Formatting by Demonstration: An Interactive Machine Learning Approach

Kesler Tanner
Data Mining Lab
Brigham Young University, Provo, Utah

Christophe Giraud-Carrier
Data Mining Lab
Brigham Young University, Provo, Utah

Dan R. Olsen Jr.
Interactive Computing Everywhere Lab
Brigham Young University, Provo, Utah

ABSTRACT

Many routine formatting tasks are subject to patterns. This is especially true of formatting actions performed by users in Excel. Excel has built-in functionality to perform some of these tasks, however their application requires the user to explicitly define logical rules. We show that by using interactive machine learning techniques, such patterns can be learned automatically by iteratively analyzing actions as they are performed by the user. This decreases the amount of work required of the user, and eliminates the necessity of explicitly defining logical rules. Our results show that many useful formatting patterns can be learned with only a few examples.

General Terms:

Human Computer Interaction, Algorithms

Keywords:

Formatting by Demonstration, Automatic Task Completion, Interactive Machine Learning, Excel

1. INTRODUCTION

Users of desktop applications, especially those who work with Excel to store, display and analyze data, often apply formatting to improve the visualization of their results. Such formatting may be as simple as highlighting every other row, known as banding, to improve contrast, and as elaborate as highlighting specific aspects of the data based on the content of certain cells (e.g., negative numbers, totals). To produce the expected effect, users must either manually repeat the same action through the data sheet or automate the process via Excel's conditional formatting option. The former is inefficient and wearisome, while the latter requires designing non-trivial logical rules.

Consider the simple and rather common case of creating a banding effect. For a large document, highlighting every other row would be very time consuming and one would much rather have a way to automate the process. However, doing so using Excel's conditional formatting requires defining two separate rules and using the functions ODD, EVEN, or MOD, as shown in Figure 1.

Designing such logical rules requires a basic level of understanding of logic and Excel, which limits the number of users who can use the available functionality. Often it is those users who lack this level of familiarity that would benefit most from having this functionality automated. Interestingly, the default placement of conditional formatting on the home ribbon of Excel as well as the number of online articles about "how to use conditional formatting" (e.g., see [5, 1, 3]) are evidence that tools for efficient formatting are valuable and in demand. Yet, assuming that the user should speak in a specific language understood by the computer but possibly foreign to the user is undesirable, and in most situations unnecessary given the clear patterns that underly most formatting activities.

Hence, rather than designing complex rules, the user should be able to show a few simple examples of what is expected while the computer *learns* patterns from the user's actions, and generalizes the patterns to the entire data sheet. In other words, the user instructs the computer to "Watch what I do" and the computer generates the appropriate result based on its observation of the user's actions. The approach is akin to the Programming by Demonstration (PbD) paradigm, that argues that it should be unnecessary for a user to learn a programming language to accomplish a repetitive task [2].

Here, we propose a novel interactive machine learning technique, *Formatting by Demonstration (FbD)*, that captures the functionality of conditional formatting, but as the name implies, only requires the user to provide examples of formatted cells rather than defining explicit rules. Like most PbD algorithms, FbD goes beyond recording simple macros that can be applied repetitiously, and instead attempts to induce patterns from the examples. By adjusting the required user input from logical rules to concrete examples, FbD helps to elicit, develop and debug rules more easily than when defining logic [11]. Furthermore, FbD helps to reduce the learning curve for automatically formatting a document, and expands the functionality to virtually all users independent of their current level of expertise. While the concept of FbD is applicable to a variety of software, we demonstrate its utility here within Excel. We show how FbD can be implemented effectively as a simple Add-in to Excel, and illustrate its use for a number of typical formatting tasks. We then revisit the FbD concept and discuss its applicability beyond Excel.

2. RELATED WORK

A number of applications of the PbD paradigm to document formatting have been proposed. Some of the earliest ones include EBE, a system that synthesizes text transformation programs from examples [12], Tourmaline, a system aimed at formatting text, especially headings, tables and references, based on examples and heuristics [10], and TELS, a system that records user actions and generalizes from them based on heuristics and user-defined, domain-specific rules [15].

