# Squares: Supporting Interactive Performance Analysis for Multiclass Classifiers

Donghao Ren, Saleema Amershi, Bongshin Lee, Jina Suh, and Jason D. Williams

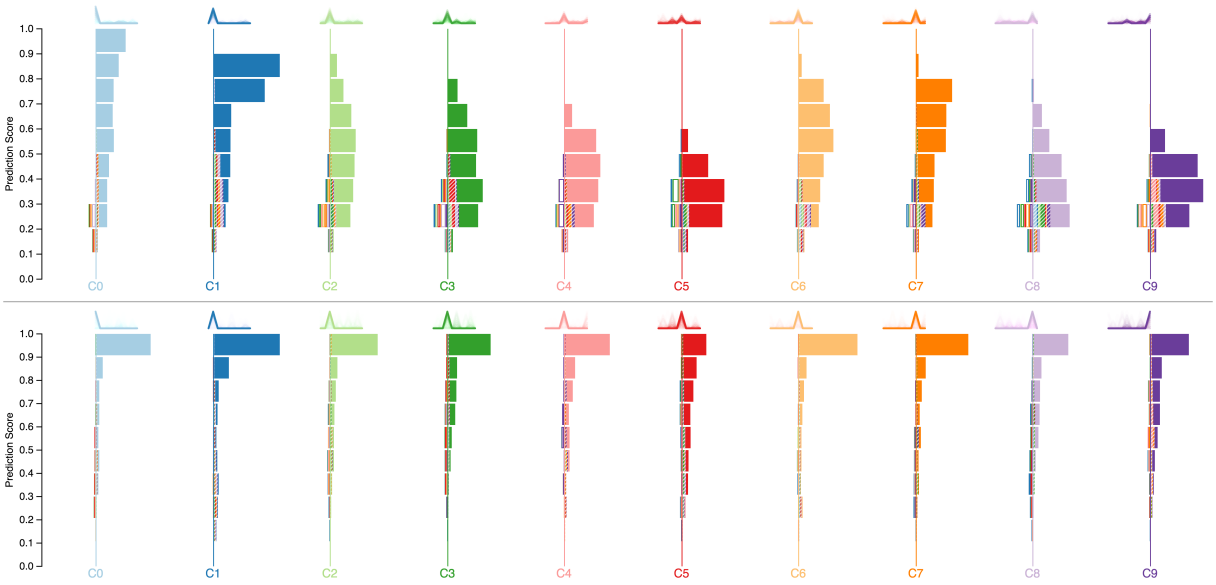



Fig. 1. Squares displaying the performance of two digit recognition classifiers trained on the MNIST handwritten digits dataset [24]. These classifiers yield the same accuracy of 0.87 (top: random forest, bottom: SVM), but show vastly different score distributions.

**Abstract**—Performance analysis is critical in applied machine learning because it influences the models practitioners produce. Current performance analysis tools suffer from issues including obscuring important characteristics of model behavior and dissociating performance from data. In this work, we present Squares, a performance visualization for multiclass classification problems. Squares supports estimating common performance metrics while displaying instance-level distribution information necessary for helping practitioners prioritize efforts and access data. Our controlled study shows that practitioners can assess performance significantly faster and more accurately with Squares than a confusion matrix, a common performance analysis tool in machine learning.

**Index Terms**—Performance analysis, classification, usable machine learning.

## 1 Introduction

Performance analysis is critical in machine learning because it influences the models (e.g., classifiers or rankers) practitioners produce. For example, practitioners often compare the performance of models generated with different algorithm parameters before deciding which parameters to use. In another example, practitioners typically iteratively develop feature-based representations of the data used in training a model, comparing performance before and after feature revisions to ensure they are improving the model as expected.

Currently, the most common way to understand model performance in machine learning is to look at summary statistics such as accuracy, precision, recall, or logarithmic loss (see [13] for a review). Another common tool for performance analysis is the confusion matrix [35] which contrasts model predictions against ground truth labels in a table of aggregated values. Although statistics and confusion matrices efficiently summarize performance, aggregated values can obscure important information about a model's behavior. For example, the two classifiers displayed in Figure 1 (a random forest classifier and a support vector machine classifier) were trained on the same dataset obtained from the MNIST handwritten digits database [24] and yield the same accuracy of 0.87 when applied to the same test dataset, but show vastly different prediction score distributions. These score distributions are important for assessing error severity and prioritizing efforts in debugging model performance. Aggregated values can also be misleading in some cases. For example, when training data is skewed, models can produce high accuracy values even when they have little or no predictive power by always predicting the majority class [38]. Summary statistics and static confusion matrices also dissociate performance from the data used in model training — a problem that has been shown to hinder applied machine learning by introducing a barrier between performance analysis and root cause debugging [29]. For example, if a statistic or confusion matrix indicates poor performance, practitioners typically have to search for offending errors in a different tool or mode before they can begin to gain insights and debug the problem (e.g., inspecting errors may reveal mislabeled instances or feature

- *D. Ren is with the University of California, Santa Barbara. E-mail: donghaoren@cs.ucsb.edu. This work was done while he was at Microsoft Research.*
- *S. Amershi, B. Lee, J. Suh, and J. D. Williams are with Microsoft Research. E-mail: {samershi, bongshin, jinsuh, jason.williams}@microsoft.com.*

*Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org.*
*Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx/*

deficiencies). Separating performance from data in this way has been shown to discourage data inspection, resulting in a trial-and-error approach to model building [29].

In this paper, we present Squares (Figure 1), a performance visualization for *multiclass* classification problems. Multiclass classification is a common task in machine learning for modeling problems that naturally have more than two disjoint classes (e.g., digit recognition [24], disease category prediction [31], and topic identification in documents [21]). Squares visually displays information used to derive several common performance metrics while providing direct access to data. This paper makes the following contributions:

- Survey results from over one hundred machine-learning practitioners at a large technology company about their classification practices and needs.
- The design and development of Squares, a performance visualization for multiclass classification problems. Squares supports estimating common performance metrics while displaying instance level distribution information necessary for prioritizing efforts and accessing data.
- A controlled study comparing Squares to an interactive confusion matrix. Results show that participants were faster and more accurate at assessing performance, at both the aggregate and instance level, with Squares. The majority of participants also preferred Squares over the common confusion matrix.

## 2 Related Work

### 2.1 Multiclass Classification Techniques

Many tools have been created to support analysis of classifiers built with specific algorithms such as support vector machines (SVMs) [10], decision trees [7, 36], and Naive Bayes [8]. In contrast, Squares can support analysis of any multiclass classifier that can output calibrated probability scores on instances [37] including SVMs, decision trees, Naive Bayes classifiers, and random forests. Multiclass classifiers can also be built by training binary classifiers and then combining their outputs to make predictions on individual instances [5]. For example, the one-vs-rest method (also known as one-vs-all) trains $N$ binary classifiers for an $N$-class problem such that each binary classifier discriminates one of the target classes from the rest of the classes. The classifier that produces the highest one-vs-rest score then determines the prediction class for each instance. With the one-vs-one (or all-vs-all) method, binary classifiers are trained on every pair of classes and majority voting is used to select the winning class prediction on each instance. Squares can support analysis of these and any multiclass classifiers that can produce scores on at least the winning class.

### 2.2 Understanding Classifier Performance

Common machine learning environments such as Weka [18], Scikit-learn [30], and AzureML [1] provide summary statistics (e.g., accuracy, precision, recall) and confusion matrices as built-in functionality. Similarly, with programming languages such as Matlab [26], R [19], and Python [3], people often use existing libraries to compute performance statistics (e.g., the `sklearn.metrics` package [30] for Python), and then plot them as basic charts. Summary statistics, however, typically only convey a single aspect of a model's performance. For example, accuracy captures the number of prediction errors whereas precision and recall emphasize false positive and false negative errors, respectively. Confusion matrices [35] are a more detailed type of summary statistics that aggregate instances by their true label and prediction in a table. Many aggregated values such as counts or percentages can, however, be misleading because they treat all predictions contributing to the value equally. For example, all false positive or negative errors within a cell of a confusion matrix are treated equally, hiding important information about error severity. Summary statistics and standard confusion matrices also separate performance from the data necessary to provide insights into performance problems. As a result, if summary statistics indicate poor performance, practitioners typically must switch tools or modes to search for and inspect data associated with the performance issue.

