Investigating the Usefulness of Metric-based Prediction Method for Spreadsheet Fault Detection

Musa Kunya[1,*], *Mohamed* Hamada[2,**], *Mohammed* Hassan[3,***], and *Saratu* Yusuf Ilu[3,****]

[1]Department of Computer Science, Bayero University, Kano. Nigeria
[2]University of Aizu, Japan
[3]Department of Software Engineering, Bayero University, Kano. Nigeria

Abstract. The ability to predict whether a specific section of a spreadsheet is faulty or not is frequently required for the development of spreadsheet functionality. Although errors in such spreadsheets are common and can have serious consequences, today's spreadsheet creation and management tools offer weak capabilities for defect detection, localization, and fixing. In this thesis, we proposed a method for predicting faults in spreadsheet formulas that can detect faults in non-formula cells by combining a catalog of spreadsheet metrics with modern machine learning algorithms. An examination of the individual metrics in the catalog reveals that they are suited to detecting data where a formula is expected to have flaws. In this framework, Recall Score of 99% was achieved and performance was compared with that of Melford. The result of the experiment reveals that the proposed framework outperforms Melford framework.

Keywords— Spreadsheet, Random Forest,Support Vector Machines, Deep Neural Networks, Adaptive Boosting, Fault Detection

1 Introduction

Google Sheets and Excel are used by between 750 million and 2 billion individuals throughout the world. Every month, about 2 billion people use Google Suite, which includes Google Sheets, whereas Microsoft Excel has an estimated 750 million to 1.2 billion monthly users worldwide. Microsoft CEO Satya Nadella gave the official statistic of over 750 million users online in 2017, referring to Microsoft Excel as the "most significant consumer product" [1].

Spreadsheets are commonly utilized in businesses for a variety of computations and decision-making. Spreadsheets, on the other hand, are prone to inaccuracy, and there have been several cases when erroneous spreadsheet formulae have resulted in large financial losses for businesses. Worryingly, because spreadsheets are frequently prepared by non-IT professionals, the error rate is frequently considered to be greater than in traditional software. McConnell claims that in the real world, there are 1 to 25 mistakes per 1000 lines of code. This rate can be reduced further depending on the underlying quality assurance techniques and processes; for example, Microsoft reports 0.5 defects per 1000 lines of code in released products. The error rate in spreadsheets, on the other hand, is thought to be around 3-5% [2].

For students and professionals in marketing, sales, business, and finance, spreadsheet software provides an easy-to-use data input platform. However, there are also chances to over type wrong numbers or formulae, or to factor in numbers inaccurately [3].

Error detection is very important to the field of software development [4], The training data required for these techniques are typically derived from collections of faulty software artifacts with explicitly labeled faults. Software metrics are commonly used as predictor variables in such a problem formulation [5] [6] [7] [8]. The values of the predictor variables are gained by standards that compute measurable characteristics of the faulty programs. Various metrics were explored for predicting faults in general software, e.g., the size, complexity, or even the modification history of a program [9].

By closing the gap in studies of prediction for fault in a non-formula cell in a spreadsheet, The thesis hope to contribute to not only techniques of creating accurate spreadsheets but also the field of software fault prediction methods.

Miscalculations may have a little impact on balance sheets for households and small enterprises, but they can have a considerably higher financial and pro-

*e-mail: musa.kunya@fulokoja.edu.ng
**e-mail: hamada@u-aizu.ac.jp
***e-mail: mhassan.se@buk.edu.ng
****e-mail: syilu.cs@buk.edu.ng

fessional impact for larger organizations. Even little spreadsheet errors can cost a company billions of dollars or damage an individual's career and reputation [3].

Even though spreadsheets are used for small applications, they are also used to develop many large programs. In recent years, a good number of errors that people make when they develop spreadsheets (Human Errors) [10].

There has been a slew of scholarly techniques to spreadsheet defect avoidance, detection, localization, and repair [11]. The efficiency of such approaches is frequently determined by their ability to anticipate the chance of a spreadsheet formula being incorrect. The irony is no matter how accurate the formula is, if it is not supposed to be there in the cell, it becomes an error/Fault.

Among the frameworks is by Koch et al., [9] developed to use a metric-based fault detection for spreadsheets. However, the usefulness of their model has not been investigated for non-formula cells such as "number where a formula is expected"; Based on our investigation no research has been conducted to examine how a model-based approach could predict faults on non-formula cells using metrics. Therefore, this research proposed to investigate the usefulness of the metric-based approach for fault detection on a spreadsheet (number where a formula is expected and vice versa). Spreadsheet error/fault can not be eliminated due to the sad ability of humans to err [3], but can be reduced to a bare minimum and eliminate unnecessary losses to households, small businesses, and giant organizations.

The research aims to propose a metric-based model for fault detection on a spreadsheet that could detect faults in non-formula cells.

The objectives of the research are:

1. To propose a framework for detecting a fault in a non-formula cell of a spreadsheet (number where a formula is expected and vice versa).

2. Implementing the framework for experimentation, performing in-depth analysis of the contribution of specific metrics will prove useful for understanding prediction results.

