Understanding and Inferring Units in Spreadsheets*

Jack Williams[1†], Carina Negreanu[1†], Andrew D. Gordon[1,2], Advait Sarkar[1,3]
[1]*Microsoft Research*
[2]*School of Informatics, University of Edinburgh*
[3]*Department of Computer Science and Technology, University of Cambridge*
{t-jowil, t-caneg, adg, advait}@microsoft.com

***Abstract*—Numbers in spreadsheets often have units: metres, grams, dollars, etc. Spreadsheet cells typically cannot carry unit information, and even where they can, users may not be motivated to provide it. However, unit information is extremely valuable: it allows us to detect and prevent an entire class of spreadsheet errors, such as accidentally adding values of different units. What if we could infer the unit of any value in a spreadsheet, with little or no work from the user?**

We present a novel method for predicting units and dimensions in spreadsheets, the first such method that combines logical constraint solving and probabilistic unit labelling. Our approach identifies and formalises the critical cells in spreadsheets that bound the user cost of unit annotation. Separately, we apply machine learning to infer probabilistic unit labels from cell text. To contextualise the accuracy of our system, we discuss the attention investment trade-off for unit inference.

I. Introduction

Spreadsheet authors often use numbers to represent real-world quantities with associated *units*. A unit could be a physical unit like second or gram; or a unit could be a domain-specific unit like dollar or euro. Each unit belongs to a more abstract *dimension*, such as time, mass, or currency.

A numeric calculation must respect the algebra of units [1] and unit checking for spreadsheets is an effective way to find spreadsheet errors [2], [3]. The task of unit error checking for spreadsheets is underpinned by the fundamental task of *unit inference*: given a cell containing a number, determine its unit.

One method for unit inference is to use explicit annotations provided by the user [3]. We formally define and measure the annotation burden this method incurs, and find that it requires significant effort from the user. In contrast, *automatic unit inference* uses an algorithm to synthesise unit annotations from information in the spreadsheet.

In this paper we study unit inference as a machine learning task. We present a new algorithm for automatic unit inference that uses logical constraints and machine learning. It combines the following forms of information found in spreadsheets: **number formats**, such as currency or time, that directly indicate the unit of a cell; **formulas**, such as =A1+A2, that constrains cells A1 and A2 to have the same unit; and **textual labels**, such as "Length (m)" or "Credit Card Charges", that suggest the unit of a nearby cell.

Our unit inference algorithm begins by applying pre-existing techniques. We use formatting, formulas, and automatically detected tables to produce unit constraints between cells [2], [4]–[6]. We cast these unit constraints into a matrix of linear equations and solve them using Gaussian elimination [1], [3], [7]. If there is no unique solution to the matrix we require additional information to infer a unit for every cell. The extent of the missing information amounts to the annotation burden a user would face without unit inference, and can be measured in distinct cell annotations. We then detect unit annotations matching known templates [2]; for example, a header of the form "Apples (kg)" would be detected by our templates for the *kilogram* unit.

These steps of our algorithm correspond approximately to the design of Chambers and Erwig [2], the closest related work, although our system predicts a unit (and hence also a dimension) for each cell, whereas their system only predicts a dimension.[1] Our template and constraint-driven algorithm delivers high precision but low recall.

A core problem is that while many spreadsheets refer to units, they do so indirectly. For example, the text cell "Credit Card Charges" strongly suggests currency but is missed by our template analysis. To address this problem we extend our unit inference algorithm with a novel *dimension inference* phase that uses pre-trained word embeddings [8] to predict a distribution over dimensions. Using our dimension inference phase we increase recall of unit inference by a factor of 30%, with some cost to precision (see Table III). On Chambers and Erwig's goal of inferring a dimension, we improve recall by a factor of 90%, with a 20% loss of precision.

Our central contribution is to frame unit inference as a machine learning task and to develop a theory for understanding the risk of machine learning error in end-user programming. Our key results are both theoretical and quantitative:

- We identify unit inference as belonging to a class of problems that sits at the intersection of the theories of mixed-initiative systems and attention investment. We show how these two theories can be interpreted to yield a guideline for the level of predictive accuracy needed to make the attention-investment trade-off attractive in a mixed-initiative

[1]Our baseline system has a more complete coverage of the Excel formula language than the one implemented by Chambers and Erwig. To detect table headers, we use TableSense [6] rather than their system. TableSense is a state-of-the-art deep learning system, but has not been explicitly compared with Chambers and Erwig's table detector.

*This extended version is the same, apart from the additional appendix, as the paper published in the proceedings of the *2020 IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC 2020)*, with DOI: 10.1109/VL/HCC50065.2020.9127254.

†Denotes equal contribution.

978-1-7281-6901-9/20$31.00 © 2020 IEEE

system (Section IV). To our knowledge, this is the first time the two theories have been directly compared and the analogies between them explicitly stated.
- We formally define the notion of *critical cells* which correspond to the unit annotation burden in spreadsheets. We measure this burden across a corpus of spreadsheets.
- We identify the subproblem of *indirect annotation inference* and present two new algorithms based on word embeddings, where the second algorithm is a refinement of the first (Section V). We show comparable results to a human baseline on the task of textual dimension inference (Section VI-C) on a hand-labelled dataset for the subproblem. We consistently reach over 60% performance on standard quantitative multi-class metrics.
- We present the first method of unit inference in spreadsheets that combines logical constraint solving and machine learning. We obtain a human-labelled dataset of 331 workbooks and measure performance using standard metrics for multi-class classification. Hence, we give the first empirical evaluation of unit inference for spreadsheets, and demonstrate that indirect annotation inference delivers better recall than heuristics for finding direct annotations.

II. Background

A. Inference: an end-user tradeoff

The opportunity and challenge of unit inference in spreadsheets belongs to a class of programming language enhancements with the following property: it asks the user to invest some attention upfront, in exchange for the possibility of a payoff in reduced attention requirements later on. A classic example of a programming language feature that exhibits this tradeoff is static typing. The upfront cost of providing type annotations might create barriers to adoption, but might also pay off in greatly reduced attention expenditure when reasoning about type errors [9].

Another example, in an end-user programming context, is Wilson et al's approach [10] to creating tests in Forms/3 [11]. Tests in Forms/3 take the form of assertions about cell values. For example, a cell that is meant to contain a positive fraction should have a value between 0 and 1. The initial investment in creating tests should pay off in the long term by preventing costly errors. However, these upfront costs are a barrier to user uptake. Wilson et al.'s solution relied on an observation regarding Blackwell's attention investment model [12]. Under this model, the user is motivated to invest attention if they believe that the (expected) cost of not investing the attention is greater than investing it now. In order to perform this calculus, the user creates a mental model of the costs of investment and non-investment. Wilson et al.'s insight was that the user can be motivated to invest attention if we introduce design features that alter this mental model – it is not actually necessary to reduce the true costs, or increase the true payoff, just to reduce the mentally modelled cost, and increase the mentally modelled payoff. They called their concrete implementation of this strategy 'surprise, explain, reward': surprise the user by showing them an unexpected consequence of their spreadsheet, explain the consequence in a way that helps them author an appropriate test, and reward them by giving them a more correct spreadsheet (reinforced through visual cues).

