# Interaction Visualizations for Supervised Learning

**Alex Bykov**
University of Washington
abykov@uw.edu

**Stanley Wang**
University of Washington
snwang@cs.washington.edu

**ABSTRACT**
In this project we create a visualization tool for analyzing machine learning classification algorithms. The key idea behind this tool is to provide the user with per-iteration performance information for the algorithm. This is done through two main views. The first view contains a scatterplot matrix of the data projected into multiple dimension pairs. Each point is labeled with its actual and predicted labels to highlight where in the dataset errors occur at each dimension. The second view provides summary statistics (classification accuracy, number of errors, etc.) at each iteration and an interface to scroll through every iteration of the algorithm. Both of these views are updated in real-time as the algorithm as running. As a test of the system, the provided implementation visualizes running a linear SVM algorithm on a breast cancer survival dataset with about 200 points and 3 dimensions. The implementation is easily extendable to other algorithms and datasets.

**Author Keywords**
Guides, instructions, author's kit, conference publications.

**ACM Classification Keywords**
H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

**INTRODUCTION**
One of the key challenges in machine learning is that it can be very difficult to track the performance of an algorithm over time. Most machine learning packages provide good interfaces to import and apply a variety of classification algorithms to a dataset. However the user is only able to see the finished result of the classifier, usually consisting of a decision boundary and some performance statistics. What is missing here is any indication of how the chosen algorithm actually performed the classification. We aim to bridge that gap by creating a tool that helps the user visualize what happens on every iteration of the classification process.


*CHI 2009,* April 4–9, 2009, Boston, Massachusetts, USA.


This tool will provide performance metrics for each iteration of an algorithm and allow the user to inspect how each point in their dataset is classified. With this scheme, the user gets both a global and local overview of the visualization process at each iteration of the algorithm.

We believe that this tool will be useful in many ways. At the most basic level, it can be used as a technique to learn how the chosen algorithm works. Oftentimes it can be challenging to choose which classifier to pick for a specific problem. It is also very bewildering when an algorithm that has great performance on one dataset has much worse performance in a different domain. Being able to examine the step-by-step execution of the classifier can lead to a deeper understanding of how the classifier actually works and why it performs the way it does on the different problems. On a similar note, this visualization can also be used as a teaching tool to help explain a new algorithm to a colleague.

Another common use case we envision is an analysis of the results by a domain expert. Given the fact that our visualization allows the user to inspect the classification performance of each data point, a domain expert will be able to glean some insight as to what anomalies in the data are causing mis-classifications. Many classification algorithms also require the user to set parameters which may determine the performance of the algorithm. By seeing which points are being commonly mis-classified, the user can alter the parameters to try and improve the classification accuracy of those specific points. One key part of our tool is that this algorithm analysis happens in a streaming fashion, while the algorithm is actually running. This will allow the user to gauge if their algorithm's performance is improving before the algorithm has terminated. If the algorithm is clearly performing worse with the newest set of parameters, the user can terminate the execution of that algorithm and try new parameters. This should greatly reduce the time required to iterate through multiple classifiers/classifier parameters.

**RELATED WORK**
There already exists several systems intended to make the task of machine learning easier. One such system is called Weka [1,2]. Weka supports a wide variety of algorithms that can be applied to different datasets. The GUI allows for exploration of the initial dataset and also some visualization of the final classifier result. Weka is especially good for visualizing tree and graph structures for the resulting classifier. Aside from Weka, there are other systems that

focus on visualizing the performance of a classifier. A program called EnsembleMatrix [3] considers the problem of comparing multiple classifiers. EnsembleMatrix draws a confusion matrix for each classifier trained on a dataset and allows interaction between each of the confusion matrices. This allows the user to see which classifiers worked better on which part of the data. Finally, there are several systems for visualizing the actual decision boundary of a classifier [4, 5]. The decision boundary is visualized in the intuitive manner - by plotting all of the points on a 2D scatterplot and then plotting the decision boundary between them. Given the fact that the dataset may have very high dimensionality, these systems allow the user to select which dimensions are shown for the scatterplot.

## METHODS

Our visualization provides a view into the performance of the classifier at any given iteration during its training phase. We plot the data on a coordinate space defined by two feature dimensions, which the user can select. Each data point is colored with its predicted class, and a compound color scheme is used to differentiate between correct and incorrect classifications for each class. Our visualization currently only supports binary classification but easily generalizes to multi-class classification. The plot is replicated with different dimensions to form a scatter plot matrix that allows the user to compare classification performance across different dimensions, since real world data sets typically have high dimensionality and are not easily separable between any two particular dimensions. For example, for a 3D dataset, the hyperplane that best separates the two classes may not be orthogonal to a particular axis.

