JCBI
Journal of Computing & Biomedical Informatics
ISSN: 2710 - 1606
Volume 07 Issue 02
2024
Research Article
https://doi.org/10.56979/702/2024

A Rule-Based Approach for Automatic Generation of Class Diagram from Functional Requirements Using Natural Language Processing and Machine Learning

Muhammad Ramzan[1], Ghunwa Saeed Sadiqi[2], Muhammad Salman Bashir[2*], Summair Raza[1], and Asma Batool[2]

[1]Department of Software Engineering, University of Sargodha, Sargodha, Pakistan.
[2]Department of Computer Science and Information Technology, Virtual University of Pakistan.
.[*]Corresponding Author: Muhammad Salman Bashir. Email: salmanbashir@vu.edu.pk

Received: March 12, 2024 **Accepted:** August 03, 2024 **Published:** September 01, 2024

Abstract: Requirement analysis is the initial and most crucial phase of the software development life cycle (SDLC). In this phase, the requirements after gathering from the user and different stakeholders are evaluated and abstraction is created in terms of a model. The generation of UML class diagrams from requirements is a very time-consuming task and hence demands the automation of the process. The researchers have proposed a number of tools and methods for the transformation of natural language requirements to UML class diagrams in the last few years. Different approaches like Natural Language Processing (NLP) and Rule based approaches were used for this purpose, but they have certain limitations. Moreover, these approaches do not extract all the relationship types of class diagrams. To resolve this issue machine learning based approaches have been used for a few years. Machine learning requires large and precise datasets to train models. In this research, a new model is proposed to generate class diagrams from requirements written in natural language more accurately using Natural Language Processing as well as the machine learning approach. NLP has helped to extract the classes, attributes, and methods while machine learning is used to extract the class relationships. To implement machine learning models we have created a dataset containing class names and relationship types i.e. aggregation, association, composition, and inheritance. The effectiveness of models is analyzed by comparing the results using accuracy metrics.

Keywords: UML Class Diagram; Machine Learning; Dataset; Class Relationships; Aggregation; Association; Composition; Inheritance.

1. Introduction

Requirements engineering is a very important yet problematic step as it requires extra time and effort for requirements elicitation and verification. Requirements that are elicited in this phase are recorded in a document called software requirements specification (SRS) in natural language. These natural language requirements can be gathered from several sources like users, existing documents, and notes. Natural language is ambiguous and inconsistent in nature and hence the requirements written in natural language are also inconsistent, incomplete, and ambiguous in nature. Hence these requirements can be interpreted differently by different analysts. Moreover, different factors like psychological, sociological and geographical can also affect the understanding of requirements. Hence before developing the system it is

the best idea to transform these requirements into less ambiguous formal language. Requirement analysts analyze the requirements in order to find and fix inconsistencies and ambiguities. However, a human analyst can also miss defects or can interpret a statement differently when he does not have enough domain knowledge. One of the main artifacts of object-oriented design is UML class diagrams because many other models can be derived from UML class diagrams. Class diagram shows the static view of the system. Figure 1.1. shows the standard class diagram.

1.1.1. Class Notation

A class diagram consists of different classes and a standard class notation consists of three sections:

1. Upper most section contains the class name.
2. Middle section consists of attributes of classes. Attributes type is separated by the colon.
3. Bottom section contains class operations. These are the services provided by the class.

1.2. Class Relationships

Every class is related to other classes in some ways. Relationships in class diagram represent the type of connection that exist among different classes. Such types of relationship are discussed in Table 1.1.

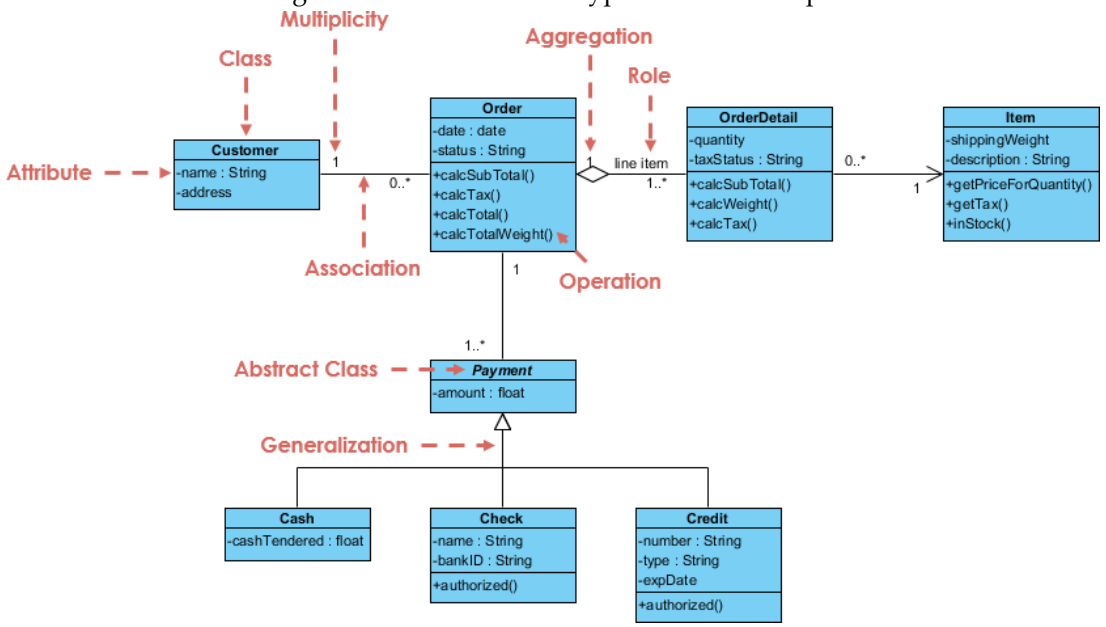

Figure 1. Standard Class Diagram

Table 1. Types of Class Relationships

Relationship Type	Purpose
Inheritance/ Generalization	It represents "is a" relationship.
Simple Association	Represents structural link between classes.
Aggregation	Represents "part of" relationship. Special type of association
Composition	Special type of aggregation. Parts destroys when whole destroy.
Dependency	Represents that changes in definitions of one class may change the other too.
Multiplicity	Represents number of instances involved in association.

