CORNET: Learning Spreadsheet Formatting Rules By Example

Mukul Singh
Microsoft
Delhi, India
singhmukul@microsoft.com

José Cambronero Sanchez
Microsoft
Redmond, USA
jcambronero@microsoft.com

Sumit Gulwani
Microsoft
Redmond, USA
sumitg@microsoft.com

Vu Le
Microsoft
Redmond, USA
levu@microsoft.com

Carina Negreanu
Microsoft Research
Cambridge, UK
cnegreanu@microsoft.com

Gust Verbruggen
Microsoft
Redmond, USA
gverbruggen@microsoft.com

ABSTRACT

Data management and analysis tasks are often carried out using spreadsheet software. A popular feature in most spreadsheet platforms is the ability to define data-dependent formatting rules. These rules can express actions such as *"color red all entries in a column that are negative"* or *"bold all rows not containing error or failure"*. Unfortunately, users who want to exercise this functionality need to manually write these conditional formatting (CF) rules. We introduce CORNET, a system that automatically learns such conditional formatting rules from user examples. CORNET takes inspiration from inductive program synthesis and combines symbolic rule enumeration, based on semi-supervised clustering and iterative decision tree learning, with a neural ranker to produce accurate conditional formatting rules. In this demonstration, we show CORNET in action as a simple add-in to Microsoft's Excel. After the user provides one or two formatted cells as examples, CORNET generates formatting rule suggestions for the user to apply to the spreadsheet.

PVLDB Reference Format:
Mukul Singh, José Cambronero Sanchez, Sumit Gulwani, Vu Le, Carina Negreanu, and Gust Verbruggen. CORNET: Learning Spreadsheet Formatting Rules By Example. PVLDB, 16(12): 4058 - 4061, 2023.
doi:10.14778/3611540.3611620

1 INTRODUCTION

Millions of users [9] perform their data management and analysis in spreadsheet software. Most popular spreadsheet platforms, such as Microsoft's Excel, allow users to define data-dependent formatting rules. These rules can express actions such as *"color red all entries in a column that are negative"* or *"bold all rows not containing error or failure"*. These rules are typically called *conditional formatting rules* (CF rules). Unfortunately, users have to write conditional formatting rules manually, which requires both programming expertise and familiarity with the particular data platform and its rule language.

A search on the Excel tech help community [8] reveals more than 10,000 questions on conditional formatting as of March 2023. We find that these posts tend to share three key struggles, preventing users from effectively using conditional formatting. First, many users are unaware of CF rules and instead manually format their spreadsheets. Second, when a user does enter a CF rule, they can fail to do so correctly (because of invalid syntax or incorrect logic) and end up manually formatting the sheet. Third, even when a user writes a valid rule that matches their desired formatting, these rules can be unnecessarily complex (and candidates for simplification) or not generalizable (may deviate from the desired behavior when data in the column changes or new data is added to the column).

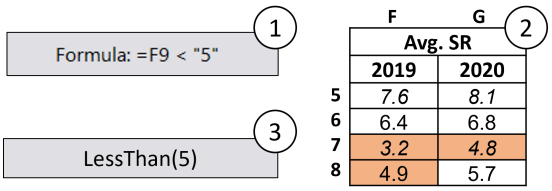


Figure 1: Example of an incorrect CF rule manually written by a user. ① A user defined a custom rule that always evaluates to false (as it compares the cell contents to the string "5" instead of the numeric literal 5). ② Because the rule does not format any cell, the user had to manually format the sheet as reflected in this figure. ③ The correct CF rule, which can be learned by CORNET from 2 examples.

Some of these struggles are reinforced by the need to write custom formulas. For example, in Excel CF rules that require logical operations like OR and AND must be written as custom formulas. Additionally, there is no validation for data types in the standard CF interface, causing surprising results. For example, a rule can (incorrectly) compare a text column to a numeric constant.

Example 1.1. Figure 1 shows a public spreadsheet where the user wanted to highlight cells with value less than 5. Unfortunately, they wrote a rule that incorrectly uses the string "5" instead of the number. Ultimately, they manually formatted the sheet.