In our case, the upfront investment is to provide unit annotations, and we might say that the payoff is a reduced requirement for attention spent to resolve unit errors. This makes a couple of simplifications: 1) In practice, unit inference should enable a wide range of beneficial user experiences beyond merely preventing unit errors, which for simplicity we consider out of scope. 2) There might be other external costs beyond attention investment, e.g., when an error results in an incorrect decision being made in the real world (i.e., a "problem" [13]) that incurs negative implications with their own costs, which we ignore.

B. Approaches to unit checking and inference

Unit and dimension inference for programming languages has an established history [1], [14], [15]. The core idea is that program identifiers are assigned unit variables and the program structure induces constraints on these variables. Unification is used to solve the unit constraints and provide assignments for the unit variables in the program. Unit checking is implemented in a similar manner. A program is unit-safe if the induced unit constraints are consistent.

Spreadsheets have been extended to support variants of unit checking and inference. Erwig and Burnett [5] develop a unit-system for spreadsheets that uses header information and formulas to derive units for cells. Their definition of unit is based on header categories and set-like operators, rather than physical units. Abraham and Erwig [4] extend this system and develop *UCheck*, which includes automatic table detection. The XeLda [3] system provides unit checking for spreadsheets in the style of Kennedy [1]. Formulas in the spreadsheet generate a set of constraints that are interpreted as a series of linear equations. XeLda does not support unit inference; a user must annotate every numeric cell in the spreadsheet with its concrete unit. Chambers and Erwig [2] develop a system for dimension inference, in response to the challenge of the user annotation burden. Building on existing techniques [4], [5], they automatically detect headers for cells and derive dimension labels from these headers. Chambers and Erwig apply their system to 40 spreadsheets and demonstrate that the approach can be used to automatically find dimension errors.

There is recent interest in applying machine learning to infer types. For instance, DeepTyper [16] is trained on a large corpus of TypeScript code, so as to predict type annotations from textual context, without logical analysis. Still, ours is the first work on machine learning to infer units, either in spreadsheets or in any programming language.

III. Understanding the Annotation Burden

Drawing a meaningful comparison between unit checking via constraint propagation (e.g., XeLda) and unit inference (e.g., Chambers and Erwig) requires an understanding of the benefits of inference. There are two factors to consider. The first factor is the effort required from the user to fully

annotate a workbook—we call this the *annotation burden*. If the annotation burden is small then deploying a non-inferential system could be justified. The second factor is the statistical performance (e.g., precision and recall) of the inference system. If the system provides incorrect unit annotations, or fails to find many unit annotations, then the merits of using the system are diminished. In this paper we give the first empirical measurements for both factors.

In the related domain of scientific computing, Orchard et al. [7] implement a system of unit inference in Fortran. They observe that given a program with unit variables, only a subset of those variables must be annotated to determine all unit variables in the program. Furthermore, they give an algorithm that finds these variables, called *critical variables*, using a series of unit constraints.

We adapt the concept of critical variables to the domain of spreadsheets. For example, given the spreadsheet:

A1=10, A2=20, A3=30, A4=(A1+A2)∗A3

only two cells must be annotated to determine the units for the complete program—these are the *critical cells*. The addition operator constrains A1 and A2 to have the same unit, therefore one annotation determines both. The multiplication operator transforms units but does not constrain them, therefore a unit must be provided for each operand. The set of critical cells (known as the *critical set*) may not be unique; in our example, both {A1,A3} and {A2,A3} are critical sets. The size of the critical set corresponds to the annotation burden a user faces if they want to unit-check a spreadsheet without the aid of unit inference.

We analyse two spreadsheet corpora to estimate the annotation burden using the frequency of critical cells. Our analysis is implemented using a subset of our unit inference algorithm (Section V). Informally, the analysis generates unit constraints from formulas and solves those constraints by casting each as a linear equation, and solving for the set of linear equations. Our implementation is standard; we defer to the work of Kennedy [1] and Orchard et al. [7] for a complete definition. Some constraints may remain unsolved, and from these we extract critical cells as described by Orchard et al. [7].

The first corpus is obtained from ENRON and the second corpus is obtained from the subset of EUSES we hand-labelled with unit information. We only analyse a subset of workbooks from their respective corpus, selected using the following criteria: each workbook must contain a formula, and each workbook must be analysed in full. We cannot analyse some spreadsheet features (discussed in Sections VI-A); workbooks exhibiting those were excluded.

In total, we analysed 1936 workbooks from ENRON, and 667 workbooks from EUSES. Table I presents the result of our analysis. For each dataset we give four statistics presented as mean and median per workbook. A numeric cell contains a number literal, or a formula that evaluates to a number, and can therefore be assigned a unit. An active cell contains a formula, or is referenced by a formula. Our analysis only exploits constraints induced by formulas. Therefore, a cell that is never referenced is vacuously critical. We only report the occurrence of critical cells from the set of active cells. Number formatting is prevalent in spreadsheets and permits a user to format a number as a currency, date, time, or percentage. A number format may provide a strong indication of the unit for a cell, but despite this, no existing work has studied the effect of formatting in unit checking or inference. We consider number formatting to be a unit annotation and report the reduction in critical cells in Table I.

(Mean/Median) across Workbooks		
	ENRON	EUSES
Numeric Cells	5333 / 1288	423 / 165
Active Numeric Cells	2781 / 644	266 / 81
Critical Cells (C.C.)	102 / 16	25 / 3
C.C. with Number Format	87 / 13	19 / 1

TABLE I: Critical Cell Analysis

From Table I it is clear that despite improvements gained from our algorithm, the annotation burden is still too high. *Any* upfront attention requirements can lead to reduced feature usage. Assuming a streamlined unit annotation interface that takes a few seconds per annotation, a mean of 19-87 critical cells per workbook still asks the user to invest several minutes of dedicated effort to unit annotation. Testing in Forms/3 had precisely the same issue; some constraints could be inferred from formulas, but the number of unconstrained cells still posed a high attention investment requirement.

IV. THE ATTENTION INVESTMENT TRADE-OFF FOR UNIT INFERENCE

We take the inference approach of Chambers and Erwig, although we aim to infer concrete physical units (instead of dimensions). Through a fully-automated process based on formulas, formatting and nearby textual labels (described in Section V), we infer the units of each critical variable without any upfront user attention requirements. By reducing the (apparent) cost to the user to zero, we can greatly reduce the barrier to adoption.