Below the scatterplot matrix, we show a plot of different metrics over time, which is measured in terms of iterations. The different metrics include the number of errors, the change in errors between iterations, and the training, validation, and test set accuracies. These metrics help inform the user about how the classifier is behaving over time and give the user insight into how the algorithm works or point out room for improvement in feature selection. For example, the user may notice that none of the plots show a good separation between classes, which suggests that the features need to be reworked, rather than the algorithm being at fault. The support for showing training, validation, and test set accuracies is helpful for putting our visualization within a typical model selection and training workflow, which generally involves training over a subset of the overall data, using a different subset to select hyperparameters and model parameters, and using the trained classifier on yet another subset to check its performance on data it has not seen before. We believe that these metrics are also very applicable to the pedagogical setting, to open the black box that is model training and show students important insights such as the bias-variance trade-off, training vs. test set performance, and the execution of specific training algorithms, such as sequential minimal optimization for support vector machines, along with the resulting changes in classification.

To implement our visualization, we use D3.js and structure the work as a web-based visualization. We chose D3.js over graphics libraries such as Matplotlib for Python and ggplot2 for R since it allowed us to exercise more control over the layout and visual presentation as well as enable interactivity, which is not possible with those libraries. Using the web also makes it easier to share the visualization with others and turn the visualization into a dashboard for display on external monitors. It also enforces a separation between the algorithm and the visualization, which can be useful for enabling communication over the internet between the machines that train the classifiers and the machine that hosts the visualization. Use cases include training a classifier on a cluster of machines running Hadoop and sending the results for each iteration back to the visualization.

We chose to use a support vector machine binary classifier for this visualization. Rather than limiting ourselves to this one case, we use the support vector classifier as purely an example of a classifier that can be integrated with the visualization. For simplicity, the classifier uses a linear kernel. We note that the linear kernel is not much different from polynomial kernels, radial basis function kernels, and other high or infinite-dimensional kernels, since even for a linear kernel, the decision boundary may not be orthogonal to a dimension, and consequentially, under our class coloring method, the different kernels are treated equivalently. We use the scikit-learn library to run the support vector classification and provide the per-iteration data that we use for the visualization. Scikit-learn uses LIBSVM and LIBLINEAR to perform linear support vector classification. We chose the LIBSVM version, which uses the sequential minimal optimization algorithm to solve the dual form of the quadratic programming problem posed by the soft-margin support vector machine [6].

## RESULTS

Figure 1 shows the main screen of the visualization. The scatterplot matrix is on the top, and the number of errors is plotted on the bottom. The data used is the Haberman's Survival dataset [7]. The scatterplot shows different features plotted against each other. It can be seen that the structure and distribution of the data vary quite a bit for each feature. Since the figure shows the state of the classifier after the last iteration, it is clear that the classifier ended up weighting the third feature very heavily and making the classification hyperplane orthogonal to the third dimension. If the user is curious about a particular misclassified data point, the user can place the mouse over a data point, which will trigger a pop-up containing information about the point. This is shown in figure 2, where a point in the plot between the second and first features is highlighted.

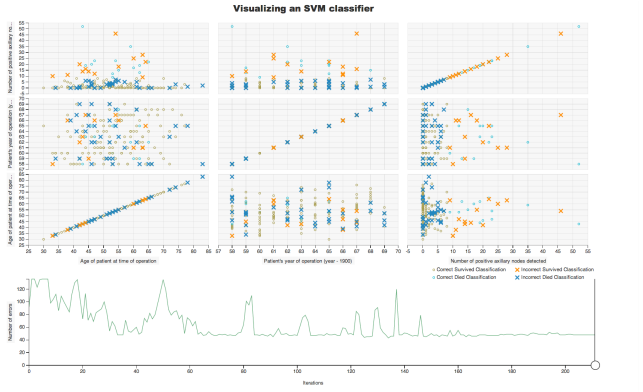


**Figure 1. The main screen, showing the scatterplot matrix for the last iteration and the number of errors per iteration on the lower chart. The prominent orange and blue marks are classification errors.**

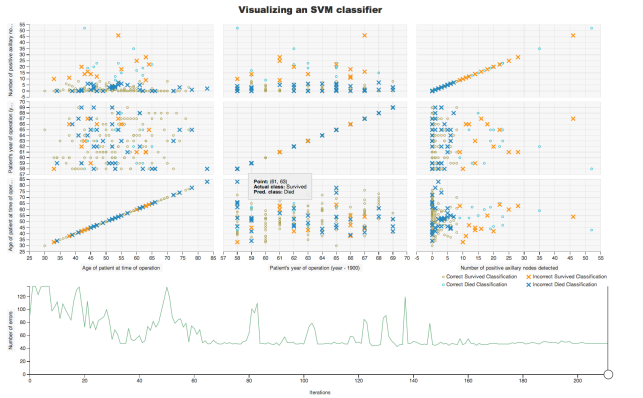


**Figure 2. The main screen, showing contextual information for a classification error under the cursor.**

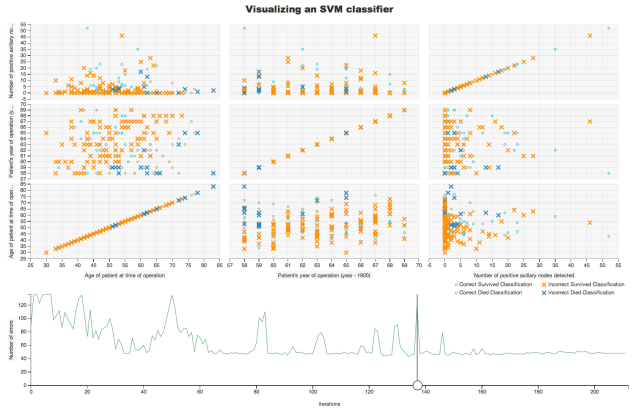


**Figure 3. The main screen, showing the scatterplot matrix for an iteration in which the number of errors sharply increased. The orange and blue marks are all classification errors.**

The plot of the number of errors shows that there are a few iterations in which the number of errors suddenly increased. The user can move the slider on the x-axis of the bottom chart to switch the scatterplot matrix to a different iteration, as shown in figure 3. The sudden increase in errors can easily be seen by the sudden increase in orange marks, which are classification errors.