Transforming natural language requirements to UML class diagram is a challenging task and demands for some automation in the process. Different approaches are proposed for the transformation of requirements into UML class diagrams. Natural language processing techniques, Rule based techniques and machine learning based approaches are remarkable ones. Many researchers have used these approaches and developed different tools to support the automation process. These include UMGAAR [1],

LIDA [2], CM Builder [3], DC Builder [4], GOOAL [5], READ[6], and RAUE [7] etc. These tools use NLP technique to evaluate the natural language requirements and transform them into class diagrams. NLP is a text processing technique. It involves different phases like parsing, text analysis and entity recognition. But it is not competent enough to extract all class diagram components [8]. Experiments show that these tools do not generate accurate class diagrams. Besides, most of the approaches do not extract all the elements of class diagrams like classes, attributes, methods and particularly advanced relationship types.

In Rule based approaches different rules and pattern are defined for the extraction of relevant information. Designing of rules require adequate domain knowledge and knowledge of software engineering concepts [9], [10]. While offering some benefits, Rules based approaches have their own limitations too. These approaches lack in flexibility. As rules are predefined hence it is difficult to handle variations in requirements. Rules are manually defined and it is time consuming and laborious task to define rules. Different interpretations of natural language requirements can lead in inconsistent and inaccurate results. Complex relationships can be difficult to capture. Requirements containing associations, multiplicity or inheritance can be challenging and may result in incorrect or incomplete class diagrams.

To resolve these issues machine learning approaches or hybrid approaches are used that combines both rule based and machine learning approaches.

1.3. Motivation and Problem Statement

The goal of this research study is to develop a system for the generation of class diagram using machine learning by identifying the type of relationship among classes as well as with the help of some NLP and rule based approaches so that it can help software designers and researchers to generate UML class diagrams effectively.

1.4. Contributions

This research study enabled us to propose a model to generate class diagrams from textual requirements using NLP and machine learning. The main target of the research is to extract relationships among classes particularly association, inheritance, aggregation and composition. A dataset is created to implement the machine learning models.

2. Related Work

Many methods and tools have been proposed in literature in the last years for creating UML class diagrams from NL requirements. Here the evaluation of some of these approaches is presented.

Bashir and Bilal [6] proposed a system called Requirements Engineering Analysis and Design (READ) to transform natural language requirements to UML class diagrams. They used the technique of NLP as well as domain ontology. This system was implemented in Python to generate class diagram by extracting classes, attributes and methods successfully. The authors implemented the feature of strong and weak threshold to refine the elements.

Abdelnabi et al. [12] studied and analyzed the existing tools and approaches for efficiency, completeness and automation degree. Authors identified the strengths as well as weaknesses of these models. All these models used different techniques and rules to extract different elements of class diagram.

Abdelnabi et al. [9] proposed a process to generate class diagrams from natural language requirements with the use of NLP technique and some heuristic rules for transformation process. The suggested approach works in five different phases. The methodology uses NLP technique and type dependency for parsing the natural language specifications. The approach generates class diagrams by extracting a variety of class elements like classes, attributes, methods and different types of relationships like associations, aggregations, composition, generalization, and multiplicity.

Nasiri and Rhazali [11] proposed a model based on transformation from computation independent model to platform independent model. For these transformations they created a platform that generates

class diagram from requirements written in natural language. For extracting the class elements, they used a tool called Stanford CoreNLP [13] the technique got the advantage of treating sentence structure of different types. Moreover, it manages the compound nouns for the extraction of classes efficiently.

Hnatkowska et al. [14] proposed a method to construct conceptual model using data frames. The data is analyzed and functional dependencies of data are recorded. This analyzed data helps in identifying classes, attributes of classes and their relationships. The explanation of method was given through a case study by real data sets processing. The quality of conceptual model depends mainly upon the quality of input data. This model facilitates in two perspectives, first it checks data quality and second it finds relationships among entities.

Utama and Jang [15] designed an algorithm that takes natural language problem statement as input and generates class diagram out of it. As class diagram comprises of class names, attributes and operations, hence this algorithm extracts all the mentioned components. The algorithm was evaluated by taking five statements from different context and class diagram was generated. This approach has recall value of 100% and precision of 92%.

Mohanan and Samuel [16] developed a tool called SUCM for the transformation of natural language requirement specification document to class diagram. The tool extracts the components of UML diagram for the generation of class diagram. But it does not represent all the aspects of system as it only supports the generation of class diagram which shows only the static aspect of system.

Jaiwai and Sammapun [17] proposed in their research an approach for the transformation of requirements from Thai language into UML class diagram. Few rules are formulated which are used for the extraction of classes and their attributes from requirements written in natural language. There are flaws in this model, it was unable to extract the operations and all types of relationships of class diagram. There is a need for other rules and pattern able to extract operations and relationships for class diagram completion.

Narawita and Vidanage [18] modified and reconstructed the UML Generator as it faced many problems and challenges. At present the UML Generator is intelligent enough to obtain the UML elements and is able to generate Use Case diagram as well as class diagram from user input resulting in the less time and cost from user and system analyst. Furthermore, as the system generates both the use cases and class diagrams, hence it provides both dynamic and static aspect of the system.

Zhong et al. [19] proposed an approach to automatically generate systems diagrams from natural language requirements. The researches proposed open domain and flexible approach. The prosed approach consists of six steps which leverage open-access tools and results in the generation of SysML diagrams. The approach uses set of five Hyperparameters specified by the user. The researchers used six case studies from different sources as input to show the benefits and applicability of the approach.

Zhang et al. [20] used the SemEval-2010 Task 8 database to evaluate a learning model. This dataset has 9 relation types and one "other" type. The nine types are Product-Producer, Cause-Effect, Content-Container, Instrument-Agency, Entity-Origin, Message-Topic, Component-Whole, Entity-Destination, and Member-Collection. Considering active and passive nature, the nine relation types are directed and the 'other' which is non-directional, makes a total of 19 types of relations. SemEval-2010 Task 8 dataset has 2717 sample test sets and 8000 sample training sets.

