Extracting Software Design from Text: A Machine Learning Approach

Islam Elmasry
Department of Computer Science
Faculty of Computers and Artificial Intelligence
Cairo University
Giza, Egypt
islam.elmasry@grad.fci-cu.edu.eg

Khaled Wassif
Department of Computer Science
Faculty of Computers and Artificial Intelligence
Cairo University
Giza, Egypt
kwassif@fci-cu.edu.eg

Hanaa Bayomi
Department of Computer Science
Faculty of Computers and Artificial Intelligence
Cairo University
Giza, Egypt
h.mobarz@fci-cu.edu.eg

***Abstract*— Extracting software design components from text is a goal of many research works. Utilizing Machine learning techniques for that goal can improve the accuracy of this process instead of using traditional Natural Language Processing (NLP) methods. In this paper, the proposed approach uses machine learning techniques to extract classes and attributes from plain text (e.g. software requirements documents) through two consequent classifiers, the first one classifies each word into a class or not using predefined features, then, the second classifier starts to classify words into an attribute or not. Finally, dependency parsing is used to define a set of rules applied to the document given the extracted classes and attributes to relate the attributes to the classes. One of the contributions of this paper is the created dataset in its final pre-processed form that could make it easier to use in the software design field in the future.**

Keywords*— *Software Design, Machine Learning, Natural Language Processing, Class Diagram.

I. INTRODUCTION

Software engineering is the activity of modeling and building well-documented software [1]. One of the most important phases in the software engineering process is software design. Software design is the process of translating software requirements into graphical models which help programmers in the implementation of the software. Software requirements are collected from users and written in natural language in a document called 'Software Requirements Specification (SRS)', then software architect or designer transforms this document into graphical models such as class diagram represented in Unified Modeling Language (UML). A class diagram consists of classes, classes' attributes, methods, and relationships between classes.

As a new trend in research, Machine Learning (ML) is used for performing some software engineering tasks. One of ML's most used methods in software engineering is Natural Language Processing (NLP) which makes computers able to understand the human language and process it like humans [2]. ML's main idea is to use the large amount of data produced all the time in training algorithms to understand these data and predict other cases.

This paper proposes a machine learning approach for extracting software design components (classes and attributes) from unstructured text, e.g., requirements documents. The rest of the paper is organized as follows; related work presented in Section 2. An overview of the proposed approach, dataset, and preprocessing is introduced in Section 3. Section 4 shows the methods. Results are shown in section 5. The conclusion is presented in Section 6.

II. RELATED WORK

Automating software Engineering activities became a vital area of research, especially with the arising of the age of data and the huge amounts of code and software repositories available online. As natural language is the widely used method of documenting software engineering activities, many attempts have been made by researchers to extract conceptual models such as UML diagrams and software architecture from natural language documents which can be requirements specification documents, design documents, source code files, etc...

In a review by Ozygit et.al. [3], they concluded that rule-based methods are used almost in all research works in the field of extracting conceptual models from the software-related text. Class Diagram was the first item in their list of most extracted models.

On the other hand, Silva et.al. [4] reported that text preprocessing is used more than other techniques such as text vectorization, and TF-IDF (Term Frequency-Inverse Term Frequency) is the most used algorithm in this area. They concluded three challenges in the area of extracting models from a text in software engineering, these are: (a) The complexity of applying NLP techniques to software artifacts due to their unique characteristics. (b) Lack of documented software, or bad documents. (c) Difficulty accessing many documents due to business restrictions.

Various types of models are extracted from software-related text. Entity Relationship Diagram was extracted in [5] from database requirements written in the Serbian language. The elements extracted were entities and relationships. This tool was tested using three test cases, and the results show that the tool can extract 100% of entities correctly, but 61% of attributes and 75% of relationships.

[6] and [7] generated use case models from requirements, [6] suggested a framework that uses Neural Networks for the generation of use cases, but in [7] a rule-based approach was used which can deal with composite sentences/requirements. A test is done with four documents for various software systems resulted in 72% precision and 69% recall.

In [8], they proposed a rule-based framework for extracting the class diagram. The framework can extract classes/objects and their attributes, and methods/operations.

Finally, a road map was drawn, for extracting classes, relationships between them, attributes, and methods using deep learning techniques. In their paper, Rigou et al. suggested using neural networks, especially Recurrent Neural Networks (RNN) for extracting class diagram components for a text describing the software. They had difficulties in finding enough data for building the appropriate dataset for this task as it is known that deep learning techniques need huge amounts of data for training and testing [9].