Of course, there is no free lunch. The catch is that inference is not perfect, and when inferred units are incorrect, the user will need to invest attention to rectify the inference (a tradeoff that has not been previously acknowledged in such work). The question is under what circumstances does this result in a situation beneficial to the user, i.e., under what conditions does the unit inference system result in a lower overall attention investment cost?

This question is precisely the one answered by the decision calculus of Horvitz's principles for mixed-initiative systems [17], but applied to the user's attention. Our key observation, which allows us to combine the theories of attention investment and mixed-initiative systems, is that the utility functions in Horvitz's calculus can be expressed in terms of Blackwell's attention units.[2]

Consider a simple model of unit errors, inference errors, and their associated attentional costs in a spreadsheet, as

[2]Although Horvitz's utility functions are reals in the interval $[0, 1]$, we do not place the same constraints on attention costs.

follows: *Over the course of interacting with a spreadsheet (authoring, editing, reading, etc.), a unit error occurs with some probability. If a unit error occurs, the user incurs an attentional cost of recovering from the unit error. However, if we have a working inference system, the cost of recovering from a unit error is zero. If there is an inference error, the user must recover from it.*

We formalise the sketch above, defining the following quantities: P_u, the probability of a unit error; P_i, the probability of an inference error; R_u, the cost of recovering from a unit error; and R_i, the cost of recovering from an inference error.

We derive the expected attentional cost to the user without and with the inference system. Without inference, the expected cost is $P_u R_u + (1 - P_u) \cdot 0 = P_u R_u$. The term $P_u R_u$ corresponds to the event where a unit error occurs, and the term $(1 - P_u) \cdot 0$ corresponds to the event where a unit error does not occur.

Similarly, we derive an expression for the expected cost *with* inference, with terms corresponding to the four cases where unit errors do and do not occur, and inference errors do and do not occur. Recall our assumption that when inference works, the cost of fixing a unit error is zero. Therefore, in the case where there is both a unit error and an inference error, we assume that resolving a unit inference error must also resolve any unit errors and therefore costs at most R_i, not $R_i + R_u$. The cost with inference is:

$$P_u(P_i R_i + (1 - P_i) \cdot 0) + (1 - P_u)(P_i R_i + (1 - P_i) \cdot 0)$$
$$= P_i R_i$$

Thus, the inference system lowers the overall attentional costs of using spreadsheets if $P_i R_i < P_u R_u$. If we now further assume our system is designed such that $R_i \leqslant R_u$, that is, the cost of recovering from a unit inference error is not higher than the cost of recovering from a unit error (a reasonable design objective), we obtain the bound $P_i < P_u$.

Thus, we arrive at a simple, calculable criterion by which we can contextualise the performance of an imperfect error-prevention system: in order for an inference system to lower the expected attentional cost to the user, the rate of inference error must be less than the natural rate of the error that the system is designed to prevent. Previous work estimates that dimension errors occur in 42.5% of spreadsheets [2], thus the error rate of our system must also not exceed 42.5%.

To arrive at our criterion, we make a number of simplifying assumptions besides those already stated, as follows:

Risk-neutrality: we assume the user is risk-neutral; that is, it is sufficient for the expected attentional cost of a system with inference to be merely lower than the expected attentional cost without inference. However, behavioural economics shows that people can be risk-averse or risk-loving, with most people being slightly risk-averse [18]. For example: given the choice of a 50% chance of winning $100, or a guaranteed win of $50, which would you choose? A risk-neutral person views both options as equivalent due to their equal expected payoff. A risk-averse person prefers the uncertain win only if the expected payoff is higher than that of the certain win; the difference between those two quantities is known as the person's *risk premium*. It is almost certainly the case that users of inference systems are slightly risk-averse, and therefore our inference system must not merely match the attention requirements of the status quo, but improve upon it by a risk premium (that might be possible to empirically determine, but has not yet been done).

No external costs: we only model *attentional* costs and utility. The full cost of an error in a spreadsheet varies according to its context; a unit error might result in incorrect real-world decisions, financial and reputational loss, and many other negative externalities. It is unclear how to model or account for these in a principled way.

Single error: we do not model multiple errors and episodes of error recovery.

Guaranteed error discovery and recovery: we do not model the likelihood of the user *not* detecting unit and inference errors, and of *not* fixing them. We assume that if a unit or inference error exists, the user always discovers it, chooses to fix it, and does so successfully. In the case where both a unit and an inference error occurs, the user discovers and fixes the inference error (which automatically fixes the unit error, see next point).

Zero-sum inference: we assume that if unit inference works, then the cost of recovering from a unit error is zero. This would be trivially the case if unit inference prevented unit errors from occurring in the first place. In this case P_u can be interpreted as the probability that a unit error *would have* occurred without the interface. This assumption and the previous one subsume another assumption we make (which Horvitz's model is particularly concerned about), namely *perfect inference of user goals*. That is, we assume that the way in which our inference system ultimately fixes or prevents unit errors is always perfectly aligned with the user's goals.

Inference has cheaper recovery: the cost of recovering from a unit inference error is less than or equal to the cost of recovering from a unit error (note a corollary design principle: incorrect inference should not be error-genic; if the inference system introduces the very error it is designed to prevent, the cost of recovering from an inference error cannot be less than the cost of recovering from a unit error).

Fixed error probabilities and costs: we model the probability of unit and inference errors to be fixed for all users and spreadsheets (e.g., interpreted as an empirical probability).

Short-term/long-term conflation: we do not distinguish between Blackwell's long-term focus (on the inference system as a whole) and Horvitz's short-term focus (on each individual opportunity for inference and user interruption). In the future we might treat these differently, using long-term empirical probabilities for the former analysis, and sheet-specific probabilities generated by our inference model for the latter.

A. Attention investment and mixed-initiative systems: two sides of the same coin?

Since our system sits at the intersection of concerns treated by both Blackwell's account of attention investment and

Aspect	Attention investment	Mixed-initiative systems
Purpose of model	To explain user behaviour	To determine system behaviour
Decision problem	Is the expected payoff of automation greater than that of non-automation? If so, the user takes action.	Is the expected utility of the (automated) action greater than that of inaction? If so, the system takes action.
Instance of concern	This model applies at each investment opportunity, that is, each time the user has an opportunity to automate something.	This model applies at each inference/automation/interruption opportunity, that is, each time the system can take an individual action.
Implementation of model	This is a long-term calculus in the user's mind. In our context, we assume a rational, learning user, who will eventually approximate P_u to be the long term rate of unit error, P_i to be the overall inference error rate.	This is a short-term calculus which the system can calculate for any given prediction. In our context, P_u would be interpreted as the sheet or cell error likelihood, and P_i would be the inference confidence in a specific prediction.

