KU LEUVEN

Knowledge Discovery on Spreadsheet Data using Semantic Information

Sarah Binta Alam Shoilee

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Big Data Analytics

Thesis supervisor:
Prof. Dr. Luc De Raedt

Assessors:
Assoc. Prof. Dr. Marc Denecker
Dr. Ir. Samuel Kolb

Mentor:
Dr. Ir. Samuel Kolb

Academic year 2019 – 2020

Preface

I would like to thank my daily supervisor Dr. Ir. Samuel Kolb for his excellent guidance and support during the process. My sincere gratitude also goes to my promoter and my assessors. Special thanks goes to KU Leuven for allowing me to work on this thesis topic. The knowledge and insights, I gathered from this experience will follow me throughout my career.

Sarah Binta Alam Shoilee

Contents

Abstract

For data processing and maintenance, a spreadsheet is the most widely used tools amongst corporations. Performing in-depth research on spreadsheets has increasingly become critical to enabling knowledge discovery as well as improving user efficiency and supporting error-free data processing. Further research in this field is essential, considering the sheer expanse and the importance of tabular data in the data community, despite the number of existing works in the area. This paper proposes an algorithm for tabular constraint learning using semantic header information, focusing on enhancing existing tabular constraint learner (TaCLe). The proposed solution will rediscover lost formulas on a poorly filed spreadsheet while conceptually analysing the header. The proposal of the header interpretation assists to recapture formulas faster with more accuracy than the previous solution, overall improving the computational performance.

Furthermore, this research outlines the usability of header information come along with tabular data, often ignored due to its complex structure. The study further proposes a formula reconstruction application, empirically validating each step of the design decision. This technique will not only be capable of formula rediscovery but also for formula suggestion, error-correction, and auto-completion in a spreadsheet environment when integrated with a right interaction model.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

CP	Constraint Programming
CSP	Constraint Satisfaction Problems
NLP	Natural Language Processing
CSV	Comma-separated format
ML	Machine Learning
POS	Parts-of-speech
LOOV	Leave One Out Validation
CV	Cross-validation
DT	Decision Tree
RF	Random Forest
LSVM	Linear Support Vector Machine
RBF SVM	Kernel Support Vector Machine
PCA	Principal Component Analysis
TaCLe	Tabular Constraint Learner
IoT	Internet of Things
ERP	Enterprise Resource Planning
tf-idf	Term FrequencyInverse Document Frequency

Symbols

c	Constraint
v	Vector
t_i	Header text assigned for vector i
w_i	Header word for vector i
β	Block
β'	Sub-block of β
$\boldsymbol{\beta}$	Set of maximal blocks
T_n	n^{th} detected table
H_n	Header for n^{th} table
$\boldsymbol{C}$	Set of Constraint Templates
$\boldsymbol{V}$	Set of vectors available
$\boldsymbol{W}$	Set of all header text available
$i = 1, \ldots, k$	Index over vectors
$j = 1, \ldots, l$	Index over constraints
$n = 1, \ldots, m$	Index over tables

Chapter 1

Introduction

A spreadsheet is one of the most frequently used data processing and preservation tools among many enterprises [44]. However, the people who maintain and modify these data often lack experience in standard data manipulation practices [14]; which can compromise data quality. The existing bodies of research on spreadsheet errors suggest that the chances of making mistakes in the data-sheet are 63%, even when the creator and maintainer is the same individual [32]. Not to mention, the problem further exacerbates when people work on data-sheets created by the other people in the workforce. On the contrary, spreadsheet developers and the corporations remain quite optimistic about the precision of the spreadsheet [32]; this fact needs serious attention to further research.

Error correction has long been a topic of great interest in a wide range of research fields from professional coding to everyday spreadsheet use. Consequently, auto-synthesising programs have gained extensive popularity over the last two decades among researchers. In a modern, digitally emerged world, auto rectification deserves new attention, since billions of people have access to computational devices such as cellphones, computers, and Internet of Things (IoT) devices [46]. The rising trend of tech device usage presents an enormous challenge of error-free data generation, given that, the flawless use from an inexperienced user is a high requirement [36]. Using everyday tools, i.e. spreadsheet reliably is very crucial, as it ties back to data processing and data maintenance.

Automation in minimising error has tremendous potential to improve the accuracy in practice significantly. This paper introduces a system where spreadsheet formulas can be auto-constructed based on the tabular data stated and using the header label of a row or column. The cells reside in the same table row, or column typically contains the same computational semantic [14]; therefore, it is rational to consider as cell clusters to justify together. Formula auto-construction on a tabular data is crucial when data-sheet is imported from other software, e.g. Enterprise Resource Planning (ERP) system. In this scenario, data is often in comma-separated format (CSV), where no formula or relational entity remains from the original data platform [27]. Manual formula reconstruction of those files could be time-consuming, tiresome and most importantly error-prone.

Regardless, in the case of a manually created spreadsheet, consistency and accuracy in formulas also remain a challenge. A considerable amount of literature exists in support of unambiguous spreadsheet computation. These studies range from code smell detection [22][14] to test case generation [3]. Although recent developments in the field of spreadsheet management led to a renewed interest, the debate continues to be the best strategies for the management of the spreadsheet. In line with other research, a successful application of the proposed system of this paper can endure error-correction, formula suggestion, and auto-completion when integrated with the right interaction model.

The work, presented in this thesis, focuses on finding tabular constraints (mathematical functions or formulas) based on the header label of each row or column (a cell array) stated in a spreadsheet. It introduces a semantic heuristic approach for formula reconstruction on a spreadsheet. The system concentrates on using the semantic information of cell arrays, mentioned in its header, on directing the search for attainable tabular constraint imposed on a spreadsheet. For each constraint, the process first locates the formula output cells and then try to fit other argument variables as per formula requirements. Alternatively, the technique proceeds by assigning left-hand-side argument for a particular arithmetic formula, and then search for the satisfiable right-hand-side argument(s) by iterating over every possible combination of available rows and columns. The repetition of this technique for every possible constraint will ultimately reveal the constraint or formula initially been on a spreadsheet.

The current work is based on the algorithm TaCLe, introduced by Kolb et al. [27], and thus named as Semantic-TaCLe. TaCLe solves the exact problem as of this thesis and does an excellent job in re-establishment of each supported formula on given tabular data using only constraint programming. However, this algorithm suffers from false-positive formula detection, as long as it satisfies the minimum restriction established by that formula constraints. The current work proposes a discriminatory constraint search approach using table header information to overcome the mentioned limitation of TaCLe. Moreover, TaCLe's generate-and-test approach experiences long waiting time when the number of cell array gets higher, despite their intelligent constraint validation technique. This research attempts to manage the computational time by eliminating the candidate variable before constraint validation.

The originality of this study lies within the benefit of having a pre-search heuristic approach; instead of investigating all cell arrays and a blind search through all possible constraints. The contribution of this paper is as follows:

- The proposed system introduces a heuristic approach based on textual information to learn tabular constraint on a spreadsheet.

- It proposes a predictive method on learning constraint existence on cell array for search space minimisation for constraint validation.

- Moreover, the implementation of the system presents a priority-based search algorithm to find satisfactory constraint with minimal effort.

Rest of the paper proceeds as follows: Chapter 2 highlights the existing research on a similar context as of this research. Chapter 3 lays out the theoretical dimensions for the current research, while Chapter 4 formalises the problem. After presenting the methodology in Chapter 5, the paper lists the results of systematic observation of the proposed system in Chapter 6, concluding in Chapter 7 by mentioning the possible application and future research scope of the proposed method.

Chapter 2

Related Work

Semantic-TaCLe borrows techniques from several fields of research, though TaCLe [27] remains the core underlying technique. This chapter discusses a few related works that inspired the current project. The whole chapter is divided into four sections; each section states the relevance of the proposed work in the mentioned field.

2.1 Spreadsheet Automation

Knowledge representation and reasoning of spreadsheet data have long been a concern in related research domain due to the non-restricted design freedom offered by the interface. The necessity of abstract formula auto-construction of spreadsheet data was first addressed and solved by Isakowitz et al. [24]. Such logical generalisation using software engineering practices remain popular in error correction or formula reconstruction in a spreadsheet environment [37][10][11]. Conceptually the purpose and intention of these papers are similar to the current paper; however, the way of minimising error and formula reconstruction differs. These techniques reconstruct general abstract formulas based on the formulas provided in the spreadsheet, as opposed to the proposed technique where no such formula information is available.

Although the implication of software development practices has significant potential to avoid error, for general users, a small automated feedback loop is rather practical [11]. In that effort, an array of technology exists for spreadsheet automation with an interactive tool. A lot of these tools use constraint solver which supports different use, e.g., intuitive addition or deleting of cell value [15][4] or error detection [14][3]. The current work also follows the baseline proposition for constraints as these papers, i.e., the cohesiveness of neighbouring cell values. Although, the available information varies depending on the context of the application.

The proposed work is inspired by two works of Abraham and Erwig [5][2] in a similar spirit of constraint ranking instead of hard constraint assignment, though used different heuristic for ranking. The need for human intervention to finalise constraint is eliminated in the current approach by using further constraint validation. Inferring constraint imposed on tabular data through constraint propagation is also useful in

the current context due to the nature of relational data presented on the spreadsheet. On the contrary, it is a matter of great concern to decide the right technique for constraint propagation algorithm for this task, given the exponential size of possible row and column combination. Two notable methodologies for constraint propagation incorporate ModelSeeker [8] and Quacq [9]. These two approaches differ from the underlying constraint validation technique of the proposed work in terms of data type and the number of tables. Besides, these techniques are more appropriate for constraint learning from spreadsheet examples; therefore, the necessity of efficient constraint learning technique for tabular data persists.

2.2 Header in Automation

A useful property to understand the spreadsheet data structure, often overlooked due to the complicated structure, is the table headers. To use this valuable information, a good number of research works are available for correct retrieval of the table header [1][16][12], although the target remains very context-specific. Inspired by these works, the proposed method only considers two general scenarios of possible header placement to exploit the benefit. From a hypothetical point of view, the current work is based on a proposition that the visual structure of the table can be transformed into a logical format [34].

Unit reasoning using header information [1] is the essence of the current work. Instead of assigning 'hard constraint' based on the unit header, the current method makes a header inspired constraint suggestion. Consequently, it connects to the line of research of table conversion to relational schema based on the header [6]. The proposed research extends this scope beyond simple relational table construction by revealing the functional definition of data that belongs to a header.

2.3 Machine Learning

With the rising popularity of machine learning (ML) in knowledge discovery field, researches adopted these techniques for spreadsheet table and header inference [6][35] and empirically validated their performance. One such paper by Koci et al. [26] performs a tabular cell classification based on the drawn feature on each spreadsheet cell and ultimately adds up to find the layout of the spreadsheet. In this paper, the author considers finding five different layout building blocks based on the conceptual and spatial features of cells. Notable to mention here, one of the deciding features for this task was 'WORDS_LIKE_TOTAL' which informs, having word token such as 'total', 'sum', 'max' and proven to be a reliable indicator to decide header cells. This observational result provides support to the concept proposed in the current research. Moreover, the experimentation section of the same paper indicates the eminent success of machine learning technique in cell value classification. The limitation of this paper applying in the current context is, it requires a lot of cell information presence which is not available in a CSV formatted tabular data.

The problem, this thesis attempts to solve is related to named entity recognition problem, as it proposes to identify words that are used to define mathematical formula derived cell. Paper [47] reveals the scope of classifying tabular entity based on header words, using only a classifier and simple bag of word features by comparing it against a heuristic rule-based learner.

From a constraint acquisition perspective, the current work is framed as the application, Constraint Seeker [7], prompting the usefulness of using machine learning in the current context. The only difference is that the current problem is unaware of its specified constraint variables, therefore, requires a more rigorous ranking strategy and pre-pruned variable domain to identify the constraint variable explicitly.

2.3.1 Knowledge Inference

The tabular spreadsheet is designed and presented in logical connections for human intervention. The linear visual clues of tables inspired many research works. The conceptual relationship presented for human interpretability is also fueling knowledge discovery, based on the recent advancement of Artificial Intelligence ranging from text analysis to bitmap analysis. Paper [23] discovers the semantic relation between table entities using the indexing structure of the table and emphasises the importance of Natural Language Processing (NLP) for understanding the table. Furthermore, the author advocates on knowledge discovery from the table by the interpretation, which is also the theme of the current paper. Nonetheless, the mentioned paper only exploits the ontological relationship between table entities, although more potential for interpretation exists.

Two early work [17] and [28], done on poorly filed XML-based document for the American companies' financial statements, are the closest to the proposed approach in the context of knowledge discovery. In these papers, information is derived from presented data based on constraints that are appropriate for the accounting context. The files, consider in these papers, belong to the financial statement which uses the exact template formatting for information years after years; this fact makes the exact token matching possible. On the contrary, the perfect matching restriction makes it harder to apply these methods in a generic context, though empirically, they have been proven to be highly reliable in the applied context.

A limited number of works have targeted to solve a similar problem with knowledge interpretation, so far. However, the few attempts made until now did not utilise the full potential of knowledge discovery through interpretation task in a tabular setting; either by limiting it in identifying ontological relationship only or by keeping the solution very domain-specific. Nevertheless, it provides great inspiration for future work by showing the promising success of such an approach.

2.4 Conclusion

Considering the recent developments in the field of machine learning and text analysis, ignorance of header information to interpret spreadsheet data is not optimal. The current work argues that the header information has optimistic potential to trace the

content in tabular data. Moreover, the potential of header information has already been explored to construct relational databases from a plain CSV or HTML tables. Realising the importance of this useful information, a considerable number of research works exists for correct header retrieval applying various approaches. Following this line of research, this thesis suggests a system architecture where formula discovery on a CSV table is made using header information with reasonably fast and reliable implementation.

Chapter 3

Background

This chapter introduces the theoretical frameworks that the proposed application is based on; hence, it is divided into three sections to deliver the concepts. The chapter begins by providing a brief overview of Constraint Programming, its solution methods and underlying assumptions. A reasonable understanding of constraint programming is necessary to a clear view of the core technology of the current application, Tabular Constraint Learning (TaCLe); which is demonstrated elaborately in the next section. The last section introduces the final theme that the current application will take advantage of, and thus set out the theoretical dimensions of the research in text categorisation.

3.1 Constraint Programming

Constraint programming (CP) is a declarative programming paradigm modelled on comprehensive techniques from artificial intelligence, computer science and operations research. It differs from empirical programming by its nature of specifying how to solve a problem. Instead of guiding a program step by step to reach a solution, the problem in a canonical form is given as an input where the constraint solver is then accountable for solving it [39]. The problem is usually defined in the form of constraints that range over a set of variables to assign feasible solutions. Typically, CP attempts to solve a combinatorial problem, which is an NP-hard problem due to the extensive number of candidate variables-value combinational pairs.