978-1-6654-4076-9/21/$31.00 ©2021 IEEE

III. Proposed Approach

In this section, the proposed approach is reviewed to achieve the goal of this research. Our goal is to extract software classes and attributes of classes as a preliminary phase. First, the data collection and dataset building processes are introduced. Then conducted experiments and problems are discussed.

A. Data

1) Preprocessing

To carry out this paper, a dataset of 79 software requirements documents is used. This dataset is published by [10] under the name PURE (Public Requirements dataset). Documents in the dataset are representing various software types and are written in an unstructured way with no unique style.

Preparing unstructured textual data for being appropriate to machine learning tasks is a difficult and critical process. In this phase, some steps were followed for preprocessing which are: cleaning up text from useless data and stop words, tokenizing text into sentences once and words once, and tagging each word with its part of speech (POS-tagging). The final size of the dataset after pre-processing is 410027 tokens.

2) Features Selection

In this task, traditional linguistic features (e.g. Part of Speech) are not enough to help in determining if the word is a class or an attribute. So, other features are extracted besides the traditional linguistic features. The process of selecting new features is based on rules used in [11] for extracting classes and attributes from requirements written in Thai. Table I. shows the chosen rules, and features selected based on them.

TABLE I. Features selected based on rules in a previous work [11]

Rule	Category	Feature
The word that appears before an attributive verb can indicate an attribute.	attribute	Attributive verb before
The word that appears after the keyword "(indicate, specify) can be an attribute.	attribute	Keyword before
When two common nouns are written next to each other, the second noun can be considered as an attribute.	attribute	Noun before
When two common nouns are written next to each other, the first noun can be considered as a candidate class.	class	Noun after

The final dataset has two forms in processing, due to the differences between classes features and attributes features. Selected features are shown in the following:

- ***Software type:*** this feature represents the software domain like business, management, education, etc.
- ***Word:*** each word in the text.
- ***POS:*** part of speech tag for each word.
- ***Is noun:*** checks if the word is a noun.
- ***Has 's:*** refers to a possessive noun phrase in the form of noun + apostrophe +s.
- ***Noun after:*** checks if there is a noun after the current word.
- ***Term frequency in the document:*** The frequency of each word in the document
- ***Class before:*** Checks if the sentence of the current word is containing class.
- ***Keyword before:*** Checks if the sentence is containing a keyword before the word (assign indicate, specify...etc.).
- ***Noun before:*** Checks if there is a noun before the current word.
- ***Attributive verb before:*** Checks if there is an attributive verb before the word (i.e. large-sized item, zip-coded email).

B. Workflow

The proposed approach uses two consequent classifiers to extract classes and attributes. After a software document is preprocessed and pos-tagged, feature vectors for the first classifier are fed the machine learning algorithm for building a model which extracts classes. After that, feature vectors for the second classifier are used for building a model which extracts attributes of the classes previously extracted from the first step, as shown in Fig. 1.

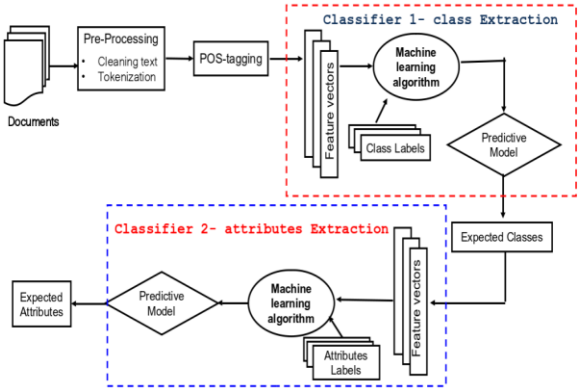

Fig. 1. The sequence flow of proposed approach

IV. Algorithms and Implementation

Carrying out this research require several experiments to be conducted. In these experiments, two traditional machine learning algorithms in addition to a simple neural network, were implemented.

After the classification process is completed, two lists of words, a classes list and an attributes list, are extracted. Then the second task is to assign attributes to classes. This task is accomplished using some rules that applied to the document.

A. Phase1: Classification

In this phase, the two machine learning algorithms and the simple neural network were applied on the dataset in its two forms to classify the document words into classes and attributes. The two algorithms are Support Vector Machine (SVM) and Naive Bayes (NB). Python and Scikit-learn were used in experiments.