TABLE II: A comparison of attention investment and mixed-initiative systems.

Horvitz's account of mixed-initiative systems, we have conducted an analysis that draws on concepts from both. In doing so, we have been able to identify a number of similarities and differences between them. In Table II, we present our comparison of the two theories.

These theories approach two different problems from two very different perspectives, but ultimately produce a mathematically identical solution (namely, to compute the expected payoff to the user of implementing a technical intervention, versus not implementing it). Therefore, when applying these theories in new contexts, it is important to consider their difference in perspective, because though the equations are the same, our interpretation of the quantities encoded varies. To our knowledge, this is the first time the mathematical equivalence of these theories has been explicitly stated, and their differences explicitly compared.

B. From unit inference to error detection

In the following sections, we describe a pipeline for inferring the unit of any numeric cell. However, merely detecting the units of cells does not by itself solve the problem of detecting unit-related errors. An additional step is needed (which may itself have a non-zero chance of error). We have not implemented any such error-detection step, but a small example of a potential application follows.

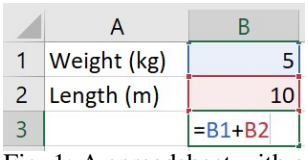

Fig. 1: A spreadsheet with a potential unit error

Consider Figure 1, which shows a spreadsheet with a potential unit error. The formula in cell B3 adds two quantities that appear to have different units, according to the user labels in cells A1 and A2. How to best assist the user depends on our belief in the relative correctness of different parts of the spreadsheet: in particular, the relative correctness of the formula and the labels. If we believe the formula is more likely to be correct than the labels, we could highlight the labels in A1 and A2 to the user as being potentially misleading. If we believe the labels are more likely to be correct, we could highlight the formula as being potentially erroneous.

How might we arrive at such a belief? One heuristic might be that the user's most recent action is more likely to be the source of an error than the state of the workbook prior to the action (because we assume the user is likely to discover and fix errors as they go along). Another might be that users are in general more likely to make errors in formulas than they are in textual labels. These options need to be tested in practice; this is a difficult problem that we leave for future work.

V. Our Algorithm for Unit Inference

Our unit inference algorithm has two steps:

1) We generate constraints over cells using three sources: formulas, inferred tables, and number formats. We then solve these constraints.
2) For the remaining set of unconstrained cells, which we call *critical cells*, we synthesise unit annotations for these cells from text in the spreadsheet.

As input to our algorithm we assign a unit variable to every numeric cell. The output of our algorithm is an assignment (or substitution) for every unit variable. A variable can be assigned a unit that is: *concrete*, denoting a known unit; *critical*, denoting an identity assignment; or *determined*, denoting a unit that contains variables from other cells. Consider the assignment $\{\mathsf{A1} \mapsto \$, \mathsf{A2} \mapsto \mathsf{A2}, \mathsf{A3} \mapsto \$/\mathsf{A2}\}$. We use unit variable names that are in one-to-one correspondence with cell addresses. The assignment for A1 is concrete; the assignment for A2 is critical, indicating that A2 is a *critical cell*; and the assignment for A3 is determined by A2. Our metrics of success are based on finding correct concrete assignments, corresponding to *precision*, and reducing the number of critical assignments, corresponding to *recall*.

Step 1 and Step 2 produce a unit assignment as output, where the purpose of Step 2 is to convert critical assignments to concrete assignments using text in the spreadsheet. We now describe each step in detail.

A. Constraint Generation (Step 1)

Constraints are generated from three sources: formulas, inferred tables, and number formats. We use an instrumented spreadsheet runtime to evaluate formulas and dynamically generate constraints when arithmetic operations are applied to numbers. Our runtime handles a large range of spreadsheet features including array formulas, implicit intersection, and worksheet functions such as VLOOKUP and SUMIF.

We detect tables in workbooks using the TableSense model [19]. Tables are used in two ways. First, a table constrains values in a row or column (depending on orientation) to have the same unit. Second, a table provides headers, which label cells—we use headers in Step 2. Our use of tables is derived from existing work [2], [4], [5], [20], although we make a strong distinction between using table headers and constraining values in a row or column. This distinction means that we benefit from the detection of a table, even if we fail to

	A	B	C	D	E
1	\$1000			2000	s_2
2	20	s_1		1500	
3	=A1/A2			=D1 − D2	

Fig. 2: Example sheet where s_i represents some string.

detect or extract annotations from headers. Cunha et al. [21] build a model capable of extracting richer constraints from a table, including functional dependencies. Combining this with table detection to improve unit inference is future work.

Figure 2 presents an example spreadsheet from which our runtime will generate the constraint set $\{\mathsf{A1} \sim \$, \mathsf{A3} \sim \mathsf{A1}/\mathsf{A2}, \mathsf{D1} \sim \mathsf{D2}, \mathsf{D1} \sim \mathsf{D3}\}$, where we write $(\sim)$ to indicate an equality constraint. The first constraint is generated by the number format for A1, and the remaining constraints are generated by formulas.

The final part of Step 1 is to solve the set of constraints and produce an assignment. We transform unit constraints to linear equations and use matrix reduction to solve them, as described by Orchard et al. [7]. The resulting unit assignment is: $\{\mathsf{A1} \mapsto \$, \mathsf{A2} \mapsto \mathsf{A2}, \mathsf{A3} \mapsto \$/\mathsf{A2}, \mathsf{D1} \mapsto \mathsf{D1}, \mathsf{D2} \mapsto \mathsf{D1}, \mathsf{D3} \mapsto \mathsf{D1}\}$.

Logical methods are precise, but fail to incorporate contextual information from nearby text. In our example, we fail to infer a concrete unit for all but A1, with A2 and D1 remaining as *critical cells*. We address the limitations of logical inference with Step 2, where unit annotations are synthesised from text.

B. Annotation Synthesis (Step 2)

We attempt to synthesise *direct* or *indirect* unit references from text cells. If a unit is found we annotate any cells within a relevant distance that share a unit with a critical cell. Relevance is determined by the vertical or horizontal distance between the cells, subject to a configurable distance, or when the text cell is a table header for the numeric cell. In our experiments we use a distance of two, and if multiple annotations are found we select the first annotation starting vertically and moving clockwise (assuming a left-to-right writing system).

A direct unit reference is found using a template-driven approach that examines the text for exact occurrences of units in a set of templates. An indirect unit reference is found using the dimension inference algorithm we describe in Section V-C.