Due to the cognitive overhead and inefficiencies in switching tools to debug machine-learning performance problems, researchers have advocated for a tighter coupling of performance with data instances [29]. For example, Gestalt [28] is an integrated machine-learning environment that includes an interactive confusion matrix that practitioners can use to directly filter an adjacent display of raw data (e.g., images or text). While Gestalt helps associate performance with data, like all confusion matrices, it still suffers from only showing aggregated values that can hide information about model behavior.

Prospector [23] goes beyond summary statistics to help data analysts better understand predictive models. It helps data analysts understand how features affect the prediction through interactive partial dependence diagnostics, and how and why specific instances are predicted as they are with localized inspection. However, Prospector relies on partial dependence for only one feature at a time, and can handle only single-class predictions.

### 2.3 Instance-based Performance Visualizations

Recently researchers have begun to investigate the use of interactive instance-based visualizations to support performance analysis and debugging in machine learning. For example, iVisClassifier [11] employs dimensionality reduction techniques [15] and instance-based visualizations including scatterplots and parallel coordinates to support examination of model behavior. Here, instances can be color coded by their true label and performance issues are indicated by the arrangement of colored points in the display (e.g., a mix of colored points may indicate poor separability of certain classes). Users can also click on individual points to view the corresponding data instances.

To cope with high dimensionality, researchers have employed projection-based methods. Frank and Hall [14] overlay color-coded instances on colormaps on a 2D projected space. Rheingans and DesJardins [32] use the self-organizing map [22] for projection, and visualize feature levels with colored contours. The interactive visualization in [17] and the iPCA system in [20] use projection-based scatterplots and coordinated views for examining model behavior and inspecting selected instances. While projection-based visualizations can convey relative performance (a scatterplot with a greater mix of colors may indicate worse performance than one where same-colored points are more clearly separated), they are not designed to convey common performance statistics such as precision and recall. Dimensionality reduction techniques also do not project onto fixed dimensions, making performance comparison between classifiers or classes difficult. Squares is designed to convey common performance metrics, while showing labels and predictions in a consistent way to make comparison easy.

Both the Visual Classification Methodology in [27] and the Class Radial Visualization in [33] visualize classifier performance via a radial arrangement of instances. Paiva et al. [27] use dimensionality reduction and neighbor joining trees (similarity trees with branches emanating radially) to arrange instances (color-coded by label) based on feature space similarity as well as classifier confidence, where instances with higher confidence predictions are located towards the leaves of the tree. This visualization can be used for comparing performance across classifiers (a tree with more mixed colored branches may indicate more confusion), but actual performance can only be estimated with a second *ClassMatch* tree that colors instances by correctness (green and red indicate correctly and incorrectly classified instances, respectively). In contrast, Squares shows ground truth labels and predictions in the same visualization, allowing for more efficient performance analysis and estimation of common performance statistics (e.g., class-level precision and recall that cannot be easily estimated with similarity trees). In the Class Radial Visualization [33], each target class is assigned to a location around the perimeter of a circle and individual instances (color-coded by prediction) are placed within the circle according to their per-class probability scores. Again here, relative classifier quality can be gauged but the layout does not support estimating common metrics which instead are shown in an adjacent panel. In both of these cases, error severity is indicated by

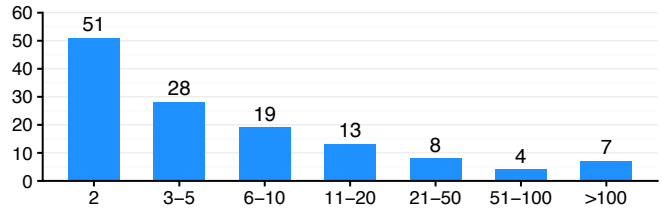



Fig. 2. Distribution of common class sizes for real classification problems (note that the total number is greater than 102 given the check-all-that-apply).

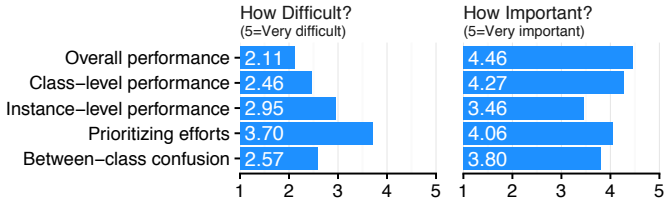



Fig. 3. Difficulty (5=Very difficult) and Importance (5=Very important) of common performance analysis tasks with current tools.

instance location in the display which helps to support prioritization in error debugging, a primary design goal in Squares as well (see Section 4.1). However, as with scatterplot displays, point-based arrangements such as these scale poorly to larger datasets due to point overlap. In contrast, Squares uses instance grouping and different visual encodings to scale up to larger datasets or number of classes.

### 2.4 Conveying Performance at Multiple Levels

ModelTracker [6] is an interactive instance-based visualization that shares our goals of supporting both aggregate and instance level performance information while providing direct access to data. However, ModelTracker only supports binary classification and it is non-trivial to extend it to multiclass scenarios. One approach proposed in [6] is to visualize each one-vs-rest classifier individually and allow users to switch between displays. This, however, makes performance estimation inefficient. Even if each one-vs-rest display is presented side-by-side, this method replicates each instance in all displays, distorting estimation of overall-classifier performance and hiding information about class imbalance. Moreover, as multiclass classifiers select the highest score as the final instance prediction, there is no clear indication of the decision boundary on each one-vs-rest classifier. That is, unless instance copies are connected in some way, it would be difficult to know which copy of the instance is chosen by the final model. Squares extends the ModelTracker approach of leveraging unit visualizations for performance estimation and access to data, but specifically addresses the characteristics of multiclass classification problems.

Perhaps most related to Squares is the system recently proposed by Alsallakh et al. in [4]. This work shares our motivations to convey overall and class-level performance (including performance statistics that can be derived from prediction distributions and prediction categories such as true positives/negatives and false positive/negatives) and help users prioritize debugging efforts by indicating error severity. The Confusion Wheel visualization they use, however, displays prediction score-based histograms in a radial display which can cause visual distortion of histogram bins. This distortion and the rotation of histograms around the wheel can also make class-level comparison difficult. The authors also explicitly avoid displaying instances in their Confusion Wheel to remove the clutter and overlap problem common in previous work. Squares builds up histograms with instances via binning, supporting instance-level performance while avoiding the clutter problem. The system in [4] also employs separate but coordinated visualizations to convey additional information including accuracy, class imbalance, and prediction scores. We argue that while coordinated views can provide insights into a variety of model behaviors, they can also add complexity to the performance analysis and debugging task. Because performance analysis and debugging is one (important) step of the iterative applied machine learning workflow, heavy weight tools are often only experimented with after trial-and-error based iterations fail [29].

With Squares, we endeavored to support the most important performance analysis tasks (as indicated by our survey of over 100 machine learning practitioners, see Section 3) within a single visualization to reduce the cognitive load of the performance analysis step. Finally, we evaluated Squares in a controlled experiment, validating the Squares visualization for performance analysis.

## 3 Survey on Multiclass Classification Practices

We conducted a survey about current practices and difficulties machine learning practitioners face when building multiclass classifiers. We distributed this survey to relevant machine learning distribution lists and user groups within a large technology company in July 2015, and obtained responses from 102 practitioners. Here we present the survey results that influenced our design of Squares.

Of our survey respondents, 37% were data scientists, 28% were software engineers, 8% were researchers, 8% were program managers and the remaining 19% had various other roles in the company including technical support, consultant, and intern. Of the respondents who self-reported their machine learning expertise level, 18% reported themselves as experts, 33% as advanced, 34% as intermediate, and 15% as novice users. For testing and examining classifier performance, 90% of respondents reported using common machine learning toolkits and libraries such as Weka [18], Scikit-learn [30], and AzureML [1], 61% reported writing custom code in general coding environments (e.g., Python, R, C#), and 28% reported using data analysis tools and spreadsheets (e.g., Excel, PowerBI [2]).