Yang and Sahraoui [21] presented a fully automated approach for the generation of class diagrams from natural language requirements. The proposed approach is based on natural language patterns as well as machine learning. For the machine learning models, authors have also created a dataset of 62 labeled diagrams that contain 624 fragments. The dataset does not contain labels for types of relationships. The proposed approach showed low precision and recall but it was a good benchmark for machine learning-based class diagram generation.

Patel and Priya [22] aimed to resolve the issue of ambiguity found in natural language. They aimed to take requirement specification as textual information, remove the ambiguity and uncertainty found in natural language, and then identify the components required to generate UML diagrams. There are different types of ambiguities but in the paper, the authors provided an approach to resolve Referential and Coordination ambiguities. Other types of ambiguities still need to be resolved.

Shweta et al. [23] developed a transformer based model to extract the components of class diagram from natural language requirements as these models generate context-dependent embedding. The results of this model were compared with the existing procedure and 9-7% enhancement was observed.

Malik et al. [24] proposed an automated approach to generate use case diagrams independent of the formalism. The approach employs a combination of NLP and network science to produce primary and external actors. It produces results in three phases, of which two use NLP and the third uses network science to generate results.

Ouaddi et al. [25] explored the advancements in the field and proposed an approach to extract the use-case diagram from natural language requirements using a chatbot. They used chatbot for the accomplishment of tasks like text summarization and analysis.

Almazroi et al. [26] proposed a method to present the class diagram according to the defined configuration and to extract the relationship between textual instruction and UML class diagram. They enabled the developers to analyze the requirements efficiently by utilizing NLP.

3. Methodology

It is very important to develop a system that can generate class diagrams from requirements effectively and accurately. As there exists no such considerable system that incorporate relationship types for generating class diagrams using machine learning. Hence we developed a machine learning model for the prediction of relationship types so that the class diagrams can be generated more effectively. We used hybrid approach for overall class diagram generation task. It includes Rule based NLP approach as well as machine learning model. The whole process consists of following modules. Input Acquisition module, NLP module, Machine learning module, Knowledge extraction module, and Class diagram generation module.

3.1. Input Acquisition Module

As the class diagrams will be generated from textual requirements provided by the user hence the very first step of our approach is getting input from user in the form of textual requirements written in English language. Input Acquisition module will enable user to input different case studies or scenarios as requirements for class diagram generation.

3.2. NLP Module

The input that is provided in Input Acquisition module will then enter in NLP module. The main purpose of NLP is to make computers able to understand and process text. To understand a text means to recognize the actual context, to perform semantic, syntactic, and lexical analysis, to create summary and extract information. The information extracted by NLP will be used by knowledge extraction module for further processing. NLP module performs its task in different steps by using different **NLP approaches** [9]. These approaches are the vital part of NLP module. The preprocessing techniques that are used by our NLP module are discussed below.

3.2.1. Spelling Checking and Correction

The first and most basic step in NLP is spelling checking and correction. Spell checking system enables us to check and find any misspelled words in the user input. If any word is found misspelled, it correct that misspelled word. Spell checking system can also suggest list of corrections for the misspelled word. For example if the user enters a misspelled word “wlak” instead of “walk”, the spell checking system will

correct the mistake and make that word "walk". We can also get the list of suggested words for correction. The list of candidate words will be displayed like {'walk', 'weak', 'flak'}.

For spell checking and correction purpose python package 'pyspellchecker' is used. This package provides the facility to find and correct misspelled words. The 'spellchecker' library uses a pre-built dictionary for the identification and correction of errors. The library is specially designed to handle spelling correction tasks. It functions independently and does not depends on any external lexical resources like WordNet.

3.2.2. Tokenization

Tokenization is the process in which the text which is generally in the form of sentence is split in sequence of tokens like words. After spelling correction tokenization is the main step in NLP processing pipeline. Tokens are considered as atomic units in processing. These can be words or subword units called morphemes or even individual characters. For example, if we input the statement "Teacher teaches the students" the tokenization module will split this into tokens as <Teacher> <teaches> <the> <students>.

In the proposed system we have implicitly performed tokenization as part of spaCy processing pipeline which is a good approach for solving problems.

3.2.3. Lemmatization

Lemmatization groups together different transformed forms of same word. It is widely used in NLP. Lemmatization reduces the word to its root form. Another approach 'stemming is also used for same purpose. But stemming is heuristic approach and can result in wrong or nonsensical words. On the other hand lemmatization analyzes the contextual meaning, structure and intended POS to generate accurate and meaningful results. Therefore lemmatization is often preferred instead of stemming.

Lemmatization is performed using spaCy. spaCy focuses on lemmatization as it is more advanced and accurate technique to reduce words to dictionary forms. For example, in a sentence the word "teach" can appear in different forms like "teaches", "taught" and "teaching". Lemmatization groups all these words as a single word teach.

3.2.4. POS tagging

The last step of NLP module is 'Parts of Speech' tagging. This phase works on the output of lemmatization. It assigns a Part of Speech category to each lemmatized token in a text e.g. noun, verb, adjective, preposition etc. The spaCy library is used for POS tagging. The en_core_web_sm model is an English Pretrained model. It allows to perform different tasks like POS tagging without need to train own model. For example, the POS tagging model will assign tags to statement tokens "Teacher teaches students" as "teacher/NN teach/VB student/NN. The description of each tagging symbol is explained in Appendix 0-I of APPENDICES Section. Figure 4.1 and 4.2 show result of NLP module.

3.3. Machine Learning Module

In the era of digitization, where data is accumulated at an exceptional speed, the capability to extract meaningful insights and predictions from this data has become paramount. Machine learning encompasses various techniques, including supervised learning (where models learn from labeled data), unsupervised learning (where models identify patterns without labeled data), semi-supervised learning (Where models learn with some labeled and a lot of unlabeled data) and reinforcement learning (where models learn by interacting with an environment). We have used supervised learning method in machine learning module. Supervised learning models require the labeled output variable as well as input variable for training samples. Hence each training sample consists of a pair of input and output variable values. The dataset that we created for machine learning module also consists of input and output pairs. The two class names are input for the model while the relationship between those classes is the output value. The model is trained for the prediction of relation among classes that are extracted from NLP knowledge extraction

module. In machine learning module we have used different algorithms to get a good result. Here we will describe the overall steps in machine learning modules as well as the algorithms that we tried and tested.