1) ***Support Vector Machine:*** Support Vector Machine (SVM) is a non-parametric supervised learning algorithm. SVM can be used for both classification and regression problems [12]. SVM is effective in high dimensional spaces, and also effective if the number of dimensions is greater than the number of samples. But if the number of dimensions is much higher than the number of samples it becomes necessary to avoid over-fitting in choosing regularization term and kernel function [13]. The linear SVM function is represented as:

$$f(x) = \sum_1^N (a_i . a) . \beta_i + c \qquad (1)$$

where a is the test instance, a_i is all of the support vectors in the training set with N input parameters, β is a vector that contains the coefficient that defines an orthogonal vector to the hyperplane, and c is the bias term [14]. In this task, SVM is implemented with linear and random basis function (RBF) kernel functions.

2) ***Naïve Bayes:*** Naive Bayes (NB) is a probabilistic classifier based on the Bayes theorem. NB assumes that features are mutually independent [15]. Bayes theorem can be represented as:

$$P(z|w_1 . \ldots . w_n) = \frac{P(z)\pi_{i=1}^n P(W_i|Z)}{\pi_{i=1}^n P(w_i)} \qquad (2)$$

where:

- z is the response variable.
- w1, …, wn are input variables.
- P(z|w1, …, wn)is the conditional probability of z given w1, …, wn.
- P(z) is the marginal probability distribution of z.
- (wi |z) is the conditional probability distribution of wi given z.
- P(wi) is the marginal probability distribution of wi.

Bayes classifier, is the function that assigns a class label as:

$$z = argmax_z P(z)\pi_{i=1}^n P(w_i|z) \qquad (3)$$

In this task, NB is implemented in Gaussian and Multi-nominal forms.

3) ***Neural Nerwork:*** Neural Networks are a set of calculations or algorithms which try to mimic the human brain. The simple form of NN consists of three parts: Input Layer that receives input data from the dataset, Hidden Layer(s) which does the processing, and Output Layer that shows the output as a value or a vector of values [16]. To perform the processing, hidden layers use an activation function, which describes how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network, and all the hidden layers use the same activation function. Also, output layer uses an activation function, but it must be different from the function used in the hidden layers [17]. The used network consists of 2 hidden layers, each layer has 9 nodes, and the input layer has a number of nodes equuals to the number of features (7 for classes and 8 for attributes). The number of hidden layers and the number of nodes in each layer was chosen after many experiments with various numbers.

The network uses the ReLU (Rectified Linear Unit) activation function in hidden layers and the sigmoid activation function in the output layer as shown in Fig. 2. ReLU was chosen because it is overcoming the limitations of the other functions, so it becomes the default activation function in modern projects. ReLU is calculated as $\mathbf{max}(\mathbf{0.0}, \boldsymbol{x})$ which means that if the value of the input is negative it will return 0, otherwise, it will return the value $(\boldsymbol{x})$.

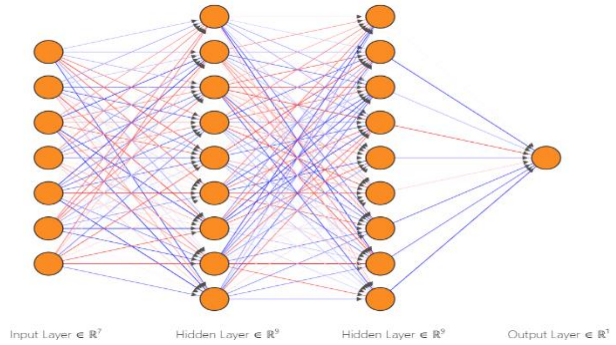

Fig. 2. The Architecture of the used Neural Network

B. Phase 2: Assigning Attributes to Classes

After the classification process is completed, the two lists of words, a classes list, and an attributes list are extracted. The task now is to assign attributes to classes. This task is accomplished using dependency parsing that used for generating rules applied to the document in order to relate the attributes to the classes. Dependency parsing is using the dependency grammar to perform syntactic analysis. Fig. 3. shows the dependency tree of a sentence, and table II. describes the dependencies used in this task. After that, a set of five rules is applied on the sentences that contain any of the classes extracted in phase 1.

Fig. 3. An example for dependency tree of a sentence

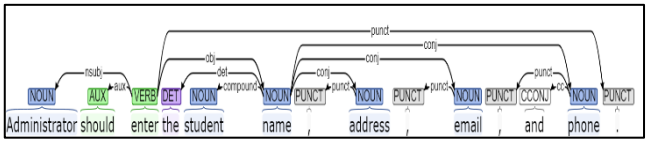

TABLE II. THE USED DEPENDENCIES AND THEIR DESCRIPTIONS

Dependency	Description
Compound	A multiword expression like "Student name" [18].
Conjunct (conj)	A relation between two elements connected by a coordinating conjunction like "and" [19].