More formally, the problem solved by CP is known as Constraint Satisfaction Problems (CSP). Given a finite set of variables $\boldsymbol{X}$ and a finite set of constraints $\boldsymbol{C}$, the NP-hard constraint satisfaction problem is to discover variable value assignments that satisfy the constraint requirements [20]. For each variable x_i, there is a domain D_i consisting of a set of possible values for x_i where $D_i = \{v_1, \ldots, v_k\}$. Each constraint c_i can be defined as a tuple <scope, relation> where the scope defines the variable arguments stated in the constraint and relation states the restriction that the concerning arguments must follow. A problem state-space with variables' candidate values and the notions of solutions should be defined to solve a CSP. [40, Chapter 6]

CSP search algorithm utilises the problem's state-space and adopts generic heuristics rather than problem-specific heuristics to discover the solution. Two conventional techniques for CSP solvers is backtracking [19] and constraint propagation [45]. The fundamental idea of constraint propagation is to eliminate a candidate from the search space by identifying nonlegal value assignments for variables. Constraint propagation downsizes the search space of each variable so that the possible values of all variables hold Local Consistency. Local Consistency can be defined through a number of concepts such as Node consistency, Arc Consistency, Path Consistency, K-consistency, Global Consistency, Consistency and satisfiability and many more [40, Chapter 6][39, Chapter 3].

Constraint propagators can be deployed as a pre-processing step before any solution search algorithm for value assignment or can be intertwined within the algorithm itself. Sometimes inference checking leads to a converging point where no valid value remains in the variable's domain or the possible solution of the problem is derived, i.e., Sudoku solver [43], in such cases, no further search is required. From empirical research, the benefit of constraint propagation in CP is realised with greater importance and thus remains the heart of constraint programming. The efficiency of CP implemented with a propagator can be further improved by adopting different domain-independent or problem-specific heuristic or constructing a structural representation of the constraint graph.

3.2 TaCLe

TaCLe (Tabular Constraint Learner)[27][33] is an unsupervised learning technique for existing logical or relational formulas over a spreadsheet. It learns formulas or constraints over mixed variable types such as integer, float or text when they are presented in a tabular format. TaCLe presents the formula discovery problem as an inverse CSP to tackle a large number of possible row or column combinations that it needs to consider as variables for constraint learning. To formalise what constraint the variable instances may hold, TaCLe has a knowledge base for each constraint, called a Constraint Template. Constraint Template defines each constraint with a tuple (**Syntax**, **Signature**, **Definition**). **Syntax** specifies the arguments, **Signature** defines the constraint requirement for candidate arguments, and **Definition** states the functional requirement [27].

Without any contextual information (i.e., headers, metadata), TaCLe considers a tabular data as an α x γ matrix, where γ is the number of rows and α is the number of columns in the table. TaCLe forms blocks by concatenating multiple adjacent rows/columns of the same type (textual or numeric) and orientation. The concept of blocks is restorative to identify logical groups that reside in neighbouring rows or columns and to reason over data type. Another advantage of this concept is that one can also consider a block β as a super-block of a sub-block β', iff β contains all the vectors from the β' in the same orientation from the same table. These logical blocks are the candidate variables for the constraint solver in the following stages. [27]

As a whole, TaCLe can be seen as a two-staged process. The first stage is

candidate generation and elimination based on the template for individual constraint. Each retrieved block is viewed as a potential candidate variable for the templates' argument but in their maximal state. In other words, each block, along with all possible sub-blocks in it, should be examined as a template argument according to the constraint template's syntax. All the properties mentioned in the Signature for each constraint are then converted to the CSP constraint and assigned to a generic CSP solver to find valid arguments. By doing so, TaCLe eliminates locally inconsistent blocks from a particular constraint's domain and in consequence, ease the search space complexity [27].

The second and final step is to inspect all the qualified blocks for each constraint and test whether they will satisfy the constraint's specification both logical and syntactical manner. At this stage, TaCLe considers not only the remaining blocks from constraint's domain (which are in their maximal form) but also all the possible sub-blocks based on the Signature of the template. In short, this process can be described as a generate-and-test process. TaCLe implements its own validation class for each constraint due to the limitation of handling floating point by generic CSP solver. TaCLe also adopts a few CSP solver optimisation techniques such as creating a dependency graph where higher-node constraints are prerequisites for the dependent constraints and removing functional redundancy.

Overall, TaCLe uses a constraint propagation approach to eliminate candidate variables from search space, thus managing the complexity better of such a combinatorial problem of a spreadsheet. Moreover, maintaining TaCLe's test classes for constraint satisfiability for the later stage helps to deal with non-integer data residing in a spreadsheet. Solving such a problem with a CSP solver is also efficient to create higher accuracy. Furthermore, introducing constraints in a template format makes it adaptable for context-specific application. In short, the automation offered by TaCLe [33] can help a less familiar user to identify formulas from raw data in a considerable amount of time but with high recall.

3.3 Text Categorisation

Classification or categorisation is a supervised machine learning technique that learns from observational data. The given data or observation may have both textual and numeric representations of information along with its associated class label. Based on the dataset, a predictive model is then constructed utilising a training procedure, which can be used to perform forecasts on unseen data. The dataset, on which the model is trained, often needs pre-processing based on the task it is trying to accomplish. For example, if textual data is present in the observation, it depends on the task whether to choose the entire text or just a segment of it. Appropriate terms or feature extraction for textual data representation has long been a subject of research in the field of ML and NLP [42][31].

3.3.1 Features Engineering

The process of information extraction from textual data has two significant parts. First, the system extracts individual 'facts' from the text through descriptive text analysis. The individual facts can be extracted by exploiting linguistic and semantic knowledge. For such approaches, part-of-speech(POS) tagging or dependency tree representation has been two standard practices [25]. Second, the pertinent facts are translated into the required input format, i.e., word vectors [21]. These parts of information extraction are connected inextricably with each other and indispensable to capture features from textual data.

3.3.2 Feature Vector

Unfortunately, unstructured text data cannot be immediately interpreted by most classifiers or by classifier-building algorithms. For this reason, feature vector representation or indexing is necessary to transform the text into a compressed, fixed-length description [42]. The choice of the vector composition method relies upon what one views as an essential unit of text. The standard 'term frequencyinverse document frequency' (tf-idf) function [41], given below, is the most often used method for vector representation:

$$tf - idf(t_i, d_j) = \frac{\frac{Count(t_i, d_j)}{|t \in d_j|}}{\frac{\sum_{j=1}^{n} Count(t_i, d_j)}{|d \in \boldsymbol{D}|}}$$

The above function is used to evaluate the weight (tf-idf score) of each word t_i that exists in text d_j to construct a weighted vector $< w_{t_1}, w_{t_2}, ..., w_{t_k} >$ for text d_j. This function, along with most indexing functions, uses weighted importance of an individual term occurrence in a text to convert it into a fixed-length vector, thereby disregarding the issue of compositional semantics [42].

On the other hand, another vector representation technique for words is word2vec [30], which has proven to carry semantic meaning by projecting words into high-dimensional vector spaces and keeping similar contextual words close to one another. In this approach, embedded vectors are constructed by training shallow neural networks on a large text corpus to reconstruct linguistic contexts of words [38] . Word2vec embedding claims to provide state-of-the-art performance on test data for measuring syntactic and semantic word similarities [30].

3.4 Conclusion

This chapter introduced the three central topics of the proposed solution. The organisation of this chapter is as follows. Section 3.1 defines constraint programming, which will further help to understand how formula learning on tabular data can be achieved using this technique. Section 3.2 describes TaCLe, an algorithm that has already solved the tabular constraint learning problem using constraint programming. Furthermore, it is the underlying algorithm for the current project. Lastly,

Section 3.3 illustrates text analysis, a well-known technique for Natural Language Processing(NLP) and the basis of the core contribution of this paper.

Chapter 4

Problem Description

In this chapter, the task of learning tabular constraint using semantic information is formally introduced. To have a good understanding of the presented problem statement, the introduction of a few terminologies is crucial. Hence this chapter begins by laying out a few definitions along with its repeatedly used notation throughout the text in Section 4.1. The later sections define the problem statement formally and demonstrate two different approaches to extract necessary information from header words for constraint learning.

4.1 Terminology

Spreadsheets and tabular data may consist of multiple tables along with one or more headers. The current system does not consider building blocks from a cell level granularity of a table, rather reason over an entire row or column and addresses them as **vector**(v). Moreover, the system emphasises more on the contextual information mentioned in table **headers**(H) for each **table**(T). To put it another way, it tries to learn formulas those range over rows or columns from their corresponding header semantics available within the same table. Thus, this paper will emphasise more on the retrieved header, that is mapped and indexed with their corresponding table, for formula learning. The process of mapping the header with its tabular will be discussed further in Section 5.1.

The goal of this task is to learn spreadsheet formulas that were initially imposed in the data values but lost due to the file formatting. The system borrowed the terminology and their notation from its previous work, TaCLe [27]. To solve this problem in CSP context, each formula is considered as an 'atomic' **constraint**(c) (a set of constraint rules) and defined as a **constraint template** with its essential knowledge base as introduced in TaCLe. The formulas that the given application intends to support is collectively called **supported constraints** and templates describing each of these formulas are stored in set $\boldsymbol{C}$; where $c \in \boldsymbol{C}$ and c is in constraint template format. Each c is described using three components (**Syntax, Signature, Definition**): **Syntax** specify the argument requirement for constraint satisfaction, **Signature** describes properties or rules of a valid argument and **Definition** es-

tablishes the logical restriction for that argument. In this constraint specification, argument variables can be both a single vector or multiple contiguous type consistent vectors, called **blocks**. Recall from Section 3.2, a block β' can be a sub-block of β when it can be said that $\beta' \subseteq \beta$. Initially, blocks are created by concatenating neighbouring homogeneous vectors (numeric or textual) together and keep merging until a different type occurs. When blocks are in their largest possible form, TaCLe addresses it as **maximal block** [27].

Let's illustrate the concept of constraint template with an example. The following text elucidates the constraint template for a spreadsheet aggregate formula, SUM. To identify evidential existence of an summation formula from a spreadsheet data representation, a constraint learner (as implemented by Kolb et al. in [27]) tries to computationally match multiple vectors $(v_1, \ldots v_n)$ to another vector v_i with a summation relation and thus can be denoted as $v_i = \text{SUM}\{v_1, ..., v_n\}$ where v_i as the **target variable** and any other variable on the right-hand-side is called **data variable**. The argument specification is controlled by the 'SUM' constraint template **signature** and have to satisfy the rule specification mentioned in the template's **signature**. Finally, the logical consistency whether the summation of vectors $\{v_1, ..., v_n\}$ equal to v_i is checked by the **definition**. The complete template for summation formula is given in 5.1.

For a particular constraint, a **target variable** is a vector to what all the other argument vectors of the same constraint are logically mapped on. The argument that provides the data for the target variable is actually the **data variable**. The proposed application assigns a value for individual vectors where the value signifies vector's chance of satisfying a constraint template or 'having a constraint' based on its **header text**. When this thesis says 'having a constraint c_j' in vector v_i, it means v_i is the target variable of constraint c_j. As a matter of fact, this application will be dedicated to forecasting the chances of 'having a formula c_j' for vector v_i.

4.2 Problem Statement

Learning formulas from spreadsheet data can be framed as an inverse Constraint Satisfaction Problem. In a spreadsheet, constraint variable values are given in a tabular format, and the task is to identify constraints from a list those are satisfiable by the given data. The learning process operates on an array of allowed constraints, namely supported constraints. The resulting solution is the list of constraints instantiated with the satisfiable argument variables. In this context, constraints are template representation of arithmetic formula supported by the application. Each constraint has a particular argument variable called the target variable. For the target variable, the allowed value is always a vector, and for data variables, it can be a block (one or more vector). The task is to find the right assignment of the constraint variables from tabular data that satisfies for each supported constraint.

Given: A set of supported constraint $\boldsymbol{C}$, a set of vector $\boldsymbol{V}$ and a set of maximal blocks $\boldsymbol{\beta}$

Find: all satisfiable assignment $c(v, B)$ for all $(v, B) \in \boldsymbol{V}$ X $\boldsymbol{B}$ and $c \in \boldsymbol{C}$; where $\boldsymbol{B} = \{\beta' : \beta' \subseteq \beta \text{ and } \beta \in \boldsymbol{\beta}\}$.

Here, $\boldsymbol{V}$ X $\boldsymbol{B}$ is the **search space** for the constraint learner to instantiate all satisfiable $c \in \boldsymbol{C}$. For some constraint $c \in \boldsymbol{C}$, it is possible that there are multiple satisfied argument pair. On the other hand, there can be constraints with no satisfiable argument. Due to the possible volume size of $\boldsymbol{V}$ X $\boldsymbol{B}$, in this paper the task of learning instantiated constraint will be divided into sub-problems. Initially, the problem is characterized as a categorization problem where the categorizes are different constraints, detail is given in Section 4.3. At the end for classification conformity, the system states the problem as a constraint validation problem. For constraint validation, this paper follows approach from TaCLe [27]; therefore, no further problem definition is given in this chapter.

4.3 Constraint Categorization

The task of learning formula constraint on a vector from its header words can be seen as a text categorization problem. In text categorization, the outcome is a set of textual instances assigned with their predicted category label or with the likelihood score of being on a certain category [42]. In formula learning context, text instances are the header words $\boldsymbol{W}$ and each formula is our predicted category c $\in \boldsymbol{C} = \{c_0, c_1, \ldots, c_n\}$ where c_0 is the null category which indicates no formula. The task of assigning formula likelihood for a vector $v_i \in \boldsymbol{V}$ based on its header word can be stated as below:

> Given header word w_i associated with its vector v_i and a predefined set of constraints category $\boldsymbol{C}$, assign a value P_{ij} to each pair $\langle v_i, c_j \rangle \in \boldsymbol{V}$ X $\boldsymbol{C}$ that indicates the likelihood of vector v_i 'having a constraint' category c_j.

For each vector v_i, a restriction $\sum_{j=0}^{l} P_{ij} = 1$ is enforced where the possible value for $c_0 \in \{0, 1\}$. Since, this is a text categorization problem, the system is going to consider learning function P_{ij} for vector v_i as a function of w_i in the following text, given that (v_i, w_i) is a tuple for vector-header word pair.

The problem of assigning likelihood for c_0 is a Boolean classification problem. The value assignment of the rest of category is only possible when $c_0 = 0$, due to the given restriction on Equation 4.2. In other words, the system first assess whether there is a constraint or not by assigning value to c_0 (0 or 1). When the system decide there is a chance of 'having a constraint' ($c_0 \neq 1$), only then it considers assigning likelihood for c_i where $i \in \{1, 2, ..., n\}$. Assigning value for P_{ij} where $j \neq 1$ is a rank categorization problem to avoid 'hard constraint' assignment. Both of these approaches are mentioned in details in the following two sub-sections.

4.3.1 Binary Categorization

To simplify the learning task of P_{ij} for all pair $\langle w_i,c_j\rangle \in \boldsymbol{W}$ X $\boldsymbol{C}$, the system assigns a bounding box value for the category c_0 of [0, 1] that is $\forall i\{(w_i, c_j) \in [0, 1]\}_{j=0}$. c_0 is the category for 'null constraints' (no constraint) which means '0' stands for having a constraint and '1' stands for not having a constraint. Once a value 1 is assigned to (w_i, c_j), i.e., $P_{i0} = 1$, due to the restriction$\sum_{j=0}^{l} P_{ij} = 1$, no further category ($c_1$, c_2, ..., c_l) value assignment is required; they are auto assigned to zero (0). This analogy is also logical, since once a text variable is categorized for 'null constraint' it should not appear in other category with a slightest likelihood.