Revisiting the example in Figure 2, suppose that s_1 = *Area (acres)* and s_2 = *Credit Card Charges*. The text s_1 is a direct reference as it matches the template where a unit occurs in parentheses at the end of a string. The text s_2 is an indirect reference to dimension *currency*, from which we synthesise the most frequent unit \$. The annotations produce an assignment $\{\mathsf{A2} \mapsto \text{acre}, \mathsf{D1} \mapsto \$\}$ which is applied to the assignment generated in Step 2, yielding assignment $\{\mathsf{A1} \mapsto \$, \mathsf{A2} \mapsto \text{acre}, \mathsf{A3} \mapsto \$/\text{acre}, \mathsf{D1} \mapsto \$, \mathsf{D2} \mapsto \$, \mathsf{D3} \mapsto \$\}$.

C. Dimension Inference Algorithm

The model can be described as a two-step process: prediction (*Predictor 1*) followed by validation (*Predictor 2*).

For prediction, we created a dictionary of dimensions and their representative units and synonyms (e.g., for the currency dimension, units include dollar, euro, money, cash, etc.), by mining WikiData [22]. We consider 14 dimensions: $\%$, *dimensionless, length, mass, angle, power, energy, speed, temperature, time, volume, force, currency, area*. We ignore dimensions encountered seldom or never (such as *luminosity*) to reduce computational cost.

We compute similarity scores between word embeddings of each unit and the header. We could either use pre-trained word embeddings (such as Glove [23], Word2Vec [8] or FastText [24]), or we can train word embeddings on spreadsheets. We have chosen to create custom FastText embeddings on a dataset generated by collecting every text snippet present in our corpora (as described in Section VI). We can motivate the choice of using static word embeddings over dynamic ones by exploring the vocabulary that is characteristic for spreadsheets. In general, dynamic word embeddings are preferred as words that have multiple meanings can retain their versatility. Fortunately, we found that 88% of the words in our corpora are mono-semantic (i.e., have a single meaning for part of speech) compared to 42% in Wiki English by using Wordnik [25].

We use cosine similarity [26] as implemented in `gensim` [27]. We create a header embedding by averaging the embedding of its words [8]. For each dimension d, we obtain a score $\mathsf{s}(d)$, the maximum of the set: $\{\, \mathsf{s}(u) \mid u \in U(d) \wedge \forall u' \in U(d).\, \phi(u, u') \,\}$. There is a score $\mathsf{s}(u)$ in this set for each unit u such that for all units u' from the same dimension, if $\cos(e(u), e(u')) \approx 1$ (meaning that u and u' are textual synonyms) and $s(u) \approx 1$ (meaning that u scores highly on this header) then $s(u') \approx 1$ (meaning that u' also scores highly). The check prevents a high artefactual dimension score produced by a single high scoring unit that is unrepresentative of similar units.

The dimension scores are transformed into a distribution by using softmax. We find that the distribution our predictor returns is variously either flat (i.e., there is no dimension or we cannot predict), or unimodal (strong signal from one dimension) or multimodal (a few possible dimensions).

For *Predictor 1* we make two simplifications. First, we only predict $\%$ when we see certain delimiters in the header (e.g., *per*). We predict for the text on each side of the delimiter and combine the predictions. Second, we predict *dimensionless* if the distribution over the other dimensions is uniform. The disadvantage of this simplification is that we cannot be certain if a header is truly dimensionless or we could not predict.

In *Predictor 2* we use constraints derived from formulas to construct equivalence classes of headers that have the same unit. We are interested in sets with more than one element as we want to validate the results of *Predictor 1* in an unsupervised manner. Equivalence for headers (denoted $\equiv$) is defined in terms of their distribution over dimensions: two headers are equivalent if their distributions share at least one dominant dimension (the intersection of their mode sets is non-empty). We ignore headers with flat distributions.

Predictor 2 has two phases, an offline training phase and an

Predictor 1 Prediction

$h \in (\textsc{Hdr} = \mathcal{P}(\textsc{Word}))$, a header, or sequence of words.
$D \in \mathcal{P}(\textsc{Dimen})$, a set of dimensions.
$U \in \textsc{Dimen} \to \mathcal{P}(\textsc{Unit})$, dimension to units map.
$e \in \textsc{Word} \to \mathbb{R}^n$, an n-dimensional word embedding.
$x \approx y$ iff $|x - y| < \epsilon$ where $\epsilon > 0$ is a given bound.

$\mathsf{predict}(h) = \mathsf{softmax}(\{\, s(d) \mid d \in D \,\})$ where
$\mathsf{s}(u) = \cos(\mathsf{mean}(\{\, e(w) \mid w \in h \,\}), e(u))$
$\phi(u, u')$ iff $\cos(e(u), e(u')) \approx 1 \wedge s(u) \approx 1 \Rightarrow \mathsf{s}(u') \approx 1$
$\mathsf{s}(d) = \mathsf{max}(\{\, \mathsf{s}(u) \mid u \in U(d) \wedge \forall u' \in U(d).\, \phi(u, u') \,\})$

Predictor 2 Validation (extends Predictor 1)

$\textsc{Hdrs} = \mathcal{P}(\textsc{Hdr} \times \textsc{Dist})$
$\textsc{Heq} = \mathcal{P}(\textsc{Hdrs})$
$H \in \textsc{Hdrs}$, header and pre-computed distribution pairs.
$\mathcal{H} \in \textsc{Heq}$, header equivalence classes.

$\mathsf{offline}(\mathcal{H}) = (\bigcup \mathcal{C}, \bigcup \mathcal{E}, \bigcup \mathcal{I}, \bigcup \mathcal{U})$ where
$\mathsf{headers}(H) = \{h \mid (h, \mathcal{D}) \in H\}$
$\mathcal{C} = \{H \in \mathcal{H} \mid \forall (h, \mathcal{D}), (h', \mathcal{D}') \in H.\ \mathcal{D} \equiv \mathcal{D}'\}$
$\mathcal{E} = \{H \in \mathcal{H} \backslash \mathcal{C} \mid \mathsf{headers}(H) \subseteq \mathsf{headers}(\bigcup \mathcal{C})\}$
$\mathcal{U} = \{H \in \mathcal{H} \backslash \mathcal{C} \mid \mathsf{headers}(H) \cap \mathsf{headers}(\bigcup \mathcal{C}) = \varnothing\}$
$\mathcal{I} = \mathcal{H} \backslash (\mathcal{C} \cup \mathcal{I} \cup \mathcal{E})$

$$\mathsf{online}(h, \mathcal{C}, \mathcal{I}, \mathcal{U}) \quad = \quad \begin{cases} \mathcal{D} & \text{if } (h, \mathcal{D}) \in \mathcal{C} \\ \bot & \text{if } h \in \mathsf{headers}(\mathcal{I} \cup \mathcal{U}) \\ \mathsf{predict}(h) & \text{otherwise} \end{cases}$$

online prediction phase. In the offline training phase, *Predictor 2* uses the context from the header's equivalence class and our definition of equivalence to partition headers from the training set into three sets: Correct ($\mathcal{C}$), Unresolved ($\mathcal{U}$), or Incorrect ($\mathcal{I}$). The error set ($\mathcal{E}$) is not used for prediction, but can be used to report errors. Prediction function online either returns a dimension distribution for the header, or fails. If the header is found in the Correct set, it returns the cached dimension distribution. If the header is found in the Unresolved or Incorrect set, it fails. Otherwise, it calls *Predictor 1*.