3.3.1. Dataset Analysis

The dataset that we created consists of an Excel file containing three columns: Class 1, Class 2, and Relationship. The Class 1 and Class 2 columns represent the names of two classes involved in a relationship, while the Relationship column denotes the type of relationship between them. There are four relationship types that we collected in dataset. These are association, aggregation, composition, and inheritance. Dataset contains 2109 elements containing 482 association, 434 aggregation, 620 composition, and 573 inheritance relationships.

3.3.2. Dataset transformation/preparation

The next step in machine learning is data transformation / preparation. As mostly machine learning models cannot operate on categorical variables. They require numerical variables for their operation. We have used a combination of both label encoding and one hot encoding. One hot encoding was used for input variables i.e. class1 and class 2.While label encoding was employed for output variable i.e. Relationship. The reason why we used one hot encoding for class data is that it efficiently transforms categorical data into binary format which allows for distinct representation of each class in a multidimensional feature space. On the other hand, our decision to use label encoding for relationship labels preserves the inherent order or hierarchy of relationship types, which is particularly relevant in scenarios where the relationship types hold a specific ranking or importance. To perform one hot and label encoding we used the scikit-learn library. The library is a powerful tool for machine learning as it provides a wide range of algorithms for preprocessing as well as for machine learning.

3.3.3. Dataset splitting

As in ML module we have created a single dataset, hence the next step is to split the dataset in 2 smaller parts. These are train and test split. For this purpose we split the dataset that we created into two subsets. The division for this purpose was is done in proportions as 80% train data and 20% test data.

3.3.4. Machine Learning Algorithms Used

In this section we will explain which have been the three algorithms that have been tried in order to solve this problem. Here, the theory and the implementation are discussed, while the results of their implementation will be discussed in next chapter.

- Random Forest Classifier
  - Algorithm

The Random Forest classifier, a prominent ensemble learning technique, has been adopted as a essential component of our methodology. Comprising multiple decision trees, the Random Forest algorithm effectively mitigates overfitting and enhances predictive accuracy through the wisdom of the crowd. Each decision tree is constructed on a subset of the training data, introducing diversity in the model's learning process. Subsequently, predictions from individual trees are integrated using majority voting, ultimately yielding a consolidated and reliable prediction [27]. This ensemble approach enables our model to capture intricate relationships within the dataset, contributing to its robustness and adaptability to complex scenarios. Greater the number of trees in forest higher is the accuracy and lower the problem of over fitting.

The working of Random Forest classifier is explained in figure 3.1 below.

  - Implementation

In our code implementation, The Random Forest model is trained using the fit method on the encoded features (x_enc) and target labels (y_enc). The predict_proba method is used to obtain class probabilities for each instance in the training data. We instantiated a Random Forest classifier with hyperparameters such as 'n_estimators' and 'max_depth', controlling the number of trees in the forest and the depth of

individual trees respectively. By training the Random Forest on our preprocessed class features and utilizing its 'predict_proba' function, we obtained class probability estimates for our machine learning model. Overall, the integration of Random Forest classifier enhances the accuracy of our classification model and contributes to the stability of the results.

- Multinomial Naïve Bayes Classifier

 Algorithm

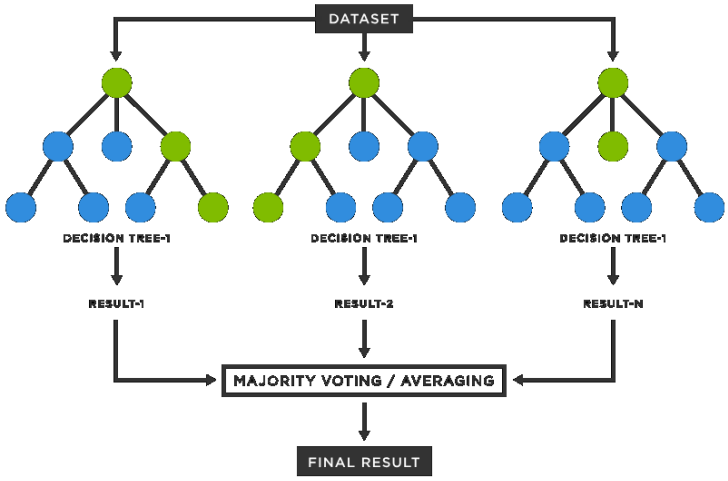

Figure 2. Working of Random Forest Classifier

The Naive Bayes algorithm, rooted in probabilistic principles, plays a pivotal role in our methodology as an elegant solution for multi-class classification. Its foundation lies in Bayes' theorem, a fundamental theorem in probability theory. Naive Bayes leverages this theorem to estimate the probability of a certain class given the observed feature values, incorporating the assumption of feature independence.

Mathematically, for a given set of class labels (y) and features (x_1, x_2, ..., x_k), the Naive Bayes classifier calculates the posterior probability of a class given the features as:

$$P\left(\frac{y}{x_1,x_2,\ldots,x_k}\right) = \frac{P\left(\frac{x^1}{y}\right)\times P\left(\frac{x^2}{y}\right)\times\ldots\times P\left(\frac{x_k}{y}\right)\times P(y)}{P(x_1,x_2,\ldots,x_k)} \quad (1)$$

In this equation, $P(y \mid x_1, x_2, \ldots, x_k)$ represents the probability of classy given the feature values, and $P(x_i \mid y)$ represents the probability of feature x_i given classy. The naive assumption of feature independence allows us to factorize the joint probability $P(x_1, x_2, \ldots, x_k)$ as a product of individual probabilities [28] [33].

 Implementation

In our implementation, the Multinomial Naive Bayes variant is employed, which is tailored for discrete features like word frequencies in text classification. The classifier is initialized, and the fit method is used to train the model on the training data.

The combined methodology utilizes the strengths of both the Random Forest and Multinomial Naive Bayes algorithms for relationship prediction. The Random Forest model provides a powerful ensemble approach, while the Naive Bayes classifier brings probabilistic modeling capabilities. . The accuracy and generalization of the models are assessed using the test data split. This approach aims to accurately predict relationships between entities, contributing to the field of machine learning and data analysis.