Figure 2 shows the distribution of responses to *"How many classes do your classifiers typically deal with?"* We allowed participants to select all answers that applied given that practitioners often work on multiple classification problems that may not all deal with the same number of classes. Among the 102 respondents, 51 (50%) reported having built 2-class (i.e., binary) classifiers, 42 (41%) built classifiers with between 3 to 20 classes (42 is the number of unique respondents who checked any of 3–5, 6–10 and 11–20 classes), and 14 (14%) built classifiers with greater than 20 classes (note that the total percentage is greater than one hundred given the check-all-that-apply).

Figure 3 shows how respondents rated *how important* various performance analysis tasks were (1–5 Likert scale, 1=Not important and 5=Very important) and *how difficult* it was to perform those tasks with their current tools (1=Very easy and 5=Very difficult).

These survey results highlight several opportunities for improvement in designing tools to support important tasks for machine learning practitioners. For example, most of our respondents reported typically dealing with no more than 20-class problems. Future researchers may therefore consider focusing on tools for 20-class problems or less for greatest impact or on tools for other machine learning techniques such as regression or hierarchical classification, which are often employed for problems involving more than 20-classes. Our results also show that, even with over 50% of respondents reporting themselves as advanced to experts in machine learning, respondents reported difficulty *prioritizing efforts in improving classifier performance*. Respondents also reported that understanding *instance-level performance* was relatively difficult with common toolkits and libraries. Interestingly, respondents reported less difficulty examining *overall-* and *class-level performance* and *between-class confusion*, yet, our evaluation of Squares compared to a common tool used by machine learning practitioners shows both a speed and accuracy improvement on each of these tasks (all rated as important by our respondents). While these results highlight issues faced by a variety of practitioners with various roles and expertise levels at a large software company, further analysis is necessary to extend these findings to a wider population.

## 4 SQUARES

In this section, we describe three main design goals influenced by our survey, our iterative design with real practitioners, and previous work. We then present the Squares visualization and interaction facilities. Note that after our controlled study (described in section 6), we further refined the visualization to accommodate larger datasets and support rich interactive exploration. We delineate the features introduced after our controlled study accordingly in this section.

### 4.1 Design Goals

Based on our survey results, we aimed to optimize Squares for 3 to 20 class classification problems (covering approximately 41% of cases reported by our survey respondents). This allowed us to make use of visual encoding techniques that are more easily discernable when there are no more than 20 categories, such as color (refer to Green-Armytage's 26-color alphabet [14]).

**G1: Show performance at multiple levels of detail to help practitioners prioritize efforts.** Our survey results suggest that overall and class-level performance should be most salient (as these were rated as most important), followed by instance-level performance details. The visualization should also help practitioners prioritize efforts in debugging performance problems. For example, classes with more severe errors (instances with high prediction scores in the wrong class) may be more problematic than classes with less severe errors.

**G2: Be agnostic to common performance metrics.** Different classification problems require optimizing for different performance metrics. For example, in diagnosing diseases, a false positive diagnosis can result in unnecessary and possibly harmful treatment, whereas a false negative may result in a lack of treatment and potential worsening of the disease. The decision of what to optimize for is therefore scenario dependent and based on the relative costs assigned to possible outcomes. To support a wide range of scenarios, the visualization should therefore attempt to be agnostic towards any specific metrics. For example, false positives should be comparably salient to false negatives to support scenarios where either could be more costly.

**G3: Connect performance to the data.** Previous work has shown that separating performance from the data necessary to provide insights into performance problems is disruptive and encourages poor practices [29]. The visualization should therefore support interactive and direct access to data where possible. Reducing disruptive switches between performance displays and data also suggests presenting both performance and data on the same screen. Data is often presented raw (e.g., images or text) or in featurized form (e.g., in a table) depending on the data type. Consequently, the visualization should be compact and flexible to support a variety of data display requirements.

### 4.2 Visualization and Interaction

Squares represents each class in a color-coded column. Each column contains a vertical axis annotated below by the corresponding class name and optional summary statistics (true/false positive/negatives and precision/recall) for that class. The vertical axes align with the leftmost axis indicating the prediction score range on instances, from lowest scores at the bottom to highest at the top. Figure 4 shows the performance of a 10-class classifier trained on the MNIST handwritten digits dataset [24] where C0 to C9 corresponds to digits 0 to 9.

#### Visualizing Count-Based Metrics

Many common performance metrics in multiclass classification are derived from different categories of prediction counts. For example, accuracy is computed as the number of correct predictions over the total number of predictions (correct and incorrect). Other metrics are derived from different types of correct (true positive and true negative) and incorrect (false positive and false negative) predictions. A false positive for class X is an instance predicted as class X but labeled as another, whereas a false negative for class X is an instance labeled as class X but predicted as another. For example, precision is computed as the number of true positives over the number of true and false positives while recall is the number of true positives over the number of true positives and false negatives. To support a wide variety of classification problems (G2), we therefore designed Squares to make both *correct* and *incorrect* predictions comparably salient, and to make both *false positive* and *false negative* errors comparably salient by using position and pattern-coding of the boxes (Figure 4 and Figure 5).

Each box in the visualization represents an instance of the data. Boxes positioned on the right side of an axis line represent instances predicted as that axis' corresponding class (column). Boxes are filled with the color of their true class (i.e., the class they are labeled as). A solid or striped fill pattern denotes instances predicted correctly or incorrectly by the classifier, respectively. Boxes on the left side of any axis line represent instances labeled as that axis' corresponding class (column) but predicted incorrectly as a different class. Boxes on the left therefore denote false negative instances and have no fill color, but are outlined with the color of the class being predicted.

Note that because false positives and false negatives are defined with respect to a given class, there is a one-to-one correspondence between each striped box (false positive) in one column and its matching outlined box (false negative) in the column indicated by the color of the striped box. That is, each error is represented twice in the display. This enables Squares to make false positive and false negative errors comparably salient (G2). Duplicating errors also does not affect between-classifier or between-class error comparisons (since errors are duplicated in each case) (G1).

With this design, Squares helps users estimate several common count-based overall and class-level performance metrics (G1, G2). For example, classifier accuracy is estimated by the number of solid boxes out of the total number of boxes on the right side of the axes lines. (Considering all boxes on both sides of the axes can distort this metric because errors are duplicated, but this again does not impact between-classifier comparisons). Similarly, class-level precision is estimated by the number of solid boxes out of the solid plus striped boxes: $\texttt{TP}/(\texttt{TP}+\texttt{FP}) =$ ■ /( ■ + ▨ ); class-level recall is estimated by the number of solid boxes out of the solid plus outlined boxes: $\texttt{TP}/(\texttt{TP}+\texttt{FN}) =$ ■ /( ■ + □ ).

#### Showing Score-Based Metrics and Supporting Prioritization

Thus far, we have described how Squares visualizes common count-based performance metrics. Some performance metrics, however, also take into consideration prediction scores on instances. For example, logarithmic loss penalizes predictions that deviate far from their true label. To support estimation of metrics that also consider score (G2), boxes on either side of any axis are positioned along the vertical dimension according to their prediction scores, with high scores to the top and low scores to the bottom (Figure 1 and Figure 4). In this way, striped or outlined boxes towards the top of a column should be penalized more than striped or outlined boxes towards the bottom. Similarly, solid boxes towards the bottom of a column should be penalized more than solid boxes towards the top.

Note that a drawback of binning boxes along the vertical dimension (for clarity of display) is that users may perceive boxes at the top of a bin as having higher prediction scores. However, in our experiences with machine learning practitioners, small score differences are less problematic than large ones. Moreover, exact scores can be revealed on demand (see Showing Instance-Level Metrics and Conveying Between-Class Confusion below) and bin resolution could be parameterized in the display to allow for finer score distinctions.

Displaying scores not only supports estimating score-based performance metrics, it allows for prioritizing efforts by prediction confidence or error severity (G1). This is in contrast to count-based metrics and confusion matrices that treat all errors equally (and all correct predictions equally).

