Automatic Evaluation of Layout Quality of UML Class Diagrams Using ML

Fatima Masood
Dept. of Software Engineering
University of Sargodha
Sargodha, Pakistan
fatimamasood141@gmail.com

***Abstract*—** This research study propose an automated methodology for assessing the layout quality of UML class diagrams, utilizing the machine learning technique. This approach is to label the layout quality of UML Class Diagrams using ML and the construction of ML models. The dataset employed for training and validation comprises over 654 UML Class Diagrams, labeled by domain experts as the ground truth for layout quality. The core of this methodology lies in ML, offering a robust framework for evaluating and improving the comprehension of UML class diagrams.

Keywords—UML Class Diagrams, Layout Quality of Class Diagrams, Machine Learning, Automatic Quality Evaluator.

I. INTRODUCTION

The Unified Modeling Language (UML) is a widely accepted industry standard for depicting software system designs through diagrams. UML encompasses diverse diagram types, each with distinct purposes and applications. This study concentrates on the class diagram[1], among the most frequently utilized diagrams in software design. Class diagrams fall into the category of diagram types designed to depict the structural aspects of software or systems. These diagrams illustrate the fundamental components of a system and their interconnections. In class diagrams, rectangles represent classes, and lines symbolize relationships between these classes. Occasionally, arrows on the lines signify the direction of the relationship.

UML diagrams serve a fundamental role in offering an abstract overview of a system's structure. Among these, UML class diagrams play a crucial role as they provide essential assistance in understanding the structure of software systems[2], [3] and sharing the understanding of the system's development by software development teams. This study is to find the layout quality i.e. "Good Quality" or "Bad Quality" of UML Class Diagrams. The quality of a layout encompasses both its aesthetic appeal and the ease with which a user can grasp the information presented in the diagram. Various layout aesthetics, supported by research findings, influence the overall layout quality. For instance, the reduction of crossing lines in a diagram is associated with enhanced quality, although there are numerous other aesthetics to consider.

There are two primary approaches for crafting Class Diagrams: firstly, as a pre-programming design to guide the development process (forward engineering), and secondly, post-development to document the design (reverse engineering) and facilitate knowledge sharing among a team of developers [4]. Diagrams formulated before development, known as forward-engineered, act as a guiding framework throughout the development phase. These are typically hand-drawn by individuals who possess an intuitive sense of layout and are adept at recognizing high-quality UML diagrams. However, it is noteworthy that human-created diagrams often overlook some basic aesthetic layout principles. Conversely, diagrams produced after development, termed reverse-engineered, serve as documentation for the system. These diagrams are usually generated automatically using a tool that translates source code into a diagram or a set of diagrams. A wide range of algorithms for creating reverse-engineered diagrams exist[5], [6], [7], [8]. Ensuring high-quality output is paramount for algorithms generating UML diagrams. Consequently, there is a necessity to evaluate the quality of these diagrams. In the case of manually created and reverse-engineered diagrams, automated assessment tools play a vital role in gauging the layout quality, and identifying areas for improvement. This research anticipates the integration of an automated quality assessment tool into a DevOps toolchain, where it can be invoked whenever a UML diagram is added to the project files.

Most of the current research on evaluating layout quality primarily concentrates on identifying key rules governing layout aesthetics [9], [10]. Previous studies have primarily aimed at determining the (relative) significance of layout aesthetics and assessing the overall layout quality of diagrams. However, there is a notable gap in research regarding the automatic evaluation of layout quality specifically for class diagrams in UML. Traditional evaluation methods involve time and user-intensive studies to gauge user preferences for different layouts. An automated evaluator could swiftly assess the quality of a layout. Hence, the principal objective of this study is to develop an automated evaluator for the layout quality of UML class diagrams using machine-learning approaches. This tool has the potential to identify strengths and areas for improvement in a diagram's layout.

The main research question of the study is:

Q: How to automatically evaluate the layout quality of UML Class Diagrams Using Machine Learning?

The main research question is subdivided into two questions:

Q1: What is the performance of an automatic layout quality evaluator?

Q2: Which features are important for determining the layout quality of UML Class Diagrams?

Utilizing a machine learning approach, this methodology involves creating an evaluator based on a dataset that serves as the ground truth for diagrams. This dataset includes image features and corresponding layout quality assessments. The contributions of this study are as follows:

1. A novel automated evaluator designed for assessing the layout quality of UML class diagrams. It has the potential to be a valuable tool for evaluating class diagram quality in both industrial and academic contexts.

2. Identification and disclosure of the most crucial aesthetics influencing the layout quality of class diagrams. These findings can play a pivotal role in enhancing the performance of automated layout algorithms.

