
21. Platt, J.C. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. Advances in Large Margin Classifiers, MIT Press (1999).
22. Rheingans, P. and desJardins, M. Visualizing High-Dimensional Predictive Model Quality. *VIS 2000*.
23. Saponas, S., Tan, S., Morris, D. and Balakrishnan, R. Demonstrating the feasibility of using forearm electromyography for muscle-computer interfaces. *CHI 2008*.
24. Stiglic, G., Mertik, M., Podgorelec, V. and Kokol, P. Using Visual Interpretation of Small Ensembles in Microarray Analysis. *CMBS 2006*.
25. Talbot, J., Lee, B., Kapoor, A., and Tan, D. Ensemble Matrix: Interactive Visualization to Support Machine Learning with Multiple Classifiers, *CHI 2009*.
26. Urbanek, S. Exploring Statistical Forests. *Joint Statistical Meeting 2002*.
27. Ware, M., Frank, E., Holmes, G., Hall, M. and Witten, I. Interactive machine learning: letting users build classifiers. *Int. J .Human-Computer Studies 56*, 3 (2001).

APPENDIX A

The current state for a data point is a function of risk and is mathematically represented using the softmax function:

$$\boldsymbol{o} = \frac{[\exp(-Risk^1), .., \exp(-Risk^c)]}{\sum_{i=1}^{c} \exp(-Risk^i)}$$

Note, that $\exp(\cdot)$ is a monotonically increasing function thus, the vector $\boldsymbol{o}$ is a normalized representation in which a high value corresponds to the class with low risk. The target state for each data point represents the ideal state given the user interaction. In the absence of interaction, the target state should match the original state. However, if the user seeks another state they can interact and then the target state of a particular point depends upon the user interaction.

Consider the case when the user presses the up arrow or locks the up direction for a cell *(i, j)* in the confusion matrix. In this case we want that all the data points with true class *i* be classified as *j*. This implies that $Risk^j$ for all those points be minimum, and we consequently assign the target vector $\boldsymbol{t}$ as zeros everywhere except the j^{th} dimension which is set to one.

Similarly, when a user presses the down arrow key or biases the *(i, j)* cell to go down, we seek a configuration where $Risk^j$ is not minimum. Consequently, we set all the entries of the target vector $\boldsymbol{t}$ to be equal to $\boldsymbol{o}$, except for the j^{th} dimension which is set to zero.

For the rest of the data points (which are unaffected by user interactions), we always set the target vector equal to the original vector. This helps regularize the problem by implying that we seek a configuration that satisfies the user preference but is closest to the original state.

APPENDIX B

The gradients of the objective function with respect to the $Cost_{ij}$ are written as:

$$\frac{\partial\ objective}{\partial\ Cost_{ij}} = \sum_{n=1}^{N} p_i^n \cdot o_j^n \left[\left(\sum_{i'=1}^{c} t_{i'}^n\right) - 1\right]$$