Visual Analytics as End-User Programming

Advait Sarkar[1], Alan F Blackwell[1], Mateja Jamnik[1], and Martin Spott[2]

[1] Computer Laboratory, University of Cambridge
advait.sarkar@cl.cam.ac.uk
alan.blackwell@cl.cam.ac.uk
mateja.jamnik@cl.cam.ac.uk
[2] BT Research and Technology, Adastral Park, Ipswich
martin.spott@bt.com

Keywords: POP-I.B. Barriers to Programming; POP-III.D. Visualisation

Abstract. We describe our view of visual analytics as a form of end-user programming that reduces expertise barriers to analytical tasks, and discuss two projects that aim to make analytical programming more visual.

1 Introduction

The activities surrounding the gathering, management, processing and analysis of large datasets are collectively referred to as "data analytics", or sometimes colloquially "big data".

Data analytics involves a great deal of end-user programming. Commonly used tools include Microsoft Excel, R [1], SPSS, Python, Java, SQL databases, Tableau, and Spotfire. The data analysts program in SQL to manipulate their data. Using tools such as Tableau and Spotfire, they are able to program visualisations of the data. Some are able to write programs using R and SPSS to perform statistical analyses on the data. A smaller number still are able use Python and Java along with machine learning libraries (e.g., scikit-learn [2] or WEKA [3]) to create sophisticated models. There is some overlap; Excel enables data manipulation as well as interactive visualisation, R enables quite sophisticated data manipulation and statistics as well as static visualisation, and Tableau does offer some basic data manipulation functionality.

In many of these tools, there is a disconnect between the programming language and the observable output; the metaphors used for each are discordant. For instance, with machine learning packages in Python/Java, the data being operated on is in a text file or database, the program is written in a separate document, and the output is typically rendered in text on a command line console. A user of such a system must have expertise that spans all these representations in order to fully understand the flow of logic and data and to properly interpret the output. Consequently, programming with machine learning packages has a very high expertise barrier to entry. Similarly, when an analyst of many time series who wishes to find clusters of similarly behaving time series cannot simply write a clustering program; they must instead make do with the features of the tools available – repeatedly visualising various aggregations of the time series in Excel or Tableau in an opportunistic, trial and error fashion, for instance.

We are pursuing a line of research where we attempt to bring the visual representations used for programming these scripts closer to the visual representations of their output, ideally making the visualisation itself the programming language with which the user specifies an analytical procedure. In the following sections, we discuss two new tools for data analysis that embody this idea.

2 Teach and Try

Teach and Try [4] is a spreadsheet-based interaction technique that can be used to build and apply sophisticated statistical models such as neural networks, decision trees, support vector machines and linear regression.

Users mark a range of cells to indicate that they have confidence in those values. The marked cells are used to train a statistical model. Once the training set has been specified in this manner, the user can select any other range of cells, potentially overlapping with the training set, in order to apply the model to those cells. If cells in the new selection are empty, they will be filled in. Otherwise, they will be coloured according to their deviation from the model prediction.