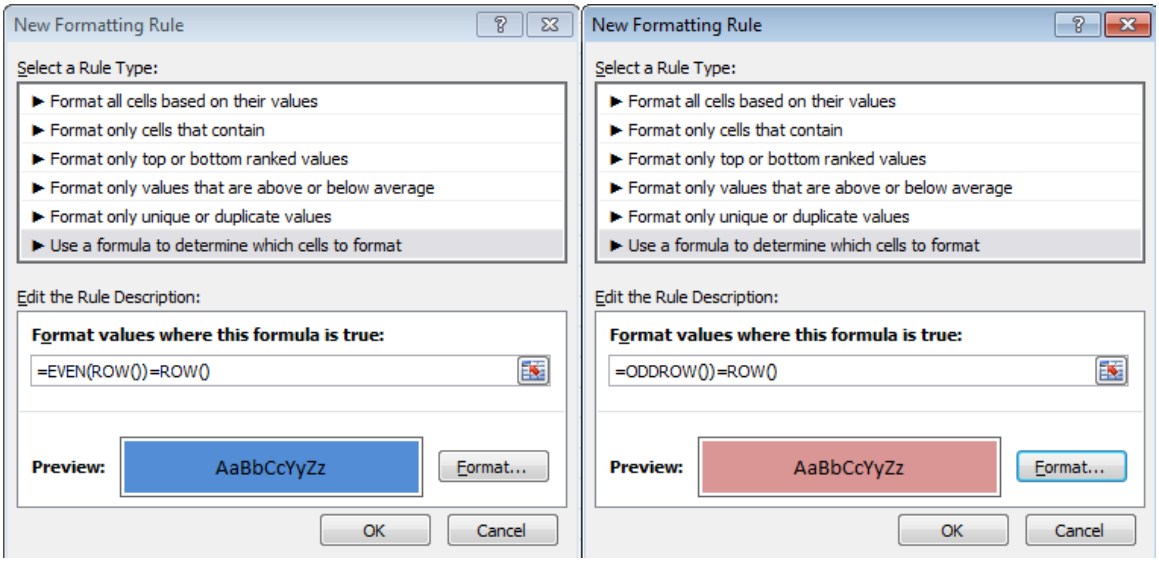


Fig. 1. Rules and Functions Needed to Produce a Banding Effect Using Conditional Formatting in Excel.

Recently, more elaborate systems have been designed, including LAPIS, a system for text editing based on constraint patterns generated by users [8], and SMARTedit, a system based on the Version Space algorithm [9], that induces text-editing programs from examples [6, 7]. Most closely related to our FbD system, since it focuses on spreadsheets rather than text documents and is also implemented in the context of Excel, is an elaborate string manipulation program synthesis system, based on a formal expression language [4].[1] FbD, on the other hand, focuses exclusively on using decision tree induction to learn cell formatting from examples in spreadsheets.

3. FORMATTING BY DEMONSTRATION

The concept of FbD for spreadsheets 1) recognizes that most formatting tasks have underlying patterns based on either the location or the content of various data cells, and 2) proposes that such patterns need not be explicitly applied or described by the user, but can be learned, and consequently generalized to a whole document, from observing a small number of relevant user actions. We use interactive machine learning to implement FbD functionality in Excel.

Our tool is implemented in C# using Visual Studio as a Excel COM Add-in, and its corresponding button, labeled *Auto-Format* for the users, is visible on the rightmost part of the home ribbon, as shown in Figure 2. By default, auto-formatting is on and the button displays "No Patterns" as expected. The fact that the tool is on does not force its usage but ensures that it is available any time the user wishes to take advantage of it, rather than have to remember to first activate the option and then perform formatting actions. When auto-formatting is on, the system continually watches for formatting changes that the user may make.

When a group of cells are formatted, the algorithm compares the cells' previous states to their current states to determine which specific action has occurred, or in the case where multiple formatting actions have occurred, it treats each action separately. For example,

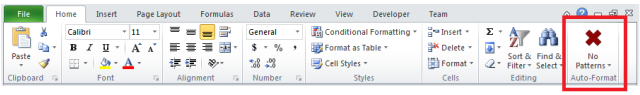


Fig. 2. Excel's Ribbon with the Auto-Format Button Added.

assume that cells A1, B1 and C1 are highlighted, and in addition, the font color of cells B1 and C1 changes. The algorithm would record two separate actions in this case. The cells are then clustered to form ranges. A range is a 1-dimensional vertical or horizontal contiguous collection of cells, here A1-C1 and B1-C1. Each range is then saved in a history of actions and grouped with similar actions. When a 2-dimensional collection of cells is modified simultaneously, the algorithm first determines which dimension is larger and then divides the block into 1-dimensional pieces according to that dimension.

The algorithm operates under the assumption that the user obeys certain felicity conditions [14] that facilitate the system's learning process. In particular, the user is assumed to perform actions *purposefully* and *perfectly*. Purposeful action means that each range that was acted upon, and each range that was skipped, was treated so intentionally. Perfect action means that we assume there exists no range between the first range and the last range that was supposed to be formatted but was not. Under these constraints, the ranges are sorted and a training set is formed starting with the first range's first cell and including all of the cells to the last range's last cell. Each of the cells that did not previously belong to a range is split into 1-dimensional contiguous strips, matching the direction and size of the previous ranges.