Learning a function to assign value for c_0 is independent of all other categories; therefore, it can be solved as an individual binary categorization problem [42]. The process considers solving inverse c_0 ($\overline{c_0}$) as a binary classification problem. In other words, $\overline{c_0}$ is the category for 'having constraints' which means a value '0' assigned to $(w_i, \overline{c_0})$ indicates vector v_i does not belongs to c_0 categories. While a value of '1' indicates a decision of v_i belongs to any other category but c_0. This paper formalizes the binary text labelling task as follow:

> Given a set of labelled instance of words $\boldsymbol{W}$: $w_i \in \boldsymbol{W}$ and $\{w_i,\overline{c_0}\}\rightarrow$ [0, 1], the task is to approximate the unknown target function ϕ that describes how word w_i ought to be classified so that function ϕ and $\overline{\phi}$ coincide as much as possible with a certain degree of confidence.

Binary classification is a supervised machine learning problem with a set of labelled instances of $\{w_i \Rightarrow \overline{c_0}\}_{c_0\in[0,1]}$ retrieved from a underlying process. In addition, the labels $\overline{c_0}$ are assumed to be noiseless which makes sense, since in practice tables are created by excel sheet practitioners for their own understandability, and word labels for formula cell (also for the non-formula cells) can be assumed well defined.

4.3.2 Rank Categorization

The label assignment for c_0 has been considered a hard categorisation problem. On the other hand, categorization for c_i where $i \neq 0$ is a rank categorization problem. While complete automation is adequate for the decision of each pair $(w_i,\ \mathrm{c}_0)_{i=0}^{k}$, partial automation is required for rest of the categories. For instance, given $w_i \in \boldsymbol{W}$ a system simply ranks the categories in $\boldsymbol{C} = \{c_1, c_2, \ldots, c_l\}$ according to their estimated appropriateness to w_i, without considering any 'hard' decision for any of them. Such a ranked list is of great advantage for other automated tools for making the final categorisation decision through validation. It helps the underlying tool with an informed search approach, instead of, a blind search through the problem space by placing the probable categories on top of the list.

Unlike c_0, the independence condition does not hold on any other constraint categories. In other words, the value assignment for $\{c_1, c_2, \ldots c_n\}$ is interdependent of each other and most importantly on c_0. Ranking categorization of $c_1, c_2, ..., c_n$ applies if and only if $\overline{c_0}$ is assigned to 1; otherwise, no further ranking is required. Moreover, due to the ranking restriction, the value for one category is also influenced

Terminology		
Symbol	Learning Context	Spreadsheet Context
c	constraint	formula
v	vector	single row or column
w_i	text assigned for vector i	header label for column/row index i
β	block	group neighbouring cell arrays of same type
β'	sub-block of β	group of cell arrays that resides in another cell arrays
$\boldsymbol{\beta}$	set of maximal blocks	set of largest possible groups
T_n	n_{th} detected table	n th table in the spreadsheet
H_n	header for n_{th} table	header for n_{th} table in the spreadsheet
$\boldsymbol{C}$	set of Constraint Templates	set of formula descriptions
$\boldsymbol{V}$	set of vectors available	set of available rows/columns
$\boldsymbol{W}$	set of all header text available	set of all header labels
$i = 1, \ldots, k$	index over vectors	
$j = 1, \ldots, l$	index over constraints	
$n = 1, \ldots, m$	index over tables	

Table 4.1: Notation used for tabular constraint learning

by other categories. So, we are interested to learn a function ψ for all $w \in \boldsymbol{W}$ that can reasonably rank constraint in $\boldsymbol{C}$. Here, the problem is to learn a function that produce a ranking score for constraint categories and it is significant enough to benefit other process or human judgement.

4.4 Conclusion

The task of formula prediction on vectors can be presented using equation 4.1 when restriction mentioned in equation 4.2 applies. Two separate approaches for obtaining the solution is described elaborately throughout this chapter. Such a text categorisation solution is useful for other processes like TaCLe to guide their constraint validation steps. The motivation was not only to guide the system to perform faster but also to make correct approximation before validation. The implementation of such a procedure coupled with other constraint solver guarantees to find solutions if one exists under certain vector. The next chapter will introduce how this text categorisation can be integrated with a constraint solver and benefit from it.

$$P_{ij}(c_j|w_i) = \begin{cases} \overline{\phi}(w_i) \in [0,1], & \text{if } j = 0. \\ \psi(w_i), & \text{otherwise} \end{cases} \tag{4.1}$$

$$\sum_{j=0}^{n} P_{ij} = 1 \tag{4.2}$$

Chapter 5

Methodology

The objective of the proposed work is to learn tabular constraint from spreadsheet rows or columns with search space reduction. This chapter is a comprehensive guide of the solution for the problem, introduced in the previous chapter. It begins with a brief overview and will then go on with mentioning detailed step-wise implementation of the process. Recall that, each row or column residing within the table is considered as a vector in our approach. Iterating over each vector from the datasheet, the system determines whether a vector (v) is a target variable for any member of the given constraint set.

From a high-level point of view, the proposed methodology of this paper contains the following steps, mentioned in Algorithm 1.

Algorithm 1: The Semantic-TaCLe approach

Step 1: Identify the tables($\boldsymbol{T}$) and their headers($\boldsymbol{H}$)

Step 2: Pre-process vector-header word pairs $\langle v_i, w_i \rangle$ for all $v \in \boldsymbol{T}$

Step 3: For each vector $v \in \boldsymbol{T}$:

(a) Predict constraint presence on v is the target variable

(b) If predicted positively: semantically rank constraints in set $\boldsymbol{C}$

(c) for each constraint $c \in \boldsymbol{C}$

1. Consider v as a target variable
2. Find remaining 'valid' input argument for c
3. Validate arguments variables assignment

The key contribution of the proposed work is step 3 of Algorithm 1. In principle, one could do constraint validation without having any sorted search space or pre-search approximation. However, using proposed estimation approach within acceptable bound will promote to shrink candidate variable for constraints before any constraint propagation. Moreover, a legitimate ranked list will place the expected

solution on top of the search list and thus facilitate to reach the desired solution faster. Besides, it assists the system to make an informed decision when more than one satisfied constraint exists for a particular vector.

On the other note, theoretically, right constraint class prediction could be achieved only using supervised machine learning model with sufficient data. However, this perspective would be quite interesting to investigate in future research. Due to the limitation of having enough annotated data, this project limits the use of classifier to predict formula vector or non-formula vector.

Another potential possibility of this work is to use only a semantic ranking technique (details in 5.3.2) to get constraint suggestion based on header information. However, such approach suffers from the fact that most of the vectors in the tabular data do not have any formula, and thus the system ends up wasting much time to find constraints in a cell array where constraint does not exist. This fact is empirically tested and verified in the evaluation section of this paper (see sub-section 6.1.2). Hence, the proposed system adopts a predictive model to find constraint existence before a constraint ranker to eliminate this downfall.

The details of the complete technique are given below.

5.1 Step-1: Table and Header Extraction

Automatic table extraction is an inescapable step to make any decision by exploring data from the table. However, there is no one single extraction solution that fits all the different variations of the table in practice [29]. From one perspective, the presence of header make the relational table human-readable; on the other hand, it operates as a root cause of making the table structure complex for auto-detection; this occurs due to the inconsistency of the header style among spreadsheets for different information representation. For example, the header can be adjacent to the table or disjoint in a neighbouring table, at the same time, it can be single-layered or multilayered to indicate the hierarchical structure of the tabular data [16].

Table auto-detection is a complex problem to deal with under the scope of the project, yet an integral part to implement the proposed solution. To deal with this situation, this project adopts solution from existing researches on this field. As a part of the process, table extraction was done using *AutoExtract* tool, introduced in TaCLe [27]. *AutoExtract* performs under two baseline assumptions: (1) there is no empty cell within the rectangular structure of a table and (2) individual table is separated from each other by at least one empty row or column. When these two assumptions hold, the identification of the table automatically is trivial.

Once the entire table is retrieved, the process mentioned above removes the header to extract tabular data only. For header identification, *AutoExtract* assumes header's existence on the edge of the table. Considering the top row(s) or left column(s) as the header of the table, when they are not consistent with their neighbouring rows and columns, is a logical proposition [12]. Moreover, headers are frequently observed at the beginning of the table rather than in the middle, coupled with data. The

Product	State	Sep	Oct	Nov	Sub-total	Column1	Product:	Sales:
Cherries	CA	200	360	230	790		Apples	2070
Bananas	AZ	350	230	150	730		Bananas	2420
Apples	TX	180	270	200	650		Lemons	740
Bananas	KS	400	240	310	950			
Lemons	AL	250	360	130	740			
Apples	FL	120	120	380	620			
Bananas	LA	330	270	140	740			
Apples	KY	110	320	370	800			

Figure 5.1: Header Identification: Scenario-1

specification of the header of a table not only assists in finding data but also supports in setting the possible orientation of the table.

Algorithm 2: Header Identification

input : $\boldsymbol{T}$-a set of Tabular data, $\boldsymbol{X}$- a set of retrieved table
output : $\boldsymbol{H}$-a set of all Headers

for *each table:* **do**
  $H_i \leftarrow X_i$ - T_i; ▷ `Scenario-1`
  if *(*$H_i \neq$ None*)* **then**
    continue
  if T_{i-1} *is compatible with* T_i **then**
    $H_i = T_{i-1}$; ▷ `Scenario-2`
  else
    $H_i \leftarrow$ None

Since the identification of header is an integral part of our approach, instead of removing it, the current system marks the header separately from tabular data and tags it as a piece of extra information to the table. Obtaining header in such a way covers only a few obvious possibilities of header placement, i.e., when the header is stated across the first row(s) or column(s). Figure 5.1 shows an example of referred scenario. This header identification procedure also covers the setting when the header of the hierarchical structure exists. For simplicity, this work chose to ignore the multi-layer header and considers only single-level header.

When the above scenario for the header does not hold, another frequent practice of header placements in the spreadsheet is to position it with a row(s) or column(s) gap from the tabular data. Such a placement of header is identified as a different table as per *AutoExtract's* assumption (2). For this reason, the system also considers the neighbouring table to examine if they can be considered as the header of the following table. One such scenario is presented in Figure 5.2. Header detection algorithm tries to match the dimension of a table (T_n) with its preceding table T_{n-1}), given that table (T_n) does not have a header already within the table. In such condition, the only restriction the preceding table has to follow is that it must

Num	Scan	Fill	Bonus	Quib	EC	Total	L1	L2	Mid Term	Mid Term
102	57	27	2	0	0	86	100	91	89.80000000000001	A
105	57	19	2	0	0	78	100	88.2	84.44	B
108	42	14	0	0	0	56	100	78	69.2	D
111	63	29	2	0	3	97	100	100	98.19999999999999	A
114	63	11	2	5	0	81	100	88	86.2	B
117	72	26	2	0	3	103	100	94	100.6	A
120	57	11	0	0	0	68	0	0	40.8	D
123	54	19	2	0	0	75	100	50	75	C
126	57	19.5	2	0	0	78.5	100	98	86.7	B
129	60	29	2	0	0	91	100	100	94.6	A
132	45	21	2	0	0	68	100	91	79	C
135	57	9	2	0	0	68	100	94	79.6	C
138	51	18.5	2	0	0	71.5	100	80	78.9	C
141	39	9	2	0	0	50	100	74	64.80000000000001	D
144	66	22	2	0	0	90	100	96	93.2	A

Figure 5.2: Header Identification: Scenario-2

contain text only cells.

The algorithm used for header detection is given in Algorithm 2. Though correct header retrieval is essential for the proposed approach, given the complexity of all possible headers identification situation, it is kept limited for two scenarios.

5.2 Step-2: Pre-processing

For any knowledge discovery technique, data pre-processing is a crucial step. Indexing header information with its vector can be tricky in situations. Nonetheless, the header extraction technique used in the current method simplifies the problem. Furthermore, not all header words appear in vector labels are equally important. Choosing the right token from sentences or word segments for information extraction is a matured field in language engineering. This section explains the selected approach of pre-processing, which is integrated into the system architecture.

5.2.1 Vector-Text Mapping

Once the system identifies the headers and the tables, the conversion of header-text to vector mapping becomes quite trivial. Since the table and it's header both have the same orientation and dimension, getting the right header text for a vector can be done by index matching. For instance, consider table T_n where its orientation and header H_n is given. If T_n is a horizontally oriented table where the observation is stated row-wise, header H_n is either a single or a couple of rows. In such a table, properties of observation are stated column-wise; thus, its label resides in the corresponding column of the header. As a matter of fact, the process generates column-wise vector from the table and tags the header text as a pair $\langle v_i,t_i \rangle$ for all $v_i \in \boldsymbol{V}$ where t_i is the found header text for v_i. In the same manner, vector-text mapping is done for vertically oriented tables and its row-wise vectors.

5.2.2 Feature Extraction

Processing raw text without complete human intervention is difficult. Most of the time, essential words are infrequent in a large text document, and makes it harder to find relevance, based on number of appearances. Moreover, it is common for words to look completely different but mean almost the same thing; whereas the same words in a different order convey a completely different meaning. For some languages, a simple task, i.e., extracting text in meaningful word-like units are often troublesome. Though it is viable to resolve some issues using the raw texts in an exact format, it is preferred to use the linguistic feature to capture useful information from text. This project follows the same principle of using linguistic property annotation to obtain useful words from text using a dependency parse tree.

In text analysis, a dependency parser carries out the sentence boundary detection and allows to iterate over token based on their dependency on the root word of the sentence. It constructs a tree describing a path from sentence root to any other words. Consequently, a word (t_1) that dependents on some other word (t_2), then on the tree t_2 is said to be the parent of t_1. In this manner, it is possible to identify words or chunks of words that are dependent grammatically. Though most of the time, the 'main verb' of the sentence stands as a root node of the dependency tree, for topic or context identification 'Noun phrases' or 'Noun chunks' place the prominent role.

For the task of spreadsheet formula learning, the system is not interested in the noun phrase itself, but any modifier that sheds more context on the noun. From a relational database point of view, a table that stores department-wise employee information for institutions where individual column stores the employee number of the respective departments as of its title. The entity that represents the cumulative information about employee number is usually labelled as total employees, all employees, and so. It is rational to consider such labels as a structure 'modifier + noun phrase'. For this reason, the system implementation has a rule-base learner to extract 'modifiers' from text labels or the 'base word' (root) itself when no modifier exists for the base word. The implementation of this feature learner is shown using a rule-based tree shown in Figure 5.3. This thesis also verifies the benefit of feature extraction using such learner comparing with a no-learner baseline (result shown in Table 6.1).

The algorithm constructs a useful feature extraction to serve the problem objective and navigates through the dependency parse tree for each header text in t retrieved in previous sub-section. Each header text t may contain one or more words in it. Using a dependency parse tree and POS tagger, the program annotates each word or token (w) in t. Using the rule-based learner, it examines dependency of each token $w \subseteq t$ with its root within t and accepts the token as the acquired feature that satisfies by the mentioned rules in Figure 5.3. The feature learner first checks whether w is an adjectival modifier that serves to modify the meaning of the NP [13]. If not, the condition is checked if w is a compound modifier and tagged as 'adjective' or 'determiner'. Lastly, when none of these rules succeeds, the system accepts the base word or 'root' word as a feature.