The five rules are defined in the following and Fig. 4. shows the pseudo code of their usage:

Rule 1: If the word is a class and the dependency of the next word is 'compound' then the next word is an attribute for it.

Rule 2: If the word is a class and contains apostrophe + s ('s) then the next word is an attribute for it if listed in the attributes list.

Rule 3: If the word is 'of' and the previous word is an attribute and the next word is a class, then the previous word is an attribute of the next word which is a class.

Rule 4: If the word is an attribute and there are other words with the dependency 'conj', then all of them are attributes (if listed in the attributes list).

Rule 5: If the sentence/requirement is composite and contains more than one class, then the attributes detected will be assigned to the nearest class.

```
Input: D: a software requirements document
       c: as a list of classes
       a: list of attributes
Output: A map of classes and their attributes

1. for each sentence in D do
2.     for each word in sentence do
3.         if dependency (word) is compound and word in attribute list then
4.             previous word is a class
5.             word is an attribute for previous word
6.
7.         if word = "of" then
8.             next word is a class
9.             previous word is an attribute of next word
10.
11.        if word = " 's " then
12.            previous word is a class
13.            next word is an attribute of word
14.
15.        if dependency(word) is conj and previous word is attribute for a class then
16.            word is an attribute for the same class
17.
18.        if count(classes) in sentence >1 then
19.            attributes in sentence assigned to the second class
```

Fig. 4. The pseudocode of the used rules

V. Experiments and Results

After conducting the experiments, it has been found that all methods used (machine and deep learning algorithms) can do the task of our approach very well with very close results. Naive Bayes classifier was implemented in two forms: Gaussian and Multi-nominal. Table III. shows the results of the NB experiments.

TABLE III. Results of Naïve Bayes classifier

Measure	Classifier 1 - Classes		Classifier 2 - Attributes	
	Gaussian	Multi-nominal	Gaussian	Multi-nominal
Accuracy	99.98%	98.66%	99.82%	98.42%
Precision	99%	98%	99%	98%
Recall	99%	89%	99%	89%
F-measure	99%	93%	99%	93%

Experiments implemented SVM with two kernel functions: Linear and random basis function. Table IV shows the results.

TABLE IV. Results of SVM classifier

Measure	Classifier 1 - Classes		Classifier 2 - Attributes	
	Linear	Random Basis Function (RBF)	Linear	Random Basis Function (RBF)
Accuracy	99.98%	98%	99.89%	98%
Precision	99%	99%	99%	99%
Recall	99%	80%	99%	80%
F-measure	99%	89%	99%	89%

Implementing Neural Network required a lot of experiments which took a long time due to tuning hyperparameters like the number of hidden layers and the number of nodes, and other parameters like the batch size and epochs number for getting better results. Best results were with the following factors:

- Number of hidden layers = 2.
- Number of nodes in each hidden layer = 9.
- Batch size = 32 and 40.
- Epochs number = 10 and 100.

Table V. shows the used batch sizes and epoch numbers, and Fig.5. – Fig. 20. show the performance curves of each experiment for the both classifiers

TABLE V. Results of Neural Network experiments

Batch Size	Epochs No.	Classifier 1 - Classes	Classifier 2 - Attributes
		Accuracy	Accuracy
32	10	99.21%	99.42%
32	100	99.86%	99.62%
40	10	99.47%	99.23%
40	100	99.77%	99.80%

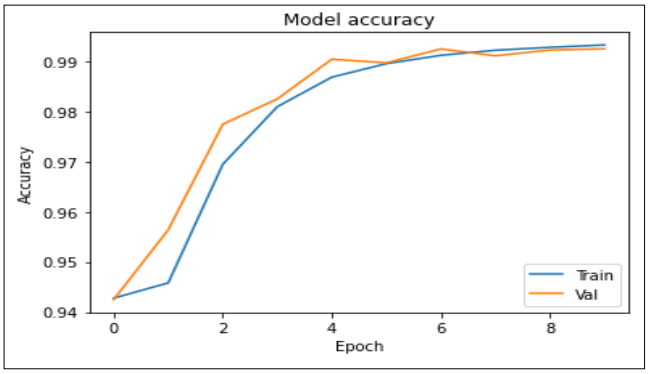

Fig. 5. Classifier 1 - Accuracy (32 batch size and 10 epochs)