Each range is then passed to a feature extraction stage. In order to understand the user's intentions, the algorithm needs to understand what distinguishes the highlighted ranges from the non-highlighted ranges. The set of all possible intentions is clearly infinite, making it necessary to restrict the set of actions to a tractable subset, such that actions in that subset are useful and can be understood. This is a well-known issue in machine learning and PbD systems. For example, SMARTedit requires the user to provide a well-crafted

[1] This system was actually included in the most recent release of Excel 2013 as the Flash Fill feature.

Table 1. Features Extracted from Individual Cells and Cell Ranges.

Features	Values
Range Features	
Mod2	*Column or row mod 2*
Mod3	*Column or row mod 3*
Cell Features	
Type	*Alpha, Alphanumeric, Numeric, Empty*
PosOrNeg	*Negative, Positive, Zero, N/A*
Value	*The actual value of the cell*
Binary Features	
Order	*Greater than, Less than, Equal, N/A*

domain description to circumscribe the version space [7]. In our context, Excel already defines part of this possible world by limiting the actions that can be performed in its editor. We further limit this definition in our choice of features. Three different types of features are extracted from each range, as shown in Table 1.

Range features extract information about the entire range, such as banding or other equally-spaced row/column highlighting. Cell features extract information about the individual cells, such as the type of value stored or the magnitude of the value stored. Finally, binary features capture information about the relative values of cells, such as whether one's value is larger than another's. The algorithm currently accounts for 2 range features, 3 cell features and one binary feature. Hence, in a range covering 3 cells (e.g., A1-C1), extracting features results in a training instance with 14 features: 3 cell feature for each of the individual cells in the range, 2 range features for the range, and 3 binary features (comparing A1 and B1, A1 and C1, and B1 and C1). In general, for a range consisting of N cells, there will be $3N$ cell features, 2 range features, and $\frac{N(N-1)}{2}$ binary features.

As a concrete example of what the training data would look like, consider the case of banding, where the first user action is to highlight cells A1 through C1, where A1 contains the value "Total", B1 contains the value 10 and C1 contains the value -15. Since the range is of size 3, there will be 14 input feature values to extract to build a training example corresponding to this action. Following the order of features in Table 1, the training example will be the vector (1 mod 2, 1 mod 3, Alpha, N/A, "Total", Numeric, Positive, 10, Numeric, Negative, -15, N/A. N/A, Greater Than, HighlightColor), where the last entry is the target action. Further highlighting of similar ranges would result in corresponding training examples that would be grouped together as a training set.

The test data is created similarly to the training data. It begins with the range following the final range of the training set and extends to the final range in the document that has the same dimensionality as the training data, as illustrated in the simple example of Figure 3. Each range is also passed to feature extraction to create test instances for the learning algorithms. As the test set is simply the complement of the training data in the document, it evolves through time as training data is added through the user's actions.

When the size of a training set (i.e., action history) reaches a minimum threshold (the default is 3), the learning algorithm attempts to find a pattern to the user's actions. The training data is presented to an ID3 decision tree learning algorithm [13], which will produce a formatting pattern only if the entropy of the induced decision tree is zero. This is because of our constraint on the user operating perfectly, and the negative effect of auto-formatting incorrectly.

Before proceeding, recall that our selected features (see Table 1) essentially define the space of patterns that our learning algorithm can induce. When no formatting actions have been performed by the user (i.e., the training set is empty), all patterns in that space are candidates in a vacuous sense, and any (or all) of them could be returned by the learning algorithm. By extension, it is clear that any time an action is performed, or equivalently any time an example is added to the learner's training set, the set of candidate patterns becomes smaller. These observations are summarized in the following theorem.

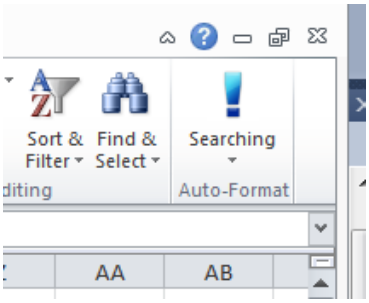


Fig. 4. The Auto-Format Button Switches from "No Patterns" to "Searching" as Learning Begins.

THEOREM 1. *Let T be a set of training examples. If no pattern is found when training the learning algorithm on T, then no pattern can ever be found by training the learning algorithm on a superset of T.*

Hence, if our system cannot find a pattern after some actions have been performed, it will never be able to do so. Note that while this may mean that there is indeed no pattern to the user's actions, that is not necessarily the case. It is possible that the user's intended pattern cannot be induced from our current set of features. In such cases, appropriately extending the set of features, a topic of future research, would solve the problem.