In this paper we present a demonstration of CORNET [1] (**C**onditional **ORN**amentation by **E**xamples in **T**ables), a system that allows users to automatically generate a conditional formatting rule from examples, thus mitigating the challenges previously outlined. CORNET takes a small number of user formatted cells as input to learn the most likely formatting rule that generalizes to other cells. CORNET

[1]The full paper can be accessed at https://arxiv.org/abs/2208.06032.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611620

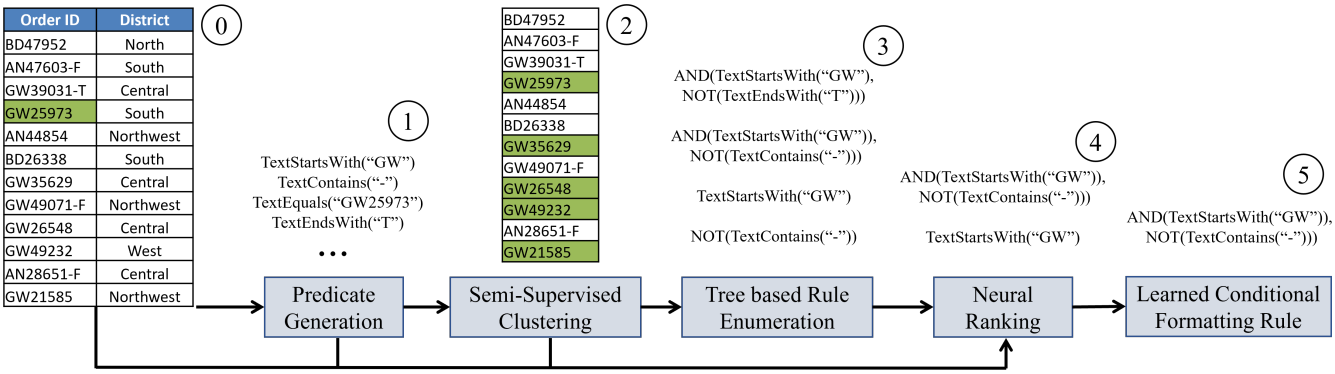


Figure 2: Cornet's architecture illustrated through an example case: ⓪ input table with a single user example (partial formatting), ① predicate generation for all cells in the table, ② semi-supervised clustering using examples and positional information to address the challenge of unlabeled cells, ③ enumerating rules over the clustered column using multiple decision trees, ④ neural ranker to score generated rules, and ⑤ final learned conditional formatting rule.

explores possible predicates for the target column, hypothesizes cell grouping via semi-supervised clustering and then learns candidate rules by employing an iterative tree learning procedure. Since multiple rules may result from this procedure, we train a neural ranker to return the most likely CF rule, given the spreadsheet data, user examples, and candidate rules' execution on the data.

Example 1.2. Consider the example shown in Figure 2, the user wants to color all IDs that start with "GW" but don't contain "-F" or "-T" at the end. To accomplish this via the Microsoft Excel GUI, the user will have to navigate 2 dialog boxes, 3 dropdown menus, perform 9 clicks, and then write the Excel formula `AND(LEFT(A2,2)="GW", NOT(OR(RIGHT(A2,2)="-F", RIGHT(A2,2)="-T")))`. With Cornet, the user can simply color the fourth cell in Figure 2 ⓪, and Cornet learns and applies the correct rule.

In our demonstration, we will show an implementation of a lightweight interface in Microsoft Excel that exercises Cornet and can successfully learn the rule in Example 1.2 and more complex scenarios. Our interface first allows users to provide one or two examples of their desired formatting. [2] They can then invoke Cornet, which will learn a symbolic rule that covers the examples and generalizes to the rest of the column. Finally, Cornet applies the rule learned to format the spreadsheet. We show how both novice and experienced spreadsheet users can use Cornet.

In the remainder of this paper, we present a high-level description of Cornet and provide an outline of our proposed demonstration.