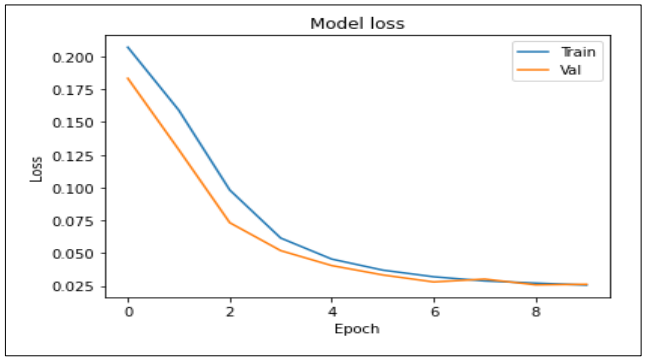

Fig. 6. Classifier 1 - Loss (32 batch size and 10 epochs)

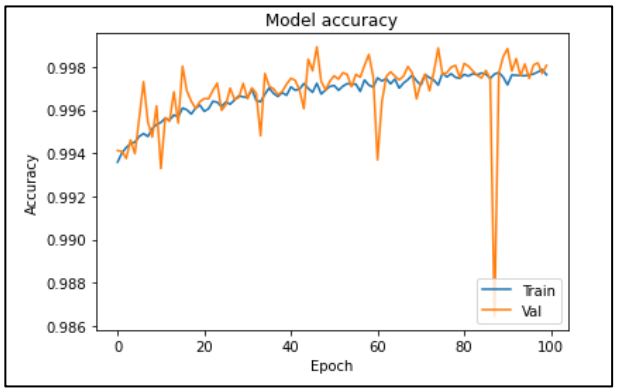

Fig. 7. Classifier 1 - Accuracy (32 batch size and 100 epochs)

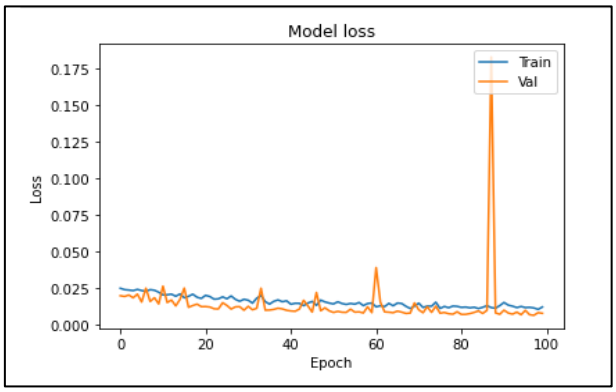

Fig. 8. Classifier 1 - Loss (32 batch size and 100 epochs)

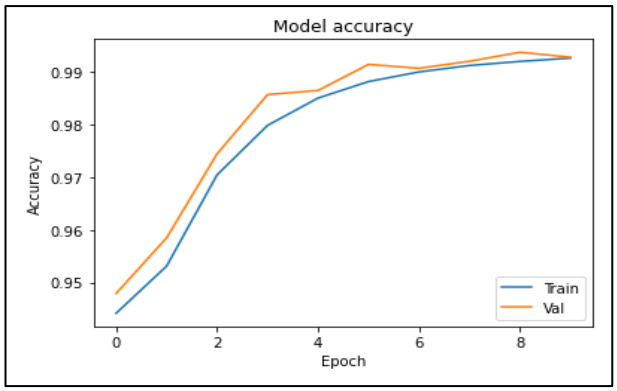

Fig. 9. Classifier 1 - Accuracy (40 batch size and 10 epochs)

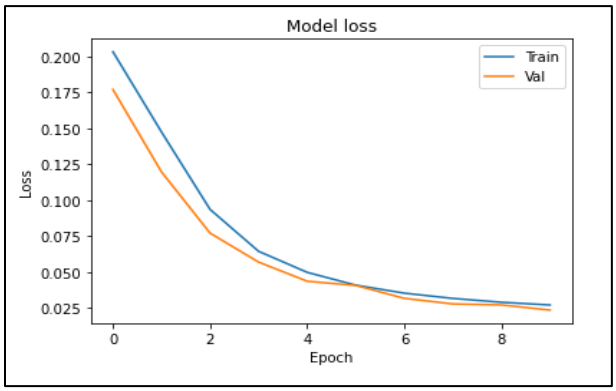

Fig. 10. Classifier 1 - Loss (40 batch size and 10 epochs)

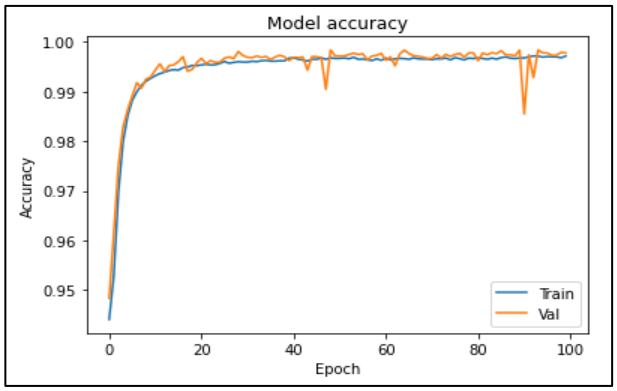

Fig. 11. Classifier 1 - Accuracy (40 batch size and 100 epochs)