Returning to our FbD system, when the learning algorithm first begins searching for a pattern, the Auto-Format button in the ribbon changes from its default state of "No Patterns" to "Searching," (see Figure 4). The ID3 algorithm executes and one of two things happens. Either a pattern is found or it is not. If no pattern is found, then by Theorem 1 no patterns will ever be found, and the Auto-Format button returns to the "No Patterns" state.

On the other hand, if a pattern is found, the user is immediately notified as the Auto-Format button changes to "Found 1 Pattern" (see Figure 5). At this point, of course, there may still be several candidate formatting patterns, as several patterns may match the current training examples. In order to make the experience as simple as possible for the user, the algorithm chooses only one as its "found" pattern, and now enters a truly interactive machine learning mode.

To confirm, or refute, the algorithm's current finding, the user can click on on the upper half of the Auto-Format button, which causes ID3 to label all test examples and hence the pattern to be applied to the rest of the document. If the user is satisfied with the result, i.e., the induced pattern matches the intended pattern, no further action is required. However, if the induced pattern does not match the intended pattern, the user can easily undo the auto-formatting by selecting the corresponding sub-option (see Figure 6).

The undo action returns the spreadsheet to its state prior to the auto-formatting and leaves the original training set unchanged. The Auto-Format button returns to "Searching" and the user may now perform further relevant actions, that in turn extend the current training set and thus reduce the number of candidate patterns the learner may induce. The above interactive process is then repeated

	A	B	C	D	E	F	G	H	I
1		Support	Confidenc	Coverage	Prevalenc	Recall	Specificity	Accuracy	
2	Support	1	0	0	0	0	0	0	
3	Confidenc	0	1	0	0	0	0	0	
4	Coverage	0	0	1	0	0	0	0	
5	Prevalenc	0	0	0	1	0	0	0	
6	Recall	0	0	0	0	1	0	0	
7	Specificity	0	0	0	0	0	1	0	
8	Accuracy	0	0	0	0	0	0	1	
9	Lift	0	0	0	0	0	0	0	
10	Leverage	0	1	0	0	0	0	0	
11	AddedVal	0	0	0	0	0	0	0	
12	RelativeRi	0	0	0	0	0	1	0	
13	OddsRatic	0	0	0	0	0	0	0	
14	Klosgen	0	0	0	0	0	0	0	
15	Interestin	0	0	0	0	0	0	1	
16		= Training Set			= Testing Set				

Fig. 3. Visual Representation of the Training Set and Testing Set, Based on Actions Range.

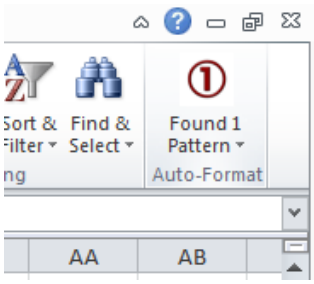


Fig. 5. The Auto-Format Button Switches from "Searching" to "Found 1 Pattern" If Learning Is Successful.

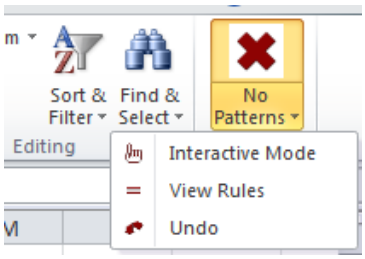


Fig. 6. Interactive Learning Mode.

until either the predicted pattern matches the intended pattern or no pattern can be found. We have found that the number of required examples differs with the layout of the Excel sheet and the type of pattern, but in most situations 3 or 4 examples are sufficient for the patterns expressible by our selected set of features.

Before we show specific applications of FbD, a few remarks about our implementation are in order.

—As an alternative to the try/undo cycle described above, our tool allows a one-at-a-time kind of approach, similar to what is found in most editors' find-and-replace functionality. These tools generally allow the user to do all replacements automatically, or to step through each occurrence and validate the replacement. Both options are useful and applicable to different situations. Automatic replacement is instantaneous, but does not provide the confidence associated with manual user input, while iterative replacement is slower, but increases confidence in the accuracy of each replacement. FbD implements a similar functionality to provide the same benefits. Instead of simply clicking the Auto-Format button, the user can select the "Interactive Mode" sub-option (see Figure 6), which allows her to step through each potential change and either accept or reject the proposed action. Not only does this option give the user greater confidence in the accuracy of the algorithm, it also allows the user to provide fewer examples (creating a more general algorithm) and to deal manually with potential boundary cases.