3. Evaluate the model for non-formula cell fault prediction using the harmonic mean of precision and recall.

The result of the study will be deployed in spreadsheet applications such as LibreOffice and MS Excell to improve the process of developing and debugging, Thus reducing the cost of errors/faults in a spreadsheet. It is also a stepping stone toward an error-free Spreadsheet for institutions.

The research is successful in determining the content of a cell, whether a formula or variable/data, which is important to the quality assurance of a spreadsheet. Therefore, this thesis has closed the gap by implementing and evaluating the performance of the cell prediction model using a metric-based model.

The remainder of the study is organized as follows: Reviewed previous work in Chapter 2 and describe the proposed method and laid out the general design of our experiments in chapter 3. Then provide details and results for the experiments in Chapter 4. Chapter 5 concludes the paper and provides an outlook on future work.

2 Related Work

Practically, coming up with a general heuristic procedure for fault prediction can be a difficult process. Given enough training data, Machine Learning methods can simplify this problem by finding a model that approximates such heuristics. A recent method by Singh et al., [12] called Melford follows such an approach and uses Machine Learning to train a classifier for the automatic prediction of "number-where-formula-expected" faults in spreadsheets. In particular, Melford trains a neural network using specific abstract representations of faulty spreadsheets. An experimental evaluation revealed that the approach can classify instances in which a number is erroneously placed instead of a formula with high precision. The applicability of Melford is still limited, as it currently can only recognize one fault type, and extending it to recognize others would require significant effort: first, one would have to design additional abstractions, which cover other fault types; second, an optimization of the neural network structure might also be required to allow for learning of an accurate classification model. In contrast, our approach does not rely on structural abstractions of faulty spreadsheets. Instead, we use the results of applying spreadsheet metrics, which encode the existing knowledge from research about possible problems in spreadsheets. As the experiments show, our set of metrics can make highly accurate predictions for various types of faults [9].

For example, the fault prediction tools UCheck by Abraham & Erwig, [13] and Dimension by Chambers & Erwig, [14] first infer and assign "data types" to spreadsheet input cells and formulas. These types are derived from the text values of header cells that are in the same row or column as the corresponding input cells. The tools then propagate the types of input cells to the formulas that refer to those inputs iteratively. If the propagated type of a formula cell does not match the derived type obtained from the header information, the cell is reported as possibly faulty.

Other approaches, such as AmCheck by Dou et al., [15], its successor CACheck by Dou et al., [16], and EmptyCheck by Xu et al.[17], focus on cell arrays.

The method of using metrics as seen in Koch et al., [9] is based on an extensive set of spreadsheet characteristics (metrics), allowing for the detection of a variety of fault types. Moreover, it is easy to extend. If a new metric, an Example for a new fault type, is added to our catalog, the underlying ML algorithm will automatically incorporate its output into the fault predic-

tion model. A cell array is a collection of columns or rows that contain formula cells that are functionally related and are surrounded by "borders" of either empty cells or cells with fixed values. The first two tools predict that a cell is faulty (smelly) if its formula deviates significantly from other formulas in the same array, whereas EmptyCheck employs a clustering algorithm to identify empty cells that should most likely contain a formula. One disadvantage of these approaches is that they are intended to detect specific types of faults, such as incorrect data types or mismatched cells in an array, which can result in limited coverage and applicability [18].

Melford's applicability is still limited because it can only recognize one fault type at the moment, and extending it to recognize others would require significant effort:

a, Additional abstractions covering other fault types would have to be designed;

b, An optimization of the neural network structure may also be required to allow for the learning of an accurate classification model. Our approach, on the other hand, does not rely on structural abstractions of faulty spreadsheets.

Instead, we use the outcomes of spreadsheet metrics, which encode existing knowledge from research about potential spreadsheet problems. As demonstrated by the experiments, our set of metrics is capable of making highly accurate predictions for various types of faults.

It has been proposed to use spreadsheet smells to predict faults by Abreu et al.,[19]. Their tool, FaultySheet Detective, uses the output of spreadsheet smells, such as complex formulas, missing inputs, or problematic dependencies by Hermans et al., [20] and By Jácome Cunha et al., [21], Technically, their approach is divided into two stages. In the first step, their method computes two sets of cells:

(i) the smelly cells and

(ii) the output cells.

Following that, the algorithm generates calculation chains for all output cells and initiates a Spectrum-based Fault Localization algorithm to predict the likelihood of each cell is faulty.

The prediction model developed by Abreu et al., [22] has the advantage of not requiring any learning phase to predict formula faults. However, an underlying assumption is the availability of reliable smell detection thresholds, i.e., if the value of the metric used to detect the smell for some cell exceeds the threshold, then this cell is considered smelly. While researchers suggested reasonable thresholds for certain smells, for example, By Hermans et al., [20], and Jácome Cunha et al.,[23]), the optimal threshold values are unknown in general and may depend on the given application domain.

The metrics model, on the other hand, can deal with situations in which individual metrics are not necessarily strong predictors for the given domain. We can achieve highly accurate predictions even in the presence of weak individual predictors by combining multiple metrics in one prediction model and automatically adjusting the weights through a learning procedure [9].