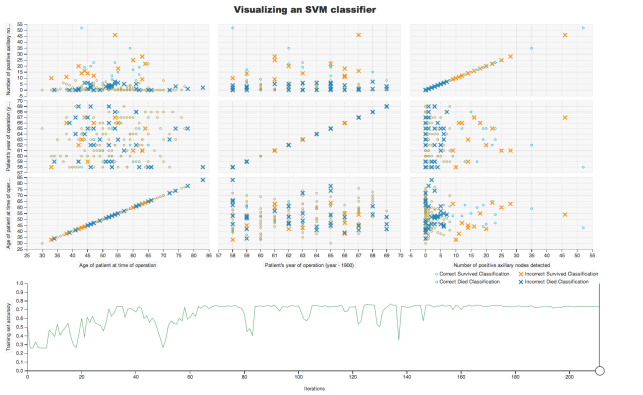


**Figure 4. The main screen, with the training set accuracy plotted below.**


**Figure 5. The main screen, with the test set accuracy plotted below.**

The user can view the overall classification accuracy on the training, validation, and test sets by selecting those metrics on the y-axis of the lower chart. Figure 4 shows the plot of the training set accuracy, and figure 5 shows the plot of the test set accuracy, with the test set replacing the training set in the scatterplot matrix. These sets need to be specified prior to running the visualization. The data points used in the scatterplot matrix are replotted to reflect whichever subset is chosen on the lower chart. Thus, the visualization allows the user to examine the performance of the classifier on the training, validation, and test sets easily.

## DISCUSSION

We believe our main contribution lies in our work enabling streaming iteration data from the training classifier. This is important since most visualizations of this nature are performed after the training is completely done, which is a long time when the training is computationally expensive or slow. This applies to complex models, especially for those

done on a distributed platform like Hadoop where the throughput is high but the latency is very high, and it often takes 24 hours to finish a job. Given our visualization's streaming capabilities, the progress of the learner can be tracked while it's still training, and the model can be iterated on much more quickly.

## FUTURE WORK

Some useful extensions for the visualization are supporting other kernels for support vector machines, supporting other algorithms, and including more visual tools to help the machine learning practitioner make use of the visualization. Also, performance could be improved to help scale the visualization to larger datasets.

Our current implementation only supports classifiers of the form $y = Wx + b$, since we utilize the W vector in the front end to classify the points in the visualization. Extending this to allow extra kernels would be very easy if we move the classification of points to the back end and send the classification of each point to the front end. This is also a valid strategy for allowing more supervised classifiers to be plotted, such as K-nearest neighbors or neural networks. Extending the binary classification setting to multiclass is also as easy as defining more classes and extending the color scheme.

More tools that we could see being useful to machine learning practitioners using our visualization include a confusion matrix as well as information about the saliency of different dimensions. A confusion matrix would give a useful, easily understandable numerical summary of the information that is already plotted in the graph, which can be important since estimating the quantities from the scatterplots can be difficult due to the large amount of data and the the existence of occlusion. Information about the saliency of different dimensions would be useful since datasets with high dimensionality have the problem that the user must choose which features to plot and examine. This is preferred over plotting the data in terms of the basis vectors found from methods like principal component analysis (PCA), since those basis vectors lack the same kind of interpret-ability as the original features, and it's not clear from the basis vectors where further improvements can be made.

Lastly, performance for the visualization can be improved. The scatterplot matrix is particularly computer-intensive when re-rendering the data points, so further research into efficient ways to draw the points or only partially draw the points would result in a lot of gains.

## REFERENCES


1. Holmes, Geoffrey, Andrew Donkin, and Ian H. Witten. "Weka: A machine learning workbench." *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*. IEEE, 1994.
2. Hall, Mark, et al. "The WEKA data mining software: an update." *ACM SIGKDD explorations newsletter* 11.1 (2009): 10-18.
3. Talbot, Justin, et al. "EnsembleMatrix: interactive visualization to support machine learning with multiple classifiers." *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2009.
4. Frank, Eibe, and Mark Hall. *Visualizing class probability estimators*. Springer Berlin Heidelberg, 2003.
5. Rheingans, Penny, and Marie Desjardins. "Visualizing high-dimensional predictive model quality." *Proceedings of the conference on Visualization'00*. IEEE Computer Society Press, 2000.
6. C.-C. Chang and C.-J. Lin. LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011.
7. Bache, K. & Lichman, M. UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science, 2013.