—While multiple formatting patterns can always be found sequentially, by performing relevant actions for each individually, the tool also supports the discovery and application of multiple patterns at once. In this case, the user would show 3 examples of one pattern (e.g., horizontal banding), which would cause the system to discover that pattern. However, instead of acting on it right away, the user may continue its formatting work in the document with a different pattern (e.g., vertical banding), until the system picks up that second pattern. The Auto-Format button would then first reflect that 1 pattern was found and later that 2 patterns had been found (see the circled number on the Auto-Format button). Upon clicking on the Auto-Format button, the

user would cause both formatting patterns to be applied to the rest of the document.

—While highlighting may be viewed as a single formatting action, our tool treats highlighting as multiple formatting, based on the color selected by the user. As a result it is possible to induce automatically more elaborate highlighting schemes such as situation where rows containing a particular value are highlighted in yellow while rows containing another value are highlighted in green (see Results section for an example).

—To provide further confidence in the algorithm and increase flexibility, the tool offers the option to export the current pattern into logical statements understood by Excel, via the "View Rules" sub-option (see Figure 6). This is possible because the models induced by ID3, namely decision trees, are comprehensible and can be post-processed easily to extract rules. Each branch in the tree represents a comparison and translates into a logical test. By properly nesting these logical statements, we can reverse engineer the tree and provide code which can be used by Excel's conditional formatting to obtain the same results. This enables the curious user to understand the algorithm's reasoning behind its formatting decisions, as well as to save the pattern and apply it to future sheets. For experienced users, this may also be an alternative to both the try/undo cycle and the step-by-step replacement; one could simply look at the rule(s) and confirm their validity.

4. RESULTS WITH FBD

To demonstrate the use and value of FbD, we show a few illustrative examples of before-after scenarios, where the before state represents the set of actions performed by the user prior to the learning algorithm suggesting that it has found a formatting pattern, and the after state represents the spreadsheet following automatic application of the discovered pattern.
Figure 7 illustrates the simple case of horizontal banding. Figure 8 extends banding to both horizontal and vertical directions and to two different colors. Figure 9 illustrates the highlighting of rows where the data shows that the goal has been missed (i.e., the value in the "Actual" column is less than the value in the "Goal" column). Figure 10 illustrates the highlighting of columns based on the content of one of the rows, here the value "John" in the row labeled "4th". Finally, Figure 11 illustrates the highlighting of rows on the basis of the content of one of the columns, here the value "Coca-Cola" in the "Category" column. Note that while the other patterns could be induced from 3 examples only, this pattern actually required 5 examples to be induced by the system.
In comparing our tool's capabilities, as illustrated above, with various tutorials on conditional formatting, we found that FbD is indeed effective on patterns where all necessary information to deduce the pattern exists within the training data, as may be expected. The tool does not have the direct ability to learn patterns where external data influences the user's decision, for example, as in highlighting all rows where the value in column X is less than the value in external cell $Y10$. It is possible to support this functionality indirectly in FbD by having the user append a column with the desired value to the end of the previous cells and include this last column in the examples provided to the algorithm. FbD will then be able to compare against this value and induce the intended pattern. Once the pattern has been applied, the user can remove the extra column.