#### Showing Instance-Level Metrics and Conveying Between-Class Confusion

Distributing boxes vertically by prediction score provides instance-level performance (G1). Some multiclass classifiers, however, produce scores for an instance across all classes (in these cases, the class with the highest score, the winning class, is selected as the predicted class).

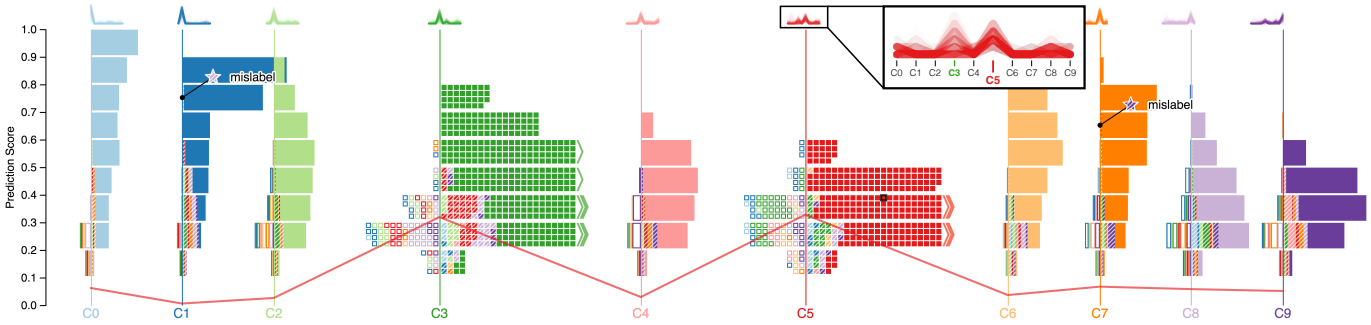



Fig. 4. Squares displaying the performance of a digit recognition classifier trained on the MNIST handwritten digits dataset [24]. All classes are represented with stacks except C3 and C5 which are expanded to boxes for more details. The solid red boxes in C5's column represents instances correctly predicted as C5 while the green-stripped boxes in that column represent instances labeled as C3 but incorrectly predicted as C5.

Squares reveals scores for an instance across all classes on demand when a user hovers or clicks on a box in the display (G1). Scores are displayed using parallel coordinates, with a polyline intersecting each axis at the corresponding score level for that class (see the polylines in Figure 4 corresponding to the selected solid box in the C5 column). Note that the polyline intersection may not align exactly with the vertical position of the corresponding box because the line intersects at the exact score location while boxes are binned along the score axis.

Instance scores across all classes also reveal between-class confusion not visible in confusion matrices that only display confusions about winning class predictions. Between-class confusion is indicated when parallel coordinates for instances have high peaks in multiple classes (i.e., have high prediction scores on multiple classes). Squares summarizes confusion information via a sparkline above each axis displaying the parallel coordinates of all instances labeled as the corresponding class (callout in Figure 4). The sparklines are aligned so that each class axis points to the same axis in the corresponding sparkline. In this way, a class with little confusion with other classes will have a single peak above the axis.

## Scaling Up to Larger Datasets or Number of Classes

Machine learning often deals with large datasets. In these cases, it may be unreasonable to display all instances individually as boxes. Squares handles larger datasets by grouping boxes together within a bin by their label, resulting in stacked bars representing proportions instead of individual boxes representing instances (Figure 5, right). Note that our visual encodings remain the same for stacks, with solid bars representing correct predictions and striped and outlined bars representing false positives and negatives, respectively. Stacked bars can also be used when the number of classes grows and more columns must be fit into the display (reducing the amount of space available per column).

The stacks view allow Squares to show overall and class-level performance as described previously. To still support viewing instance-level performance (G1) with larger datasets or more classes, Squares allows users to toggle individual classes between stacks and boxes by clicking on their column.

For assigning a unique color to each class, Squares currently uses D3's 20 category color scale [9] for over 12 classes, the ColorBrewer [16] paired colors for 6 to 12 classes, and the ColorBrewer main colors for less than 6 classes. To support a larger number of classes in the future, different colors may be obtained by using semantically-resonant colors [25].

**After the controlled study:** We improved Squares further by (1) introducing the strips view (Figure 5, middle) in addition to the boxes and stacks view and (2) preventing overflow boxes or strips by replacing them with a truncation indicator (Figure 5, left). The strips view serves as a trade-off between boxes and stacks by allowing for more instances to be displayed than with the boxes view, and better granularity for instance selection than the stacks view. Each strip in the strips view represents 10 boxes by default (this could, however, be parameterized in the display), and is color- and pattern-coded in the same

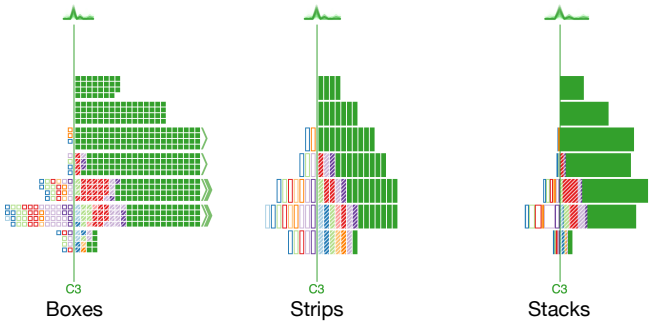



Fig. 5. The strips and stacks view group boxes together when there are a large number of classes or instances. Users can toggle between boxes (left), strips (center), and stacks (right) at the class level.

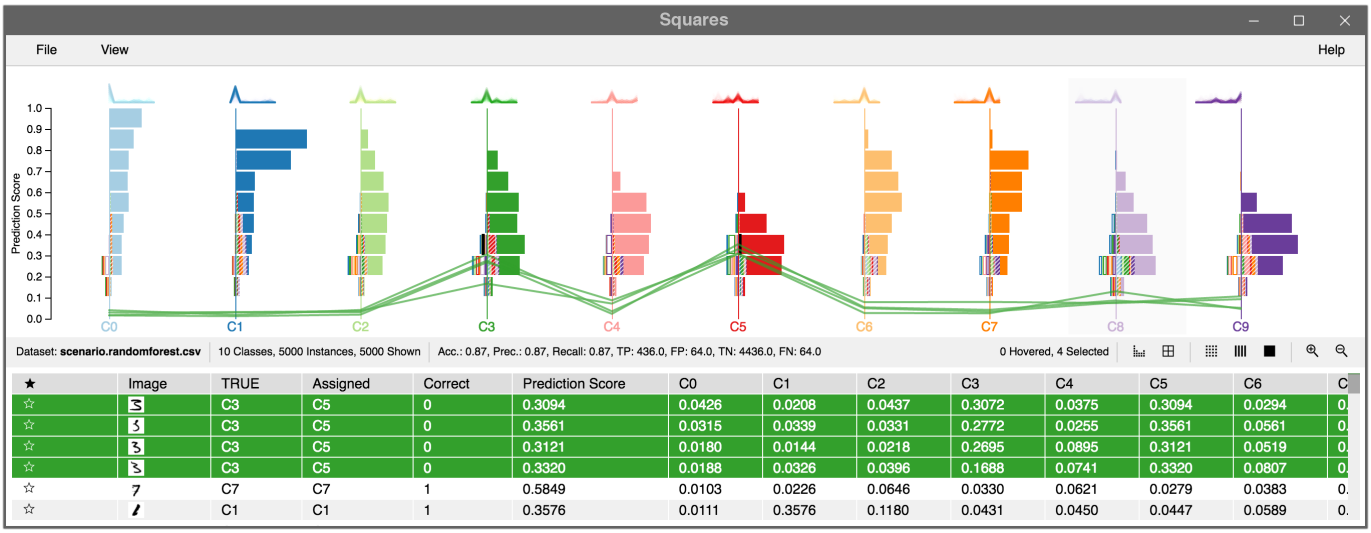
way as the boxes it represents. The truncation indicators use $\rangle$, $\rangle\rangle$ and $\rangle\rangle\rangle$ to represent 1–10 boxes, 11–100 boxes, 101–1000 boxes, etc. This allows users to still perceive the distribution (i.e., compare two bins and see which has more instances) when there are too many boxes or strips to display.

