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).

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 numeric literal 5. Because the rule does not format any cell, the user had to manually format the sheet as reflected in this figure. The correct CF rule, which can be learned by CORNET from 2 examples.

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. 3 The correct CF rule, which can be learned by CORNET from 2 examples.
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 drop-down menus, perform 9 clicks, and then write the Excel formula \( \text{AND(LEFT(A2,2)="GW", NOT(OR(RIGHT(A2,2)="-F", RIGHT(A2,2)="-T")))} \). With \text{Cornet}, the user can simply color the fourth cell in Figure 2 (0), and \text{Cornet} learns and applies the correct rule.

In our demonstration, we will show an implementation of a lightweight interface in Microsoft Excel that exercises \text{Cornet}. With \text{Cornet}, we can successfully learn the rule in Example 1.2 and provide an outline of our proposed demonstration.

**2 SOLUTION SKETCH**

A core challenge in \text{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 \text{Cornet} is meant to mitigate this challenge by generating predicates that characterize values in the target formatting column, hypothesizing labels for the entire column by clustering over the generated predicates, synthesizing many rules that satisfy the hypothesized groups, and using a dedicated ranker that further refines the likelihood of generated rules. Figure 2 shows an overview of these steps, which we describe in following sections.

### 2.1 Predicate Generation

\text{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

---

**Table 1: Supported predicates and their arguments for each data type.**

<table>
<thead>
<tr>
<th>Numeric</th>
<th>Datetime</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>greater(c, n)</td>
<td>greater(c, n, d)</td>
<td>equals(c, s)</td>
</tr>
<tr>
<td>greaterEquals(c, n)</td>
<td>greaterEquals(c, n, d)</td>
<td>contains(c, s)</td>
</tr>
<tr>
<td>less(c, n)</td>
<td>less(c, n, d)</td>
<td>startsWith(c, s)</td>
</tr>
<tr>
<td>lessEquals(c, n)</td>
<td>lessEquals(c, n, d)</td>
<td>endsWith(c, s)</td>
</tr>
<tr>
<td>between(c, n_1, n_2)</td>
<td>between(c, n_1, n_2, d)</td>
<td></td>
</tr>
</tbody>
</table>

\(n, d\), and \(s\) are generated from cell values, column statistics (mean, percentiles, min, max) and popular constants (0, 1, 100). These predicates are
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

$$(p_1(c) \land p_2(c) \land \ldots) \lor (p_3(c) \land p_4(c) \land \ldots) \lor \ldots$$

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 $\lambda_n$ or fewer nodes ($\lambda_n = 10$ 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 $\lambda_L$ ($\lambda_L = 0.8$ as a default).

2.4 Candidate Rule Ranking
Because the rule learning step by construction produces multiple candidate rules, CORNET orders 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.

3We 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 3 (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"))).

**Step 4 (Completing Task):** 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 3 (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 2 first cells in the district column that are either "North" or "South".

**Step 1 (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.

### REFERENCES