5. CONCLUSIONS AND FUTURE WORK

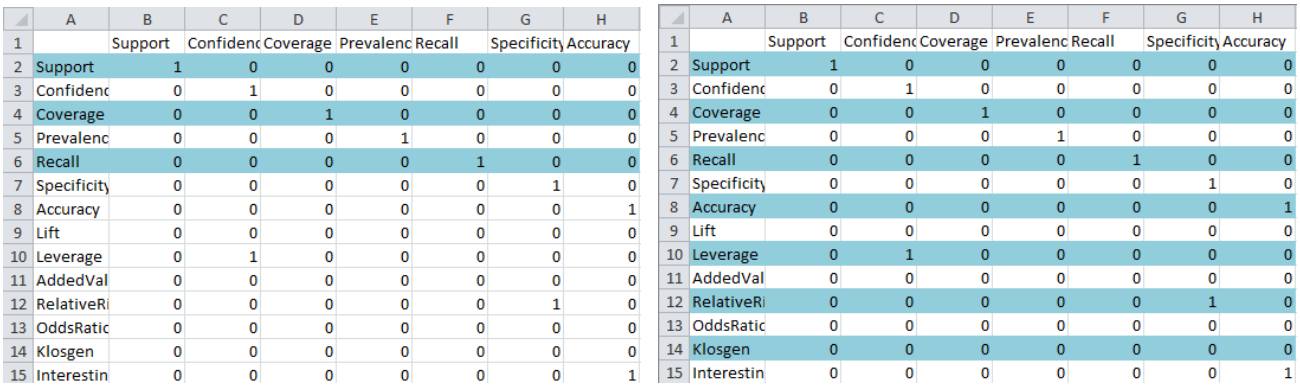
We have introduced the concept of Formatting by Demonstration as an alternative to rule-based conditional formatting, and showed how we have implemented it as an Add-in within Excel. Our results suggest that even with a relatively small set of features, it is possible for an interactive machine learning algorithm to induce a number of interesting formatting patterns with very few examples required from the user.
There are several ways our tool could be extended. As discussed above, some patterns are not found by our system not because they do not exist but because they cannot be expressed with the set of features we have selected. It is, of course, possible to extend this set. However, it would be most interesting to find ways to do so semi-automatically when the system fails but the user has an intended pattern in mind. As far as the current rule extraction is concerned, the tool does not support nesting of condition within rules. In terms of auto-formatting, the completion defaults to the complement of the training set. Rather than having to consider the whole document, it would be interesting to allow the user to mark the area of the spreadsheet over which she wishes the tool to operate and auto-format. The current implementation assumes that the use acts perfectly. A more realistic scenario would allow allow some error to be made by the user. Given enough training data, the system should be able to recognize what examples may be erroeneous and generalize from the correct examples only. Finally, it may be useful to consider running the tool in a kind of strict pattern mode, where it only offers to auto-format the document when only 1 pattern is found.
Most importantly, the concept of FbD is clearly not restricted to Excel. We used Excel here only as a vehicle to demonstrate FbD and present a viable implementation thereof. It is possible to generalize the methodology to any application where formatting is routinely used. As stated earlier, the main assumption of FbD is that most formatting tasks have underlying patterns based on either the location or the content of various pieces of data and these patterns can therefore be induced from specific actions performed by the user. Hence, for an arbitrary application, the implementation of FbD consists of:

(1) Setting up a mechanism to observe and record user actions
(2) Defining a set of features that may be used to characterize a relevant subset of user intentions
(3) Creating training examples from user actions
(4) Using interactive machine learning to induce patterns from these examples

As formatting is common to all editors, it is clear that FbD has broad applicability. For example, assume the application is an Integrated Development Environment (IDE), where the user has her own code formatting style. The IDE is likely to offer ways to customize the format by altering default preferences. However, as in the case of conditional formatting discussed in this paper, this is a burden on the user, who would much rather go ahead and start coding, and let the IDE learn her specific preferences. Starting with features, such as tabbing and parenthesis style, one could design an FbD that observes the user behavior, builds relevant training data and interactively learns formatting preferences.

6. REFERENCES

[1] Contextures. Excel conditional formatting — examples. Online at http://www.contextures.com/xlCondFormat03.html, 2013.



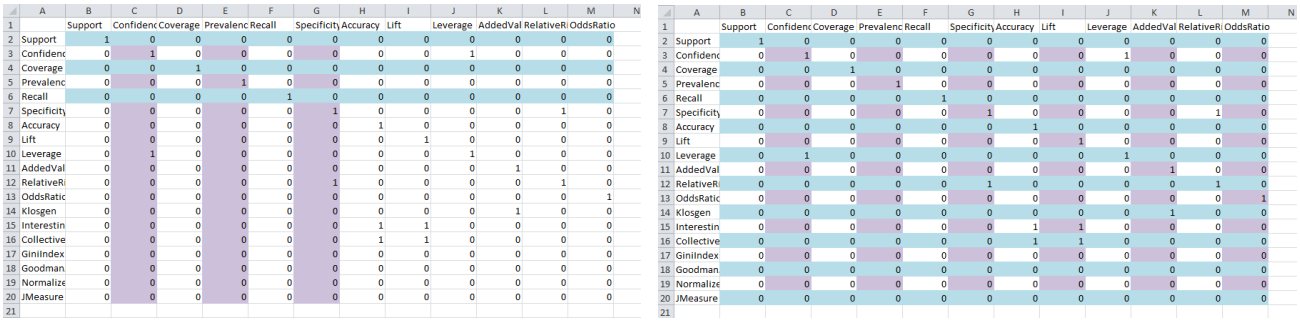
	A	B	C	D	E	F	G	H
1		Support	Confidenc	Coverage	Prevalenc	Recall	Specificity	Accuracy
2	Support	1	0	0	0	0	0	0
3	Confidenc	0	1	0	0	0	0	0
4	Coverage	0	0	1	0	0	0	0
5	Prevalenc	0	0	0	1	0	0	0
6	Recall	0	0	0	0	1	0	0
7	Specificity	0	0	0	0	0	1	0
8	Accuracy	0	0	0	0	0	0	1
9	Lift	0	0	0	0	0	0	0
10	Leverage	0	1	0	0	0	0	0
11	AddedVal	0	0	0	0	0	0	0
12	RelativeRi	0	0	0	0	0	1	0
13	OddsRatio	0	0	0	0	0	0	0
14	Klosgen	0	0	0	0	0	0	0
15	Interestin	0	0	0	0	0	0	1