- SVM

 Algorithm

The second algorithm that we tried is SVM or Support Vector Machine. SVM is a well-known Supervised Learning algorithm. It is used to solve classification and also the regression problems. In Machine Learning, it is basically used for classification problems. SVM algorithm aims at creating a decision boundary or a best line boundary, which can separate n-dimensional space into classes for us to place the new data point in the right category easily later on. These best line boundaries are called a

hyperplane. SVM selects the vectors/ extreme points which assist in creating the hyperplane. These points are known as support vectors, coining the term Support Vector Machine for the algorithm [29] [32].

Mathematically, given a set of training examples represented as pairs of feature vectors (x_i) and corresponding labels (y_i), where i = 1, 2, ..., n, and $y_i \in \{-1, 1\}$ for binary classification or $y_i \in \{1, 2, ..., K\}$ for multi-class classification, the SVM aims to solve the optimization problem:

$$Minimize: \frac{1}{2}||w||^2 + C\sum_i \xi_i \quad (2)$$

$$Subject\ to: y_i(w.x_i - b) \geq 1 - \xi_i, \forall_i = 1,2,3,\dots,n \quad (3)$$

$$\xi_i \geq 0, \forall_i = 1,2,3,\dots,n \quad (4)$$

Here, w is the weight vector that defines the hyperplane, b is the bias term, C is a regularization parameter controlling the trade-off between maximizing the margin and minimizing the classification error, and ξ_i are slack variables that allow for misclassifications. The SVM aims to minimize the L2-norm of w while ensuring that each training example is correctly classified or within a certain margin. The below figure 4.9 shows two categories which are classified using a hyperplane.

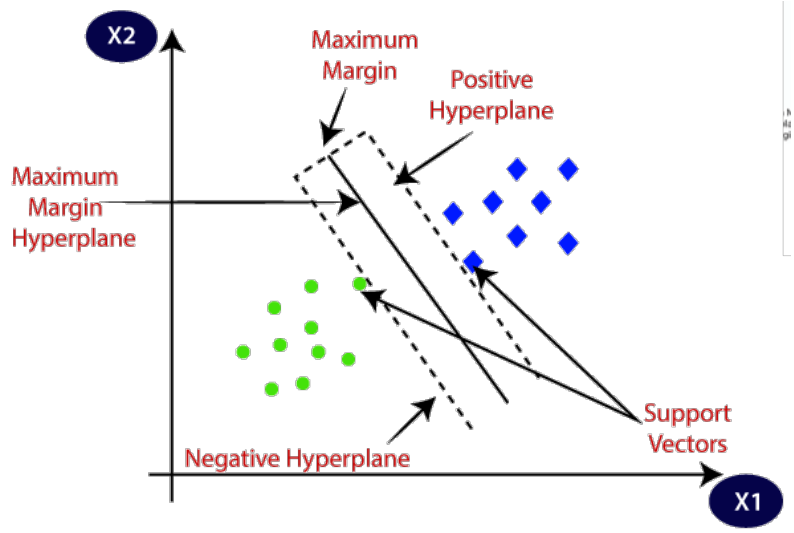

Figure 3. Classification using Hyperplane

- Implementation

In our implementation, SVM was chosen as a robust model to predict the relationship types based on the class features. Utilizing libraries such as scikit-learn, we instantiated an SVM classifier and explored different kernel functions, such as linear and radial basis function (RBF), to tailor the model to our dataset's characteristics. Through feature preprocessing, including one-hot encoding for class features and label encoding for relationship types, we prepared the data for SVM training. The SVM's inherent ability to capture complex decision boundaries was instrumental in effectively categorizing relationship types based on the provided class information.

- KNN
- Algorithm

K-Nearest Neighbors (KNN) is a versatile and intuitive classification algorithm that excels in scenarios where data points with similar features tend to share the same labels. At its core, KNN predicts the label of a data point by examining the labels of its k-nearest neighbors in the feature space [30]. The algorithm's simplicity lies in its reliance on local information to make predictions, making it robust in handling complex decision boundaries and non-linear relationships.

- Implementation

In our implementation, the KNN algorithm played a pivotal role in predicting relationship types based on the provided class features. Following a preparatory step where we split our data into training and testing sets, we instantiated the KNeighborsClassifier from scikit-learn. The parameter 'n_neighbors'

was set to 100, indicating that the classifier would consider the labels of the 100 nearest neighbors when making predictions.

3.4. Knowledge Extraction Module

In knowledge extraction module components of class diagram i.e. class names, methods and relationship types are identified. This modules takes the output of NLP and machine learning dataset as input for the extraction of class components. Knowledge extraction module has three phases.

3.4.1. Class Extraction

In the first step of knowledge extraction module, classes are identified. Classes are generally extracted from nouns. Hence all the common noun <NN> and proper noun <NNP> (e.g. human, system) tags are extracted mapped as classes.

In a sentence of the form (Sub-Verb-Obj); the subject and object are extracted and mapped as classes.

In a noun phrase like (Noun +Noun); the first noun is mapped as a class e.g. “Patient Name”

3.4.2. Methods Extraction

Methods identified by the system are the verb phrases of the sentence associated with the class in the form NP: VP.

In a verb phrase which consists of a lexical verb associated with a noun, then the verb is mapped as a method of the noun which is a class.

In a verb phrase which consists of an action verb associated with a noun, then the verb is mapped as a method of the noun which is a class

In a sentence of the form (NN + VB + NN + NN + NN) the VB is mapped as candidate method of the class.

Verb phrases of the form (VB + NN) are mapped as candidate methods of the subject as class.

3.4.3. Attribute Extraction

All the adjectives in a sentence in the form of <JJ> tag are extracted and mapped as attribute of the class.

In a noun phrase like (Noun +Noun); if first noun is a class then second noun will be mapped as class attribute e.g. “Patient Name” Attributes are also identified by common nouns as well as nouns that come after possessive pronoun <PP$> tag.

3.4.4. Relationship types Extraction

The type of relationship is extracted by machine learning module. The classes that are identified above using rules are passed to machine learning module. The classifier that we developed will be applied on these classes and based upon the experience it gained through training data, it will predict the type of relationship among these classes.