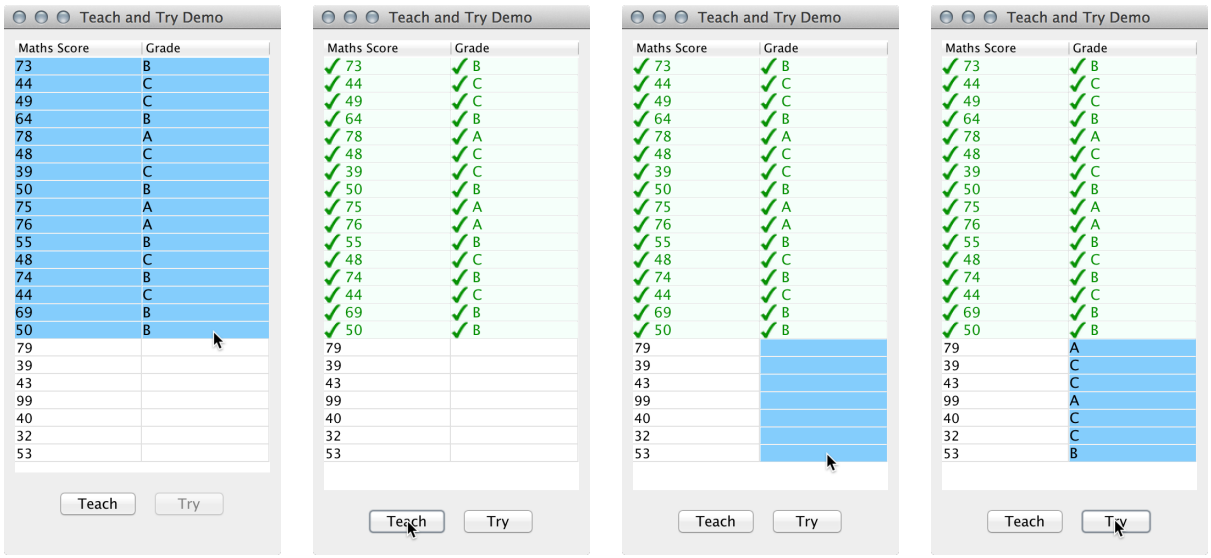

Fig. 1: The "filling" capability of the prototype, depicted as a sequence of actions from left to right. This resembles an actual scenario presented to the participants of our study, in which they were asked to imagine themselves as a maths teacher grading students based on their score on a recent exam. Each row is a student. The first column is their score, and the second column is their grade. A score of 75 or above gets an A, 50 to 74 gets B, and 49 or below gets C. Some of the Grade column is pre-populated, and the participants were asked to use the software to "quickly grade the remaining students". The model being implicitly built in these figures is a decision tree classifier.

This sequence of operations is illustrated in Figure 1. The user first makes a selection of a number of rows, and then clicks on the "Teach" button. At this point, the selected rows are added to a training dataset. The cells are visually marked as "taught" by colouring the text green, colouring the background a faint green, and placing a green check mark icon in the cell.

Next, the user selects cells from a single column and clicks on the "Try" button, as in panels 3 and 4 of Figure 1. The variable in the selected column is now interpreted as the *target* or *dependent* variable for the statistical model, and the variables in the unselected columns are interpreted as the *feature vector* or *independent* variables. The software interprets cell selection bounds as parameters for the model, and this allows the user to build and apply a relevant model with a single interaction. This is contingent on the data in the spreadsheet being laid out in a well-defined relational schema. Upon clicking the "Try" button, the software trains a statistical model on the rows of data previously taught, and applies the model to the rows containing the current selection. If the cells are empty, then the model's predictions are used to populate the

cell contents. If cells in the new selection already contain values, as in Figure 2, their values are not altered. Instead, they are coloured on a red-green scale to reflect the deviation of those values from the expectations of the model, and a question mark icon is added.

Using the familiar metaphor of the spreadsheet, Teach and Try unifies the visual representation of the programming language, the visual representation of the data, and the visual representation of the model output.