The ExceLint add-in by Singh et al., [12] detects faulty formulas automatically by assuming that spreadsheets follow “rectangular-like” patterns, i.e., formulas in the same row or column are very likely to have the same semantics. As a result, any formula that does not fit into a rectangular-like layout is most likely flawed. ExceLint first transforms all formulas into a two-dimensional vector representation known as fingerprints to find such layouts. The sum of the relative column/row distances from the formula cell to the other cells referred to in the formula defines the value of each coordinate. Following that, a binary decomposition algorithm finds fingerprint regions by recursively dividing the spreadsheet into two parts in such a way that each split minimizes the normalized Shannon entropy of the fingerprints in both subdivisions. Finally, the tool generates repair suggestions for formulas that deviate from the norm. Suggestions are generated that would result in a moderate reduction of the spreadsheet's overall entropy, which is typical for genuine fault fixes. ExceLint detects a variety of fault types caused by incorrect formulas. However, because the approach does not consider the structure of the formulas, it is unable to identify certain types of faults that may occur. For example, as a result of incorrectly applied functions or operators.

By adding new metrics to the catalog that are indicative of the specific faults, this approach by Koch et al., [9] can include these and other fault types. Only FaultySheet Detective and ExceLint are directly comparable with our framework because they are also designed to detect various fault types in spreadsheet formula cells. The other techniques discussed either focus on detecting a single type of fault or detect faults in cells that do not contain formulas.

For spreadsheets, a framework has been designed to use metric-based fault detection. However, the model's use for non-formula cells, such as "number where a formula is expected," has not been studied; according to our investigation, no research has been done to see how a metric-based technique could predict defects on non-formula cells. As a result, the goal of this study was to see how beneficial a metric-based approach is for fault identification on a spreadsheet. Due to the unfortunate nature of humans to make mistakes, spreadsheet error/fault cannot be completely eradicated, but it can be minimized to a minimal minimum, preventing avoidable losses to households, businesses, and large organizations.

3 Experimental Methodology

3.1 Research Design

The fundamental idea behind our method is to train an ML model for fault prediction using a set of spreadsheets containing tagged defects. The ML problem's predictor variables correspond to a diverse range of metrics. Computing the values for the metrics for the labeled spreadsheets yields the learning dataset. As a result, our approach's formal inputs are:

A series of training spreadsheets with flaws in which all formulae are labeled as correct while data is labeled as erroneous, as well as a set of metrics, each of which accepts a spreadsheet as input and outputs a value for each formula in the spreadsheet.

The result is a fault predicting classifier that yields the likelihood that a formula in a previously unknown spreadsheet is faulty given a collection of observations derived by our metrics for that formula. The overall procedure for creating this classifier consists of four main steps: Cataloging of Metrics, Preparing Experimental Dataset, Material and Equipment, and Training (optimization) and Evaluation Aspects.

Before we get into the technical intricacies of each stage, we'll go through the basics of a spreadsheet metrics catalog that we created specifically for this project and will subsequently utilize as a foundation for our experimental evaluations.

3.2 Cataloging of the Metrics

Consider a combination of various metrics to achieve high prediction accuracy and recall, the previous model uses the blocks of metrics, metrics computed by cell, metrics computed per formula cell, metrics computed per formula, and metrics computed per worksheet. The aim is to use as little as possible number of metrics (attributes) with the highest possible accuracy and recall.The flowchart for searching the optimum metrics can be seen in Figure 1.

3.3 Algorithm for searching the right metrics

The pseudo code for searching the optimum metrics that represent the algorithm for prediction can be seen below.

Step 1 Begin
Step 2 Extract dataset from Corpus
Step 3 Select Metric Properties
Step 4 Perform Training, Prediction and tuning algorithm metrics
Step 5 If Result Not Improved
pass control to Step 3
Else
Produce model
Step 6 End

The Quality assurance tool is going to be developed for a spreadsheet using 22 metric-based parameters for detecting where the formula is supposed to be,

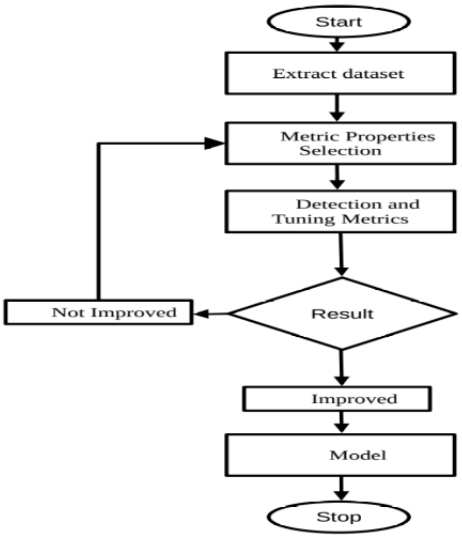

Figure 1. Flowchart for searching the right metrics

it will train by using Random Forest, AdaBoost, SVM SDG, and Deep Neural Networks, the algorithm with the highest F1 score be adopted. In Table 1 are the metrics to be used as follows:

3.4 Source(s) of Data

Enron Errors is a subset of the Enron spreadsheet corpus by Jannach et al., [11] containing real-world faults [24]. The Enron corpus itself consists of over 15,000 spreadsheets that were found in emails of Enron employees after the bankruptcy of the company in 2001. From this larger corpus, researchers extracted 26 faulty spreadsheets in a tool-supported process by Jannach & Schmitz, [11]. Since the spreadsheets of the corpus cover various application problems, we consider this case unrestricted.