For example the predicted relationship type between class “Human” and “Heart” is “Composition” as predicted by the generated classifier model. Figure 4.10 shows the predicted relationship type among “Human” and “Heart”

3.5. Diagram Generation Module

The final phase of the proposed system is Class Diagram generation. The knowledge extracted in previous phase becomes the input of diagram extraction module. By using class names and relationship types, the module generates class diagram in the format of plan UML diagrams.

The system then shows a success message and asks to save the diagram in system. On confirmation our system generates a file in the format. wsd which can be opened in VS code and diagram can be visualized.

4. Results and Discussions

At initial we employed the ML models directly after generating features through encoders and implementing train test split model. The models employed in this way did not performed up to the mark. The accuracy obtained by different models is presented in Table 4.1.

Table 2. Results of ML models without using Random Forest Classifier

Data split (Train: Test)	Multinomial Naïve Bayes	SVM	KNN
90:10	62	63	43
80:20	61	62	37
70:30	56	58	40

As it can be clearly seen through the table that none of the model showed good accuracy while the KNN model performed the least. To solve the issue we combined these models with Random Forest classifier is particularly designed in a way that it enhances accuracy by using multiple decision trees.

4.1. Results of ML models without Random Forest Classifier

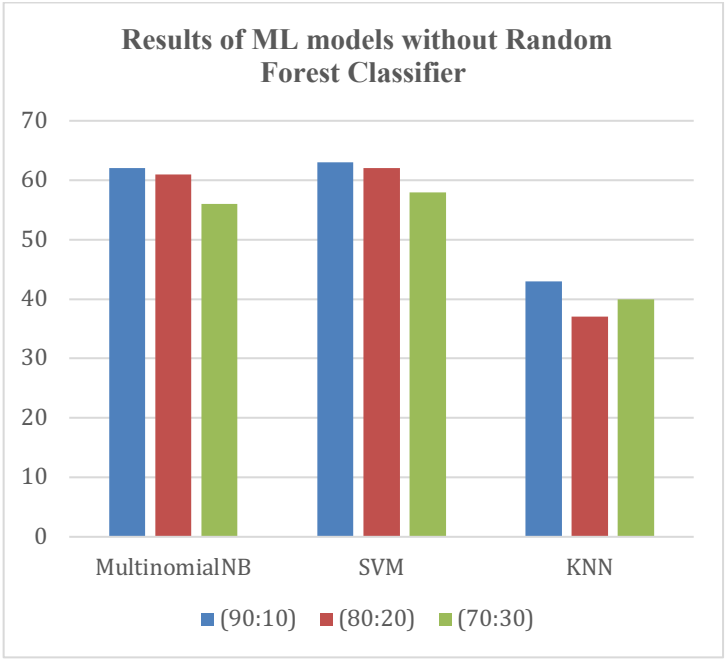

Figure 4. Bar Graph showing results of ML models without Random Forest Classifier

4.1.1. Confusion Matrix for SVM:

The confusion matrix for SVM without applying Random Forest Classifier is as below:

[[23 5 18 3]
[5 33 2 7]
[10 8 36 2]
[1 16 0 42]]

4.1.2. Number of Inaccurately classified records:

As the accuracy of models without applying random Forest Classifier was low hence the number of records that were inaccurately classified by the models was high. The number of inaccurately classified records for SVM for 90:10 train test split is 77.

Number of records inaccurately classified by SVM: 77

4.2. Results of ML models combined with random forest classifier

As the accuracy of models was not good at initial employment, hence we combined these models with Random Forest classifier. . By training the Random Forest on our preprocessed class features and utilizing

its 'predict_proba' function, we obtained class probability estimates for our machine learning model. The combined models showed enhanced accuracy as shown in Table 5.2 below.

Table 3. Results of ML models combined with random forest classifier

Data split (Train: Test)	Multinomial Naïve Bayes	SVM	KNN
90:10	95	96	95
80:20	95	95	95
70:30	94	94	93

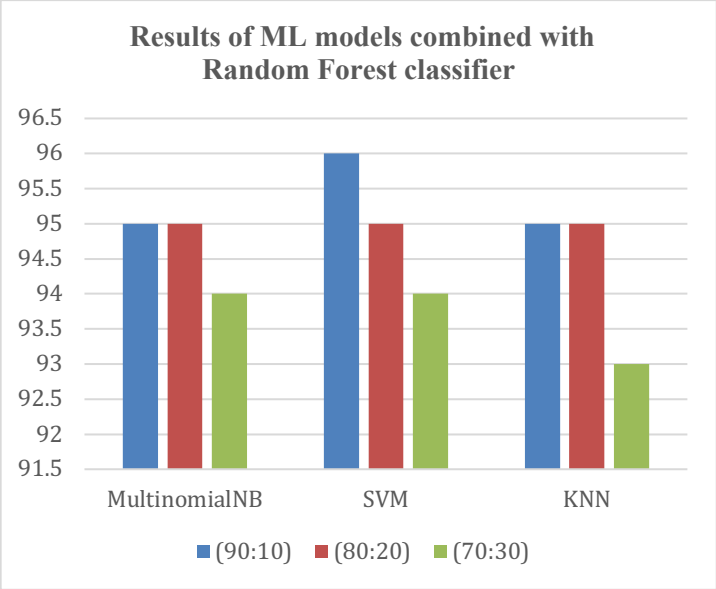

Figure 5. Bar Graph showing results of ML models with Random Forest Classifier

4.2.1. Confusion Matrix for Naive Bayes

The confusion matrix for Naïve Bayes after applying Random Forest Classifier is as below:

[[72 0 17 0]
[1 97 0 0]
[0 0 124 0]
[0 0 0 111]]

4.2.2. Number of Inaccurately classified records

As the accuracy of models after applying random Forest Classifier was high hence the number of records that were inaccurately classified by the models was less. The number of inaccurately classified records for Naïve Bayes is 18.

Number of records inaccurately classified by Naive Bayes: 18.

4.2.3. Records inaccurately classified by Naive Bayes

array(['Aircraft', 'Album', 'Assignments', 'Attendees', 'Bank','CameraDemo', 'Canary', 'Catalogs', 'Check Damage','Checking with Interest', 'Company', 'Composition layer','Crankshaft', 'Degree', 'Display', 'Elephant','ElevatorButton','Enrollment'], dtype=object).