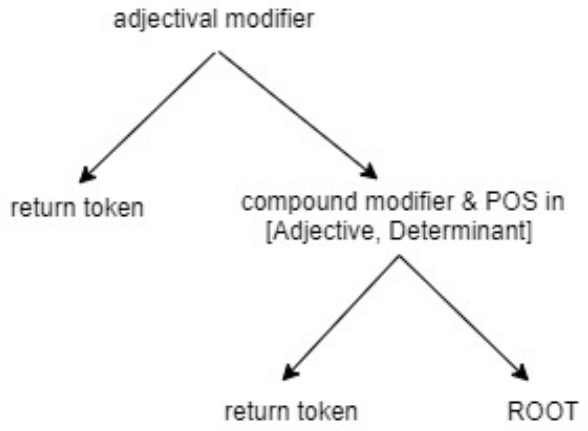


Figure 5.3: Rule based learner for feature extraction

5.3 Step-3: Constraints Learning

Learning formula's using contextual information given on the spreadsheet is the key contribution of the current research. Recall from Section 3.2, one of the existing methods of learning formulas on a spreadsheet is introduced by TaCLe [27] where the author applied constraint solver on blocks retrieved from the table. The tool itself is efficient enough to identify all the existing constraints in the tabular data. However, to do so, they have to reason over all the possible blocks that satisfy the constraint signature. The present approach is going to adopt the same concept from TaCLe, but instead of reasoning over all the blocks, it is going to make some heuristic predictions and eliminate some inconsistent vector's from search space and prioritise constraints for validation. The overview of the constraint learning algorithm is given in Algorithm 3

In this stage, the overall task of learning constraint can be seen as a job for two process learners. The first one is the main learner, who is responsible for reconstructing formulas on the complete file. The main learner accepts learned Tables($\boldsymbol{T}$), Header($\boldsymbol{H}$), supported constraints $\boldsymbol{C}$ and maximal blocks($\boldsymbol{\beta}$) as input argument and produce learned constraints($\boldsymbol{S}$) as output. Though the system uses the concept of blocks (maximally contiguous type-consistent vectors) for final constraint validation, initially it focuses on vectors to identify likely constraints. From all the table in $\boldsymbol{T}$, it generates vector $\boldsymbol{V}$ where all $v \in \boldsymbol{V}$ and tags it with their corresponding header word as mentioned in Section 5.2.1. Then, the learner aims to find tabular constraint by enumerating over each vector v. The main learner sub-assigns this task of learning constraint for vector v to another procedure and later collects the found constraint on v.

The second one is the constraint learner for a particular vector, denoted here as LEARNCONSTRAINT. It designates v as the target variable for each constraint $c \in \boldsymbol{C}$. To elaborate more on that, the system considers v as the target variable assignment and all generated blocks along with its sub-blocks as rest of the constraint

argument (data variable arguments). The benefit is once the learner reaches a constraint solution for v, the system checks no other constraint and generates a maximum one constraint per vector. Before applying constraint validation, the system applies the vector through two predictive models where the first one (ϕ) predicts the possibility of having 'any' constraint and the second one (ψ) ranks all constraint ($c \in \boldsymbol{C}$). The detail process of these two predictive models is given in the following sub-sections.

Algorithm 3: Constraint Learning using Header Information

input : $\boldsymbol{T}$- Tabular data, $\boldsymbol{H}$- Headers, $\boldsymbol{C}$-Constraint List, $\boldsymbol{\beta}$- Maximal Blocks
output : $\boldsymbol{S}$- Learned Constraint List
Algorithm `Learner()`
  for $T_n \in \boldsymbol{T}$ **do**
    for *each orientation of* T_n **do**
      $\boldsymbol{V_n} \leftarrow GENERATE(\mathrm{T}_n)$
      for $v_i \in \boldsymbol{V_n}$ **do**
        $\boldsymbol{S} \leftarrow \mathbf{S} \cup$ LEARNCONSTRAINTS$(tuple(v_i, w_i), \boldsymbol{C})$
      end
    end
  end
  return $\boldsymbol{S}$
Procedure `LEARNCONSTRAINTS`$(tuple(v_i, w_i), \boldsymbol{C})$
  if $\phi(w_i) = 0$ **then**
    return; ▷ `null constraint for vector` v_i
  end
  else
    $\boldsymbol{C} \leftarrow \psi(w_i, \boldsymbol{C})$; ▷ `Rank constraint set` $\boldsymbol{C}$ `based on` w_i
    for $c \in \boldsymbol{C}$ **do**
      if *VALIDATION*$(v_i, c) \neq$ None **then**
        $S \leftarrow$ VALIDATION(v_i, c, β)
        return S; ▷ `valid constraint found`
      end
    end
  end

5.3.1 Predict Constraint Existence

The purpose of the initial algorithm, applied on a vector v_i given the header word w_i, is to learn the possibility of 'having any constraint'. Based on the w_i , a predictive model will try to forecast whether vector v_i can be a target variable for any of the constraint $c \in \boldsymbol{C}$. The analogy behind this algorithm is when a meaningful header exists for a given table, the header label for any row or column will indicate what the vector in question contains. If a vector holds the result of a mathematical function

over other vectors, it is natural to expect hint from the vector's header label. This phenomenon is used to shrink the search space for constraint learner by pruning some vectors whose label suggests otherwise.

The approach considers a null constraint category(c_0) within $\boldsymbol{C}$ as a possible constraint label. The acceptable value for $c_0 \leftarrow [0, 1]$ which makes the problem of assigning value to c_0 a binary classification problem. If constraint existence learner function predicts the value '1' as the value for c_0, one can say that the vector v_i belongs to null constraint category which means the vector v_i is not likely to be a target variable for any of the supported constraints. Since the system is more interested in predicting a constraint category right than identifying not having a constraint, it trains a binary classifier $\phi(w_i)$ for the complement of c_0 ($\overline{c_0}$).

To train such a function $\phi(w_i)$, all generated vector from the spreadsheet data are labelled; each (v, w) pair with '0' or '1' based on the formula existence. By constructing a variant of TaCle, the system collects all the vector header and automatically annotate them with class labels (0 or 1). It assigns class label '1' ($\overline{c_0} = 1$) to those vector who hold the resultant value of any constraint found in TaCLe; otherwise annotate with label 0 ($\overline{c} = 0$). Systematic retrieval of the headers and indexing with vectors is done as mentioned in Section 5.1 and 5.2.1. The collected header labels are often 'chunk' of words rather than a single word. The system makes use a rule-based learner (mentioned in Section 5.2.2) to select an influential word for constraint existence prediction to train a model. Next, we convert the selected header word into a word vector using word2vec algorithm [30] to extract the semantic meaning of it. To construct a predictive model, a Decision Tree Classifier is trained using Python Scikit-learn library (version 0.23.1) for a maximum tree depth 4 and using maximum feature size 100. More specification of model training and the motivation behind choosing this particular model is given in Section 6.2.2.

5.3.2 Rank Constraints

Based on the trained model from seen examples, the system predicts the likability of each vector of being on the null constraint class. If the function $\phi(w_i)$ assigns a negative label $\overline{c_0}$ ($\overline{c_0} = 0$) on v_i from the current datasheet, then the vector disqualifies of being a target variable for any constraint and thus no further search for the constraint is required for v_i. If function $\phi(w_i)$ suggests otherwise, the system ranks the constraint list $\boldsymbol{C}$ depending on the similarity score between the category name and the header word. Such a ranking algorithm sort $\boldsymbol{C}$ in descending similarity score to place the best matches on top and thus helps the constraint validation step to find satisfiable constraint faster.

This similarity between each word and constraint name is measured based on their position on a high dimensional space. The system implementation uses word2vec based similarity score offered by Python library Spacy, as this algorithm claims to have state of the art solution for capturing semantic similarity among words [30]. The underlying mechanism behind the conversion of words into vectors is to group semantically similar words together and to achieve that the words end up in high dimensional space with their vector co-ordinates. In the semantic similarity learning

context, word2vec reaches high accuracy when it is fitted with the right amount of data. For this project, due to the limitation of genre-specific spreadsheets, the vector conversion is done based on an already fitted natural language dictionary 'en_core_web_lg', offered by Spacy (version 2.3).

After training this NLP model with sufficient data, it can produce a 'word embedding' once it receives any word: a multi-dimensional meaning representation of a particular word. This high-dimensional representation helps similar topic word cluster together and thus hold a higher similarity score in between. The similarity score of word vectors can be measured using vector computation. The proposed application used cosine similarity to calculate relatedness between header word and constraint template names. Let us consider a frequently occurred header word in our dataset, 'total'. The cosine similarity of the words, i.e., Cumulative, Difference, and Product with 'total' are 0.5273877, 0.46938312, 0.2831078 respectively; though these similarity scores are subject to the training data of the NLP network. In this paper, the similarity score is the representative of how two related two words appear in natural language text. Though word embedding construction on spreadsheet dataset was more appropriate, this step is skipped for now for the sake of the simplicity of the current project.

However, training on an existing dictionary of natural language does provide with a reasonable guess to capture the meaning of header word (evaluated on 6.1.1) for constraint ranking based on similarity. Taking into account that ranking constraint is not a 'hard categorisation' problem. In scenarios where the ranking algorithm fails to make the best possible prediction on the dataset, it will end up finding the satisfying constraint by iterative inspection in the constraint list. The concern remains in such a context is how faster a constraint validation mechanism will reach to the solution.

5.3.3 Constraint Validation

For constraint validation, the current paper borrowed approach from TaCLe with an only exception of pre-assigning target variable before any other constraint variable assignment. While iterating over the vector of the table for constraint validation, each vector v is placed as a target variable for all $c \in \boldsymbol{C}$. First, the system assesses v's compatibility as target variable based on the constraint signature and then look for other argument variables whose resultant value v may contain, upon applying the formula described by any constraint $c \in \boldsymbol{C}$. Accept from the target variable, maximal blocks(β) from all tables are considered as an input variable for constraint argument. The algorithm for constraint validation is stated in Algorithm 4.

Once the vector v is placed as the target variable of constraint c, the system conducts a restriction feasibility examination based on the signature of template c to access whether vector v meets the criteria to be a target variable of c. If this condition does not hold, the algorithm move on to the next constraint. On an opposite scenario when v meets the restriction imposed on the target variable of c, the system then assigns v as the target variable of c. Having this partial assignment, the task is to find a suitable variable(s) that matches with input variable restriction of c. Like TaCLe, this application also uses a constraint solver for this task where

Algorithm 4: Constraint Validation

input : v- Target Variable, $\boldsymbol{\beta}$- Maximal Blocks, c-Constraint
output : S- Satisfied constraint
Procedure `VALIDATION`$(v, \boldsymbol{\beta}, c)$

```
if c.target_variable ≠ v then
    return None
end
if assign (v, β, c) = None then
    return None
end
if solution (v, β, c) = None then
    return None
else
    S ← solution(v, β, c)
    return S
end
```

the input variable is the maximal blocks along with all sub-blocks combinations in it. Upon generating each possible sub-block, the system checks the satisfiability of the signature restriction for c. In the end, the system returns all sub-blocks that are considered as a valid candidate by the constraint solver.

If no valid input argument is found for constraint c, then the system returns empty block assignment and move on to the next constraint. In case, there is a valid input block(s) for constraint c. The system is going to assess each valid argument logically whether they hold constraint c among the input block(s) and target variable v. Upon a positive response from this last assessment, the system returns a learned constraint to LEARNCONSTRAINT procedure to include the learned constraint with its assigned variables and ultimately end the constraint search for vector v. If the input block does not satisfy the constraint logically, then the next block is taken for validation until there is no valid block left.

Supported Constraints

The proposed system currently supports only aggregate spreadsheet formulas, i.e., SUM, PRODUCT, MAX, MIN, COUNT and AVERAGE. This choice of constraints is rational since aggregate formulas are the most used formulas in a corporate setting. Moreover, in two databases of spreadsheet for scientific use, Enron and EUSUS, contain a large number of cell with data value from aggregate operation [26]. Hence, the system is designed in such a way that it supports well these formulas' constraint representation. These formulas can be described using constraint templates as shown in Table 5.1.

The constraint template representation in Table 5.1 of formula SUM shows the both orientation horizontal and vertical. This template also supports PRODUCT,

Syntax	Signature	Definition
$B_2 = SUM_{row}\boldsymbol{B_1}$	B_2 and $\boldsymbol{B_1}$ are numeric columns($\boldsymbol{B_1}$) ≤ 2 rows($\boldsymbol{B_1}$) = length(B_2)	$B_2[i] = \sum_{j=1}^{columns(\boldsymbol{B_1})} row(i, \boldsymbol{B_1})[j]$
$B_2 = SUM_{col}\boldsymbol{B_1}$	B_2 and $\boldsymbol{B_1}$ are numeric rows($\boldsymbol{B_1}$) ≤ 2 columns($\boldsymbol{B_1}$) = length(B_2)	$B_2[i] = \sum_{j=1}^{rows(\boldsymbol{B_1})} column(i, \boldsymbol{B_1})[j]$

Table 5.1: Constraint Template representation of formula SUM [27]

MAX, MIN, COUNT and AVERAGE. The only variation has to make in definition based on the operation. In the syntax of the template, the right-hand-side argument is the target vector(B_1) of the constraint. The left-hand-side argument ($\boldsymbol{B_1}$), stated in bold, indicates more that one vector, namely a block. Signature specifies the rule for $\boldsymbol{B_1}$ and B_2. Finally, the definition states the operation itself that the data value has to match.

This work is expanded to all 33 constraints supported by TaCLe in paper [27]. The unsupported constraint templates by the current system, but supported by TaCLe, are mostly look-up operations or conditional aggregate operations. For conditional aggregate constraints, the semantic approach given in this paper would be almost the same as the supported aggregate constraints, except the system has to separate conditional aggregate and aggregate operation from header text if such information exists. Another potential solution is to place a semantic dependency between any aggregate operation and its corresponding conditional aggregate operations. For look-up operation, mapping one vector to another would be more trivial if the system can reveal the naming convention for foreign-key operations. However, all these propositions are subject to further research.

5.4 Conclusion

This chapter exhibits the implementation of the proposed system architecture in an explicit manner. The entire process is composed of three steps, namely data extraction, data processing and constraint learning. The first two steps are described concisely as these steps are only pre-requisite of the main solution. The final step is constructed elaborately and emphasised more than the other two steps based on its research contribution. The following chapter will present a detailed analysis of each decision choice made during the process.

Chapter 6

Evaluation

The main objective of this paper is to rediscover formulas on tabular data with minimal effort. To achieve that, it aims to improve an existing solution (TaCLe) by making it faster and with less false-positive formula. The system architecture proposed in this paper is based on two baseline assumption. (1) Tabular constraint learner spends a significant amount of time on vectors where no constraint exists. (2) Header label can provide a reasonable estimation to find the right constraint for vectors. Through a controlled study, this chapter aims to validate these baseline assumptions. This test experimentally verifies these propositions and the system performance by answering the three main question given below:

Q1: Ranking constraints based on vector's header label may help to find the valid constraint faster. Is this proposition true? Does the search space ordering based on similarity rank can improve the constraint search?

Q2: For the majority of the vector, the algorithm makes an effort to find a constraint on vectors where no constraint exists. Does the introduction of a pre-pruning strategy on vectors improve the algorithm?

Q3: Including semantic heuristic in TaCLe, the proposed algorithm attempts to improve TaCLe. Does this algorithm manage to enhance TaCLe?

By finding answers to the question as mentioned above on a limited dataset, this paper attempts to establish the feasibility of the proposed approach. To structure this chapter, these questions are answered in three different sections. Each section is further designed to explore sub-questions that demands to be addressed to reach a conclusion.