INFO1 contains spreadsheets developed by civil engineering students during an Excel course by Getzner et al., [25]. The students had to solve two tasks (construction-related calculations) by modeling spreadsheets. Consequently, the collection includes spreadsheets with similar, albeit long and complex computations, which is why we regard it as a restricted case.

The modified EUSES by Hofer et al., [4] corpus contains artificial faults that were injected into spreadsheets of the original EUSES corpus by Elbaum et al., [26], such that each spreadsheet contains exactly one faulty formula. Faults were injected by randomly selecting one formula cell of each spreadsheet and modifying it using a randomly chosen mutation operator. The authors considered only operators from Abraham & Erwig, [13], which correspond to the introduction of mechanical faults, i.e., typos. Examples of the operators include alternations of arithmetic or logical connectives, replacement of formulas by constants, etc.

C. No	Attributes
1	Column number (position) of the cell in the worksheet.
2	Column distance to relative i.e., distance to the next cell in the same column that has the same cell type; we use a predefined and configurable maximum value for this metric. Adapted from Pattern Finder to provide a numeric score.
3	Row distance to relative i.e., distance to the next cell in the same row that has the same cell type; we use a predefined and configurable maximum value for this metric. Adapted from Pattern Finder to provide a numeric score.
4	Range references to cell, i.e., number of range references referring to this cell. Counts multiple references from the same cell individually.
5	Direct references to cell, i.e., number of direct references (not as part of range references or named ranges; multiples are counted individually.)
6	Number of any references to the cell, i.e., sum of metrics 4 and 5.
7	Number of direct and range references in other worksheets that refer to this cell.
8	Row number (position) of the cell in the worksheet.
9	Standard deviation (column) i.e., the absolute difference to the mean of the numeric values in the same column (for cells with numeric values).
10	Standard deviation (row) i.e., the absolute difference to the mean of the numeric values in the same row.
11	Successors i.e., the number of cells that are referring to the cell by either direct or range references; also known as fan-in and visibility.
12	Number of successors in other worksheets.
13	Number of non-empty or referenced blank cells in the worksheet.
14	Number of formula cells in the worksheet.
15	Column number of the right-most non-empty cell in the worksheet.
16	Number of R1C1-unique formulas in the worksheet.
17	Position of the worksheet in the workbook's sheet tab.
18	Number of cells that are referenced by cells of this worksheet.
19	Number of cells in other worksheets that are referenced by cells of this worksheet.
20	Row number of the bottom-most non-empty cell in the worksheet.
21	Number of cells that refer to cells of this worksheet.
22	Number of cells in other worksheets that refer to cells of this worksheet.

Table 1. Adopted metrics (C.= Column)[9]

Thus, while the dataset comprises real-world spreadsheets augmented with injected faults, the synthetic case represents a simulation of real-world scenarios occurring due to typos. Table 2 contains the summary corpus for extracting dataset.

Fritz is the tool (executable jar file) that was used to compute the metric values for each of the input datasets. Additional information regarding Fritz's requirements, functionalities, and usage can be discovered on the tool's web page (https://spreadsheets.ist.tugraz.at/index.php/software/fritz/).

Corpus	Scenario	Spreadsheets	Formulas
Enron Errors	Unrestricted	26	16.790
INFO1	Restricted	119	174.493
EUSES	Synthetic	576	284.109

Table 2. Statistics of the Research Datasets [11] [25] [4]

Corpus	Scenario	Spread sheets	Instances	TRUE	FALSE
Enron Errors	unrestricted	26	6,790	8,687	8,103
INFO1	restricted	119	174,493	5,157	169,336
EUSES	synthetic	576	284,100	181,283	102,817
Total		721	475,383	195,127	280,256

Table 3. Data set Statistics table [11] [25] [4]

3.5 Preparing Experimental Dataset

This approach involves getting ready the dataset before applying data mining. However, in the context of Enron, Info1, and EUSES Worksheets, the steps involved in data preprocessing are data cleaning, which involves dealing with missing and noisy data. Data transformation involves transforming the data into a form suitable for mining, data scaling involves scaling the data within a common appropriate scale for all features, and dimensionality reduction involves data selection (through forward feature selection) and data extraction. to optimize storage efficacy, retaining data information, and reducing analysis costs.

We generated a dataset with 475,383 instances and 21 Attributes connected by 721 Sheets after cleaning the data. Table 3 summarizes the statistical properties of the above data descriptions.

We also select the most relevant basic variables from the clean data set using an appropriate dimensionality selection method (forward selection and random forest) to select variables in order of their variation ranking; based on this, we only select 21 basic features that we consider to be the most relevant basic variables for predicting whether a cell contains formula or not. We choose at random 80 percent of the total summation from the basic variables as the training set and the remaining 20 percent as the testing set.