2 SOLUTION SKETCH

A core challenge in Cornet is to learn from few user-provided examples—often just one. Traditional programming-by-example (PBE) systems [5, 6] can carry out an efficient search because they can derive constraints by relating the inputs and outputs for their task. In conditional formatting, however, the output provides a relatively weak binary signal (whether a row is formatted or not). The design of Cornet is meant to mitigate this challenge by ① generating predicates that characterize values in the target formatting column, ② hypothesizing labels for the entire column by clustering over the generated predicates, ③ synthesizing many rules that satisfy the hypothesized groups, and ④ using a dedicated ranker that furthers refines the likelihood of generated rules. Figure 2 shows an overview of these steps, which we describe in following sections.

[2]Text rules can be learnt accurately by providing just one or two examples. Numeric columns typically require more examples because of their large search space.

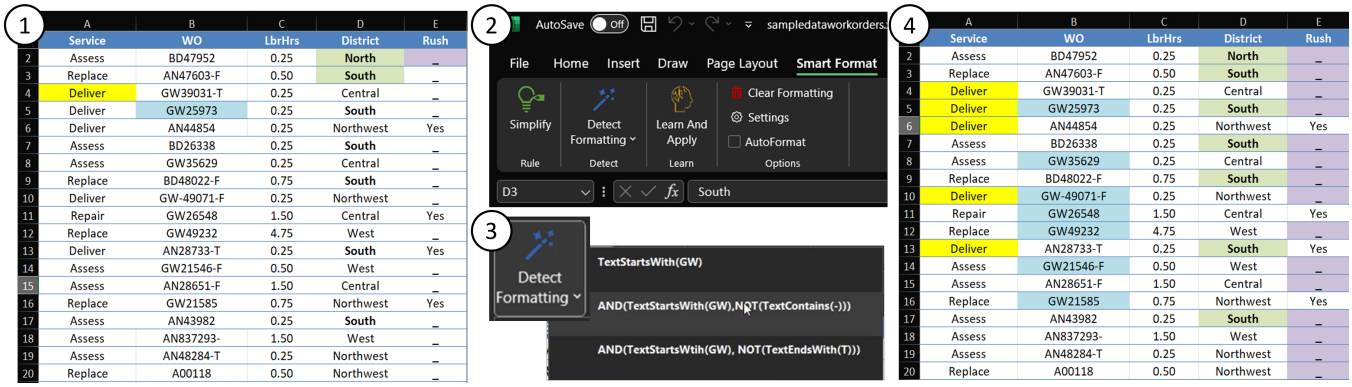
Table 1: Supported predicates and their arguments for each data type. c corresponds to a cell value, n is a numeric literal, d is a datetime literal, and s is a string literal. Cornet uses type-specific generators to instantiate n, d, and s, as necessary.

Numeric	Datetime	Text
greater(c, n)	greater(c, n, d)	equals(c, s)
greaterEquals(c, n)	greaterEquals(c, n, d)	contains(c, s)
less(c, n)	less(c, n, d)	startsWith(c, s)
lessEquals(c, n)	lessEquals(c, n, d)	endsWith(c, s)
between(c, n_1, n_2)	between(c, n_1, n_2, d)	

2.1 Predicate Generation

Cornet computes properties of cell values in the form of boolean predicates—a boolean-valued function that takes a cell c, zero or more arguments and returns true if the property that it describes holds for c. To avoid type errors, all predicates are assigned a type t and they only match cells of their type. Mixed-type columns are assigned *Text* type. We chose predicates, shown in Table 1, based on the operations supported by popular spreadsheet platforms.

Given a column of cells and a predicate, the goal is to initialize each additional argument to a constant value such that the predicate returns true for a strict subset of the column. We use different generators for each predicate type. For example, contains uses tokens obtained by splitting on non-alphanumeric delimiters and also tokens from a prefix trie. For numeric predicates, constants are generated from cell values, column statistics (mean, percentiles, min, max) and popular constants (0, 1, 100). These predicates are