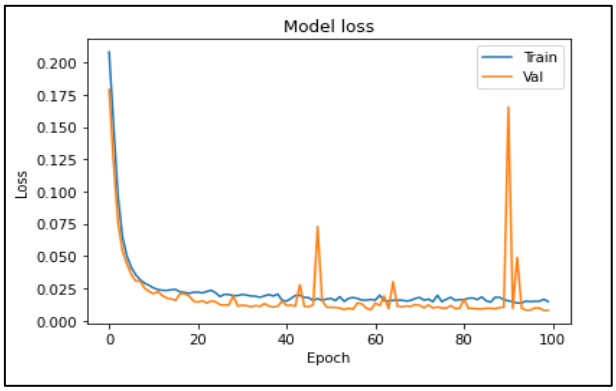

Fig. 12. Classifier 1 - Loss (40 batch size and 100 epochs)

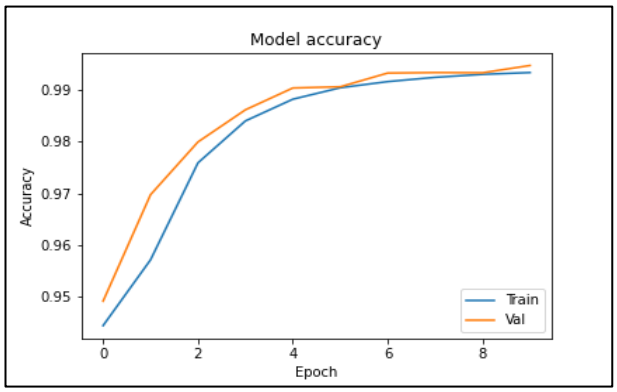

Fig. 13. Classifier 2 - Accuracy (32 batch size and 10 epochs)

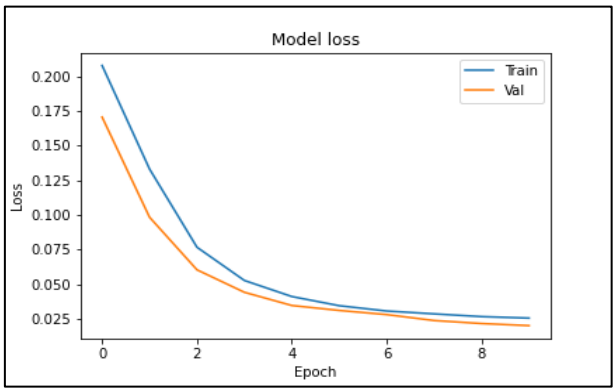

Fig. 14. Classifier 2 - Loss (32 batch size and 10 epochs)

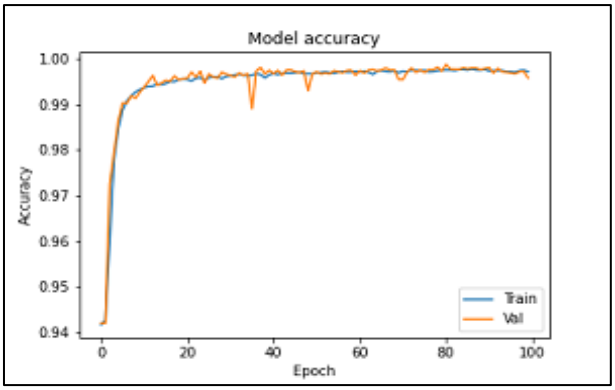

Fig. 15. Classifier 2 - Accuracy (32 batch size and 100 epochs)

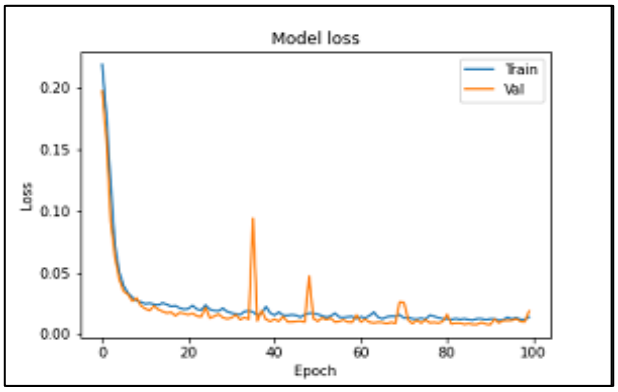

Fig. 16. Classifier 2 - Loss (32 batch size and 100 epochs)