3.6 Material and Equipment

For the described problem, Variety of algorithms can be used, ranging from Logistic Regression, Support Vector Machines, and the most recent Deep Learning techniques. Even though forecasting the relative performance of different ML algorithms for specific datasets is challenging, we evaluate algorithms from different families. However, some general considerations should be made when choosing an algorithm. First, in terms of dataset sizes, the number of labeled spreadsheets in some applications could be low.As a direct consequence, the chosen algorithm should be stable even though only limited data is available. Furthermore, the algorithm should scale well in cases

where there is a huge amount of data. Second, the importance of features varies across datasets. Our metric catalog is designed to describe a broad range of fault types. However, if a specific spreadsheet collection lacks certain fault types, the learning algorithm should disregard the corresponding metrics.

As a result, the learning algorithm used is known to perform well on small to medium learning datasets, and have an explicit or implicit mechanism for automatically selecting or highlighting the most informative metrics (feature selection). Taking these factors into consideration, the Random Forests algorithm is a better match for the given learning problem. Random Forest is an ensemble approach that employs the bagging concept to train N decision trees on N randomly selected subsets of the training set. The final classification model is then calculated as an average of the individual tree results. Aside from variance reduction via bagging, Random Forest constructs each of the underlying decision trees with a specifically sampled subset of features. This results in a more accurate representation of the various features in different trees and a decrease in the correlation between the features.

Support Vector Machine (SVM) has been shown to be effective in a variety of domains, so they were included as a baseline approach. Because traditional methods can be slow when training SVM on large datasets, we also use the faster but sometimes less accurate gradient descent method in its learning phase (SVM Stochastic Gradiend Decent). SVM training is heavily influenced by the value of the regularization parameter C, which determines the degree to which the model may misclassify the training data. Choosing an appropriate value for this parameter is critical because it directly influences a model's ability to generalize beyond the training dataset.

Adaptive Boosting is a well-known ensemble technique that is similar to Random Forest. Adaptive Boost trains N "weak learners" sequentially on weighted variants of the dataset. As a result, the algorithm corrects the shortcomings of previously trained weak classifiers by emphasizing misclassified observations from the previous iteration. Every newly trained weak learner attempts to find a model that classifies the most highly ranked observations correctly first. An Adaptive Boost model determines a classification result given an observation as the weighted sum of the results of the underlying weak models, with the weights determined by the prediction performance of the individual models.

Deep Neural Networks have been used successfully to solve a variety of classification problems. In Melford, they were also used to predict "number-where-formula-expected" faults. In this study, we employ a network architecture similar to that of Melford. Our feed-forward neural network, in particular, has 1 hidden layers with 128 neurons each and employs a rectified linear unit activation function. Rather than feeding abstractions to the network as Melford did, we feed our computed metric values to the 64 input neurons.

Score	Data	Random Forest	Ada Boost	Deep Neural Network	SVM SDG	Average Score
Acc	enron	0.9898	0.9818	0.7826	0.7468	0.8753
Acc	euses	0.9984	0.9958	0.9148	0.6303	0.8848
Acc	info1	0.9982	0.9958	0.9127	0.6306	0.8843
Pre	enron	0.9931	0.9874	0.8917	0.747	0.9048
Pre	euses	0.9987	0.9966	0.9331	0.6481	0.8941
Pre	info1	0.9985	0.9964	0.9087	0.6464	0.8875
Rec	enron	0.9875	0.9779	0.6693	0.7866	0.8553
Rec	euses	0.9988	0.9969	0.9336	0.9219	0.9628
Rec	info1	0.9987	0.9967	0.9596	0.9295	0.9711
F1	enron	0.9903	0.9827	0.7646	0.7663	0.8760
F1	euses	0.9988	0.9967	0.9334	0.7611	0.9225
F1	info1	0.9986	0.9967	0.9334	0.7625	0.9228
Average		0.9958	0.9918	0.8781	0.7481	

Table 4. Results for datasets with Algorithms over the Accuracy(Acc), Precision(Pre), Recall(Rec), and F1 score metrics.

4 Result and Discussion

It can be noticed from the Result table (Table 4). The performances of Random Forest and AdaBoost is on par with each other, it is because we used for both, Ensemble algorithms Decision Tree learners, the only difference is the final result is determined by voting in the former and bagging in the later.

We further pick the F1 Score to compare the results because Accuracy, Precision, and Recall alone can be deceiving when it comes to classification models.

The recall score for the EUSES data set for the algorithm SVM SDG gives a score of 1 that is 100% which is highly unlikely for a machine learning algorithm, it has been a surprise for us to have such a score. SVM SDG was supposed to produce one of the worst results because, we didn't have to compute power for convergence of SVM and it is used as a control trainer to represent Logistic Regression classifier, so we used SVM SDG which those not take time to train. We intend to further investigate from an algorithmic and data perspective.

Looking at the result table one might think or suggest that such high scores are overfitted, well it is, but because theoretically, all the possible scenarios of the spreadsheet have reflected in the data set, it would not affect the performance of the model.