3. The proposed automatic evaluator is evaluated to validate its ability to assess the layout quality of class diagrams that were not part of the training dataset.

This study is for the automated assessment of class diagram layouts. As previously discussed, the layout of class diagrams plays a crucial role in facilitating communication and understanding of designs. While evaluating layout quality is inherently valuable, it also contributes to the broader assessment of designs represented by UML class diagrams. The assessment of UML designs holds practical applications in various domains, such as quality assurance in industrial settings and evaluating student assignments in educational contexts. However, assessing design quality introduces an additional layer of complexity, requiring consideration of design semantics and domain-specific factors. Key considerations include the clarity of responsibility allocation and the meaningfulness of relationships between classes. A broader discussion of evaluating the quality of UML designs can be found in the Ph.D. thesis of Christian Lange[11].

II. BACKGROUND AND RELATED WORK

A. Background on layout aesthetics of diagrams

Properties of a diagram layout known as layout aesthetics can influence the subjective perception of the diagram. Given that UML class diagrams can be viewed as a subtype of mathematical graphs, it is pertinent to consider aesthetics from the broader field of general graph layout. Research has explored both general graph layout aesthetics and those specifically tailored to UML class diagrams.

Four levels of design principles dictate the layout of UML diagrams.[12]

1. General principles, applicable across all diagram types, involve guidelines such as ensuring alignment and preventing element obstruction.

2. Principles specific to mathematical or abstract graphs include minimizing the number of line crossings and bends.

3. UML diagram-specific principles include the recommendation to group similar elements.

4. The fourth level of principles focuses on enhancing audience comprehension, employing strategies such as highlighting items through color or size.

The majority of UML diagram research centers on principles from the second level. This inclination is likely due to the higher quantifiability and measurability of these principles compared to those at other levels. This focus is crucial for this study, as the aesthetics need to be identified through image processing and subsequently translated into numerical features for the machine learning aspect to function. It's important to note that this study confines its scope to the graphical properties of diagrams, excluding metrics related to the semantics of class diagrams.

UML are fundamental tool for engineers to rationalize systems without delving into technological details[13]. The use of models facilitates communication among stakeholders in system development, and visual notations are preferred over pure code. The introduction of roundtrip engineering, an approach supporting the automatic or semi-automatic integration of models and source code, is highlighted as a means to enhance consistency throughout the software development process. The research conducted by Khaled [14] and Wang [15], focuses on the generic comparison of UML modeling tools, particularly in the context of round-trip engineering.

There are seven prevalent aesthetic criteria for graph drawings and metrics to evaluate the presence of each criterion [16]. In related research, [17] into graph layout aesthetics with a specific focus on UML, introducing additional aesthetics. These aesthetics are empirically assessed to discern their correlation with user preferences. Expanding on this the Ph.D. thesis [10] presents an extensive array of aesthetics categorized into different groups. In subsequent work [18], a comprehensive summary of guidelines for the aesthetic quality of UML diagrams across various levels, drawing from prior research.

The layout aesthetics discussed in previous studies can be categorized as follows:

A1: Line crossings involve points where two lines intersect, and minimizing them enhances diagram clarity [10], [16]

A2: Minimizing line bends is crucial for maintaining line continuity and aiding user comprehension[10], [16].

A3: Orthogonality applies to lines and node orientation, emphasizing horizontal or vertical alignments for improved layout quality [10], [16].

A4: Line lengths should be neither too long nor too short to facilitate grouping and separation.

A5: Diagram drawing size, aspect ratio, and minimizing width contribute to a homogeneous distribution and reduced scrolling[18].

A6: Increasing symmetry enhances diagram understandability, with a focus on both local and global symmetry(Eichelberger and Schmid, 2009[18]).

A7: Maximizing the minimum angle between lines improves distinguishability, especially crucial at low rendering resolutions[10], [16].

A8: Class placement involves uniform distribution, avoiding proximity extremes, and placing high-degree classes near the center [10], [18].

A9: Overlapping should be minimized among nodes and between nodes and edges to ensure readability[10], [18].

A10: Minimizing the difference in class sizes and keeping them as small as possible contributes to an aesthetically pleasing layout[10].

B. *Related Work*

The development of a tool using deep learning techniques, specifically, Convolutional Neural Networks (CNN), to automatically classify images as either UML class diagrams or non-UML class diagrams[19]. The study employs a dataset[20] comprising 3282 images, including 1635 UML class diagrams and 1647 non-UML class diagrams. Two CNN models, one with and one without regularization techniques, are implemented for classification. The experiments involve data augmentation, and the best accuracy on the test set is considered for evaluation. The research aims to explore the effectiveness of deep learning in solving the UML class diagram classification problem.