(a) Before

	A	B	C	D	E	F	G	H
1		Support	Confidenc	Coverage	Prevalenc	Recall	Specificity	Accuracy
2	Support	1	0	0	0	0	0	0
3	Confidenc	0	1	0	0	0	0	0
4	Coverage	0	0	1	0	0	0	0
5	Prevalenc	0	0	0	1	0	0	0
6	Recall	0	0	0	0	1	0	0
7	Specificity	0	0	0	0	0	1	0
8	Accuracy	0	0	0	0	0	0	1
9	Lift	0	0	0	0	0	0	0
10	Leverage	0	1	0	0	0	0	0
11	AddedVal	0	0	0	0	0	0	0
12	RelativeRi	0	0	0	0	0	1	0
13	OddsRatio	0	0	0	0	0	0	0
14	Klosgen	0	0	0	0	0	0	0
15	Interestin	0	0	0	0	0	0	1

(b) After

Fig. 7. Horizontal Banding



(a) Before

(b) After

Fig. 8. Horizontal and Vertical Banding With Different Colors

[2] A. Cypher and D.C. Halbert. *Watch What I Do: Programming by Demonstration*. The MIT Press, 1993.

[3] T. French. Excel conditional formatting. Online at http://spreadsheets.about.com/od/advancedexcel/tp/090822-excel-conditional-formatting-hub.htm, 2013.

[4] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 317–330, 2011.

[5] S. Harkins. 10 cool ways to use Excel's conditional formatting feature. Online at http://www.techrepublic.com/blog/10things/10-cool-ways-to-use-excels-conditional-formatting-feature/3166, 2012.

[6] T. Lau. *Programming by Demonstration: A Machine Learning Approach*. PhD thesis, Department of Computer Science and Engineering, University of Washington, 2001.

[7] T. Lau, S.A. Wolfram, P. Domingos, and D.S. Weld. Programming by demonstration using version space algebra. *Machine Learning*, 53(1-2):111–156, 2003.

[8] R.C. Miller. *Lightweight Structure in Text*. PhD thesis, Computer Science Department, School of Computer Science, Carnegie Mellon University, 2002.

[9] T. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[10] B.A. Myers. Tourmaline: Text formatting by demonstration. In A. Cypher, editor, *Watch What I Do: Programming by Demonstration*, chapter 14. The MIT Press, 1993.

[11] C.G. Nevill-Manning. Programming by demonstration. *New Zealand Journal of Computing*, 4(2):15–24, 1993.

[12] R. Nix. Editing by example. *ACM Transactions on Programming Languages and Systems*, 7(4):600–621, 1985.

[13] J.R. Quinlan. Inductive learning of decision trees. *Machine Learning*, 1:81–106, 1986.

[14] K. VanLehn. Felicity conditions for human skill acquisition: Validating an ai-based theory. Technical Report CIS-21, Xerox Corp., Palo Alto, CA. Palo Alto Research Center, 1983.

[15] I.H. Witten and D. Mo. TELS: Learning text editing tasks from examples. In A. Cypher, editor, *Watch What I Do: Programming by Demonstration*, chapter 8. The MIT Press, 1993.

	A	B	C	D	E
1		Activity	Actual	Goal	
2	Monday	Pushups	52	50	
3	Tuesday	Situps	90	100	
4	Wednesd	Pullups	10	10	
5	Thursday	Jogging	22	20	
6	Friday	Pushups	45	50	
7	Saturday	Situps	90	100	
8	Sunday	Pullups	12	12	
9	Monday	Jogging	30	35	
10	Tuesday	Pushups	60	60	
11	Wednesd	Situps	120	150	
12	Thursday	Pullups	11	12	
13	Friday	Jogging	35	35	
14					

(a) Before

	A	B	C	D	E
1		Activity	Actual	Goal	
2	Monday	Pushups	52	50	
3	Tuesday	Situps	90	100	
4	Wednesd	Pullups	10	10	
5	Thursday	Jogging	22	20	
6	Friday	Pushups	45	50	
7	Saturday	Situps	90	100	
8	Sunday	Pullups	12	12	
9	Monday	Jogging	30	35	
10	Tuesday	Pushups	60	60	
11	Wednesd	Situps	120	150	
12	Thursday	Pullups	11	12	
13	Friday	Jogging	35	35	
14					