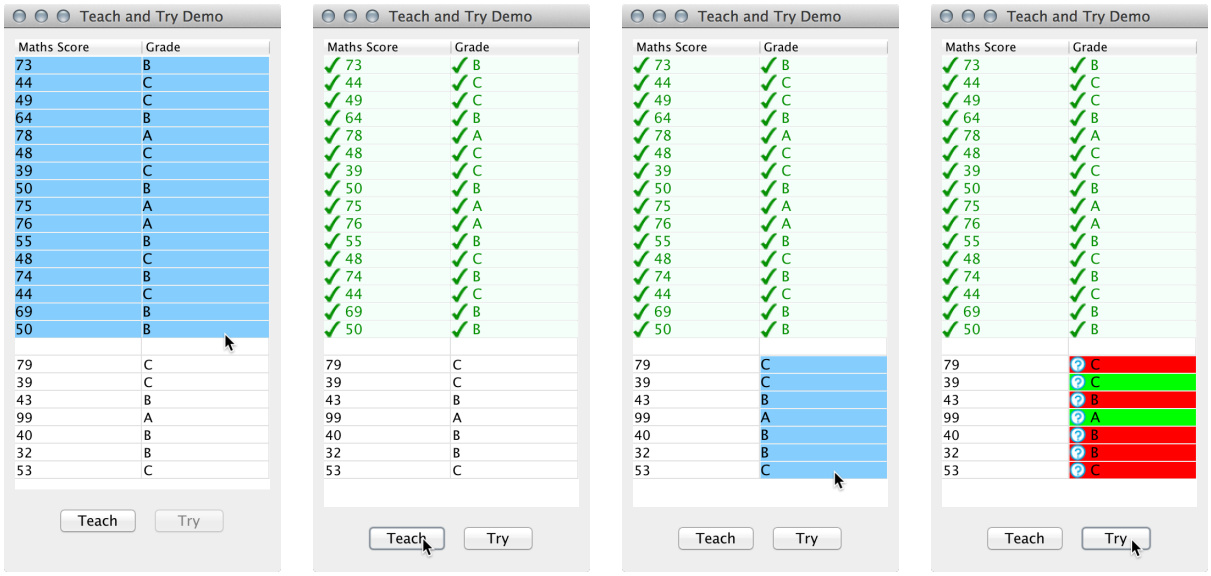

Fig. 2: The "evaluation" capability of the prototype, depicted as a sequence of actions from left to right. This resembles an actual scenario presented to the participants of our study. They were asked to imagine themselves as a maths teacher assessing the competency of a colleague, who had graded the latter section of the students. The first section of the Grade column is pre-populated with "correct" data as per the marking scheme from Fig. 1, and the participants were asked to use the software to "check whether the new teacher understands how to grade papers". Here, the machine learning task is that of classification, so there are only two colours assigned to the cells; green for "correct" and red for "incorrect". In other scenarios, where the target variable was numerical and continuous, the cells were coloured according their percentage error on a linear red-green scale.

3 Gatherminer

To understand the Gatherminer software, we must first discuss its underlying visualisation: the gatherplot. The gatherplot is motivated by the observation that while there are many excellent types of data visualisation available, a lot of data-preprocessing is often required before the visualisations yield a compelling insight. The idea behind the gatherplot is that a strategic automatic layout algorithm can vastly improve the perceptibility of analytic insights, even to those without domain expertise in interpreting such visualisations.

The gatherplot is an augmented time series visualisation. It takes the form of a colour-mapped matrix where each row is an individual time series and each column is an individual time point. A cell is therefore a single data point within a single time series, coloured according to the magnitude of its value. This representation has a number of useful properties [5]. For instance, it is compact, and allows for the visualisation of each original data point without ad-hoc aggregation of the data; each cell can be shrunk to a single pixel in size before the

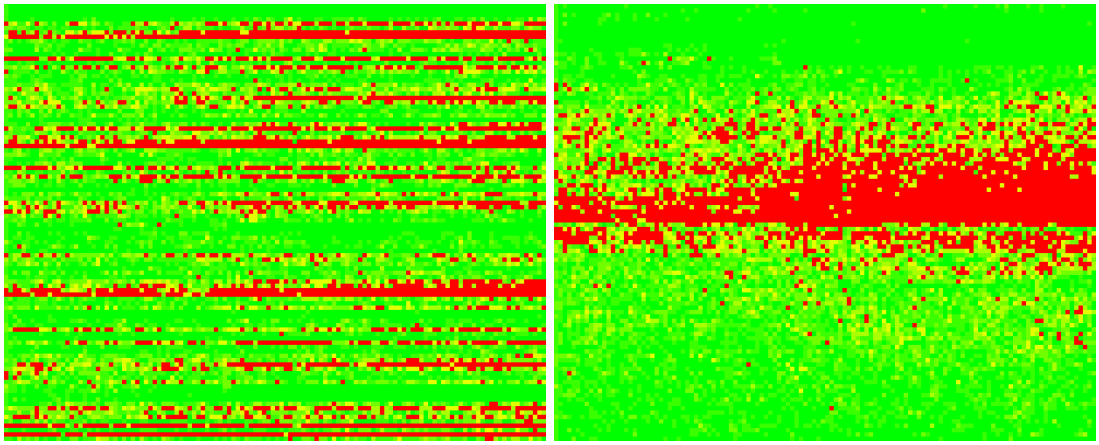

Fig. 3: Two plots showing the improvement in analytic insight achieved by applying intelligent layout algorithms to existing visualisations. Left: an ordinary colour-mapped data matrix. Right: a gatherplot produced by strategically reordering the data rows before visualisation. Each row of cells is an individual time series, with time on the horizontal axis. Each data point is coloured according to its value. Here, we have a linear red-green scale, with green corresponding to low values and red corresponding to high values.

visualisation ceases to be lossless. The tradeoff is that a difference in hue is not as perceptible as the difference in position afforded by, e.g., a line graph [6].