We have 458 headers in the *Correct Set*, 132 in the *Unresolved Set*, and 56 in the *Incorrect Set*. The strengths of this method are that it is extensible (as we acquire more data we can extend the sets and improve coverage), and economical (it can significantly reduce the running cost of the algorithm as we can bypass *Predictor 1*).

VI. Evaluation

A. Dataset description

We create a corpus by selecting spreadsheets from Enron [28], Fuse [29] and Euses [30]. After data cleaning and table detection we retain 14,281 unique workbooks. We use 3,610 workbooks to produce two test datasets, split between *Annotated* and *Text(unit)*. Remaining workbooks are used to create the dataset for the offline phase of *Predictor 2*, and for training custom word embeddings. In Enron some individuals authored several spreadsheets; to avoid overfitting, the same individual's spreadsheets are not used in both test and training.

Annotated is a hand-labelled subset of Euses. The annotated set contains 867 workbooks of which 823 had formulas, and 456 (of 823) contained a unit tag. Mainstream spreadsheet implementations have an extensive set of features and our modified runtime only implements a subset. Features we do not handle include structured table references, as well as implementations of some functions. Our coverage per formula is high, over 90%, but we choose not to selectively evaluate formulas from a workbook. Instead, we only consider workbooks that our runtime can evaluate completely (331 of 456 workbooks). Within these there were 1130 critical cells. These cells constitute our unit inference test set. The labelled annotations for these cells span 8 dimensions and 35 units.

Text(unit) is a dataset consisting of 760 table headers, originally matching the template *Text(unit)*. We strip off the *unit* in brackets and transform the unit to its dimension; the final dataset contains 760 items of the form *Text, dimension*. The headers were gathered from Fuse (65%), Enron (25%) and Euses (10%). For headers that occurred multiple times with different dimensions (e.g., *Capacity* had both volume and power units attached), we randomly selected one to resolve ambiguity. There are 18 recorded dimensions with *time, length, mass* the most common.

To establish a baseline for human performance on this task, we asked three English-speaking researchers who were unfamiliar with the dataset to independently label each header with their best guess for its dimension, selecting from one of the 18 dimensions. We observed high inter-rater agreement (Fleiss' $\kappa = 0.73$ [31]). We did not observe fatigue or learning effects; label correctness did not significantly change over the course of the labelling exercise for any participant.

We prepare two sets of aggregated labels to serve as benchmarks: the first aggregates by majority vote (for ties, we used the label from $P3$ who had the highest correctness overall); the second chooses the correct label if any of the 3 raters had chosen it. (If none chose correctly, the majority is taken. If there is no majority, we default to $P3$.)

B. Unit inference evaluation

We evaluate the effect of direct and indirect annotation synthesis using an ablation study. The preparatory phase of our analysis runs logical unit inference, as described in Section V-A, on the 331 workbooks from the *Annotated* set. After logical inference 1130 critical cells remain. We implement three algorithms that attempt to synthesise a unit annotation for each critical cell, and we compare the synthesised annotation with the labelled annotation. Our algorithms are: a baseline that always synthesises USD, an algorithm that only synthesises *direct* annotations, and an algorithm that synthesises *direct* and *indirect* annotations using our dimension inference algorithm. We selected the baseline as our dataset is skewed towards currencies: 280/1130 of all unit tags

		Unit Prediction			Dimension Prediction		
Metric	Type	Always USD	Direct	Direct + Indirect	Always Currency	Direct	Direct + Indirect
Precision	Micro	24.8% (280/1130)	91.1% (112/123)	49.7% (164/330)	42.5% (480/1130)	93.5% (115/123)	65.2% (215/330)
	Macro	0.8%	58.0%	44.8%	5.3%	78.8%	39.4%
	Weighted	6.1%	82.6%	67.4%	18.0%	97.7%	79.8%
Recall	Micro	24.8% (280/1130)	9.9% (112/1130)	14.5% (164/1130)	42.5% (480/1130)	10.2% (115/1130)	19.0% (215/1130)
	Macro	3.3%	25.3%	24.3%	12.5%	34.8%	31.8%

TABLE III: *Performance for annotation synthesis*. For micro precision and recall we report absolute numbers.

Metric	Type	Human baseline		Baseline	Our model
		Majority	Optimal		
Precision	Macro	62.5%	69.3%	1.7%	62.6/63.1%
	Weighted %	83.8%	87.8%	4.89%	68.7/69%
Recall	Macro	57.3%	61.2%	7.69%	60.9/61.1%
Accuracy	Top 1	80.7%	85.3%	22.1%	66.8/67.4%
	Top 3			54.9%	79.9/80.4%
	Top 5			69.8%	90.7/91%

TABLE IV: *Dimension inference performance*. Human baseline, naïve baseline and our model (*Predictor 1 / Predictor 2*). Note: Top 1 accuracy = micro precision = micro recall [34].

are USD. The inclusion of the baseline allows us to highlight this disparity. We report the performance of unit and dimension prediction in Table III using a selection of standard metrics including multi-class precision and recall [32]. In line with the `scikit-learn` package [33], when precision and recall are undefined we assign 0. We use micro, macro, and weighted metrics as our test datasets are unbalanced. Micro-precision is calculated globally by counting the total true positives, false negatives and false positives. Macro-precision calculates precision for each class, and finds their unweighted mean, ignoring label imbalance. Weighted precision calculates precision metrics for each class, and finds their average weighted by support (the number of true instances for each label). We include Top-k accuracy for dimension inference, in line with previous work [16]. If we fail to infer an annotation we predict *unknown*, which we treat as a false negative; consequently, micro precision and recall are different.

The inclusion of *indirect* annotations significantly improves recall, at the cost of precision. Micro recall in dimension prediction improves by a factor of 90%, from 115/1130 to 215/1130, with a decrease in precision by a factor of 20%. Micro recall in unit prediction improves by a factor of 30%, from 112/1130 to 164/1130, while precision drops below 50%. We focus on the *micro* and *weighted* metrics as the dataset has high class imbalance and we prioritise correct results for common units and dimensions.