4.3. Comparison of MultinomialNB, SVM and KNN models

There are multiple performance measurement matrices that can be used to evaluate performance of ML tools. We used three measures i.e. accuracy, precision and recall. The comparison ML models in terms of performance measures is presented in Table 5.3 below.

Table 4. Comparison of MultinomialNB, SVM and KNN models

Model	Accuracy	Precision	Recall

Multinomial NB	95	95	95
SVM	95	95	95
KNN	95	96	95

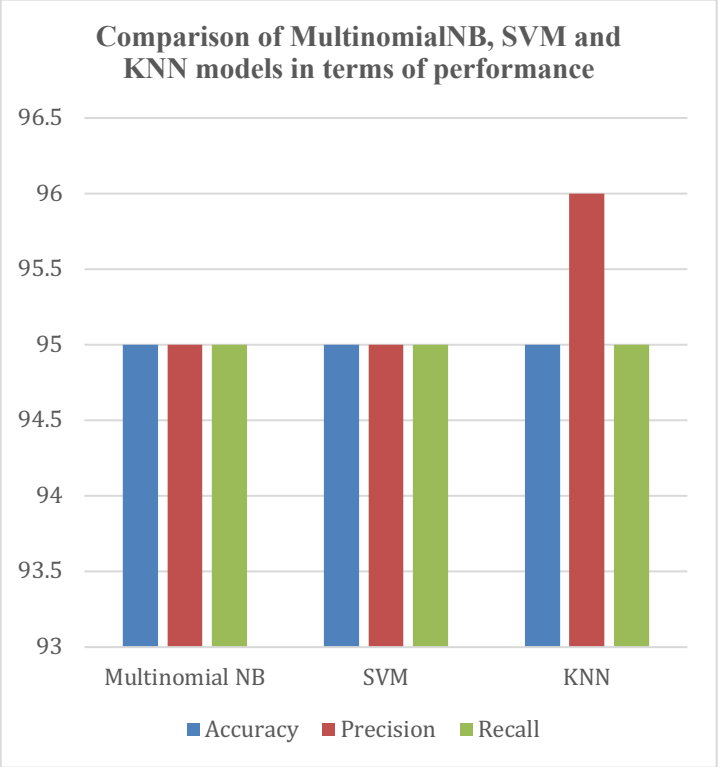

Figure 6. Comparison of MultinomialNB, SVM and KNN models in terms of performance

4.4. Summary

The UML Class diagram plays a vital role in Object Oriented design. It serves as a building block and represents the structure of a system. Class diagrams are generated from requirements written in natural language. The process of manual class diagram generation can be highly challenging and error-prone. This is because generating class diagrams is a time consuming process. Moreover due to ambiguous nature different interpretations of natural language requirements can lead in inconsistent and inaccurate class diagrams. These challenges demand the automation of this process to mitigate the problems. Automation of class diagrams generation has many key benefits. It saves time of the designers, ensures the consistency and accuracy and support maintenance. This automation of UML class diagrams is itself very difficult and challenging task. However many techniques and approaches have been proposed to resolve the problem. Natural language processing techniques, Rule based techniques and machine learning based approaches are remarkable ones. In this research work our main focus was to develop a system for the generation of class diagram using machine learning as well as with the help of some NLP and rule based approaches. Our system takes natural language requirements as input and after applying combination of NLP, Rules based and machine learning approaches it generates a class diagram. The salient feature of this research study was the creation of dataset to be used in machine learning models to train a classifier that can predict the relationship types of class diagram. The identification of the relationship type in a class diagram is important as they help users to understand the interaction between different classes. Our system focuses on four relationship types. These are association, aggregation, composition and inheritance. The system is evaluated using evaluation metrics.

5. Future Work

The purpose of the proposed system is to generate the class diagram using machine learning that incorporate the identification of relationship types. The scope of proposed system is limited to

identification of association, aggregation, composition and inheritance. As the created dataset contain these four relationship types. If more relationship types include in machine learning dataset they can predict all relationship types and can generate more accurate class diagrams. Furthermore the correctness of class diagram produced by our system is limited and it can be explained as we used limited NLP tools and rules for identification of classes. By normalizing the input sentence and by adding more patterns to rule based part we can improve the work. From a broader perspective our approach work as a baseline for researchers to generate more accurate and comprehensive system. The dataset we presented can serve as the foundation for researchers and can be used in automatic class diagram generation.