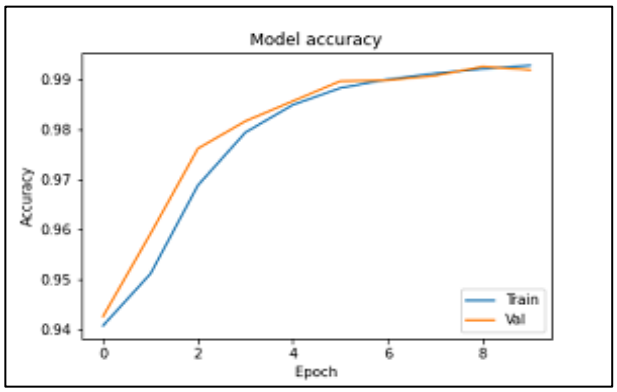

Fig. 17. Classifier 2 - Accuracy (40 batch size and 10 epochs)

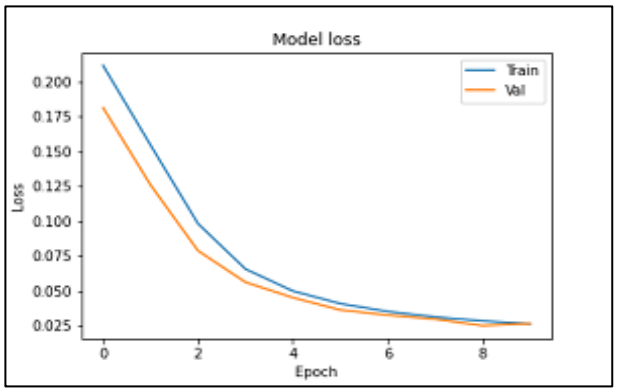

Fig. 18. Classifier 2 - Loss (40 batch size and 10 epochs)

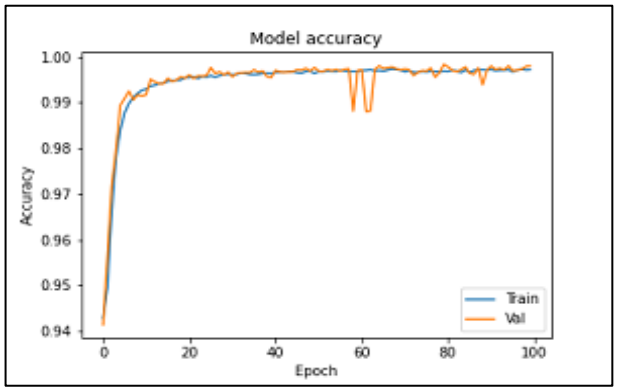

Fig. 19. Classifier 2 - Accuracy (40 batch size and 100 epochs)

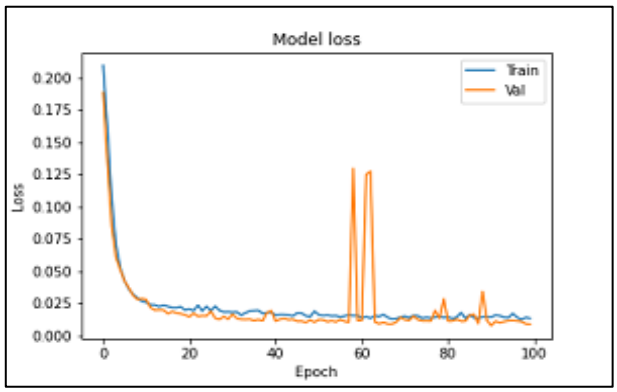

Fig. 20. Classifier 2 - Loss (40 batch size and 100 epochs)

Results may appear to be over-fitted, but this is due to the similarity of most types of software in large parts of the design. For avoiding the over-fitting, some methods were tried such as:

- Changing The size of the dataset.
- Reducing the Dimensionality by removing some features because the less redundant data, the better decision quality [20].
- Regularization: which is any changes to the learning algorithm for lowering its generalization error [21].

It is noticed that precision, recall, and f-measure have the same values in some cases, which can be explained as follows:

$Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$, $F1 = \frac{2TP}{2TP+FP+FN}$

Where TP is True Positive, FP is False Positive, and FN is False Negative.

The three measures can have the same value (0) if TP is 0, But this is not the case here. here, If $FP = FN$, then the three measures will have the same value.