	A	B	C	D	E
	Service	WO	LbrHrs	District	Rush
2	Assess	BD47952	0.25	North	_
3	Replace	AN47603-F	0.50	South	_
4	Deliver	GW39031-T	0.25	Central	_
5	Deliver	GW25973	0.25	South	_
6	Deliver	AN44854	0.25	Northwest	Yes
7	Assess	BD26338	0.25	South	_
8	Assess	GW35629	0.25	Central	_
9	Replace	BD48022-F	0.75	South	_
10	Deliver	GW-49071-F	0.25	Northwest	_
11	Repair	GW26548	1.50	Central	Yes
12	Replace	GW49232	4.75	West	_
13	Deliver	AN28733-T	0.25	South	Yes
14	Assess	GW21546-F	0.50	West	_
15	Assess	AN28651-F	1.50	Central	_
16	Replace	GW21585	0.75	Northwest	Yes
17	Assess	AN43982	0.25	South	_
18	Assess	AN837293-	1.50	West	_
19	Assess	AN48284-T	0.25	Northwest	_
20	Replace	A00118	0.50	Northwest	_

Figure 3: The Cornet user interface. ① Spreadsheet in Excel where the user has formatted a few cells as examples. ② Cornet UI in Excel ribbon as an Add-In listed under tab named *Smart Format*. *Learn and Apply* learns and applies the top rule on the selected column. ③ *Detect Formatting* lists the top 3 rules for the current column. ④ The final sheet after applying the rules.

then used for both clustering and constructing formatting rules. We generate these predicates only for the column the user is currently formatting to prevent the predicate space from exploding.

2.2 Semi-supervised Clustering

Because the user has provided few examples, rule generation at this stage would only reflect the properties of these positive examples, but may not generalize to the full column. To address this shortcoming, we first predict the expected output of the rule on the remainder of the unformatted cells. We exploit positional information to constrain this problem: we assume users annotated their column top-down. Any cells above (and between) formatted examples are intended to have no formatting, while any cells after the last formatted example are true unassigned cells.

We perform iterative clustering over our predicates to produce k clusters, where k is the number of unique formats in the column plus one cluster for unassigned cells. Formatted examples and intentionally unformatted rows are never reassigned. Unassigned cells are reassigned at each iteration based on the sum of minimum and maximum distance to any cluster element, where distance is computed as the symmetric difference between sets of predicates.

2.3 Candidate Rule Enumeration

Cornet supports rules that can be built as a propositional formula in disjunctive normal form over predicates. In other words, every rule is of the form

$$(p_1(c) \wedge p_2(c) \wedge \dots) \vee (p_j(c) \wedge p_{j+1}(c) \wedge \dots) \vee \dots$$

with p_i a generated predicate or its negation. Our goal is to strike a balance between expressiveness and simplicity.

We greedily enumerate promising candidates by iteratively learning decision trees, which use our predicates as features to classify cells into the groupings produced by our semi-supervised clustering. Each decision tree then corresponds to a rule in disjunctive normal form [1]. We identify and address three challenges: variety in rules, simplicity of rules, and coping with noisy labels. To ensure variety, the root feature is removed from the set of candidate splits after each iteration. To ensure simplicity, we only accept decision trees with λ_n or fewer nodes (λ_n=10 as a default). To deal with noisy labels, we only require decision trees to have perfect accuracy on user examples. In our greedy tree learning approach we consider labeled cells to be twice as important as unlabeled ones and we stop learning rules once accuracy falls below λ_t ($\lambda_t = 0.8$ as a default).

2.4 Candidate Rule Ranking

Because the rule learning step by construction produces multiple candidate rules, Cornet reorders candidates and returns the top rule to the user. To perform this ranking, Cornet uses a neural ranker that considers, features of the data, the rule execution on that data [10] and intrinsic properties of the rule [3], such as what predicates and literals are used. Information about the column is captured by turning the column data into a sequence of words and using a pre-trained language model [2] to obtain cell-level embeddings. These embeddings are augmented with information about the execution of the rule through cross-attention [7]. Information about the rule is captured by handpicked features. We then concatenate the vectors of handpicked features and the vector capturing the column data and rule execution. Finally, we learn a linear weight vector to reduce this feature vector to a single ranking score. [3]

3 DEMONSTRATION

The participants can use Cornet on any workbook of their choice. For this demonstration we use the Work Orders workbook from a public internet tutorial [4]. The dataset contains information about work orders for a hypothetical services company. The data has 1000 rows and 22 attributes including order ID, district, number of technicians, service date and service type. We implemented a simple add-in for Microsoft Excel (called *Smart Format*) to demonstrate Cornet. Figures 3 and 4 show a screenshot of this add-in.