Dataset

Tabular data used in the experiments are in CSV format where tables are listed along with its header and other descriptive text. For experimentation, this chapter considers only those CSV files where the table header can be detected based on the two scenarios described in section 5.1. The tables and its formula is extracted

by running the software, TaCLe. Then, using the header extraction approach, described in section 5.1 and 5.2.1, each formula is mapped to its header text. Later, manual intervention on these automatically detected word-formula pairs is required to annotate the false-positive occurrence generated by TaCLe.

The purpose of the manual intervention is to set up the ground truth to measure application performance and to train predictive machine learning models. When there are multiple 'valid' constraints for a given vector, TaCLe ends up finding all of them and struggles to decide which one of them is the intended constraint in the original datasheet. For example, a frequent case of false-positive constraint is the formula 'PRODUCT' on a zero (0) valued cell. Mathematically, it can be established as the resultant cell for any other cell that is multiplied by another (zero) where in real it was a missing value cell. Likewise, when an aggregate formula, i.e., SUM is detected over a block of vectors, the target variable is logically the maximum valued vector over the same block of vectors. In such cases only, we manually annotated the false-positive occurrence based on the header words and the location in the spreadsheet.

For experimentation, the system evaluation considers both real spreadsheet from Euses dataset[18] and synthetic data the same one as TaCLe used to validate its performance [27]. These synthetic data are not randomly generated tabular data, but artificially created by spreadsheet professionals for tutorials. Hence, it is reasonable to consider these datasheets as an authentic representation of the spreadsheet community. For experimentation and observation, the system runs our code on 10 real datasets and on 10 synthetic datasheets, which remain consistent across experiments. More detail experiment specific setup and data selection will be presented in the respective sub-sections.

6.1 Effectiveness of Constraint Ranking

This section assesses the effectiveness of the implemented constraint list ranking and thus attempts to address Q1. To reach an answer for Q1, this paper would like to answer a few more questions that can be considered as the sub-questions of Q1. The questions are given below:

Q1(a): Does the constraint list ranking manage to provide meaningful information for constraint search?

Q1(b): Is it possible to find the correct formula only by calculating semantic similarity between header word and formula name?

Q1(c): Does constraint ranking influence run-time of the algorithm based on similarity score?

The following text will answer these questions in three sub-sections and will try to draw a conclusion on Q1.

6.1.1 Heuristic Based Search Space Improvement

The application accepts the extracted feature from header text for constraint ranking of each vector. Moreover, the system substitutes the constraint name with a more semantically significant word to describe the constraint function, as the purpose is to capture semantic similarity. This sub-section evaluates this design choice for constraint ranking. To empirically validate this approach, this sub-section compares and contrasts among the constraint rank quality implemented in three different ways: 'without constraint name substitution', 'without feature extraction' and 'with the final approach'.

Experimental Setup:

TaCLe considered 33 constraints for its construction. Therefore, it did not cover all the possible formulas of spreadsheets. In paper [27], such constraints are denoted as **supported constraints** among which some of them were a non-functional constraint, only to support other constraint findings. The current sub-section did not cover all the supported constraints used by TaCLe. It considers only the constraints which can be found in the chosen spreadsheets for semantic analysis.

This experiment investigates the semantic significance of constraint ranking based on the header word. First, the system creates a mapping dictionary where each constraint is mapped against the list of words, occurred as its target variable header in the selected spreadsheets (given in Appendix A). Upon constructing a word base, constraint ranking for each word is conducted in three different ways. This sub-section introduces a scoring system to understand the performance of the ranking, which is called **semantic score**. The score indicates how well words, assigned to a particular constraint, estimates the correct constraint. The semantic score can be described as:

$$score_{c_j} = \frac{\sum_{w \Rightarrow c_j} Rank_w(c_j)}{\|j\|} \tag{6.1}$$

where c is the j^{th} constraint in the list, w is a word that belongs to the word list for c and $Rank$ is a function that returns the constraint position in the sorted constraints list for word w. Then, the score has been normalized by dividing it by the number of supported constraints on a scale of 0 to 1 where 0 being the best score and 1 being the worst. The score indicates that the predictability of constraint c given it's seen header word.

Result:

Table 6.1 shows the result of the experiment. The first two column of the table state the constraint name along with its semantic substitution. The last three columns of the table mention the semantic score when constraint ranking is done in three different approaches.

The third column of the table lists the semantic score for each constraint when the ranking of the constraint set is done without any semantic substitution or feature

extraction. In this case, each word listed under all constraints try to rank the constraint list C based on their semantic with the constraint names. Then each constraint is scored using Equation 6.1. A closer observation to the column score reveals that the system is capturing some information from semantic similarity, as most of the constraints' similarity score is less than 0.5. Note that, the score scale is from 0 to 1 where 0 being the best. In other words, it can be said that the ranked list seems to be slightly better than a random guess in most cases. Such a conclusion can be re-established from the 'overall' average score for all constraint, mentioned in the last row.

Though the naive similarity ranking performs sufficiently for some constraints, yet fails to do so in many cases. This is also reasonable since not all the formula constraints are self-explanatory by its name, stated in the constraint template. Moreover, the mention of orientation and conditional information does not add any value for semantic similarity learning. Next, the system conducts the same experiment only by substituting a well-describing word for the formula constraints. From the scores, stated in the fourth column of Table 6.1, one can observe a significant improvement in ranking score. The system manages to make an noteworthy estimation for the assigned constraint, except for two constraints.

Lastly, the system tries to improve the ranking score by implementing feature extraction (mentioned in 5.2.2) on the header text. The contrast between the fourth and the fifth column of Table 6.1 shows rank score improvement for almost every column. A simple proposition on this fact is when a spreadsheet practitioner labels a row or a column, s/he not only places the formula information but also refers to the formula entity name. Though the entity information is interesting for relational formulas, it does not contribute much for constraint categorization; thus discarded in the current context. The final strategy with 'constraint name substitution' and 'important feature extraction' makes the best-ranked list among these three approaches. However, the ranking strategy failed to provide the desired estimation for 'product' and 'sum-if' constraint templates.

Q1(a): The proposed constraint ranking strategy manages to produce the best score in comparison to the other two approaches. Moreover, the overall similarity score for this approach is 0.218 which is a substantial outcome on a scale of 0 to 1 where 0 being the best. This sub-section can conclude that the intended constraint ranking does provide meaningful information to identify constraints.

Q1(b): Though the average rank score is significant enough to improve search space, but it is not a perfect score (0). Therefore, the semantic scores of each constraint project that only sorting constraint based on word similarity is not enough to find the exact desired constraint.

To summarize, the proposed constraint ranking strategy failed to produce a perfect score to predict the right constraint. However, it is adequate to adopt this approach for further constraint validation. It is logical to believe that having the right constraint at the top of the search list may improve the constraint search

Constraint Name	Substituted Word	Semantic Score (a) Naive Ranking	(b) Substituted Ranking	(c) Final Ranking
sum (col)	cumulative	0.596	0.231	0.163
difference	difference	0.31	0.282	0.224
sum (row)	cumulative	0.29	0.306	0.188
max (row)	maximum	0.5	0.176	0.176
min (row)	minimum	0.8	0.235	0.118
product	multiplication	0.59	0.682	0.671
sum-product	cumulative	0.1	0.471	0.118
rank	rank	0.175	0.147	0.147
foreign-key	foreign key	0.4	0.147	0.147
lookup	look up	0.425	0.059	0.059
running-total	cumulative	0.05	0.118	0.118
series	serial	1.0	0.118	0.118
sum-if	conditional summation	0.25	0.618	0.618
max-if	conditional maximum	0.3	0.235	0.353
max (col)	maximum	0.583	0.255	0.157
min (col)	minimum	0.5	0.147	0.235
average (col)	average	0.212	0.074	0.103
Overall		0.439	0.253	0.218

Table 6.1: Comparison Table of Semantic Score among Naive Ranking (without word substitution and feature extraction), Substitution Ranking (with constraint name substitution), and Final Ranking (with feature extraction)

experience. The next topic will discuss to what extent this strategy indeed improves constraint learning performance.

6.1.2 Influence in Run-time

To measure the influence of ranking in run-time, the system integrates the ranked constraint list proposition in tabular constraint learning. First, for each vector, it sorts the templates based on header word similarity and then it applies constraint validation approach from TaCLe. The current process iterates over individual vector of tabular data and make partial assignment by considering current vector as target variable for each supported constraint. Unlike TaCLe, instead of, considering all maximal blocks and sub-block combination for all the constraint arguments, the current approach first assigns the target block and then validates other argument variables. Next, it searches for a satisfiable constraint over the vector and moves to the next vector once any feasible constraint is found or reaches at the end of the constraint list. The approach can be described using the following pseudo-code:

For each vector v:

Step-1: Get the header word w for vector v

Step-2: Rank the constraint list $\boldsymbol{C}$ based on the similarity score for (w, c) for $c \in \boldsymbol{C}$

Step-3: Run TaCLe considering c as a target variable for each c

Step-4: Stop when the first satisfied constraint is found for v or end of $\boldsymbol{C}$

This implementation is the same as Algorithm 3, except the conditional block for null constraint prediction in LEARNCONSTRAINTS function. This experiment is established without the predictive classifier to asses the performance of similarity ranking on TaCLe without any other influence.

Set-up

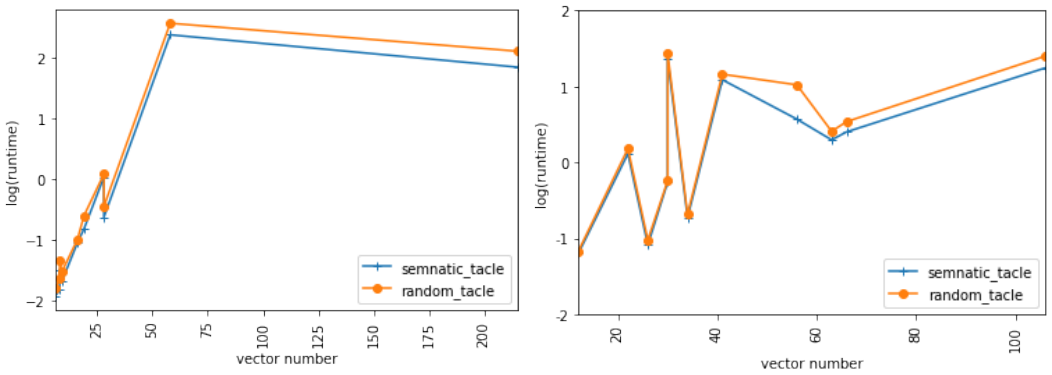
TaCLe has implemented a concept of dependency graph among constraints to aid constraint propagation. For this reason, the current suggested approach comply only with root-level constraints from TaCLe's hierarchical dependency tree. Therefore, for this experiment, the system only considered eight constraints, i.e., EQUAL, PROJECT, SUM, MAX, MIN, COUNT, AVERAGE, PRODUCT. Two of these constraints are non-functional constraints (EQUAL, PROJECT) and thus excluded from the findings report.

This paper includes a no-semantic baseline to compare with the proposed ranking approach. This baseline implementation follows a similar approach for constraint learning as stated in this sub-section. However, it does not rank the constraint list based on the similarity score, rather assigns a score randomly and sorts based on that. Therefore, the only difference between these two ranking approaches is the presence of semantic information. And thus the effect of having such heuristic in finding constraints can be measured.

All experiments were run using Python 3.8.3 on a Windows 10 operating system with Intel Core i7 1.99GHz processor speed on a 16GM RAM embedded machine. The similarity score measured in Step-2 is created by word embedding similarity score, offered by Python library Spacy. Each datasheet was computed 10 times and reported the average run-time.

Result:

The result of this experimentation is stated in Figure 6.1. The Figure 6.1a reports the logarithm of average run-time (in seconds) required for each spreadsheet from synthetic dataset. Each file's run-time is sorted according to the vector number and plotted in the figure. The graphical representation reveals the computational time differences between semantic ranking approach and random ranking approach when the vector number is sufficiently large (more than fifty). However, the time difference remains almost the same except a few decimal points (can be observed in Table 6.2) for spreadsheets whose vector number are comparatively smaller.



(a) Synthetic Spreadsheet

(b) Real Spreadsheets

Figure 6.1: Log plots of computation-time for ranked constraints implementation and baseline randomly ranked constraint implementation

Figure 6.1b conveys the same information as of Figure 6.1a, but for real spreadsheets. The figure states average run-time in logarithmic form over Y-axis and vector number for spreadsheet in X-axis. For this dataset, run-time report also exposes the same trend as synthetic dataset. Even from the real spreadsheet which contains everyday header words by people, the system is able to produce a reasonable estimation by sorting constraints. As a result, it ends up reducing overall computational time (see Table 6.3).

Q1(c): From the observation for both real and synthetic spreadsheet, the experiment result suggests that semantic information based constraint ranking influences algorithm run-time when the vector size is greater than 50.

Q1: From the empirical observation of sub-question (a), (b), and (c), this section can conclude that semantic similarity has the potential to sort the constraint in a useful manner. Besides, the run-time comparison experiment also shows positive bias towards semantic ranking approach, though the difference in computation time is very limited.

6.2 Effectiveness of Constraint Existence Prediction

From the previous experiment, the computation time improvement can be observed in fractional seconds between semantically ranked approach and randomly ranked approach. Moreover, time improvement is hardly observed unless a large number of vector exists in the spreadsheet. An assumption was made in favour of the ranking approach, which is applicable if the satisfied constraint is found over a vector. When there is a constraint located over a vector, placing that constraint on top of the list will ultimately diminish the overall computation time. In contrast, the situation is different when there is no constraint for most of the vectors. In this sub-section,

File name	Vector Number	Run-time in seconds	
		Semantic Ranking	Random Ranking
sum_table_from_guide	6	0.145	0.166
fruits	8	0.161	0.192
week_2_busn_store_2	8	0.223	0.263
magic_ice_cream	9	0.185	0.217
school	16	0.347	0.367
inventory	19	0.438	0.545
demo	28	1.03	1.081
example	28	0.532	0.636
score	58	10.734	12.988
fy92	215	6.3	8.194
Overall		2.0095	2.4649

Table 6.2: Computations Time for Synthetic Spreadsheets with ranked constraint implementation

File name	Vector Number	Run-time in seconds	
		Semantic Ranking	Random Ranking
134 document	12	0.301	0.31
030902	22	1.123	1.2
accessiblearchives	26	0.337	0.36
031001	30	0.772	0.786
STT841pub	30	3.898	4.178
2000 reser School	34	0.484	0.505
Technical 20Salary 20	41	2.972	3.198
io a3 wb1 reichwja xl97	56	1.769	2.78
financial outlook sta	56	1.346	1.507
results subjectGrade	66	1.495	1.711
Financial 20Compariso	106	3.474	4.06
Overall		1.6337	1.8722

Table 6.3: Computations Time for Real Spreadsheets with ranked constraint implementation

the paper proceeds to investigate if early detection of 'any' constraint presence can improve the learning algorithm along with the semantically ranked list. To examine this fact, the author put forward three sub-questions:

Q2(a): Is it possible to predict any constraint existence for a vector from its header word?

Q2(b): How to select a predictive model that best suits the purpose?

Q2(c): Can a supervised predictive model help constraint search by pruning candidate variables?

Dataset

To support this study, a word dataset is constructed with all the vector (both with or without having a formula) headers appears in the previously selected spreadsheets, both real and synthetic. From the header inspection, the system gathers 358 header words where 332 are not assigned to any constraint's target variable, but only 26 of them had constraints assigned for their respective vectors. Next, pre-processing of the header segments is done using POS tagging and dependency parse tree, in the same manner as mentioned earlier (in the sub-section 5.3). Finally, an annotation was made systematically by labelling constraint header words with class-1 and otherwise with class-0.