The classification of UML diagrams using various deep learning architectures and transfer learning techniques[21]. Ahmed and Huang employed machine learning techniques to categorize role stereotypes within UML class diagrams, aiming to swiftly provide developers with insights into role stereotypes[22]. The study[21] involves collecting a dataset of 3231 images, including different types of UML diagrams (class, activity, use case, sequence) and non-UML images. Five popular neural network architectures, namely MobileNet, DenseNet169, NasNetMobile, ResNet152V2, and InceptionV3, are evaluated for multiclass classification using both semi-trainable and fully trainable transfer learning.

Concerning the automated assessment of (class) diagram layouts, the automatic quality layout evaluator demonstrates versatility with potential applications in industrial settings for automated quality assurance within DevOps toolchains, educational contexts for grading student assignments, and algorithm design for graph layouts. The approach employs (linear) regression models constructed through machine learning, utilizing features extracted from UML class diagram images via image processing. By utilizing a dataset of 600+ UML Class Diagrams with expert-labeled layout quality, the study showcases the feasibility of automatic evaluation, analyzes the most influential features on layout quality, evaluates the performance of the layout evaluator, and contributes a labeled dataset for replication and further advancements in the field[23].

1) *Classifying layout quality*

No existing literature is there, where the objective is to utilize layout aesthetics for systematically estimating the overall quality. Störrle[12] categorizes diagrams as either 'good' or 'bad' for his study, but notes that this classification is not a central focus. In his classification, diagrams adhering to positive aesthetics and avoiding violations of negative aesthetics are deemed as 'good'.

2) *Prior Work*

Regarding the utilization of image processing to identify features in UML diagrams, prior research conducted by Karasneh[24], [25] resulted in the development of a system. This system is designed to interpret an image of a UML diagram, extracting its semantic content, which includes classes and relationships. Subsequently, the system generates an XMI file representing the UML model based on this extracted information.

Additionally, there is an automated classifier capable of determining whether an arbitrary image corresponds to a UML class diagram[26]. Their approach is in line with ours, as it involves utilizing image processing to identify features and subsequently employs machine learning to train a model capable of distinguishing class diagrams from other images. The distinction lies in the scope of our study, where we aim to recognize various aspects and intricate details of UML diagram layouts. Notably, their approach yields a binary classifier (UML class diagram or not), whereas our objective is to develop an evaluator capable of assigning a numerical value (on a 5-point scale) to estimate the layout quality of any UML class diagram.

III. RESEARCH METHODOLOGY

We utilized a dataset with labeled ground truth for the layout quality, achieved through manual labeling of diagrams by a group of UML experts. Our approach involves applying supervised machine learning to the set of images, ground truth, and features.

A. *Dataset*

We have a dataset of 654 diagrams with the following criteria:

C1: The image must depict a class diagram, focusing specifically on diagrams representing a system's structure. Other structural diagrams, such as component or package diagrams, are excluded to maintain dataset homogeneity. Future work may explore extending the study to include these diagram types.

C2: Classes must be portrayed as rectangular boxes, aligning with the UML standard for drawing class diagrams. Diagrams with classes represented by rectangles with rounded corners are excluded for improved image recognition performance.

C3: The image must be generated by a tool, not drawn by hand. This scoping decision simplifies image recognition approaches.

C4: Screenshots of UML modeling tools displaying diagrams within applications, including toolbars and menus, are excluded to prevent misrecognition of menu bars and toolbars as rectangles.

C5: The image should display the entire diagram to ensure comprehensive coverage and avoid corner cases.

C6: The image should only include UML notation-defined diagram elements, excluding additional annotations or icons except for UML-comment-boxes. This decision accommodates common practices in software documentation.

C7: The image should not be overly simple, requiring a minimum of four classes and at least three relations in a diagram.

C8: No duplicate diagrams are included to enhance the integrity of the dataset.

B. Defined layout aspects of quality

In the following section, there is a series of layout quality aspects (labeled A1 to A10) and a set of image features (labeled F1 to F16) as the foundation for computing or approximating these aspects [23].

A1 — Line crossings: Minimize the number of lines crossing each other, and maximize the angles of the line crossings. Computed from features F1 (Line crossings) and F2 (Crossing angles).

A2 — Line bends: Minimize the number of line bends. Computed using feature F3 (Line bends).

A3 — Orthogonality: Ensure rectangles are on an orthogonal grid, and lines are drawn horizontally or vertically. Extracted using features F4 (Line angles), Line orthogonality, and Rectangle orthogonality.