For promoting insight, the visualisation format alone is not sufficient; a layout algorithm must now be applied. For this, a number of clustering, sorting or optimisation methods can be employed, but we focus on one method here. Our prototype reorders the time series such that those which are most similar are placed close together, hence the "gather". Specifically, the final layout minimises the sum of pairwise distances between neighbouring time series. That is, we use a sliding exponentially-weighted distance metric:

$$\text{distance}(T^{(a)}, T^{(b)}) = \sum_i \sum_j \left| T_i^{(a)} - T_j^{(b)} \right| \cdot \mathrm{e}^{-(|i-j|)} \tag{1}$$

Where $T^{(a)}$ and $T^{(b)}$ are two time series whose elements are indexed by i and j respectively. For n series we find an ordering $\{T^{(1)}, T^{(2)}, ..., T^{(n)}\}$ that minimises $\sum_{i=1}^{n-1} \text{distance}(T^{(i)}, T^{(i+1)})$. The specific metric is unimportant as it can be changed for the task at hand.

A similar technique of reshuffling data series was proposed by Bertin with his 'reorderable matrix' [7]. Bertin proposed a visual procedure where pieces of paper representing rows of a matrix were cut and manually reordered on a flat surface. Several years hence, we now have the tools and sufficiently advanced knowledge of clustering techniques to adapt this method for large datasets.

The resulting visualisation clearly exposes interesting analytical features such as outliers and inflections. For example, in Fig.3, on the left is the colour-mapped matrix visualisation of a time series dataset in the order that the dataset appears in the original data file. On the right is a gatherplot of the same dataset, which has been laid out so that similar time series are closer together. In the gatherplot, a group of time series with consistently high values is easily spotted in the middle of the graph.