6.2.1 Predictability of Constraint from the Header Text

The fact of whether constraint existence, in general, is predictable from the header word is itself a matter of experimentation. The motivation behind this experiment is to examine how much any learning algorithm can gather information from existing observation and perform reasonably on forecasting class for unseen data. In a nutshell, the effort is to evaluate how efficient the header texts are to hint learner about formula existence. The experimentation of constraint prediction begins with training a couple of naive machine learning models on the observed constraint header words.

Experimental Setup:

Before training a predictive machine learning model, feature extraction (mentioned in 5.3) on header text and vector conversion was performed. The learning framework considers two vector conversion approaches: Word2Vec and tf-idf. Word2Vec is known for its performance on establishing semantic similarity between words. On the other hand, tf-idf is the most commonly used vector conversion approach when it comes to text information analysis.

For model training, the system divides the dataset of 358 words into train and test sets in a ratio of 70% and 30% respectively, ensuring that the given equilibrium of limited positive class remains the same. From the title, it can be inferred that the training set trains the model and test set report the result. No further validation

	Word2Vec			Tf-Idf		
	Logistic Regression	Naive Bayes	Decision Tree	Logistic Regression	Naive Bayes	Decision Tree
Accuracy	0.98	0.98	0.93	0.96	0.44	0.99
Precision	0.88	0.80	0.5	1	0.11	0.89
Recall	0.88	1	0.88	0.5	0.88	1.0

Table 6.4: Performance metric comparison between word2vec and tf-idf

technique has been adopted, since the sole purpose of this experiment is to identify how much predictable the constraints vectors are from their textual information. The system trained the following machine learning models to report the result on test set: Logistic Regression, Naive Bayes and Decision tree.

Result:

Table 6.4 lists the trained models' performance on test data; while they were trained on vectors generated from the training dataset. All these models are trained on two different vectors, and their test results are listed alongside within the table. The scoring metric indicates better results for Word2Vec vector. Moreover, in the current context, the system is more interested in predicting positive class and therefore emphases more on higher recall and allows lower precision. On that account, this paper will proceed with Word2Vec algorithm for vector conversation for the rest of the process.

Q2(a): If one takes a closer look into the Table 6.4, the accuracy and recall for every trained models are quite significant for Word2Vec produced vector.
Thereupon, it is rationally presumable that with such vector conversion, constraint existence can be sufficiently learnt. Nonetheless, the appropriate model selection for this predictive learner is a subject to further experimentation.

6.2.2 Model Selection

This paper further continues to investigate in finding the appropriate machine learning model for constraint prediction classifier. For this experiment, the process used cross-validation (CV) and hyperparameter tuning for models, as now it is more interested in choosing the model that provides the best performance to predict formula existence on any vector. This sub-section uses a train, test and validation set to compare different algorithms, as a model validation practice. Though the comparison is made on the accuracy metric, the process is interested in getting the best recall for class-1 in our test set. Important to mention here, due to the limited number of positive class presence in our dataset, accuracy can sometimes be misleading for choosing a model; thus, we compare both accuracy and class-1 recall.

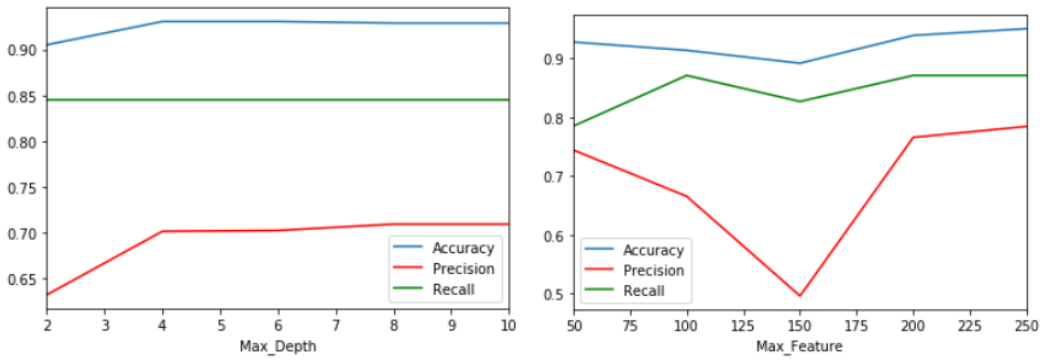
Experimental Setup:

One of the crucial factors in model training is to balance train and test dataset. The models' prediction on the test set may not be accurate if they are very different from the training set. However, in the current context, the objective of the selected model is to perform adequately in the unseen datasheet. Since word appearance varies contextually from spreadsheet to spreadsheet, it is also reasonable to validate models in a similar fashion. Instead of splitting word dataset, like previous sub-section, this experiment is going to split datasheet by datasheet to construct train, test and validation dataset. For example, if the system accepts 15 spreadsheets for model training, then it is going to consider 5 spreadsheets for validation and 5 spreadsheets for testing to make a 3:1:1 data split.

The system uses 4-Fold cross-validation to make model training more reliable. For each execution, 20 datasheets were split into four chunks where the process alternatively kept one chunk for validation and the rest three chunks for training. Then, it collects header words from the training set and annotates them with the class labels (0 and 1) based on the formula presence, similar to the previous experiment. Then, the constructed pipeline trained each model on the word vector to learn function $\phi(w)$ that predicts formula existence. This experiment only considers word2vec vector conversion, since the result of the previous sub-section suggests that the trained model on word2vec tends to perform better than tf-idf.

The system also considers training same machine learning algorithm with different parameters for hyperparameter tuning. Hyperparameters are those parameters that the model is not going to learn itself from training dataset; instead, it has to set before model parameter learning. This experiment trains and contrasts among models with different parameter but the same algorithm and also among different algorithms. The predictive functions are trained with different models, and their cross-validation scores are reported for comparison. Due to the class imbalance in the dataset, weighted importance has imposed on the models to learn positive class correctly.

Upon tuning hyperparameter and CV score comparison, the system chooses the best-learned hyperparameter for each algorithm. The system pipeline finally conducted a Leave One Out Validation (LOOV) on the dataset to choose the best algorithm among them. Applying LOOV helps to overcome the limitation of having a small dataset partially. LOOV is the best-known validation technique, though often avoidable in practice due to its high computational time. In this technique, the training set always leaves one example out of the dataset and later validate the model on that omitted example. The system repeatedly keeps doing it for each example in the dataset and end up having n-iteration; where n is the number of example in the dataset. LOOV reports on the average of n-iteration and projects the best representation of the model on a given dataset.



(a) CV score w.r.t max depth

(b) CV score w.r.t max feature

Figure 6.2: Decision Tree HyperParameter Tuning

Result

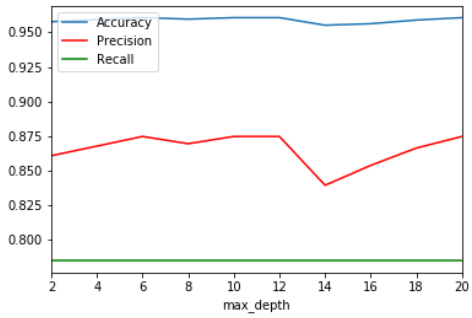
In this sub-section, the machine learning pipeline considers comparatively complex model than the previous experiment. Such a choice is made to learn an acceptable predictive classifier through the limited dataset. This section evaluates on Decision Tree (DT), Random Forest (RF), Linear Support Vector Machine (LSVM) and Kernel Support Vector Machine (RBF SVM).

Decision Tree

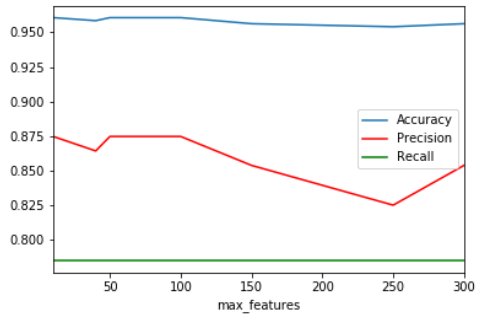
For the decision tree, the system tunes two distinct parameters of the model, which is the maximum depth limit and the maximum feature of the tree. The vector generated from header words contains 300 dimensions. Therefore, the investigation on finding the most significant dimensional feature is quite reasonable. With increasing depth and feature number, the model increases its complexity as well. Thus the system is concerned to find a combination of depth and feature number where the model fits well with the trained dataset and also performs satisfactorily on the validation set. The system examines on maximum depth [2, 4, 6, 8, 10] and maximum feature [50, 100, 150, 200, 250] in each combination. The result of the CV for increasing the maximum depth of the tree and feature number is given in Figure 6.2.

Figure 6.2a shows the cross validation score with respect to increasing depth for the tree. Different feature numbers for each depth of the tree are grouped together and their average CV score is mentioned on the figure. The same goes for Figure 6.2b where CV scores are grouped and average based on feature number.

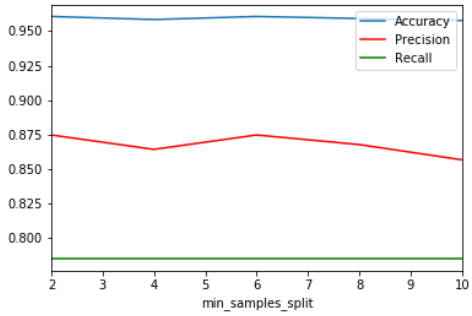
In terms of recall, feature size 100, 200, 250 provides a better score than the other two features 50 and 150. The feature-length 100, 200 and 250, all three of them perform very close in terms of predicting positive class irrespective of the depth considered for the tree. Recall score got significantly better for max feature 100, but the precision drops when the tree gets more complicated with the rise of max depth. For feature 200 and 250, recall remains the same with every combination



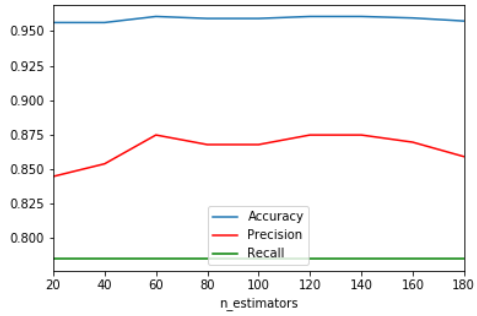
(a) Maximum Tree Depth Representation



(b) Maximum Feature Number Representation



(c) Minimum Sample Split Representation



(d) Number of tree representation

Figure 6.3: Random Forest HyperParameter Tuning

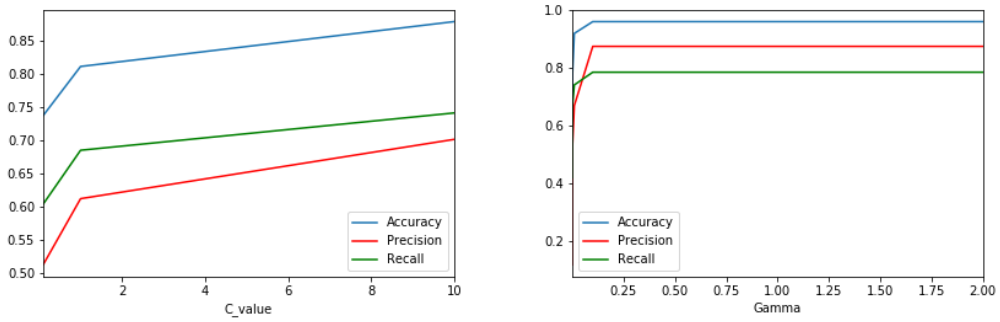
of maximum depth, and the accuracy and precision score trend also stays identical between these two feature lengths. Details of this performance metric can be found in Appendix B.1.

The system is going to investigate further on feature-length 100 and 200 with maximum tree depth 4 to find the best performing model. The feature size 250 will be ignored due to its added complexity. Considering the maximum tree depth 4 is also logical as tree performance remains almost the same on average for different features, so we accept the minimum one.

Random Forest

Tuning hyperparameter for Random Forest is more complicated than the decision tree, as it has more flexibility in parameters. Therefore, the system considers a random search through the following the hyperparameters: maximum tree depth, maximum number of features, minimum number of leaf nodes and maximum number of trees. The result, grouped by each parameter, is shown in Figure 6.3 (details in Appendix B.2).

Despite having a high-performance score in accuracy and precision, this model will not be considered further due to its complex structure for such a small dataset



(a) CV score w.r.t increasing C value (b) CV score w.r.t increasing Gamma value

Figure 6.4: Support Vector Machine HyperParameter Tuning

which often leads to overfitting. Moreover, different combinations of RF parameter does not have any effect on the positive class recall value. Another motivation for rejecting this model is its narrow decision boundary to fit the data point of the positive class, shown in Figure 6.5.

Linear Support Vector Machine

For Linear Support vector machine algorithm, the system considers two different loss functions 'hinge loss' and 'squared hinge' and tried different regularization parameter (C value). However, the Linear SVM performs constant across different combinations of these two parameters and end up having a CV accuracy 0.93, where the class-1 precision is 0.686 and recall 0.785 ((Details can be found in Appendix B.3)). Therefore, the system accepts the Scikit-learn library default parameter for linear SVM 'square hinge' and C-value 1.0 for LOOV validation.

Kernel Support Vector Machine

To examine hyperparameters for Kernel Support Vector Machine, the system considers different smoothing values (γ) for decision boundary and different regularisation (C-value) parameters. The Gamma value starts from a tiny number such as 10^{-5} goes all the way to till number 2. The system ensures to train the model with very smooth decision boundary to a very harsh one. For inverse regularisation parameter, the system tried 3 different values, C-value 0.1, 1 and 10.

The result from the hyper-tuning parameter is stated in Figure 6.4. From the figure, it is observant that smoothing does not have any effect on the CV score when gamma value is higher than 0.1. The C value graph 6.4a illustrates that SVM reaches the best result for all the considered metrics (accuracy, precision, recall) at gamma value= 0.1 (details in Appendix B.4). Though to avoid over-fitting, the next experiment will consider gamma value 0.1, 0.01, 0.001 with constant C value = 1 for LOOV validation. Though imposing less regularization (inverse C value) the performance metric improves, to keep generalization C = 1.0 value is chosen.

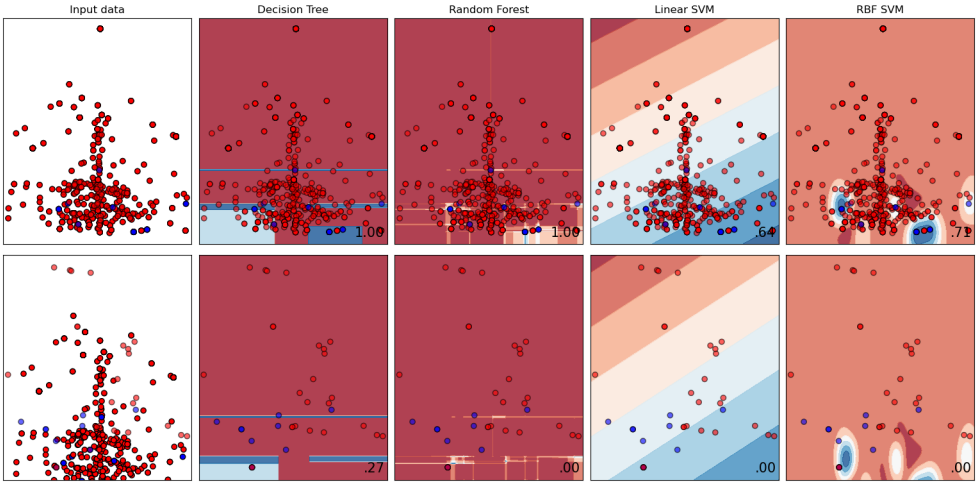


Figure 6.5: Decision Boundary for Models: Decision Tree, Random Forest, Linear SVM, and RBF SVM

The result emerged from the analysis of this sub-section helps the system to choose the best parameter for each algorithm. Finally, this paper plots the decision boundaries in a 2D space for the best-learnt parameter for each algorithm (Figure 6.5). The process converted the word vector into 2 components by applying principle component analysis (PCA) to fit data points into a 2D representation. In this 2-rowed plot, the first row shows the data point from the training set while the second row plots the test data points. For all the sub-plots the decision boundary is created using trained data. Therefore, having a closer look to the second, it is observant that how the test set data points is complying the decision boundaries which is further used to justify models. In other words, model visualisation helps to understand the decision boundary better and to comprehend on model nature in fitting data points.