As with boxes and stacks views, the strips view orders errors towards the axis line such that truncation indicators are more likely to encode only correctly classified instances. In cases where a mix of instances are truncated (e.g., errors and non-errors) the truncation indicator is colored gray. Users can then hover over the truncation indicators to reveal a summary of truncated instances or click on them to view the corresponding instances in the table. Note that because this may interfere with visual performance analysis, users are advised to toggle to the stacks view or downsample the data in these cases.

## Connecting Performance to Data

Squares displays performance information that can signal classifier problems. However, previous work has shown that viewing the actual data indicating performance issues is key to providing insights [29]. Squares therefore supports direct access to instances (G3) by allowing users to click on boxes, strips, or stacks to reveal corresponding instances or groups of instances in an adjacent table (Figure 6).

**After the controlled study:** In our initial design, to allow users to examine instance attributes, Squares only displayed selected instances in an adjacent table on demand (when clicked). After the controlled study, we enhanced the coupling between Squares and the table view in the following manner: 1) Through bi-directional selection linking, Squares allows users to access instance properties by selecting boxes, strips, or stacks from the visualization, and use the table to find interesting instances (e.g., instances with a certain attribute) and review them in the visualization (Figure 6). 2) Squares allows users to apply filters based on attribute values to focus on interesting and important




| ★ | Image | TRUE | Assigned | Correct | Prediction Score | C0 | C1 | C2 | C3 | C4 | C5 | C6 | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ☆ | | C3 | C5 | 0 | 0.3094 | 0.0426 | 0.0208 | 0.0437 | 0.3072 | 0.0375 | 0.3094 | 0.0294 | 0. |
| ☆ | | C3 | C5 | 0 | 0.3561 | 0.0315 | 0.0339 | 0.0331 | 0.2772 | 0.0255 | 0.3561 | 0.0561 | 0. |
| ☆ | | C3 | C5 | 0 | 0.3121 | 0.0180 | 0.0144 | 0.0218 | 0.2695 | 0.0895 | 0.3121 | 0.0519 | 0. |
| ☆ | | C3 | C5 | 0 | 0.3320 | 0.0188 | 0.0326 | 0.0396 | 0.1688 | 0.0741 | 0.3320 | 0.0807 | 0. |
| ☆ | | C7 | C7 | 1 | 0.5849 | 0.0103 | 0.0226 | 0.0646 | 0.0330 | 0.0621 | 0.0279 | 0.0383 | 0. |
| ☆ | | C1 | C1 | 1 | 0.3576 | 0.0111 | 0.3576 | 0.1180 | 0.0431 | 0.0450 | 0.0447 | 0.0589 | 0. |

Fig. 6. Bi-directional coupling between the visualization and table allows users to view instance properties in the table by selecting boxes, strips, or stacks from the visualization, or locate interesting instances found in the table in the visualization.

patterns in a subset of the data. 3) Users can bookmark and tag instances of interest, annotated in the visualization with a star and the tag text (Figure 4), to keep track of interesting instance-level findings.

## 5 Usage Scenarios

Here we describe example usage scenarios inspired by observations and feedback from actual practitioners using Squares on their own data (e.g., Twitter Sentiment dataset, entity extraction dataset for an address extractor) during its iterative design. To improve clarity, we describe these scenarios in the context of the MNIST handwritten digits dataset [24], relating the scenarios back to figures in this paper.

**Comparing models:** Mary wants to build a handwritten digit classifier for an app she is creating to automatically scan receipts into a digital format. The commercial machine learning toolkit she is using to build her classifier supports a variety of machine learning models. She decides to experiment with two different models, a support vector machine (SVM) and a random forest model. She trains and tests the models on handwritten digits data she has collected, and is surprised to observe that both models show the exact same accuracy of 0.87. She loads the classification results of both models into Squares to investigate further (see Figure 1). With Squares she immediately sees that although both models show the same accuracy performance, the solid colored stacks in the SVM model are positioned much higher on the vertical dimension relative to the solid colored stacks in the random forest model. This indicates that the random forest model is much less confident in its correct predictions than the SVM. In fact, in examining instances in the random forest model (see Figure 4), she can see that some correctly classified instances have similar prediction scores on incorrect classes (indicated by the polyline in Figure 4 showing a similar prediction score for the hovered instance on both the correct class C5 and the incorrect class C3). This is problematic because it means that even slight variations or noise in the data may easily flip predictions from correct to incorrect (ideally, correct predictions will have high scores for the correct class and low scores for all others). After examining her models in Squares, she decides to use the SVM model in her app because of its superior performance.

**Improving a model:** Bob works for a postal service and is tasked with creating a handwritten digit classifier for an automated mail sorting machine that can distribute mail to the correct delivery workers according ZIP codes they service. Not being an expert in machine learning, Bob decides to use a random forest model, which has relatively few hyperparameters to understand and tune. He loads the classification results into Squares, sees that the classifier has low confidence in its predictions because many instances are distributed towards the bottom of each axis, and decides to investigate the performance issues. First, he expands some of the classes into strips view to locate any severe errors that he should deal with first. He notices some strips of the wrong color towards the top of both C1 and C7, which indicates the wrong prediction with high confidence. He clicks on each strip to view the corresponding instances in the table and sees that in both cases the instances were mislabeled; the classifier was actually predicting the instances correctly, but they were just given incorrect labels before training (which explains their high prediction scores). He bookmarks these instances with "mislabel" tags to fix the labels later (see starred and tagged instances in Figure 4). Bob then examines some classes with high between-class confusion. He notices the sparklines above class C5 has two strong peaks on the correct class C5 and the incorrect class C3 indicating confusion between 3s and 5s (see callout in Figure 4). He inspects some of the C3 instances incorrectly assigned to class C5, by clicking on one of the green striped stacks in C5 and viewing the instances in the table (see Figure 6). He can see that some of the 3s do in fact resemble 5s (e.g., '3' in the second row of the table in Figure 6). He hypothesizes that these errors could be fixed by increasing the resolution of pixels used as features in his model. He goes back to his data and first fixes the mislabels that he found using Squares. He then reprocesses the data to double the input resolution he was using (from $7 \times 7$ to $14 \times 14$ pixels) and retrains his random forest classifier. He reloads the results into Squares to see a boost in performance with fewer errors and more confident correct predictions.

## 6 Evaluation

We conducted a controlled experiment to evaluate the effectiveness of Squares for performance analysis. The experiment consisted of two parts (Figure 8). In the first part, we compared Squares to an interactive confusion matrix in estimating overall-, class-, and instance-level performance because confusion matrices are one of the most commonly used tools for examining classifier performance as many standard performance metrics can be derived from their cell values [34]. We created an interactive confusion matrix as recommended by previous work to provide more direct access to data [29]. In the second part of the study, we examined the distribution capabilities that Squares supports for estimating score-based metrics and class confusion according to instance scores on all classes. Because confusion matrices provide no support for viewing score distributions and only show between-class confusions on winning classes (not confusions indicated by scores on all classes), we only evaluate Squares in this part.

### 6.1 Conditions: Confusion Matrix vs. Squares

We compared the following two visualizations:

**Confusion Matrix:** An interactive confusion matrix (see Figure 7) displaying instance counts in each cell and color-coded to emphasize cells with larger counts (mapping white to 0 and red to the largest cell count). Cells on the diagonal show the numbers of correct predictions for each class (cf. solid boxes in Squares); non-diagonal cells in a column show false positives for the corresponding class (cf. striped boxes in Squares); non-diagonal cells in a row show false negatives for the corresponding class (cf. outlined boxes in Squares). Although it is possible to remove the diagonal from the color coding to emphasize errors, we found the uniform coloring scheme to be most prevalent in existing machine learning tools and toolkits (e.g., Scikit-learn [30], AzureML [1]). We therefore chose to compare Squares to this standard display instead of creating a specialized confusion matrix optimized for our study tasks.

**Squares:** Our interactive visualization for multiclass performance analysis. (In this experiment, we used our initial design which did not support the strips view, truncation indicators, bookmarking, and enhanced coupling between Squares and the table view.)