C. Dimension inference algorithm

We evaluate dimension inference on the dataset *Text(unit)* (only those dimensions we consider as per Section V-C; the final size of the dataset is 737). The task is to predict the dimension of *unit* using only *Text*. For the heuristic baseline we predict the most common dimension(s). Table IV shows that *Predictor 1* and *Predictor 2* significantly outperform the heuristic baseline and are comparable to human performance in macro precision and recall. In our model we do not see a significant difference between macro and Top 1 (micro) performance. Since the dataset exhibits class imbalance, this suggests that the model performs similarly for frequent and rare dimensions, validating our unit dictionary. In contrast, the human baselines are good (70-80% correctness) at detecting mass, length, speed, and area, but significantly worse (less than 50% correctness) at detecting volume, power, %, and energy. Although our sets for *Predictor 2* are small, they improve performance modestly, likely because ENRON and FUSE have many similar or identical headers. The main improvement is the reduction in false negatives for *length*.

We inspected 140 headers that our optimal human baseline correctly labelled, but the model did not. We hoped to identify systematic deficiencies in the model and opportunities to improve it. Informally, we found that these mis-predicted headers formed three categories: 1) headers containing *distractor words*, which mislead the model (e.g., "Average Life of Loan" is misclassified as a currency, due to the influence of the word "Loan".); 2) headers requiring *domain knowledge* (e.g., "National Gross Domestic Product" is misclassifed as mass instead of currency); and 3) headers with *language understanding* issues, suggesting deficiencies in the word embeddings (e.g., "Median age" is misclassified as length).

D. Does this satisfy our criterion?

Previous work estimates dimension error rate in spreadsheets as 42.5% [2] . The criterion derived in Section IV suggests that an inference precision of $1-42.5\%$ $(= 57.5\%)$ is the minimum sufficient to justify the attention investment trade-off. Our precision for direct and indirect inference exceeds this threshold, but there is clear opportunity for improvement.

VII. CONCLUSION

We have framed unit inference in spreadsheets as a machine learning problem. We present a novel analysis of the trade-off between the attention costs of unit errors and unit inference errors. We present a new algorithm for inferring the units of numeric cells in spreadsheets, using constraints derived from formulas, tables, formatting, and textual labels. Our work shows that textual dimension inference improves recall while maintaining reasonable precision, when compared to the estimated rate of dimension error in spreadsheets. In future, we aim to improve our detection of textual annotations, improve our training datasets and word embeddings, and improve our estimates of the natural unit-related error rate through analysis of larger corpora.

VIII. Acknowledgements

We would like to thank Nate Kushman, Nuno Lopes, Tom Minka, and Sruti Srinivasa Ragavan for their advice; Martin Erwig for sharing the source code of the system he developed with Chris Chambers; Shi Han and Ran Jia for assistance with table detection and knowledge sharing; Dany Fabian and Gavin Smyth for engineering support; our unit annotator Kathryn Smyth; Anusha Iyer for help designing the unit annotation tool; and our dimension annotators.

References

[1] A. J. Kennedy, "Programming languages and dimensions," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-391, Apr. 1996. [Online]. Available: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-391.pdf

[2] C. Chambers and M. Erwig, "Reasoning about spreadsheets with labels and dimensions," *Journal of Visual Languages & Computing*, vol. 21, no. 5, pp. 249 – 262, 2010.

[3] T. Antoniu, P. A. Steckler, S. Krishnamurthi, E. Neuwirth, and M. Felleisen, "Validating the unit correctness of spreadsheet programs," in *Proceedings of the 26th International Conference on Software Engineering*. IEEE Computer Society, 2004, pp. 439–448.

[4] R. Abraham and M. Erwig, "Ucheck: A spreadsheet type checker for end users," *J. Vis. Lang. Comput.*, vol. 18, no. 1, p. 71–95, Feb. 2007. [Online]. Available: https://doi.org/10.1016/j.jvlc.2006.06.001

[5] M. Erwig and M. Burnett, "Adding apples and oranges," in *Practical Aspects of Declarative Languages*, S. Krishnamurthi and C. R. Ramakrishnan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 173–191.

[6] H. Dong, S. Liu, S. Han, Z. Fu, and D. Zhang, "Tablesense: Spreadsheet table detection with convolutional neural networks," in *AAAI*. AAAI Press, 2019, pp. 69–76.

[7] D. A. Orchard, A. C. Rice, and O. Oshmyan, "Evolving fortran types with inferred units-of-measure," *J. Comput. Science*, vol. 9, pp. 156–162, 2015. [Online]. Available: https://doi.org/10.1016/j.jocs.2015.04.018

[8] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," in *ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 32. JMLR.org, 2014, pp. 1188–1196.

[9] Z. Gao, C. Bird, and E. T. Barr, "To type or not to type: quantifying detectable bugs in javascript," in *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, S. Uchitel, A. Orso, and M. P. Robillard, Eds. IEEE / ACM, 2017, pp. 758–769. [Online]. Available: https://doi.org/10.1109/ICSE.2017.75

[10] A. Wilson, M. Burnett, L. Beckwith, O. Granatir, L. Casburn, C. Cook, M. Durham, and G. Rothermel, "Harnessing curiosity to increase correctness in end-user programming," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2003, pp. 305–312.

[11] G. Rothermel, L. Li, C. DuPuis, and M. Burnett, "What you see is what you test: A methodology for testing form-based visual programs," in *Proceedings of the 20th international conference on Software engineering*. IEEE, 1998, pp. 198–207.

[12] A. F. Blackwell, "First steps in programming: A rationale for attention investment models," in *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*. IEEE, 2002, pp. 2–10.

[13] D. Kulesz and S. Wagner, "Asheetoxy: a taxonomy for classifying negative spreadsheet-related phenomena," *arXiv preprint arXiv:1808.10231*, 2018.

[14] M. Wand and P. O'Keefe, "Automatic dimensional inference," in *Computational Logic - Essays in Honor of Alan Robinson*, 1991.

[15] J. Goubault and R. Jaurès, "Inférence d'unités physiques en ML," *Journées Francophones des Langages Applicatifs, Noirmoutier*, 1995.

[16] V. J. Hellendoorn, C. Bird, E. T. Barr, and M. Allamanis, "Deep learning type inference," in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, G. T. Leavens, A. Garcia, and C. S. Pasareanu, Eds. ACM, 2018, pp. 152–162. [Online]. Available: https://doi.org/10.1145/3236024.3236051

[17] E. Horvitz, "Principles of mixed-initiative user interfaces," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 1999, pp. 159–166.

[18] C. A. Holt and S. K. Laury, "Risk aversion and incentive effects," *American economic review*, vol. 92, no. 5, pp. 1644–1655, 2002.