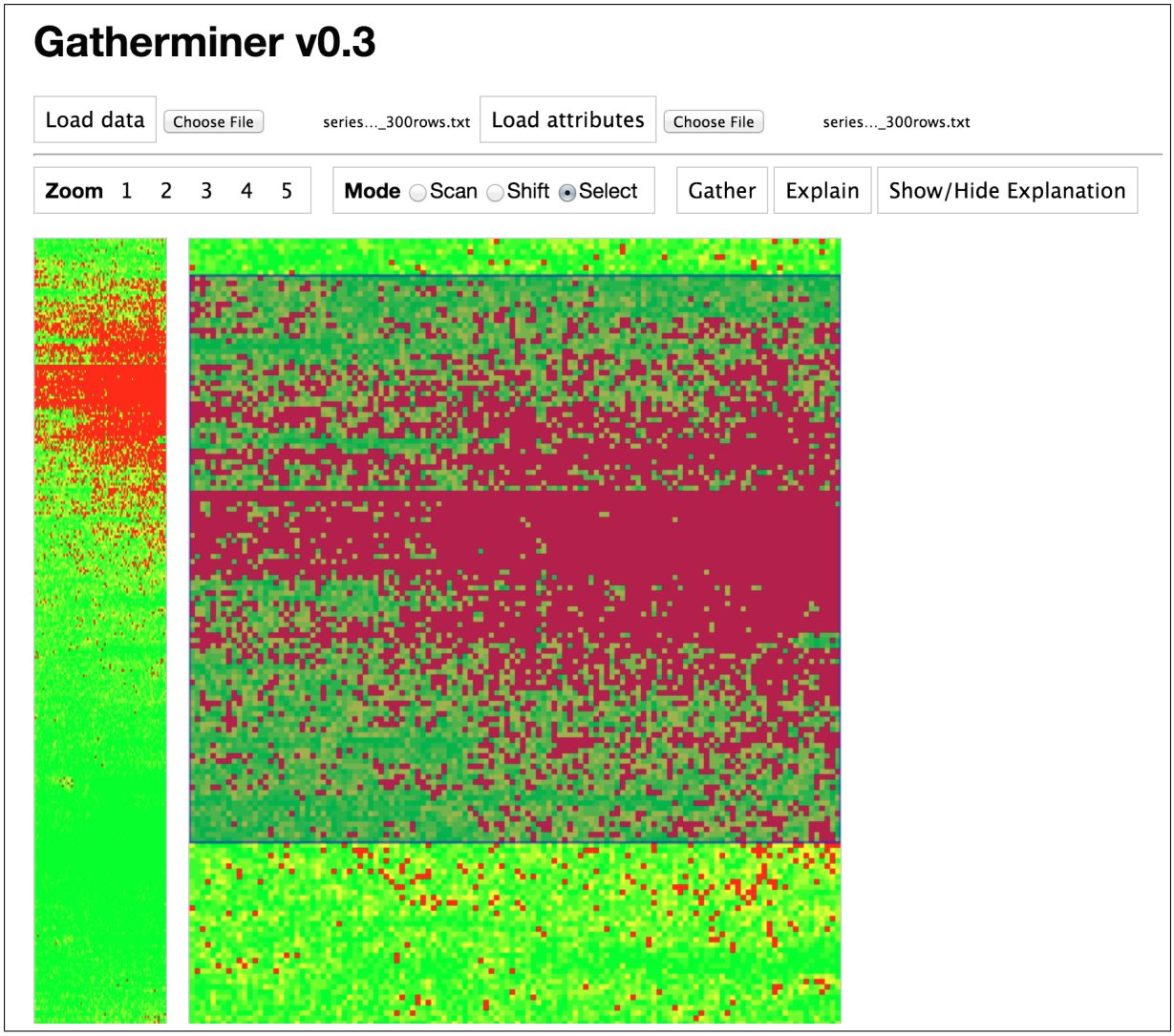

Fig. 4: Gatherminer screenshot depicting a zoomed region of interest being selected for machine learning.

The Gatherminer software leverages the gatherplot visualisation to provide an intuitive interface for rule mining and decision tree learning on multivariate time series data. With visual artefacts such as coloured blobs and streaks appearing prominently in gatherplot visualisations, the next logical step is to use the visualisation itself as the interface for querying the data.

Gatherminer (Fig. 4), in a manner similar to the interactive machine learning applications presented by Fails and Olsen [8] or Wu and Madden [9], allows users to select regions of interest in order to mark them as 'interesting'. This constitutes manual annotation of a subset of data points. In Fails and Olsen's *Crayons* application, the user drew on an image to convey certain learning parameters. By directly annotating the image, the user interactively built a classifier for portions of an image, e.g., a classifier that detects a human hand against a background. Similarly, in Gatherminer, the user draws to annotate. While *Crayons* performed image classification directly on image data, Gatherminer extends that style of interaction to arbitrary data, as long as an appropriately useful visualisation can be rendered.

Once the selection is made, Gatherminer builds a decision tree in order to discover which attributes of the time series best classify the interesting (selected) regions from the non-interesting ones. Thus, the user asks the software to 'explain' regions of interest by querying for explanatory attributes. These explanations are displayed in multiple formats, including bar charts, aggregate