In the experiments, the models were evaluated on spreadsheets with the same origin as the training data (ENRON, EUSES, and INFO1) with Learning Algorithms (Random Forest, Adaboost, Deep Neural Network, and SVM SDG) over the Accuracy, Precision, Recall, and F1 score metrics,Table 5 shows the general schema of the resulting dataset. The dataset statistics are presented.

In terms of Accuracy score, the learning algorithm Random Forest and INFO1 corpus gave the best score of 0.9998. in terms of Precision score, Learning algorithms Random Forest, AdaBoost, and INFO1 corpus gave the best score of 0.9997 each. In terms of Recall

Algorithm	enron	euses	info1	Average Score
Random Forest	0.9903	0.9988	0.9986	0.9959
AdaBoost	0.9827	0.9967	0.9967	0.9920
Deep Neural Network	0.7646	0.9334	0.9334	0.8771
SVM SDG	0.7663	0.7611	0.7625	0.7633
Average Score	0.8760	0.9225	0.9228	

Table 5. Statistics of the Research Datasets [11] [25] [4]

Score, Learning algorithm SVM and EUSES corpus gave the best score of 1. while in terms of F1 score, Random Forest EUSES and INFO1 corpus gave the best score of 0.9998. Averagely Random Forest produces the best score of 0.9990 while SVM the worst score of 0.5261.

Depending on Accuracy, Precision, and Recall alone can be deceiving because they don't translate into real-world performance, as we compute the F1 score which is the Mean Harmonic of precision and recall scores. The F1 scores were extracted for a better comparison of the model performance in table 9 with EUSES having the best score Averagely of 0.9394 and Enron having the worst score of 0.7402. Table 6 shows the general schema of the resulting dataset for F1 score.

While the SVM approach was completed in a fair amount of time on the Enron Errors dataset, it took an unusually long time to train on the others. As a result, instead of using a comprehensive optimization technique, we moved to the SVM SGD variation of the algorithm, which uses stochastic gradient descent. As a result, we were able to train SVM models in a reasonable amount of time, but the models' performance suffered. Given the findings of both SVM variants on Enron Errors and SVM SGD on the other datasets, we may rule out both variants from further consideration due to their poor performance when compared to the other approaches.

4.1 Results of individual Metrics

We present Algorithmic performance results for the three Dataset, Shown based on Datasets score, In figures 2-5 with the best and worst scores.

The classification performance results for the three datasets are shown based on the Accuracy score in Figure 9 wth Random Forest and AdaBoost having best scores and SVM SDG having the worst score.

The classification performance results for the three datasets are shown based on the Precision in Figure 10 with Random Forest and AdaBoost having the best scores and SVM SDG having the worst score.

The classification performance results for the three datasets are shown based on the Recall score in Figure 11 with Random Forest and AdaBoost having the best scores and SVM SDG having the worst score.

The classification performance results for the three datasets are shown based on the F1 score in Figure 12

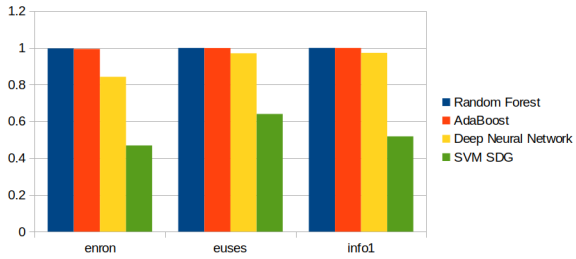

Figure 2. Comparison of Accuuracy Score of the learning Algorithms

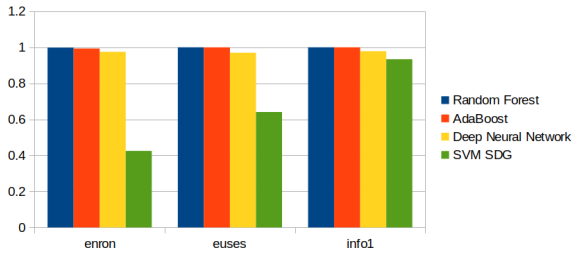

Figure 3. Comparison of Precission Score of the learning Algorithms

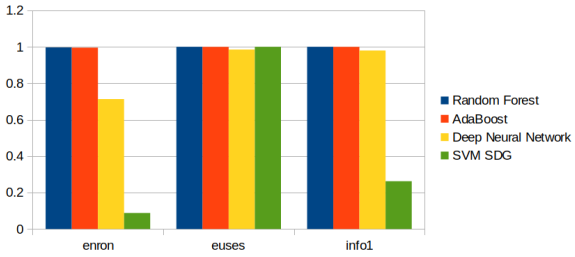

Figure 4. Comparison of Recall Score of the learning Algorithms

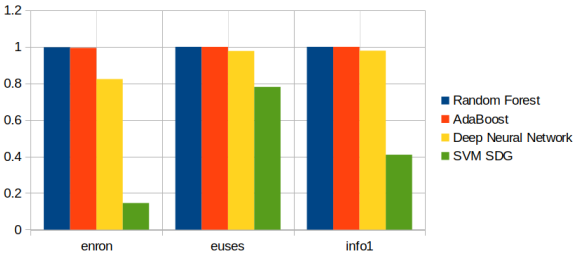

Figure 5. Comparison of F1 Score of the learning Algorithms

with Random Forest and AdaBoost having best scores and SVM SDG having the worst score.