[19] H. Dong, S. Liu, S. Han, Z. Fu, and D. Zhang, "Tablesense: Spreadsheet table detection with convolutional neural networks," in *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI'19)*, January 2019. [Online]. Available: https://www.microsoft.com/en-us/research/publication/tablesense-spreadsheet-table-detection-with-convolutional-neural-networks/

[20] C. Chambers and M. Erwig, "Dimension inference in spreadsheets," in *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, Sep. 2008, pp. 123–130.

[21] J. Cunha, M. Erwig, J. Mendes, and J. Saraiva, "Model inference for spreadsheets," *Automated Software Engineering*, vol. 23, no. 3, pp. 361–392, 2016.

[22] D. Vrandecic and M. Krötzsch, "Wikidata: a free collaborative knowledgebase," *Commun. ACM*, vol. 57, no. 10, pp. 78–85, 2014.

[23] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[24] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017. [Online]. Available: https://www.aclweb.org/anthology/Q17-1010

[25] E. McKean, "Wordnik," in *The Routledge Handbook of Lexicography*. Routledge, 2017, pp. 473–484.

[26] C. Allen, I. Balazevic, and T. Hospedales, "What the vec? towards probabilistically grounded embeddings," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 7467–7477. [Online]. Available: http://papers.nips.cc/paper/8965-what-the-vec-towards-probabilistically-grounded-embeddings.pdf

[27] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[28] F. Hermans and E. Murphy-Hill, "Enron's spreadsheets and related emails: A dataset and analysis," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 7–16.

[29] T. Barik, K. Lubick, J. Smith, J. Slankas, and E. Murphy-Hill, "Fuse: a reproducible, extendable, internet-scale corpus of spreadsheets," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 486–489.

[30] M. Fisher and G. Rothermel, "The euses spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms," in *Proceedings of the first workshop on End-user software engineering*, 2005, pp. 1–5.

[31] J. L. Fleiss, "Measuring nominal scale agreement among many raters." *Psychological bulletin*, vol. 76, no. 5, p. 378, 1971.

[32] M. Hossin and M. Sulaiman, "A review on evaluation metrics for data classification evaluations," *International Journal of Data Mining & Knowledge Management Process*, vol. 5, no. 2, p. 1, 2015.

[33] 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 E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[34] B. Shmueli, "Multi-class metrics made simple, part II: the F1-score," Dec 2019. [Online]. Available: https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1

APPENDIX

A. Extra details for Predictor 1

This appendix provides extra details that can be useful for reproducibility (e.g. for reaching the results of *Predictor 1* as shown in Table IV) and for future model extensions. In particular we address how to train word embeddings for tabular data using *FastText* [24] and provide further intuition for our modelling choices.

1) Processing choices for training word embeddings: In our work we experimented with various static word embeddings (either pre-trained on a large external corpus or trained on Excel tables) and found that *FastText* produces the best performance when trained on tabular data.

Instead of learning vectors for words directly, *FastText* represents each word as an n-gram of characters. This helps capture the meaning of shorter words and allows the embeddings to have a sense of suffixes and prefixes. It also helps with rare or out of vocabulary words as even if a word is not present in the training set it can be broken into n-grams. Unsurprisingly these are important considerations for tabular data.

In order to use *FastText* on our data we transformed every table to a list of tokens. We kept only non-numerical cells and removed the columns that had words from a banned set. We generated the set of forbidden words by looking at the histogram of word occurrences in our training data. We identified words that are likely to be non-informative for the context of the spreadsheet (e.g. 'yes', 'true').

For training, we used most of the default settings with the exception of learning rate (0.05), number of epochs (20), number of threads (maximum number of CPUs), size of the context window (4), size of the embedding vectors (300) and length of n-grams (2).

We created a random selection of workbooks for training embeddings (11538 workbooks) from a mix between ENRON, FUSE and EUSES . We had to eliminate spreadsheets with significant duplication. For instance in ENRON data the file name contains the name of the person who authored the spreadsheet and it is common for one person to duplicate their template across many spreadsheets. We keep just one of those workbooks to avoid too much overlap. In our best experiment we added the documents created from Excel workbooks to a well known dataset of business articles, Reuters-21578, Distribution 1.0 which can be found at http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html. We chose articles that are short so there is no significant discrepancy.

In our work we have investigated other approaches as well. For Glove embeddings one can fine-tune to new data (for example by using Mittens), but the technique is not very reliable and it did not produce good results in our case. For FastText there is currently no equivalent. Another approach that we leave to future work is to train word embeddings on new data and then aligning the embedding space to a richer space of pre-trained embeddings on a large corpus.

2) The intuition behind our modelling choices: As previously mentioned, we can motivate the choice of using static word embeddings over dynamic ones by exploring the vocabulary that is characteristic for spreadsheets. In our study we found that 88% of the words in our corpora are mono-semantic (i.e. have a single meaning for part of speech) compared to 42% in Wiki English by using Wordnik [25].

The only assumption that we make in our work is that the embedding of mono-semantic words is within a small error from the correct embedding and that error is isotropic.

In order to find how similar are two words (for example 'cm' is closer in meaning to 'metre' than to 'volume') we use the standard cosine similarity metric, whose inherited error remains low and bounded, and avoids issues identified with Euclidean distance [26]. Intuitively let's consider two vectors in 2D, a and b that are the correct (oracle) representations for two words. Our assumption states that our embeddings will be within a maximum (small) error of ϵ (another 2D vector) from the correct representations. So the similarity error δ_{err} can be bounded by:

$$\begin{aligned}
\delta_{err} &\leqslant |\cos(a-b+2\epsilon)-\cos(a-b)| && (1)\\
&= |\cos(a-b)\cos(2\epsilon) && (2)\\
&\quad -\sin(a-b)\sin(2\epsilon)-\cos(a-b)|\\
&\approx 2\epsilon|\sin(a-b)| \text{ to first order in } \epsilon. && (3)
\end{aligned}$$

For the words that are truly similar to each other we can see that δ_{err} should stay small. This is an intuitive explanation for 2D and in higher dimension we need more machinery to support this argument, but we can formulate it using a similar line of reasoning.

As shown in *Predictor 1* we define functions $\mathsf{s}(u)$ and $\mathsf{s}(d)$ that compute the similarity score of a unit or dimension to input header h. To aggregate the unit similarities into a dimension similarity we pick the highest unit similarity per dimension that satisfies a weak version of transitivity, defined by predicate check. This constraint is a desirable property for faithful embeddings and we can show that it is not violated if the embedding error is consistent with our assumption. To exemplify, if *depth* and *metre* are very similar and *metre* and *cm* are truly similar, we require *depth* and *cm* to also be very similar. If this relation is not true we have identified a false-positive similarity, an artefact of the propagated error (i.e. the two words are not truly similar, but appear to be due to the error in our embedding).