(b) After

Fig. 9. Missing Target Goal

	A	B	C	D	E	F	G	H	I	J	K	L
1		Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10	
2	1st	Kade	Kade	Tyler	Brady	Scott	Kade	Skyler	Kade	Kade	Joe	
3	2nd	Steven	James	Scott	Steven	Joe	Steven	James	James	Steven	Skyler	
4	3rd	Brady	Tyler	Joe	Kade	Skyler	Brady	Ryan	Tyler	Brady	Ryan	
5	4th	John	Ryan	John	John	Ryan	John	John	Ryan	Scott	John	
6	5th	Tyler	Skyler	James	Skyler	Kade	Joe	Brady	Skyler	Joe	Tyler	
7	6th	Scott	John	Steven	Ryan	Steven	Scott	Steven	John	Skyler	James	
8	7th	Joe	Scott	Brady	Tyler	Brady	Tyler	Kade	Scott	Ryan	Kade	
9	8th	Skyler	Joe	Kade	Scott	John	Skyler	Joe	Joe	James	Steven	
10	9th	Ryan	Brady	Skyler	Joe	Tyler	James	Scott	Brady	John	Brady	
11	10th	James	Steven	Ryan	James	James	Ryan	Tyler	Steven	Tyler	Scott	
12												

(a) Before

	A	B	C	D	E	F	G	H	I	J	K	L
1		Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10	
2	1st	Kade	Kade	Tyler	Brady	Scott	Kade	Skyler	Kade	Kade	Joe	
3	2nd	Steven	James	Scott	Steven	Joe	Steven	James	James	Steven	Skyler	
4	3rd	Brady	Tyler	Joe	Kade	Skyler	Brady	Ryan	Tyler	Brady	Ryan	
5	4th	John	Ryan	John	John	Ryan	John	John	Ryan	Scott	John	
6	5th	Tyler	Skyler	James	Skyler	Kade	Joe	Brady	Skyler	Joe	Tyler	
7	6th	Scott	John	Steven	Ryan	Steven	Scott	Steven	John	Skyler	James	
8	7th	Joe	Scott	Brady	Tyler	Brady	Tyler	Kade	Scott	Ryan	Kade	
9	8th	Skyler	Joe	Kade	Scott	John	Skyler	Joe	Joe	James	Steven	
10	9th	Ryan	Brady	Skyler	Joe	Tyler	James	Scott	Brady	John	Brady	
11	10th	James	Steven	Ryan	James	James	Ryan	Tyler	Steven	Tyler	Scott	
12												

(b) After

Fig. 10. Highlighting Columns Based on Row Values

	A	B	C	D	E	F
1	Product Name	Unit Price	Units In Stock	Units On Order	Reorder Level	Category
2	7-Up	23	35	0	0	Snapple Group
3	Amp	38	86	0	0	Pepsi
4	Barq's	18	39	0	10	Coca-Cola
5	Crush	17	29	0	10	Snapple Group
6	Dr Pepper	25	6	0	0	Coca-Cola
7	Fanta	30	15	0	10	Coca-Cola
8	Monster	25	120	0	25	Coca-Cola
9	Mountain Dew	30	29	0	0	Pepsi
10	Pepsi	21	22	30	30	Pepsi
11	Red Flash	15	13	70	25	Coca-Cola
12	Sierra Mist	31	31	0	0	Pepsi
13	Simply Orange	19	17	40	25	Coca-Cola
14	Sobe	30	0	0	5	Pepsi
15	Sprite	22	53	0	0	Coca-Cola
16	Sunkist	15	39	0	5	Snapple Group
17						

(a) Before

	A	B	C	D	E	F	G
1	Product Name	Unit Price	Units In Stock	Units On Order	Reorder Level	Category	
2	7-Up	23	35	0	0	Snapple Group	
3	Amp	38	86	0	0	Pepsi	
4	Barq's	18	39	0	10	Coca-Cola	
5	Crush	17	29	0	10	Snapple Group	
6	Dr Pepper	25	6	0	0	Coca-Cola	
7	Fanta	30	15	0	10	Coca-Cola	
8	Monster	25	120	0	25	Coca-Cola	
9	Mountain Dew	30	29	0	0	Pepsi	
10	Pepsi	21	22	30	30	Pepsi	
11	Red Flash	15	13	70	25	Coca-Cola	
12	Sierra Mist	31	31	0	0	Pepsi	
13	Simply Orange	19	17	40	25	Coca-Cola	
14	Sobe	30	0	0	5	Pepsi	
15	Sprite	22	53	0	0	Coca-Cola	
16	Sunkist	15	39	0	5	Snapple Group	
17							

(b) After

Fig. 11. Highlighting Rows Based on Column Values