References

1. Deeptimahanti, D. K., & Babar. M. A., (2009). "An Automated Tool for Generating UML Models from Natural Language Requirements, in ASE2009" - 24th IEEE/ACM International Conference on Automated Software Engineering. https://doi.org/10.1109/ASE.2009.48.
2. Overmyer S., Lavoie B. and Rambow O. (2001). "Conceptual Modeling through Linguistics Analysis Using LIDA." The 23rd Int. Conf. on Soft. Eng. (ICSE'01), Canada, pp. 401-410.
3. Harmain, H., & Gaizauskas, R. (2003). "CM-Builder: A Natural Language-based CASE Tool." Jou. of Auto. Soft. Eng., vol. 10(2), pp. 157-181.
4. Herchi, H., & Abdessalem, W. (2012). "From user requirements to UML class diagram." Int. Conf. on Comp. Related Knowledge (ICCRK' 2012), Tunisia.
5. Perez-Gonzalez, H. & Kalita, J. (2002). "Automatically generating object models from natural language analysis" in Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pp. 86-87.
6. Bashir, N., Bilal, M., Liaqat, M., Marjani, M., Malik, N., Ali, M., (2021). "Modeling Class Diagram using NLP in Object-Oriented Designing.
7. Joshi, S. D., & Deshpande, D. (2012). "Textual Requirement Analysis for UML Diagram Extraction by using NLP." International Journal of Computer Applications (0975 – 8887) Volume 50 – No.8, July 2012.
8. Hamza, Z. A., & Hammad, M. (2019). "Generating UML Use Case Models from Software Requirements Using Natural Language Processing." in 2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO), 2019, pp. 1-6: IEEE.
9. Abdelnabi, E. A., Maatuk. A. M., Abdelaziz, T. M., & Elakeili, S. M. (2020). "Generating UML Class Diagram using NLP Techniques and Heuristic Rules." 20th international conference on Sciences and Techniques of Automatic control & computer engineering (STA), Sfax, Tunisia, December 20-22, 2020.
10. More, P., & Phalnikar, R. (2012). "Generating UML diagrams from natural language specifications." International Journal of Applied Information Systems. 2012. 1, 8, pp. 19–23.
11. Nasiri, S., Rhazali, Y., Lahmer, M., Chenfour, N., (2020). "Towards a Generation of Class Diagram from User Stories in Agile Methods" Procedia Computer Science 170 (2020) 831–837.
12. Abdelnabi, E. A., Maatuk. A. M., & Hagal. M. (2021). "Generating UML Class Diagram from Natural Language Requirements." A Survey of Approaches and Techniques.
13. Klein, D., & Manning, C. (2007). Stanford Parser 1.6. Stanford Natural Language Processing Group, City
14. Hnatkowska, B., Huzar, Z., and Tuzinkiewicz, L. (2020). "A Data-Driven Conceptual Modeling. In Integrating Research and Practice in Software Engineering." (pp. 97-109). Springer, Cham.
15. Utama A. Z. & Jang D.-S. J. J. o. K. M. S., "An Automatic Construction for Class Diagram from Problem Statement using Natural Language Processing," vol. 22, no. 3, pp. 386-394, 2019.
16. Mohanan, M., & Samuel, P. (2018). "Natural Language Processing Approach for UML Class Model Generation from Software Requirement Specifications via SBVR." International Journal on Artificial Intelligence tools, Vol 27, No. 06, 1850027, 2018.
17. Jaiwai, M., & Sammapun, U. (2017). "Extracting UML class diagrams from software requirements in Thai using NLP." 14th International Joint Conference on Computer Science and Software Engineering (JCSSE) (pp. 1-5). IEEE.
18. Narawita, C. R., & Vidanage, K. (2017). "UML Generator – Use Case and Class Diagram Generation from Text Requirements." International Journal on Advances in ICT for Emerging Regions, 2017, 10 (1).
19. Zhang, R., Meng, F., Zhou, Y., & Liu, B. (2018). "Relation classification via recurrent neural network with attention and tensor layers," Big Data Mining and Analytics, vol. 1, pp. 234-244, 2018.
20. Zhong, S., Scarinci, A., Cicirello, A. (2023) "Natural Language Processing for systems engineering: Automatic generation of Systems Modelling Language diagrams." Knowledge-Based Systems 259 (2023) 110071, pp. 1-18.

21. Yang, S., & Sahraoui, H. (2022). "Towards Automatically Extracting UML Class Diagrams from Natural Language Specifications." In ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.
22. Patel, G. A., & Priya, A. S. (2014). "Resolve the Uncertainty in requirement Specification to generate the UML Diagram." IEEE - International Conference on Advances in Engineering and Technology-(ICAET 2014).
23. Malik, M.I., Sindhu, M.A. and Abbasi, R.A. (2023), "Extraction of use case diagram elements using natural language processing and network science", PLoS ONE, Vol. 18 No. 6. https://doi.org/10.1371/ journal.pone.0287502.
24. Shweta, X., Suyash, M., & Suryansh, C. (2023). Advancing Class Diagram Extraction from Requirement Text: A Transformer-Based Approach. In Proceedings of the 20th International Conference on Natural Language Processing (ICON) (pp. 433-441).
25. Ouaddi, C., Benaddi, L., Souha, A., Jakimi, A., & Ouchao, B. (2024). A sketch of an approach for discovering UML use-case diagrams from textual specifications of systems using a chatbot. In 2024 IEEE 12th International Symposium on Signal, Image, Video and Communications (ISIVC) (pp. 1-6). IEEE.
26. Almazroi, A. A., Abualigah, L., Alqarni, M. A., Houssein, E. H., AlHamad, A. Q. M., & Elaziz, M. A. (2021). Class diagram generation from text requirements: An application of natural language processing. Deep Learning Approaches for Spoken and Natural Language Processing, 55-79.
27. Raza, A., Siddiqui, HUR., Munir, K., Almutairi, M., Rustam, F. (2022). "Ensemble learning-based feature engineering to analyze maternal health during pregnancy and health risk prediction." PLOS ONE 17(11): e0276525.
28. Rish, I. (2001). "An empirical study of the naive Bayes classifier" IJCAI 2001 workshop on empirical methods in artificial intelligence 3 (22), 41-46.
29. Ejaz, F., Tanveer, F., Shoukat, F., Fatima, N., & Ahmad, A. (2024). Effectiveness of routine physical therapy with or without home-based intensive bimanual training on clinical outcomes in cerebral palsy children: a randomised controlled trial. Physiotherapy Quarterly, 32(1), 78-83.
30. Shah, A. M., Aljubayri, M., Khan, M. F., Alqahtani, J., Sulaiman, A., & Shaikh, A. (2023). ILSM: Incorporated Lightweight Security Model for Improving QOS in WSN. Computer Systems Science & Engineering, 46(2).
31. Evgeniou, T., & Pontil, M. (2001). "Support Vector Machines: Theory and Applications." 2049. 249-257. 10.1007/3-540-44673-7_12.
32. Khan, M. F., Iftikhar, A., Anwar, H., & Ramay, S. A. (2024). Brain Tumor Segmentation and Classification using Optimized Deep Learning. Journal of Computing & Biomedical Informatics, 7(01), 632-640.
33. Guo, G., Wang, H., Bell, D., & Bi, Y. (2004). "KNN Model-Based Approach in Classification."
34. Imran, A., Li, J., Pei, Y., Akhtar, F., Yang, J.-J., & Dang, Y. (2020). Automated identification of cataract severity using retinal fundus images. Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization, 8(6), 691-698. Taylor & Francis.
35. Latif, J., Xiao, C., Tu, S., Rehman, S. U., Imran, A., & Bilal, A. (2020). Implementation and use of disease diagnosis systems for electronic medical records based on machine learning: A complete review. IEEE Access, 8, 150489-150513. IEEE.