We will demonstrate two scenarios: a workflow where the user requests a rule from Cornet after providing a few formatting examples, and a workflow where Cornet, running in the background, can hand-raise and suggest a rule without prompting. The two workflows are designed to address different users' Excel expertise.

[3]We also designed a heuristic based symbolic ranker that performs slightly worse than the neural ranker. More details about the symbolic ranker are included in full paper.

An experienced user performing sophisticated formatting tasks might prefer selecting rules themselves, exerting more influence over Cornet. A novice user, on the other hand, might be unaware of CF and benefit from proactive suggestions for feature discovery.

3.1 User Requested Suggestion

For this demonstration, we will guide the participants through four steps. We use Figure 3 to guide the demonstration.

Step ① (Providing Examples): The user colors a few cells as examples for each column that needs to be formatted. For our scenario, the user wants to format ID, District, Service and Rush column, so they color one cell in each such column. Their intent for each column is as follows:

(1) *Service* – Fill yellow for all services that are delivery related.
(2) *WO* – IDs starting with *"G"* or *"AN"* and not ending with *"T"*.
(3) *District* – Bold and fill green for North and South districts.
(4) *Rush* – Fill magenta for missing value cells denoted by *"_"*.

Step ② (Learning Formatting Rules): To learn and apply rules, the user has 2 options. (1) The user can select which rule to apply by clicking on *"Detect Formatting"* which opens a drop-down list of the top 3 rules that Cornet learned for the column that is currently active. (2) The user can also choose to directly learn and apply the top ranked rule by selecting the desired columns and clicking *"Learn and Apply"*. Option (2) allows for an in-context experience, as it can be triggered via a keyboard hotkey `(Ctrl+Shift+O)`.

Step ④ (Updating Rules): Like with other PBE systems, the user can improve the rules learned by Cornet by providing more examples. For example, the user can give one example and learn the rule and if it is incorrect, the user can uncolor some cells or color new cells and relearn the rule. For this demonstration, lets consider the user wants to also color cells starting with *"AN"* in the *WO* column. The user colors the second cell (*"AN47603-F"*) and then detects formatting rule. The user can now choose `AND(OR(Begins("GW"),Begins("AN")),NOT(EndsWith("T")))`.[4]

3.2 Background Suggestion

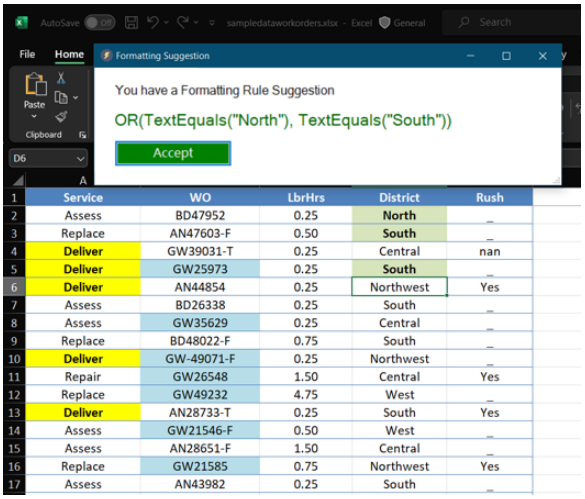
We now describe the hand-raise workflow of Cornet where the user edits the spreadsheet as normal and Cornet proactively suggests the top-ranked formatting rule to the user. This workflow can help novice users who may be unfamiliar with Cornet or conditional formatting in general. For this demonstration, we will guide the participants through three steps that exercise this proactive workflow. We use Figure 4 to guide the demonstration.

Step ① (Opening Reference Data): Similar to the first demonstration, the user opens the desired workbook in Excel. For this demonstration, we use the same Work Orders workbook as before.