Note that, like with Squares, we connected the confusion matrix to a sortable table for displaying selected instances as rows (and including per-class score information). Participants could click on cells in the confusion matrix to view corresponding instances. We also chose not to include any summary statistics for both conditions for two reasons: 1) there is no consensus on what summary statistics (totals, accuracy, precision/recall, etc.) to show in a confusion matrix, and 2) we wanted to examine how well participants can understand and use the visualizations as opposed to how well they can read and compare numbers.

### 6.2 Tasks and Datasets

We created six tasks representing common performance analysis needs (see Section 3). Tasks 1–3 pertain to estimating overall-, class-, and instance-level performance, respectively. Tasks 4–6 pertain to estimating score-based performance and confusion between classes. Specifically, the tasks were:

- T1: Select the classifier with the larger number of errors (this required displaying two visualizations side-by-side).
- T2: Select one of the two classes with the most errors.
- T3: Select an error with a score of .9 or above in the wrong class.
- T4: Select the classifier with the worst distribution (this required displaying two visualizations side-by-side).
- T5: Select one of the two classes with the worst distribution.
- T6: Select the two classes most confused with each other.

For each task, we used the Chars74K images dataset [12] that contains 62,992 synthesized characters from various computer fonts. To understand the impact of class size on each performance visualization, we created both a small and large class-size dataset by choosing a list of 5 and 15 English letters as the classes, respectively, and avoiding naturally confusing pairs such as "I" and "J."

We generated classifiers for each task using the SVM algorithm. For T1, we generated two classifiers for comparison: a "good" classifier with 0% of the data mislabeled and a "bad" classifier with 7% of the data mislabeled by randomly flipping ground truth labels before passing the data through the SVM. For T2, we introduced errors by randomly selecting two classes, and adding 5% and 2% mislabels between them for the small and large class-size cases, respectively. For T3, we introduced errors by randomly adding 5% mislabels over all classes and 1% mislabels between a random pair of classes. For T4, we again generated two classifiers for comparison, this time by creating a "good" classifier with 0% of the data mislabeled and a "bad" classifier by blurring the scores with a nonlinear function ($s' = s^{0.6}$) and then re-normalizing them. For T5, we blurred the scores of two randomly selected classes. Finally, for T6, we increased scores of the confused classes by a random value within $[0, 0.5]$ and then re-normalized.

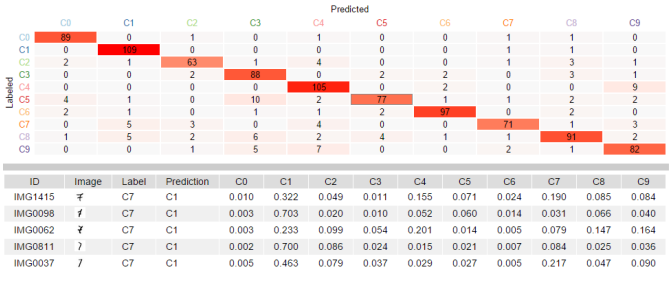



Fig. 7. The interactive confusion matrix used in our controlled study. Each cell shows instance counts and is color-coded the cells by the number. Selecting cells, columns or rows reveals selected instances in the adjacent sortable table (below the matrix).

### 6.3 Study Design and Procedure

We ran the comparison part of the study as a 2 (Visualization: Squares vs. ConfusionMatrix) $\times$ 2 (Class-Size: Small vs. Large) $\times$ 3 (Task: T1, T2, and T3) within-subjects design (see Part 1 in Figure 8). Small class size classifiers had 5 classes and Large ones had 15. To control ordering and learning effects, we prepared two versions of our datasets, and counterbalanced the order of the visualizations and datasets. We also created a third version of our datasets for tutorial and practice. We fixed the order of class size (from Small to Large) and the order of tasks (T1, T2, and T3). In addition, we ran the second part of the study as a 2 (Size: Small vs. Large) $\times$ 3 (Task: T4, T5, and T6) within-subjects design (see Part 2 in Figure 8).

Each session began with a brief introduction to the study topic (i.e., multiclass classification) and an overview of the study procedure. We then asked the participants to fill in a background questionnaire.

In Part 1, we began each condition with an introduction to the corresponding visualization and a hands-on tutorial with 6 practice tasks (Small $\times$ T1–T3 and Large $\times$ T1–T3). Participants then practiced on their own with another 6 tasks. For the actual study tasks, participants performed three variants for each of the three tasks (T1–T3) per class size (Small and Large). Before starting a task, participants were asked to read the task description and then press a "Start" button to reveal the task. Participants could select an answer from a multiple choice list of all possible options displayed on screen or select an answer by clicking on rows in the table for instance-level tasks. Each question had an "I don't know" option that would move participants to the next task. Participants were given 60 seconds to complete each task. If an answer was not submitted within 60 seconds, the system moved participants to the next task.

After each set of tasks per class size, participants completed a post-class-size questionnaire. After completing tasks for both class sizes in a condition, participants filled in a post-visualization questionnaire. After completing both conditions, participants filled out a questionnaire comparing the two visualizations.

Participants went through a similar process in Part 2 but only with Squares and the three remaining tasks (Small $\times$ T4–T6 and Large $\times$ T4–T6). At the end of the study, participants completed a post-study questionnaire asking about their experience. The study lasted approximately 90 minutes and participants received a $20 lunch coupon.

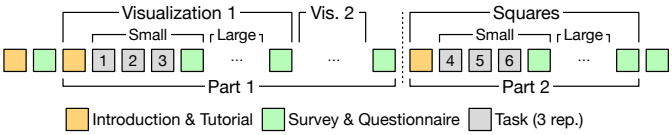



Fig. 8. The study consists of two parts. In the first part, we compared Squares to an interactive confusion matrix. In the second part, we evaluate only Squares for estimating score-based metrics.

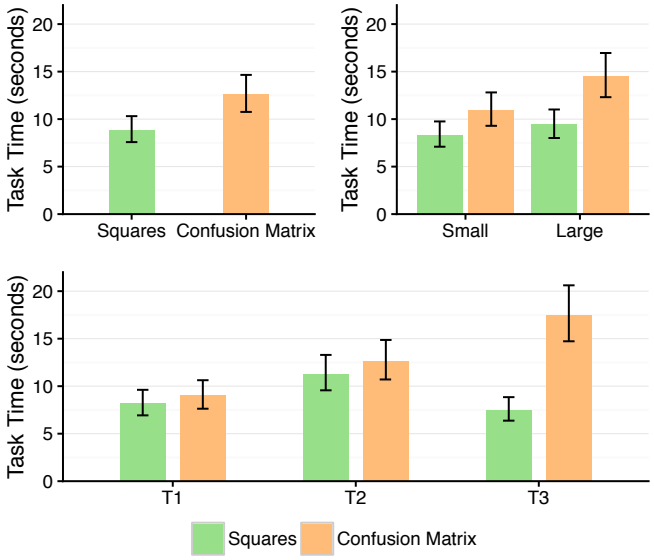



Fig. 9. Task time means. (top-left) per visualization: participants perform tasks faster with Squares. (top-right) per visualization and class-size: participants tended to perform better with Squares as class size increased from Small to Large. (bottom) broken down by visualization and task: participant performance with Confusion Matrix was disproportionately more affected on T3 tasks compared to Squares. Error bars represent 95% confidence intervals.

### 6.4 Participants and Equipment

We recruited 24 participants via an internal mailing list for people interested in machine learning at a large technology company. Eligible participants had to have built at least one machine learning model in the past using any existing tools, be familiar with the terms false positive and negative, and not be color-blind. Participants included data scientists, engineers, researchers, program managers, and interns. 63% of participants self-reported having passing (3 participants) or some knowledge (11) of machine learning while 42% reported being knowledgeable (8) or an expert (2). Participants also reported being moderately familiar to familiar with confusion matrices (mean = 3.6, median = 4 on a 5-point Likert scale from 1=Not at all familiar to 5=Very familiar).

We ran up to four participants at a time, performing tasks independently in each session in a conference room. Each participant worked on a 2.0 GHz Windows 8 Lenovo laptop with 8GB RAM. We attached a 24" Dell LCD display running at a $1920 \times 1200$ resolution to each laptop, as well as a mouse and keyboard. We turned the laptop away from participants; participants could only look at the attached monitor and use the attached mouse and keyboard.