Final Model

At this stage, the system conducted a LOOV validation to determine the final classifier among the best-found models from the previous sub-section. The selected models are Decision Tree(max_feature= 100, 200, max_depth=4), Linear SVM(loss= 'hinge', C=1.0) and RBF SVM(Gamma=1.0 , C=1.0). The result is stated in Table 6.5.

From the result, it is apparent that Decision Tree Classifier with parameter max_depth 4 and max_feature set to 100 is the best-performed algorithm so far. Therefore, this paper accepts this model for system integration.

Q2(b): To answer this question, this sub-section begins with hyperparameter tuning of 4 different classification algorithms. The experiment pipeline conducted a custom cross-validation technique; instead of using the built-in technique offered by Python library. The process split spreadsheets in 4-folds, rather splitting the header word examples, to remove bias between train and validation set and to project its

Classifier	Accuracy	Precision	Recall
DecisionTree(depth= 4, feature= 100)	0.927	0.755	0.907
DecisionTree(depth= 4, feature= 200)	0.889	0.59	0.815
LinearSVM(loss ='squared hinge', C=1.0)	0.918	0.672	0.79
RBF SVM(gamma=1.0, C=1.0)	0.942	0.765	0.836

Table 6.5: Leave one out validation score for different algorithm accepted with their best learn parameter

true performance on an unseen spreadsheet. In the end, this paper conducted a comprehensive LOOV validation to estimate a reliable performance of the model on the restricted dataset.

6.2.3 Improvement in Implementation

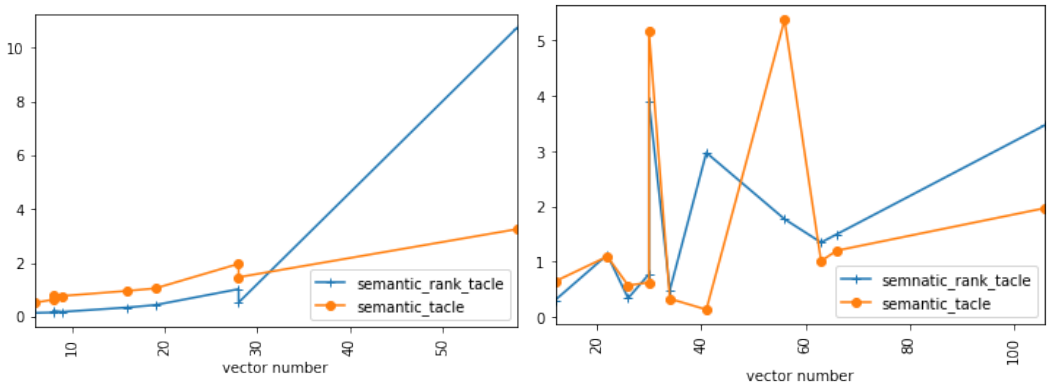
From a theoretical point of view, having a predictive classifier before constraint validation should improve the performance of the constraint learner. This predictive strategy discards vectors with less likelihood of having a constraint which is identified by the classifier. The baseline assumption is that most of the vectors in a tabular data is not a target variable of any constraint. However, the behaviour of this approach in practice is the subject of the current experimentation. This sub-section compares between, Semantic-TaCLe without classifier and Semantic-TaCLe. The aim is to measure the performance of implementing a classifier before the constraint ranking and validation.

Experimental Setup:

After discovering the best-performed model, the system integrates the classifier in its implementation (Algorithm 3). This experiment compares the final algorithm and the implementation done on experiment 6.1.2. The final algorithm consists of both semantic classifier and the constraint list ranking. Therefore, the ultimate comparison will be between the presence and absence of a constraint existence predictor. This experiment uses the same dataset as 6.1.2 and run the two different version of the proposed algorithm: without a predictive classifier and with a predictive classifier. Each of these implementations is executed 10 times, and the computational time is observed.

Result:

The Figure 6.6 shows that run-time improvement appears when the vector size grows over 30 for the synthetic dataset. Though the real dataset consists of few files where computational time for Semantic-TaCLe exceeds the without classifier version, the opposite scenario occurs in most of the cases. Moreover, from Table 6.7,



(a) Synthetic Spreadsheet

(b) Real Spreadsheets

Figure 6.6: Computation time Semantic-TaCLe without Classifier vs Semantic-TaCLe

computational time comparison between these two approaches reveals the same information.

On the other hand, the performance comparison between these two approaches in terms of precision and recall shows significant improvement (Table 6.6). For Semantic-TaCLe without classifier, there was an early stopping strategy that stops the constraint search for vector as soon as it gets a satisfiable constraint, even for the non-functional ones. The final algorithm allows constraint search after reaching the non-functional constraints and stops when it detects a functional constraint. This flexibility helps Semantic-TaCLe to reach recall as same as TaCLe and also supports to improve precision by getting all the right feasible constraints. On the downside, this flexibility could be one reason for Semantic-TaCLe's surplus of time for the real dataset. Important to mention here, Semantic-TaCLe loads its classifier from a hard-drive file which is also responsible for some additional time.

Q2(c): From recall score (in Table 6.6), this is evident that pre-pruning strategy manages to eliminate appropriate vectors, rather discarding vector that has constraints. Moreover, the intuition was to help the algorithm to get rid of the vectors that have less chance to be a target vector for any constraint and consequently learn formulas with minimal effort.

This experiment compares two implementations of with or without having a predictive classifier for class existence. For performance comparison, this experiment takes both computational time and confusion metric into account. In the chosen dataset, this is not obvious whether it can save time for the algorithm or not, due to its limited size. This section will conclude by answering question Q2.

Q2: The observations from this section show improvement in precision and recall than the previous approach. However, it needs further validation on a larger dataset to reveal the significance of having a pre-search classifier with respect to run-time, though the current experiment shows some signs of improvement in performance.

6.3 Effectiveness of Proposed Approach

This section evaluates the ultimate objective of this project which is if the proposed algorithm is sufficient to enhance TaCLe. To measure the improvement, the following text will compare performance from two different perspective. First, it will measure whether the current approach can help TaCLe to reduce false-positive. Second, it will compare whether there is improvement in run-time.

6.3.1 Influence in Precision and Recall

Experimental Setup:

The first comparison is made to understand performance contrast in terms of precision and recall. The system selects 5 unseen spreadsheets both synthetically generated or collected from existing database Euses. On this unseen datasheet, the system pipeline applied all three variants of the algorithm: TaCLe-the baseline algorithm, Semantic-TaCLe without classifier-semantic ranking in original TaCLe environment, and Semantic TaCLe-the final proposed algorithm with both predictive learner and ranked constraints.

To measure precision and recall of the new test set, setting up the ground truth was important. Upon manual annotation of the intended formula, the system is going to measure the confusion matrix by each algorithm and represent the result.

Result:

Table 6.6 shows a comparison of the three mentioned algorithms. Kolb et al. [27] demonstrates that the algorithm TaCLe has a perfect recall on the tested spreadsheet, though it suffers from limited precision. Semantic-TaCLe without classifier tries to recreate the performance of TaCLe, but fall a little short in both precision and recall. Due to the algorithm's early stopping strategy, it is reasonable not to have perfect recall, though it used TaCLe as an underlying process. In this algorithm, the system stops learning constraint for a vector when any formula over that vector is discovered. If the algorithm finds a false-positive constraint for a specific vector, it will never reach the intended formula for that vector using 'Semantic-TaCLe without a classifier' algorithm. Consequently, this version of algorithm suffers from limited precision and sometimes with limited recall too, unless it has an accurately ranked constraint list.

On the other hand, Semantic-TaCLe provides better precision and recall than the original TaCLe on the selected test set. The current algorithm considers a little tolerance for checking another constraint even if a constraint is found on a particular vector, thus reaches the same performance as TaCLe in terms of recall. Furthermore, due to the early prediction of constraint existence and constraints' prioritisation support to get rid of false positives; thus, it ends up with higher precision than its baseline algorithm.

	Precision	Recall
TaCLe	0.857	1.0
Semantic-TaCLe without classifier	0.846	0.917
Semantic-TaCLe	0.923	1.0

Table 6.6: Performance metric of TaCLe, Semantic-TaCLe without classifier and Semantic-TaCLe in terms of precision and recall.

6.3.2 Influence in Run-time

This part of the paper investigates the time performance difference between the final algorithm of this paper semantic-TaCLe and its foundation algorithm TaCLe. When there are γ positively predicted vectors in tabular data for constraint existence, the system requires to run TaCLe for γ times. However, constraint ranking and partial assignment save some effort of running the complete TaCLe. In the worst-case scenario, if there are α vectors in tabular data and all of them predicted positively for having a constraint, the computational time end up as $O(tacle\_runtime^{\alpha})$. However, this situation is highly unlikely in practice.

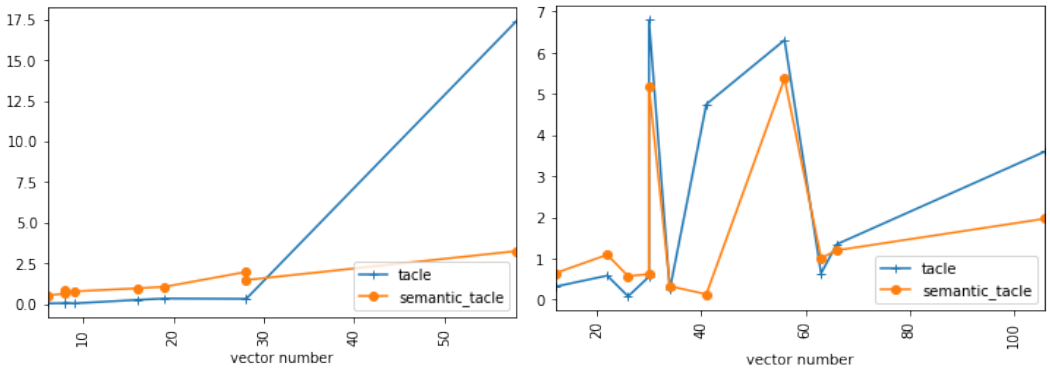
Experimental Setup:

By setting up a computational time performance test, this paper assesses Semantic-TaCLe's run-time in practice. For each dataset, both TaCLe and Semantic-TaCLe were executed using the same machine as mentioned in sub-section 6.1.2. The dataset used in this experiment is the same as all other experiments: Synthetic Spreadsheet, Real Spreadsheet and Test Spreadsheet.

Result:

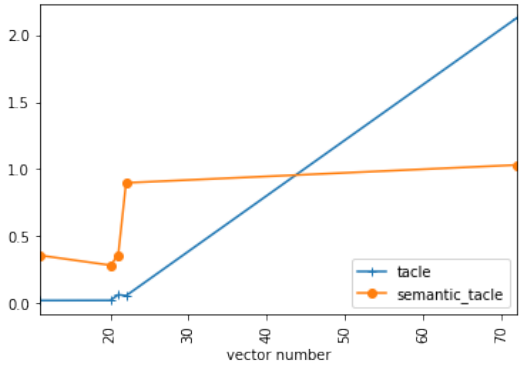
The result of the above experiment can be found in Figure 6.7 and Table 6.7. For all three datasets, Semantic-TaCLe reveals the same behaviour as per computation time. File with vector number less than 30, Semantic-TaCLe performs slower than the original TaCLe, whereas, TaCLe takes of its run-time after vector size 30. In conclusion, it can be said that Semantic-TaCLe performance is better when the vector size is sufficiently large. Thus such a scenario can be considered as Semantic-TaCLe's ideal application.

Q3: On the limited dataset chosen as the test set, this is evident from the experiment that the current approach overcomes the downfall of TaCLe which is limited precision and large computational time with increased vector number. Therefore, this chapter can conclude that the adopted approach has significant potential to improve tabular constraint learner without considerable computational cost.



(a) Synthetic Spreadsheet

(b) Real Spreadsheet



(c) Test Spreadsheet

Figure 6.7: Computation time comparison between TaCLe and Semantic TaCLe

	Average Run-time		
	TaCLe	Semantic-TaCLe without Classifier	Semantic-TaCLe
Synthetic Spreadsheet	2.091	1.532	1.275
Real Spreadsheet	2.294	1.634	1.646
Test Spreadsheet	0.4568	0.5846	0.5846

Table 6.7: Run-time Comparison in seconds

6.4 Conclusion

Overall, the step-by-step experimentation on each decision choice for the system architecture exhibits the potential of the proposed system. This chapter gradually verifies the system components starting from the semantic constraint ranking to predictive classifier implementation. Finally, it projects the complete system performance and compares it with TaCLe. The results show promising responses not only in TaCLe's improvement but also toward the header inference technique for knowledge discovery in a spreadsheet environment.

Chapter 7

Conclusion

This project designed a text-based, pre-search heuristic technique for constraint validation and evaluates such an approximation approach in tabular constraint learning. The evaluation of the proposed approach revealed higher precision than its baseline software TaCLe and sufficiently good recall to apply it in an interactive environment. The system is easily expendable for practical use by adding more constraint templates and training on the industry-based contextual spreadsheet.

Formula prediction and constraint prioritisation are necessary when the tabular data is too big and exhaustive search to learn constraints are expensive. If a table has α number of row and columns (cell arrays) and the number of the possible formula is l, to rediscover all intended formula from cell values will take exponential time; this makes the problem an NP-hard problem. The proposed approach downsizes α by forecasting constraint presence and considers only those cell arrays for constraint validation those the algorithm considers credible for formula search. Moreover, ranking formula based on likelihood increases chances of finding feasible constraint early and assures correct formula finding when more than one constraint is satisfiable on a cell array.

This paper demonstrates its key concept in formula reconstruction implementation. However, the proposed idea is equally applicable in the context of formula suggestion, auto-completion, and error correction. Formula suggestion could be the most exciting application of the proposed approach for a novice user, who is unaware of spreadsheet formula syntax. From the header label of any cell array and a very few data examples, the system is eligible to suggest appropriate formula to the user for that particular row or column. Such an implementation is also beneficial for spreadsheet auto-completion and error correction. From the system's formula prediction mechanism, an error correction software can place more bias on a predicted constraint from header label when the most of the value from the cell array satisfies the predicted constraint and classify inconsistent cell values as an error. Similarly, auto-completion can be seen as value inference problem based on already stated data in the cell array, and its header label and thus a suitable possible application of suggested technology in this paper.