Step ② (Normal Editing): The user edits the spreadsheet to format the *District* column based on the intent described earlier. In this scenario, the user wants to format *"North"* and *"South"* districts. The user starts editing the sheet and formats the first 2 cells in the district column that are either *"North"* or *"South"*.

Step ③ (Rule Suggestion): Cornet can learn rules in the background as users are normally editing their spreadsheet. After three examples, Cornet is able to learn a rule in the background and can suggest this to the user, as shown in Figure 4, without any prompting from their end. Similarly to the prior scenario, the user can then decide to either accept the suggested rule, modify parameters, or give more examples to learn a better rule. The user accepts this rule and its application automatically formats the entire column as intended.

[4]The user can also manually update the rule from the Conditional Formatting menu.



	Service	WO	LbrHrs	District	Rush
2	Assess	BD47952	0.25	North	_
3	Replace	AN47603-F	0.50	South	_
4	Deliver	GW39031-T	0.25	Central	nan
5	Deliver	GW25973	0.25	South	_
6	Deliver	AN44854	0.25	Northwest	Yes
7	Assess	BD26338	0.25	South	_
8	Assess	GW35629	0.25	Central	_
9	Replace	BD48022-F	0.75	South	_
10	Deliver	GW-49071-F	0.25	Northwest	_
11	Repair	GW26548	1.50	Central	Yes
12	Replace	GW49232	4.75	West	_
13	Deliver	AN28733-T	0.25	South	Yes
14	Assess	GW21546-F	0.50	West	_
15	Assess	AN28651-F	1.50	Central	_
16	Replace	GW21585	0.75	Northwest	Yes
17	Assess	AN43982	0.25	South	_

Figure 4: Interactive formatting rule suggestion by Cornet. The user edits the spreadsheet as usual and Cornet learns rules in the background, surfacing them after three examples have been provided. Cornet can help users who are unaware of conditional formatting rules in spreadsheet platforms.

REFERENCES

[1] Hendrik Blockeel and Luc De Raedt. 1998. Top-down induction of first-order logical decision trees. *Artificial intelligence* 101, 1-2 (1998), 285–297.
[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423
[3] Kevin Ellis and Sumit Gulwani. 2017. Learning to Learn Programs from Examples: Going Beyond Program Structure. In *IJCAI 2017* (ijcai 2017 ed.). https://www.microsoft.com/en-us/research/publication/learning-learn-programs-examples-going-beyond-program-structure/
[4] Microsoft Excel. 2023. Sample Excel Wrkbooks. https://www.contextures.com/xlsampledata01.html#wos. Last Accessed: 2023-03-24.
[5] Sumit Gulwani. 2011. Automating String Processing in Spreadsheets using Input-Output Examples. In *PoPL'11, January 26-28, 2011, Austin, Texas, USA*. https://www.microsoft.com/en-us/research/publication/automating-string-processing-spreadsheets-using-input-output-examples/
[6] Vu Le and Sumit Gulwani. 2014. FlashExtract: a framework for data extraction by examples. In *Programming Language Design and Implementation*. ACM, 542–553.
[7] Kuang-Huei Lee, Xi Chen, Gang Hua, Houdong Hu, and Xiaodong He. 2018. Stacked Cross Attention for Image-Text Matching. In *Computer Vision – ECCV 2018*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer International Publishing, Cham, 212–228.
[8] Microsoft. 2022. Excel Help Forum. techcommunity.microsoft.com/t5/forums/searchpage/tab/message?q=conditional%20formatting. Accessed: 2022-06-30.
[9] Joseph N. 2022. Number of Google Sheets and Excel Users Worldwide. https://askwonder.com/research/number-google-sheets-users-worldwide-eoskdoxav. Last Accessed: 2022-07-30.
[10] N Natarajan, D Simmons, N Datha, P Jain, and S Gulwani. 2019. Learning Natural Programs from a Few Examples in Real-Time. In *AIStats*. https://www.microsoft.com/en-us/research/uploads/prod/2019/01/AIStats19.pdf