### 6.5 Results

#### Part 1: Task Time and Accuracy

Since the task times were skewed, we performed a log transform of the data prior to analysis. Because there is a tradeoff between task time and accuracy (i.e., participants could reduce task time by answering carelessly), we performed our statistical analyses of task time conditioned on the participants answering correctly. We analyzed the task time data with a linear mixed model: $\log(\texttt{TaskTime}) \sim \texttt{Visualization} \times \texttt{ClassSize} \times \texttt{Task} + (1|\texttt{ParticipantID})$.

We assess the absolute fit of the model with two pseudo-$R^2$ values. For only fixed effects, the marginal pseudo-$R^2$ was 0.221. For fixed and random effects, the conditional pseudo-$R^2$ was 0.515. We conducted an ANOVA with the model using both Satterthwaite's and Kenward-Roger's approximations of degrees of freedom. We found a significant main effect of Visualization ($p < .001$), with Squares ($M = 8.84s$[1]) being faster than the confusion matrix ($M = 12.57s$) (Figure 9, top-left). We also found a significant main effect of Class-Size ($p < .001$), with tasks involving Small class sizes taking less time ($M = 9.54s$) than those involving Large ones ($M = 11.65s$). We also found a significant main effect of Task ($p < .001$). Post-hoc tests indicate that T1 took less time ($M = 8.59s$) than both T2 ($M = 11.93s$, $p < .001$) and T3 ($M = 11.44s$, $p < .001$).

We found a significant interaction effect between Visualization and Class-Size ($p = .012$), indicating that participants performed better with Squares as class size increased from Small to Large (Figure 9, top-right). We also found an interaction effect between Visualization and Task ($p < .001$), indicating that participants' performance with the confusion matrix ($M = 17.42s$) was disproportionately more affected on T3 tasks compared to Squares ($M = 7.51s$) (Figure 9, bottom). Finally, we found an interaction effect between Class-Size and Task ($p = .001$).

For accuracy, we performed an analysis using a generalized linear model with binomial distributions (i.e., logistic regression). To avoid overfitting, only main effects are modeled. We found a significant main effect of Visualization ($p < .001$), with Squares leading to significantly more correct answers ($M = 98.9\%$) than the Confusion Matrix ($M = 91.4\%$). We also found a main effect of Task ($p < .001$), with both T1 ($M = 96.9\%$) and T2 ($M = 98.7\%$) leading to significantly more correct answers than T3 ($M = 92.7\%$). We did not find a significant effect of Class-Size.

| Visualization | T1 | | T2 | | T3 | |
|---|---|---|---|---|---|---|
| | 5 | 15 | 5 | 15 | 5 | 15 |
| **Helpfulness** | | | | | | |
| Squares | 4.3 | 4.3 | 4.7 | 4.1 | 4.1 | 4.7 |
| Confusion Matrix | 3.7 | 3.7 | 3.8 | 3.5 | 3.5 | 3.1 |
| **Preference** | | | | | | |
| Squares | 20 | 17 | 23 | 20 | 23 | 23 |
| Confusion Matrix | 5 | 7 | 2 | 5 | 2 | 2 |

Table 1. Subjective responses: (top) means of participant responses on how helpful (5=Very helpful) the visualization was by task for each class size; (bottom) the numbers of participants who preferred the visualization by task for each class size.

#### Part 1: Subjective Responses

From our study questionnaires, participants indicated that Squares was more helpful for all tasks (T1–T3), in particular for T3 ($M = 3.46$ for Confusion Matrix vs. $M = 4.69$ for Squares on a 5-point Likert scale with 5 = Very helpful). Table 1 shows participant responses on how helpful the visualization was for each task per class size.

In terms of overall preference, 83% (20) of participants indicated preference for Squares for 5-class problems and 79% (19) for 15-class problems. Table 1 shows a similar trend per task.

#### Part 2: Task Time, Accuracy, and Subjective Responses

The second part of our study only involved score distribution-based questions with the Squares visualization. Therefore, here we only report descriptive statistics (i.e., means). Overall, participants completed the three tasks quickly ($M = 9.9s$, $SD = 4.9s$) with high accuracy ($M = 95\%$) regardless of class sizes. They also reported that Squares was helpful for completing the tasks. Table 2 shows a breakdown of average task time (in seconds), accuracy, and participant reported helpfulness (5-point Likert scale with 5=Very helpful) on all tasks per class size.

[1]Least squares means of $\log(\texttt{TaskTime})$ are transformed back to seconds and reported.

| Task | Time (s) | | Accuracy (%) | | Helpfulness | |
|---|---|---|---|---|---|---|
| | 5 | 15 | 5 | 15 | 5 | 15 |
| T4 | 5.6 | 5.3 | 100 | 99 | 4.6 | 4.6 |
| T5 | 8.5 | 8.6 | 97 | 97 | 4.6 | 4.6 |
| T6 | 13.0 | 18.1 | 84 | 91 | 4.2 | 4.4 |

Table 2. Task time means, accuracy, and reported helpfulness (5=Very helpful) for tasks T4–T6 for each class size.

## 7 Discussion and Future Work

### 7.1 Efficient Performance Analysis

Machine learning is a complex and time-consuming process where practitioners must experiment with a wide variety of model inputs. Tools for more efficiently assessing performance of alternative models and prioritizing efforts in debugging issues can help ease the process. Our study results show that Squares helps practitioners estimate multiclass performance not only faster but also more accurately than an interactive confusion matrix for our overall-, class-, and instance-level analysis tasks. This is particularly interesting given that respondents of our survey on current classification practices reported that measuring overall- and class-level performance was relatively easy with current tools. Several participants also commented on Squares' presentation of performance information at multiple levels of detail as one of the things they liked most (e.g., *"I could scan a lot of information quickly," "Much easier to spot errors," "Granular and at the same time general overview of the classifiers is great"*). The inefficiency of the confusion matrix, particularly for instance-level tasks where participants had to search for severe errors by sorting the adjacent table (standard in current tools), was also mentioned by several participants in their questionnaire responses (e.g., *"Not possible to find high/low score instances efficiently," "I would prefer to have a way to look at the error points in one place rather than searching and fishing them"*).

Instance-level score information has been shown in previous work to help practitioners prioritize efforts in improving model performance [6]. Score distributions can, for example, help practitioners distinguish between classifiers reporting similar summary statistics or direct practitioners towards more severe errors or problematic classes. Most common performance analysis tools for machine learning, however, do not visualize scores. Squares supports estimating score-based performance by design. Part 2 of our evaluation also shows that participants were able to efficiently and accurately answer our score-based performance tasks. Several participants also commented on score information as one of the things they liked most about Squares (e.g., *"Easy to visualize error scores, making strong FP/FN stand out"* and *"Seeing the distribution of scores is very helpful"*).

Note that we designed the confusion matrix used in our evaluation based on those appearing in common machine learning toolkits and added interactivity to better support error analysis. Future work may consider comparing Squares to alternative confusion matrix designs, such as confusion matrices without color shading along the diagonal to emphasize shading on errors or confusion matrices that support row/column reordering. Future work may also compare Squares to other performance visualizations proposed in the literature.

### 7.2 Preference and Familiarity

Our questionnaire responses also show that our participants preferred Squares on all tasks even though on average they reported being already familiar with the confusion matrix. Participant comments about their preference include: *"[Squares] gave a better insight of what is going on in the classifier at the very low level," "I would use this right now if I could,"* and *"Had fun for the first time while classifying!"* Of the participants who preferred the confusion matrix, some commented about its familiarity and simplicity: *"Confusion matrices are familiar, so easy to understand"* and *"more straightforward, lower learning curve."* Others commented about the ability to see numbers: *"I prefer having numbers than pure display" and "numbers help in some case."* Interestingly, participants commented that they wanted both visualizations to contain more numbers: *"Add some numbers to [Squares], and everyone will give up on confusion matrices," "Wanted numbers for total sum [in Squares]," and "It would be nice to have the error total for each row/col (minus correct) [in confusion matrix]."* While Squares can and does display some summary statistics (removed in our study for fair comparison to the confusion matrix) and some tools augment confusion matrices with statistics (e.g., sums, precision, recall), different scenarios require optimizing for different metrics making it impossible to know in advance which to display. Alternatively, we could display all available metrics; this, however, would likely make performance analysis and comparison cognitively taxing.