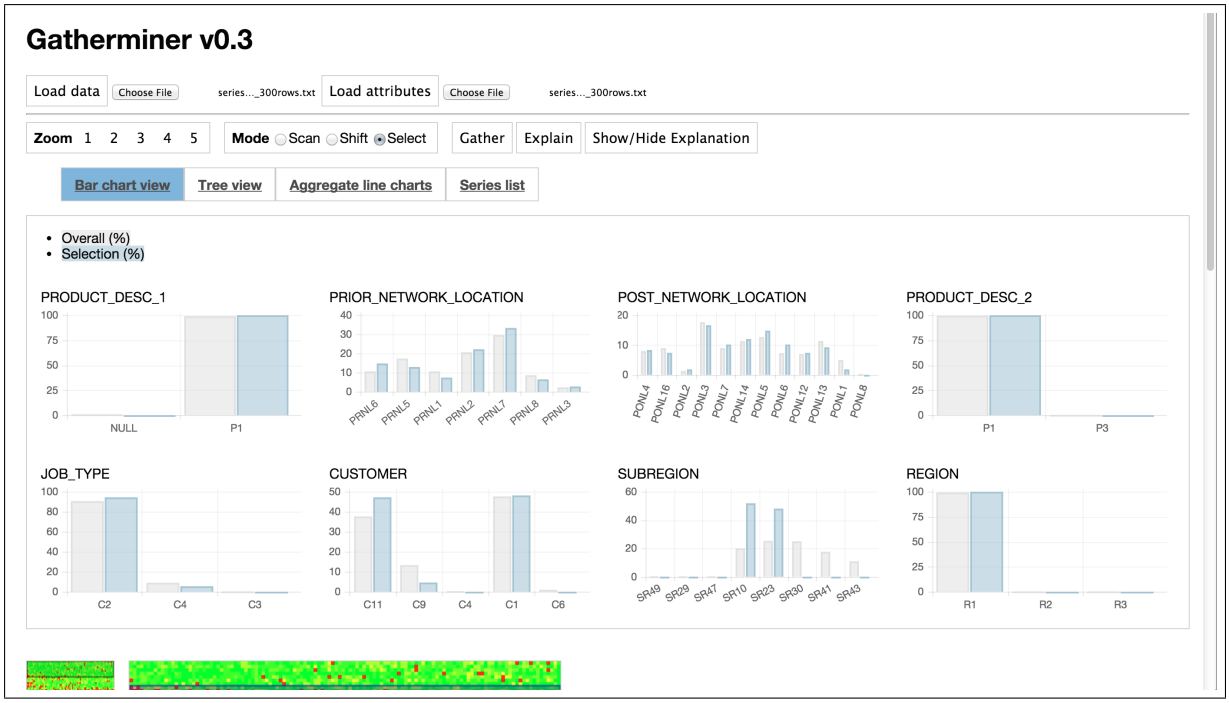

Fig. 5: A region of interest being 'explained' using bar charts. In particular, the "subregion" charts show that the subregions 'SR10' and 'SR23' occur much more frequently in the region of interest than in the overall dataset.

line charts, and the decision tree itself. These features foster a process of iterative exploration where the user refines their selections according to their understanding of the data.

As with Teach and Try, with Gatherminer we have brought various disparate representations closer to each other. In this instance, the data visualisation is used to specify the parameters for a machine learning procedure.

4 Conclusion

Our research focuses on the potential of visual analytics as an end-user programming activity. By bringing the interaction language closer to the visual representations of its output, that is, by attempting to unify 'how we tell the computer to do it' with 'what we want the computer to do', we hope to reduce the expertise requirements of these analytical procedures.

We live in an age where the application of sophisticated analytics to ever-expanding volumes of data is becoming increasingly useful. However, the tools we use to conduct analytics are still very primitive; supposedly visual tools only provide a thin, artificial interface layer over statistical programming. Consequently, advanced data analytics is the preserve of a small group of highly-skilled professionals, despite the fact that a much larger population could benefit from access to sophisticated analytics. We hope that our novel interfaces for analytics which exploit interactive visualisation reduce the programming knowledge (and potentially domain knowledge) required of data workers.

5 Acknowledgements

Advait Sarkar is supported through a EPSRC Industrial CASE studentship sponsored by BT Research and Technology, and also by a Premium Studentship from the University of Cambridge Computer Laboratory.

References

1. R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.
2. F. Pedregosa and et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
3. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.
4. Advait Sarkar, Alan F Blackwell, Mateja Jamnik, and Martin Spott. Teach and try: A simple interaction technique for exploratory data modelling by end users. In *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium on*, pages 53–56. IEEE, July 2014.
5. MC Hao, Umeshwar Dayal, Daniel A Keim, and Tobias Schreck. Multi-resolution techniques for visual exploration of large time-series data. *EuroVis*, pages 1–8, 2007.
6. William S Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American statistical association*, 79(387):531–554, 1984.
7. Jacques Bertin. *Graphics and graphic information processing*. Walter de Gruyter, 1981.
8. Jerry Alan Fails and Dan R. Olsen. Interactive machine learning. *Proceedings of the 8th international conference on Intelligent user interfaces - IUI '03*, page 39, 2003.
9. Eugene Wu and Samuel Madden. Scorpion: Explaining Away Outliers in Aggregate Queries. *Proceedings of the VLDB Endowment*, 6(8):553–564, 2013.