VI. Conclusion

In this paper, a machine learning-based approach is proposed and applied for extracting software design components (classes and attributes) from text such as requirements documents or any other software description text. The greatest challenge in this work was data, due to the limited availability of requirements documents and software descriptions. Therefore, the needed dataset is created from scratch, which is the process that takes most of the work time.

Applying the proposed approach proved that machine learning algorithms and deep learning outperform other tools used in this field. After classifying words into "classes" and "attributes", the dependency tree was used for generating rules to assign the extracted attributes to the extracted classes. In future work, more data will be collected to increase dataset size and investigate new features to apply deep learning algorithms like Recurrent Neural Network and Convolutional Neural Network for extracting other components of the class diagram.

REFERENCES

[1] B. Bruegge and A. Dutoit, Object-Oriented Software Engineering Using UML, Patterns, and Java, 2010, p. 5.

[2] P. Goyal, S. Pandey and K. Jain, Deep Learning for Natural Language Processing, 2018, p. 24.

[3] F. Bozyiğit, Ö. Aktaş and D. Kılınç, "Linking software requirements and conceptual models: A systematic literature review," Engineering Science and Technology, an International Journal, vol. 24, no. 1, pp. 71-82, 2021.

[4] P. Rodrigo da Silva, V. d. Santos, E. Souza, G. Meinerz, K. Felizardo and N. Vijaykumar, "Extraction of Useful Information from Unstructured Data in Software Engineering: A Systematic Mapping," in XXIII Ibero-American Conference on Software Engineering (CIBSE) - Experimental Software Engineering (ESELAW), 2020.

[5] K. Kuk, M. Angeleski and B. Popovic, "A Semi-automated generation of Entity-Relationship Diagram based on Morphosyntactic Tagging from the Requirements Written in a Serbian Natural Language," in IEEE 19th International Symposium on Computational Intelligence and Informatics and 7th IEEE International Conference on Recent Achievements in Mechatronics, Automation, Computer Sciences and Robotics, 2019.

[6] I. A. O. Ahmed and M. E. E. Daleel, "Automated Use Case Diagram Generation with Non-functional Requirements using Neural Network," International Journal of Applied Information Systems, vol. 12, no. 34, pp. 1-4, 2020.

[7] Z. A. Hamza and M. Hammad, "Generating UML Use Case Models from Software Requirements Using Natural Language Processing," in 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO), 2019.

[8] M. R. Sahoo and M. M. Padhy, "NLP based Knowledge Extraction for Software Modeling: A Review," International Journal of Engineering Research & Technology (IJERT) NCRTAPSE, vol. 8, no. 1, 2020.

[9] Y. Rigou, D. Lamontagne and I. Khriss, "A Sketch of a Deep Learning Approach for Discovering UML Class Diagrams from System's Textual Specification," in 1st International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET), 2020.

[10] A. Ferrari, G. O. Spagnolo and S. Gnesi, "PURE: A Dataset of Public Requirements Documents," in IEEE 25th International Requirements Engineering Conference (RE), 2017.

[11] M. Jaiwai and U. Sammapun, "Extracting UML class diagrams from software requirements in Thai using NLP," in 14th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2017.

[12] "SVM Explained," [Online]. Available: https://paperswithcode.com/method/svm. [Accessed 22 June 2021].

[13] "Support Vector Machines - Scikit-Learn 0.24.2 Documentation," [Online]. Available: https://scikit-learn.org/stable/modules/svm.html. [Accessed 22 June 2021].

[14] H. Kour, J. Manhas and V. Sharma, "Comparative Study of Different Machine Learning Techniques in the Diagnosis of Dementia," in Rising Threats in Expert Applications and Solutions, 2021.

[15] M. Maniruzzaman, M. Rahman, B. Ahammed and M. M. Abedin, "Classification and prediction of diabetes disease using machine learning paradigm," Health Imformation Science and Systems, vol. 8, no. 7, 20.

[16] S. Sinhal and R. Nanda, "Understanding the Role of Artificial Neural Networks in the Prediction of Mental Health Diseases," in Rising Threats in Expert Applications and Solutions, 2021.

[17] J. Brownlee, "How to Choose an Activation Function for Deep Learning," 18 January 2021. [Online]. Available: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/. [Accessed 4 July 2021].

[18] "Compound," [Online]. Available: https://universaldependencies.org/u/dep/compound.html. [Accessed 26 June 2021].

[19] "conj," [Online]. Available: https://universaldependencies.org/u/dep/conj.html. [Accessed 26 June 2021].

[20] J. Brownlee, Machine Learning Mastery With Python, 2016, p. 52.

[21] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, Massachusetts Institute of Technology, 2016, p. 117.