The Random Forest Result confusion matrix for the spreadsheet class is shown in Tables 6-8, we only showed Random Forest confusion matrices because in

Class Category	Predicted Status Categories	Predicted Status Categories
	Formula Cell	Non-Formula Cell
Formula Cell	1587	16
Non-Formula Cell	20	1735

Table 6. Confusion matrix table for Random Forest on ENRON data.

Class Category	Predicted Status Categories	Predicted Status Categories
	Formula Cell	Non-Formula Cell
Formula Cell	20431	37
Non-Formula Cell	33	36319

Table 7. Confusion matrix table for Random Forest on EUSES data.

Class Category	Predicted Status Categories	Predicted Status Categories
	Formula Cell	Non-Formula Cell
Formula Cell	20514	57
Non-Formula Cell	35	36214

Table 8. Confusion matrix table for Random Forest on INFO1 data.

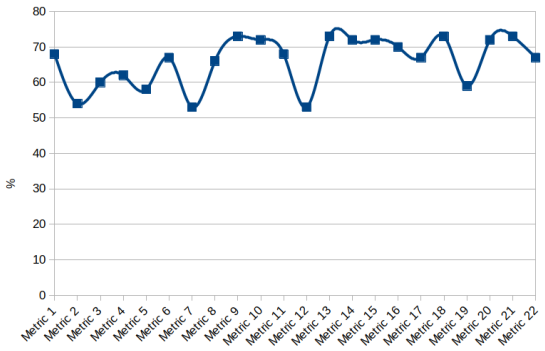

Figure 6. Metric Contribution Analysis

the preliminary running testing of the algorithms it shows more promising among the rest.

We performed ranking of the attributes used for the training and present it in form of percentage to clearly see the contribution of each and every attributes in the corpus,with metric 9 and 13 having the highest rank of 73%. it can be seen in the Figure 6.

4.2 Baseline Comparison

In addition, we compare Melford to our model, as well as the effectiveness of the various models.

The first baseline is a simple statistical classifier that remembers every 5x5 context and predicts outcomes based on the frequency of outcomes in the training set. For instance, if the 5x5 stencil shown in Figure 14 occurred 10 times in the training set and the center cell contained a F 9 times and a N once, we record this exact distribution (Num F = 9) and (Num N = 1).

In the Table 10 and Figure 15, the F1 score for each Melford network model is shown in red color while the result of our study is in green color. It can be ob-

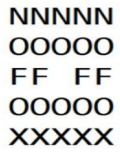

Figure 7. Melford 5× 5 abstract context for cell [12]

Models	Average F1 Score
Random Forest	0.9959
AdaBoost	0.9920
Deep Neural Network	0.8771
SVM SDG	0.7633
5x5-FF1	0.3650
5x5-FF4	0.3000
9+9-FF4	0.0800
9x9-FF4	0.2950
9x9-LSTM	0.2800
5x5-LSTM	0.2400
Stencil	0.1400
SVM	0.1700
CUSTODES	0.7600

Table 9. Comparison of F1 Score of the learning Algorithms

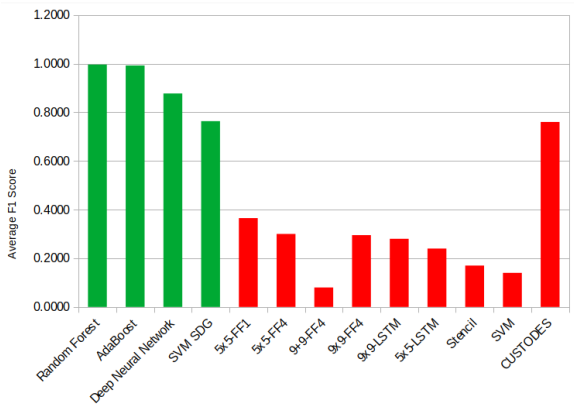

Figure 8. Comparison of F1 Score of the learning Algorithms

served that there is improvement in performance between green bars and red bar which represent Melford models. There is an improvement of 23.59% between our best model developed in Random Forest and the best score in the Melford model CUSTODES.

5 Conclusion and Recommendation

We set out to investigate the usefulness of metric-based fault detection for fault finding in a spreadsheet, which led us to narrow our objectives to detecting a fault in a non-formula cell where a formula is expected (Koch et al., 2019) which was suggested as a future area of investigation. We were able to develop a model that predicts whether a cell is supposed to contain a formula or a value with a Harmonic mean of 0.9990 averaging across the three datasets used to cover all

the possible scenarios of real-life spreadsheets used in the day to day activities of businesses, institution, and other industries. The evaluation of the model was done using Accuracy, Precision, Recall, and F1 score, including a Confusion Matrices for the Random Forest that produced the best result among the rest of the Algorithms. Then we tried comparing the result with the state-of-the-art framework used for achieving prediction where the formula is expected (Melford) in a spreadsheet. From the result we have gotten it is clear that the metric-based fault-finding in a spreadsheet can not only detect error associated accuracy of the formula (study of the base paper) and where a formula is expected (our study) in a spreadsheet but also future errors that may arise as a result of continues use of spreadsheets.