### 7.3 Scalability

Our survey responses on multiclass practices showed that a large proportion (41%) of classification tasks involve between 3–20 classes. While confusion matrices are generally considered scalable to many classes (simply requiring more rows/columns to the matrix), we were surprised to see an interaction between Visualization and Class-Size on task time in our study. Specifically, we found that Squares does not seem to impose a significant overhead with the growth of class size or with the type of task, as opposed to the confusion matrix, which was associated with a significant decrease in efficiency on larger class sizes and on T3. This result is likely because the confusion matrix was not necessarily designed for efficient visual perception, but rather simplicity and efficient use of screen space. For scalability to larger datasets, Squares employs an interactive stacks and strips views to display performance while still supporting direct access to subsets of data, and supports toggling between the three views (boxes, strips, and stacks).

Squares aims to support 3–20 class classification problems. We evaluated Squares with up to 15 classes showing that, as the number of classes increase, model- and class-level performance estimation appear to scale well (see Figure 9). We hypothesize this is because model- and class-level performance estimation are supported via differentiating textured/outlined boxes from solid ones. What may become difficult is distinguishing between different errors within the same column (i.e., differentiating between adjacent textured/outlined boxes of different colors). This is partially mitigated through interactions but could be further improved in future work by optimizing color assignments to minimize the number of adjacent boxes with similar colors. In addition, while we found that Squares could support displaying up to 15 classes as columns in a single row, further investigation is needed to validate how well Squares extends to up to 20 classes (perhaps by distributing columns across multiple rows so that stacks receive enough pixel space for efficient visual analysis). Extending Squares to support classification problems with more than 20 classes is also an open area for future work.

## 8 Conclusion

We presented Squares, an interactive performance visualization for multiclass classification problems. Squares displays information used to derive several common performance metrics and helps practitioners prioritize efforts in debugging performance problems while supporting direct access to data. Our design of Squares is informed by a survey we conducted on classification practices and needs, iterative feedback from real users, and previous work. Our controlled study showed that Squares helps practitioners assess performance significantly faster and more accurately than a common confusion matrix, and was preferred over the confusion matrix by most participants.

Squares continues a trend in recent work on usable machine learning, advocating for integrated tools that connect various steps in the machine learning process. Squares specifically supports multiclass classification. Future work may investigate instance-based visualizations for other machine learning tasks such as ranking and extraction.


## Acknowledgments

We thank the support and feedback from the Machine Teaching Group at Microsoft Research.

## REFERENCES


[1] Microsoft Azure Machine Learning. https://studio.azureml.net. Accessed: 2016-03-31.

[2] Microsoft Power BI. https://powerbi.microsoft.com/. Accessed: 2016-07-31.

[3] Python. https://www.python.org/. Accessed: 2016-03-31.

[4] B. Alsallakh, A. Hanbury, H. Hauser, S. Miksch, and A. Rauber. Visual methods for analyzing probabilistic classification data. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1703–1712, 2014.

[5] M. Aly. Survey on multiclass classification methods. *Neural Networks*, (11):1–9, 2005.

[6] S. Amershi, M. Chickering, S. M. Drucker, B. Lee, P. Simard, and J. Suh. ModelTracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*, pages 337–346, 2015.

[7] M. Ankerst, C. Elsen, M. Ester, and H.-P. Kriegel. Visual classification: an interactive approach to decision tree construction. In *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 392–396. ACM, 1999.

[8] B. Becker, R. Kohavi, and D. Sommerfield. Visualizing the simple bayesian classifier. *Information Visualization in Data Mining and Knowledge Discovery*, 18:237–249, 2001.

[9] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[10] D. Caragea, D. Cook, and V. G. Honavar. Visual methods for examining support vector machine results, with applications to gene expression data analysis. Technical report, 2005.

[11] J. Choo, H. Lee, J. Kihm, and H. Park. iVisClassifier: An interactive visual analytics system for classification based on supervised dimension reduction. In *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology - VAST '10*, pages 27–34, 2010.

[12] T. E. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP)*, pages 273–280, 2009.

[13] C. Ferri, J. Hernández-Orallo, and R. Modroiu. An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, 30(1):27–38, 2009.

[14] E. Frank and M. Hall. Visualizing class probability estimators. In *Proceedings of European Conference on Principles of Data Mining and Knowledge Discovery*, pages 168–179. Springer, 2003.

[15] A. N. Gorban, B. Kégl, D. C. Wunsch, and A. Y. Zinovyev. *Principal Manifolds for Data Visualization and Dimension Reduction*, volume 58. Springer, 2008.

[16] M. Harrower and C. A. Brewer. ColorBrewer.org: an online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1):27–37, 2003.

[17] F. Heimerl, S. Koch, H. Bosch, and T. Ertl. Visual classifier training for text document retrieval. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2839–2848, 2012.

[18] G. Holmes, A. Donkin, and I. H. Witten. Weka: A machine learning workbench. In *Proceedings of Australian New Zealnd Intelligent Information Systems Conference - ANZIIS '94*, pages 357–361. IEEE, 1994.

[19] R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.

[20] D. H. Jeong, C. Ziemkiewicz, B. Fisher, W. Ribarsky, and R. Chang. iPCA: An interactive system for PCA-based visual analytics. *Computer Graphics Forum*, 28(3):767–774, 2009.

[21] B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. In *Proceedings of the 15th European Conference on Machine Learning - ECML '04*, pages 217–226, 2004.

[22] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[23] J. Krause, A. Perer, and K. Ng. Interacting with predictions: Visual inspection of black-box machine learning models. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*, pages 5686–5697, 2016.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[25] S. Lin, J. Fortuna, C. Kulkarni, M. Stone, and J. Heer. Selecting semantically-resonant colors for data visualization. *Computer Graphics Forum*, 32(3):401–410, 2013.

[26] Mathworks. MATLAB and Statistics Toolbox Release 2012b, 2012.

[27] J. G. S. Paiva, W. R. Schwartz, H. Pedrini, and R. Minghim. An approach to supporting incremental visual data classification. *IEEE Transactions on Visualization and Computer Graphics*, 21(1):4–17, 2015.

[28] K. Patel, N. Bancroft, S. M. Drucker, J. Fogarty, A. J. Ko, and J. Landay. Gestalt: Integrated support for implementation and analysis in machine learning. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology - UIST '10*, pages 37–46, 2010.

[29] K. Patel, J. Fogarty, J. A. Landay, and B. Harrison. Investigating statistical machine learning as a tool for software development. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '08*, pages 667–676, 2008.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[31] S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, C.-H. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. P. Mesirov, T. Poggio, W. Gerald, M. Loda, E. S. Lander, and T. R. Golub. Multiclass cancer diagnosis using tumor gene expression signatures. volume 98, pages 15149–15154, 2001.

[32] P. Rheingans and M. DesJardins. Visualizing high-dimensional predictive model quality. In *Proceedings of Visualization 2000*, pages 493–496, 2000.

[33] C. Seifert and E. Lex. A novel visualization approach for data-mining-related classification. In *Proceedings of the 13th International Conference Information Visualisation*, pages 490–495, 2009.

[34] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[35] J. T. Townsend. Theoretical analysis of an alphabetic confusion matrix. *Perception & Psychophysics*, 9(1):40–50, 1971.

[36] S. van den Elzen and J. J. van Wijk. BaobabView: Interactive construction and analysis of decision trees. In *Proceedings of the IEEE Conference on Visual Analytics Science and Technology - VAST '11*, pages 151–160, 2011.

[37] B. Zadrozny and C. Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Proceedings of the 18th International Conference on Machine Learning - ICML '01*, volume 1, pages 609–616, 2001.

[38] X. Zhu and I. Davidson. *Knowledge Discovery and Data Mining: Challenges and Realities*. IGI Global, Hershey, PA, USA, 2007.