From a theoretical point of view, the system suggests better performance when trained with a similar contextual spreadsheet. Creating context-based word em-

bedding from sufficient spreadsheet data will provide more accurate direction for formula learning using header labels. Future research work would be interesting to validate this theory empirically. Moreover, inspiring from this work, it will be reasonable to investigate further to an extent where spreadsheet formula learning is possible only from the header text. Along with the presence of formula, maybe, it is possible to predict which cell arrays are precisely connected by the functional constraint in spreadsheet from the header info; all these propositions are subject to further research.

Nevertheless, the current system is reliable enough to rediscover formula using tabular constraint learning with minimal effort. It manages to provide support to its underlying algorithm, TaCLe to improve learning precision and manage computational time when the tabular data has too many rows and columns. The system currently supports the spreadsheet's aggregate formulas, which is the most commonly used formulas in enterprises. This paper not only proposes an intelligent technique for formula learning but also demonstrates the convenience of the presence of valuable contextual information, i.e., header. It reveals the potential of contextual information interpretation to extract existing knowledge from a data-sheet.

Appendices

Appendix A

Collected Header Words w.r.t. Constraint Name

Constraint Name	Header Words
sum (col)	totals , total 2003-04 funds available, total, total education technology funds requested , total liabilities and equity, all grades, total assets, total income, total 2004-05 appropriations, totals, total 2004-05 funds available, total 2003-04 appropriations, income
difference	teachers who do not meet nclb standards, profit before taxes, operating profit, obj 1, gross profit
sum (row)	total eligible cost, total, sub-total, obj 1, total erdf
max (row)	total eligible cost, total
min (row)	non-obj
product	total cost, profit before taxes, net sales, operating profit, gross profit
sum-product	total education technology funds requested
rank	rank, position
foreign-key	id, type, salesperson, label
lookup	contact name, contact
running-total	total
series	id
sum-if	items sold total, sales
max-if	max items sold
max (col)	total liabilities and equity, max, the highest grade in class
min (col)	the lowest grade in class , min
average (col)	rbw avg, look avg, fifo avg, average

Table A.1: Constraint Name-Header word disctionary

Appendix B

Hyperparameter Tuning Results

B.1 Decision Tree

B.2 Random Forest

B.3 Linear Support Vector Machine

B.4 Kernel Support Vector Machine

Parameter		Classification Report		
Max_Depth	Max_Feature	Accuracy	Precision	Recall
2	50	0.851	0.558	0.785
4	50	0.947	0.791	0.785
6	50	0.947	0.791	0.785
8	50	0.947	0.791	0.785
10	50	0.947	0.791	0.785
2	100	0.925	0.694	0.871
4	100	0.917	0.688	0.871
6	100	0.904	0.607	0.871
8	100	0.912	0.67	0.871
10	100	0.912	0.67	0.871
2	150	0.899	0.507	0.827
4	150	0.899	0.507	0.827
6	150	0.899	0.507	0.827
8	150	0.882	0.479	0.827
10	150	0.882	0.479	0.827
2	200	0.921	0.695	0.871
4	200	0.934	0.719	0.871
6	200	0.947	0.805	0.871
8	200	0.947	0.805	0.871
10	200	0.947	0.805	0.871
2	250	0.93	0.709	0.871
4	250	0.956	0.803	0.871
6	250	0.956	0.803	0.871
8	250	0.956	0.803	0.871
10	250	0.956	0.803	0.871

Table B.1: Hyperparameter For Decision Tree tuning

Parameter				Classification Report		
$n_estimators$	max_depth	$min_samples_split$	$max_features$	Accuracy	Precision	Recall
80	20	2	50	0.961	0.875	0.785
20	2	2	20	0.961	0.875	0.785
20	14	4	40	0.956	0.854	0.785
20	18	4	30	0.956	0.854	0.785
160	18	4	50	0.961	0.875	0.785
80	8	8	100	0.961	0.875	0.785
20	14	10	250	0.952	0.797	0.785
60	10	2	50	0.961	0.875	0.785
180	14	10	150	0.956	0.854	0.785
100	10	10	30	0.961	0.875	0.785
180	14	10	300	0.956	0.854	0.785
160	8	8	40	0.961	0.875	0.785
60	12	10	50	0.961	0.875	0.785
40	16	10	250	0.956	0.854	0.785
60	18	8	50	0.961	0.875	0.785
160	2	8	300	0.956	0.854	0.785
100	20	10	10	0.961	0.875	0.785
140	6	4	20	0.961	0.875	0.785
80	2	4	20	0.956	0.854	0.785
180	8	8	20	0.961	0.875	0.785
120	20	6	30	0.961	0.875	0.785
100	18	10	300	0.956	0.854	0.785
140	18	4	20	0.961	0.875	0.785
180	8	8	300	0.956	0.854	0.785
160	12	10	20	0.961	0.875	0.785

Table B.2: Hyperparameter For Random Forest tuning

Parameter		Classification Report		
Loss	C_value	Accuracy	Precision	Recall
hinge	0.1	0.93	0.686	0.785
hinge	1.0	0.93	0.686	0.785
hinge	10.0	0.93	0.686	0.785
squared_hinge	0.1	0.93	0.686	0.785
squared_hinge	1.0	0.93	0.686	0.785
squared_hinge	10.0	0.93	0.686	0.785

Table B.3: Hyperparameter For Linear Support Vector Machine tuning

Parameter		Classification Report		
C_value	Gamma	Accuracy	Precision	Recall
0.1	2.0	0.961	0.875	0.785
1.0	2.0	0.961	0.875	0.785
10.0	2.0	0.961	0.875	0.785
0.1	1.0	0.961	0.875	0.785
1.0	1.0	0.961	0.875	0.785
10.0	1.0	0.961	0.875	0.785
0.1	0.1	0.961	0.875	0.785
1.0	0.1	0.961	0.875	0.785
10.0	0.1	0.961	0.875	0.785
0.1	0.01	0.89	0.609	0.653
1.0	0.01	0.934	0.7	0.785
10.0	0.01	0.934	0.7	0.785
0.1	0.001	0.504	0.297	0.436
1.0	0.001	0.908	0.632	0.785
10.0	0.001	0.917	0.655	0.785
0.1	0.0001	0.443	0.029	0.391
1.0	0.0001	0.513	0.298	0.477
10.0	0.0001	0.908	0.632	0.785
0.1	1e-05	0.443	0.029	0.391
1.0	1e-05	0.443	0.029	0.391
10.0	1e-05	0.513	0.298	0.477

Table B.4: Hyperparameter For Kernel Support Vector Machine tuning

Appendix C

Source Code

The detailed source code of this project can be found in: https://github.com/ML-KULeuven/tacle/tree/semantic-native/tacle

Bibliography

[1] R. Abraham and M. Erwig. Header and unit inference for spreadsheets through spatial analyses. In *Proceedings - 2004 IEEE Symposium on Visual Languages and Human Centric Computing*, pages 165–172, 2004.

[2] R. Abraham and M. Erwig. Goal-directed debugging of spreadsheets. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pages 37–44. IEEE, 2005.

[3] R. Abraham and M. Erwig. AutoTest: A tool for automatic test case generation in spreadsheets. In *Proceedings - IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2006*, pages 43–50, 2006.

[4] R. Abraham and M. Erwig. Inferring templates from spreadsheets. In *Proceedings - International Conference on Software Engineering*, volume 2006, pages 182–191, 2006.

[5] R. Abraham and M. Erwig. GoalDebug: A spreadsheet debugger for end users. *Proceedings - International Conference on Software Engineering*, pages 251–260, 2007.

[6] M. D. Adelfio and H. Samet. Schema extraction for tabular data on the web. *Proceedings of the VLDB Endowment*, 6(6):421–432, 2013.

[7] N. Beldiceanu. A Constraint Seeker : Finding and Ranking Global Constraints from Examples A Constraint Seeker : Finding and Ranking Global Constraints from Examples. (September 2011), 2011.

[8] N. Beldiceanu and H. Simonis. A model seeker: Extracting global constraint models from positive examples. In *International Conference on Principles and Practice of Constraint Programming*, pages 141–157. Springer, 2012.

[9] C. Bessiere, R. Coletta, E. Hebrard, G. Katsirelos, N. Lazaar, N. Narodytska, C.-G. Quimper, and T. Walsh. Constraint acquisition via partial queries. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[10] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, and C. Wallace. End-user software engineering with assertions in the spreadsheet paradigm. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 93–103. IEEE, 2003.

[11] M. Burnett, C. Cook, and G. Rothermel. End-user software engineering. *Communications of the ACM*, 47(9):53–58, 2004.

[12] Z. Chen and M. Cafarella. Automatic web spreadsheet data extraction. In *Proceedings of the 3rd International Workshop on Semantic Search over the Web*, pages 1–8, 2013.

[13] M.-C. De Marneffe and C. D. Manning. Stanford typed dependencies manual. Technical report, Technical report, Stanford University, 2008.

[14] W. Dou, S. C. Cheung, and J. Wei. Is spreadsheet ambiguity harmful? detecting and repairing spreadsheet smells due to ambiguous computation. In *Proceedings - International Conference on Software Engineering*, number 1, pages 848–858, 2014.

[15] M. Erwig, R. Abraham, S. Kollmansberger, and I. Cooperstein. Gencel: A program generator for correct spreadsheets. *Journal of Functional Programming*, 16(3):293–325, 2006.

[16] J. Fang, P. Mitra, Z. Tang, and C. L. Giles. Table header detection and classification. *Proceedings of the National Conference on Artificial Intelligence*, 1:599–605, 2012.

[17] D. Ferguson. Parsing financial statements efficiently and accurately using c and prolog. In *Practical Applications of Prolog Conference*, volume 97, 1997.

[18] 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*, pages 1–5, 2005.

[19] S. W. Golomb and L. D. Baumert. Backtrack programming. *J. ACM*, 12(4):516524, Oct. 1965.

[20] F. Grandoni and G. F. Italiano. Algorithms and constraint programming. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4204 LNCS(507613):2–14, 2006.

[21] R. Grishman. Information Extraction : Techniques and Challenges Why the Interest in Information Extraction ? *Information Extraction A Multidisciplinary Approach to an Emerging Information Technology*, i:10–27, 1997.

[22] F. Hermans, M. Pinzger, and A. van Deursen. Detecting and refactoring code smells in spreadsheet formulas. *Empirical Software Engineering*, 20(2):549–575, 2015.

[23] M. Hurst. Towards a theory of tables. *International Journal on Document Analysis and Recognition*, 8(2-3):123–131, 2006.

[24] T. Isakowitz, S. Schocken, and H. C. Lucas. Toward a Logical/Physical Theory of Spreadsheet Modeling. *ACM Transactions on Information Systems (TOIS)*, 13(1):1–37, 1995.

[25] Z. Kiziltan, M. Lippi, and P. Torroni. Constraint detection in natural language problem descriptions. *IJCAI International Joint Conference on Artificial Intelligence*, 2016-Janua:744–750, 2016.

[26] E. Koci, M. Thiele, O. Romero, and W. Lehner. A machine learning approach for layout inference in spreadsheets. *IC3K 2016 - Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, 1:77–88, 2016.

[27] S. Kolb, S. Paramonov, T. Guns, and L. De Raedt. Learning constraints in spreadsheets and tabular data. *Machine Learning*, 106(9-10):1441–1468, 2017.

[28] W. Kornfeld and J. Wattecampst. Automatically locating, extracting and analyzing tabular data. *SIGIR Forum (ACM Special Interest Group on Information Retrieval)*, pages 347–348, 1998.

[29] D. Lopresti and G. Nagy. A tabular survey of automated table processing. In *International Workshop on Graphics Recognition*, pages 93–120. Springer, 1999.

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[31] G. Miner, J. Elder IV, A. Fast, T. Hill, R. Nisbet, and D. Delen. *Practical text mining and statistical analysis for non-structured text data applications*. Academic Press, 2012.

[32] R. Panko. What We Don't Know About Spreadsheet Errors Today: The Facts, Why We Don't Believe Them, and What We Need to Do. In *Proceedings of the EuSpRIG 2015 Conference on Spreadsheet Risk Management*, 2016.

[33] S. Paramonov, S. Kolb, T. Guns, and L. De Raedt. TaCLe: Learning constraints in tabular data. *International Conference on Information and Knowledge Management, Proceedings*, Part F1318:2511–2514, 2017.

[34] V. Paramonov, A. Shigarov, V. Vetrova, and A. Mikhailov. Heuristic Algorithm for Recovering a Physical Structure of Spreadsheet Header. *Advances in Intelligent Systems and Computing*, 1050(18):140–149, 2020.

[35] D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table extraction using conditional random fields. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 235–242, 2003.

[36] S. G. Powell, K. R. Baker, and B. Lawson. A critical review of the literature on spreadsheet errors. *Decision Support Systems*, 46(1):128–138, 2008.

[37] K. Rajalingham, D. Chadwick, B. Knight, and D. Edwards. Quality control in spreadsheets: a software engineering-based approach to spreadsheet development. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 10 pp. vol.1–, 2000.

[38] X. Rong. word2vec Parameter Learning Explained. pages 1–19, 2014.

[39] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier Science Inc., USA, 2006.

[40] S. Russell and P. Norvig. *Artificial intelligence: a modern approach*, pages 202–233. 2002.

[41] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[42] F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[43] H. Simonis. Sudoku as a constraint problem. In *CP Workshop on modeling and reformulating Constraint Satisfaction Problems*, volume 12, pages 13–27. Citeseer, 2005.

[44] J. Tyszkiewicz. Spreadsheet as a relational database engine. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD 10, page 195206, New York, NY, USA, 2010. Association for Computing Machinery.

[45] D. Waltz. Understanding line drawings of scenes with shadows. In *The psychology of computer vision*. Citeseer, 1975.

[46] We Are Social. Trends shaping social in 2019. Technical report, 2019.

[47] W. Wong, D. Martinez, and L. Cavedon. Extraction of named entities from tables in gene mutation literature. *ADCS 2008 - Proceedings of the Thirteenth Australasian Document Computing Symposium*, (June):49–52, 2008.

Master's thesis filing card

Student: Sarah Binta Alam Shoilee

Title: Knowledge Discovery on Spreadsheet Data using Semantic Information

UDC: 621.3

Abstract:
For data processing and maintenance, a spreadsheet is the most widely used tools amongst corporations. Performing in-depth research on spreadsheets has increasingly become critical to enabling knowledge discovery as well as improving user efficiency and supporting error-free data processing. Further research in this field is essential, considering the sheer expanse and the importance of tabular data in the data community, despite the number of existing works in the area. This paper proposes an algorithm for tabular constraint learning using semantic header information, focusing on enhancing existing tabular constraint learner (TaCLe). The proposed solution will rediscover lost formulas on a poorly filed spreadsheet while conceptually analysing the header. The proposal of the header interpretation assists to recapture formulas faster with more accuracy than the previous solution, overall improving the computational performance. Furthermore, this research outlines the usability of header information come along with tabular data, often ignored due to its complex structure. The study further proposes a formula reconstruction application, empirically validating each step of the design decision. This technique will not only be capable of formula rediscovery but also for formula suggestion, error-correction, and auto-completion in a spreadsheet environment when integrated with a right interaction model.

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Big Data Analytics

Thesis supervisor: Prof. Dr. Luc De Raedt

Assessors: Assoc. Prof. Dr. Marc Denecker
Dr. Ir. Samuel Kolb

Mentor: Dr. Ir. Samuel Kolb