It is however clear that with a careful and strategic reshuffling of the metrics many errors associated with a spreadsheet can be detected with a significant score of harmonic mean. So it is up to the future researchers to outline the errors they want to cover, follow an iterative process of metric selection till the required score is achieved. Our study may have been limited because of a lack of computing power, which can be seen in the implementation of SVM and DNN on the dataset. We were not able to train until the convergence of the scores. We believe with a huge capacity for computing better scores perhaps even better than the score gotten from decision tree ensemble learners can be obtained. We also intend to find a more real-world spreadsheet and test our model even though we tried as much as possible to see that our dataset has reflected on all the possible scenarios of the real world, but with advances in academia, business, and other industries, soon these datasets will be obsolete.

References

[1] N. Joseph, *Number of Google Sheets and Excel users worldwide* (2021), https://askwonder.com/research/number-google-sheets-users-worldwide-eoskdoxav

[2] A. Mukhtar, B. Hofer, D. Jannach, F. Wotawa, Journal of Systems and Software p. 111119 (2021)

[3] Stuart Leung, *Sorry, Your Spreadsheet Has Errors (Almost 90% Do)* (2014), https://www.forbes.com/sites/salesforce/2014/09/13/sorry-spreadsheet-errors/?sh=6cbe2bb756ab

[4] B. Hofer, A. Riboira, F. Wotawa, R. Abreu, E. Getzner, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **7793 LNCS**, 68 (2013)

[5] A. Zeller, *Learning from 6,000 projects: Mining models in the large*, in *Proceedings - 10th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2010* (2010), pp. 3–6, ISBN 9780769541785

[6] A. Musa, H. Mohamed, M.A. Farouq, M. Hassan, *An Intelligent Plant Dissease Detection System for Smart Hydroponic using Convolutional Neural Network* (2021), pp. 345–351, ISBN 9781665438605

[7] E.O. Jessica, H. Mohamed, S. Ilu Yusuf, M. Hassan, *The Role of Linear Discriminant Analysis for Accurate Prediction of Breast Cancer* (2021), pp. 340–344, ISBN 9781665438605

[8] J.J. Tanimu, M. Hamada, M. Hassan, S.Y. Ilu, *A Contemporary Machine Learning Method for Accurate Prediction of Cervical Cancer*, in *SHS Web of Conferences* (EDP Sciences, 2021), Vol. 102, p. 04004

[9] P. Koch, K. Schekotihin, D. Jannach, B. Hofer, F. Wotawa, IEEE Transactions on Software Engineering **PP**, 1 (2019)

[10] R.R. Panko, Article in Journal of Organizational and End User Computing (2005)

[11] D. Jannach, T. Schmitz, Automated Software Engineering 2014 23:1 **23**, 105 (2014)

[12] R. Singh, B. Livshits, B. Zorn (2017)

[13] R. Abraham, M. Erwig, Journal of Visual Languages Computing **18**, 71 (2007)

[14] C. Chambers, M. Erwig, Journal of Visual Languages and Computing **20**, 269 (2009)

[15] W. Dou, S.C. Cheung, J. Wei, Proceedings - International Conference on Software Engineering pp. 848–858 (2014)

[16] W. Dou, C. Xu, S.C. Cheung, J. Wei, IEEE Transactions on Software Engineering **43**, 226 (2017)

[17] L. Xu, S. Wang, W. Dou, B. Yang, C. Gao, J. Wei, T. Huang, 25th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2018 - Proceedings **2018-March**, 423 (2018)

[18] R. Zhang, C. Xu, S.C. Cheung, P. Yu, X. Ma, J. Lu, Journal of Systems and Software **126**, 87 (2017)

[19] R. Abreu, J. Cunha, J.P. Fernandes, P. Martins, A. Perez, J. Saraiva, Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014 pp. 111–120 (2014)

[20] F. Hermans, M. Pinzger, A. Van Deursen, Proceedings - International Conference on Software Engineering pp. 441–451 (2012)

[21] J. Cunha, J.P. Fernandes, P. Martins, J. Mendes, J. Saraiva, Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC pp. 243–244 (2012)

[22] R. Abreu, J. Cunha, J.P. Fernandes, P. Martins, A. Perez, J. Saraiva, Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014 pp. 625–628 (2014)

[23] J. Cunha, J.P. Fernandes, H. Ribeiro, J. Saraiva, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **7336 LNCS**, 202 (2012)

[24] F. Hermans, E. Murphy-Hill (????)

[25] E. Getzner, B. Hofer, F. Wotawa, Proceedings - 2017 IEEE International Conference on Software Quality, Reliability and Security, QRS 2017 pp. 102–113 (2017)

[26] S. Elbaum, G. Rothermel, S. Karre, M. Fisher, IEEE Transactions on Software Engineering **31**, 187 (2005)