A4 — Line lengths: Ensure lines are not too long or too short, with uniform lengths. Computed from features F7 (Average Line length), F8 (Line length variation), F9 (Longest line), and F10 (Shortest line).

A5 — Diagram drawing size: Minimize the size of the diagram, considering the coverage of rectangles (F11: Rectangle coverage) and the aspect ratio (F12: Aspect ratio).

A6 — Symmetry: Although symmetry is an aesthetic, no specific image features are computed due to the complexity of a computational algorithm for evaluating symmetry.

A7 — Line angular distances: No feature is computed as the image processing used only identifies standalone rectangles and lines, lacking connections between elements.

A8 — Class placement: Consider the uniform distribution of rectangles and avoid them being too close or too far apart. Features include Rectangle distribution and Rectangle proximity. However, image processing does not support high-level analysis involving connections between elements, so additional features for connected rectangles are not defined.

A9 — Overlapping: No feature is defined, as the image processing struggles to interpret overlapping elements.

A10 — Node sizes: Minimize rectangle sizes, and minimize the difference among their sizes. Defined using features F15 (Rectangle size) and F16 (Rectangle size variation).

C. Definitions of image features

In this section, features are defined. These features serve as input for machine learning. It is assumed that the image processing stage produces a set of rectangles (representing classes) and lines (representing relations between classes). In addition to features related to diagram aesthetics (F1–F16), two additional features are incorporated to characterize the diagram itself (F17 and F18).[23]

F1 — Line Crossings: This feature calculates the ratio of actual line crossings to the maximum possible crossings, addressing variations in diagram complexity.
F2 — Crossing Angles: The average crossing angle of intersecting lines in the diagram.

F3 — Line Bends: The average number of bends per line in the diagram.

F4 — Line Angles: The average deviation angle from orthogonality for each line.

F5 — Line Orthogonality: The ratio of orthogonal lines to the total number of lines.

F6 — Rectangle Orthogonality: Evaluates the orthogonality of rectangles in both horizontal and vertical directions.

F7 — Average Line Length: The average length of all lines in the diagram.

F8 — Line Length Variation: Measures the variation in line lengths using standard deviation.

F9 — Longest Line: The length of the longest line in the diagram.

F10 — Shortest Line: The length of the shortest line in the diagram.

F11 — Rectangle Coverage: The ratio of the total area covered by rectangles to the entire diagram area.

F12 — Aspect Ratio: The ratio of the diagram's width to its height.

F13 — Rectangle Distribution: Measures the variance in coverage of quadrants by rectangles.

F14 — Rectangle Proximity: The average distance from the center of each rectangle to the center of other rectangles.

F15 — Rectangle Size: The average area of all rectangles in the diagram.

F16 — Rectangle Size Variation: Measures the variation in rectangle sizes using standard deviation.

F17 — Number of Rectangles: Indicates the count of rectangles detected, corresponding to the number of classes.

F18 — Number of Lines: Indicates the count of lines detected, corresponding to the number of relationships between classes.

D. *Machine Learning Approach*

The Random Forest algorithm has been chosen as the primary machine learning model in this study. Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputs the mode of the classes for classification tasks or the mean prediction for regression tasks. This algorithm is particularly well-suited for our task, as it excels in handling high-dimensional data, mitigates overfitting, and provides robust performance.

Random Forest's strength lies in its ability to aggregate the predictions of multiple decision trees, each trained on a random subset of the dataset. This randomness in both data selection and feature choice enhances the model's generalization capabilities and resilience to noise in the data. Additionally, Random Forest provides a built-in feature importance measure, which will aid in identifying the most influential image features contributing to the evaluation of UML Class Diagram layout quality. The versatility, robustness, and interpretability of Random Forest make it a suitable choice for the complex task of assessing the aesthetic quality of UML Class Diagram layouts through machine learning.

IV. RESULTS AND EVALUATIONS

This section presents the outcomes of the machine learning algorithm (Random Forest) and its assessment. Python programming was employed for the implementation of the machine learning components in this research.

A. *Methodology*

To assess the layout quality of UML class diagrams, a machine learning approach was employed using the Random Forest algorithm. The dataset, consisting of 654 instances and 19 features, was preprocessed using the pandas library in Python. Descriptive statistics were computed, and one-hot encoding was applied to categorical variables. The dataset was then split into training 75% (490 instances) and testing 25% (164 instances) sets using the sci-kit-learn library.

For establishing a baseline, the mean of the 'RectOrth2' feature was used as a predictor. Subsequently, a Random Forest Regressor model with 1000 decision trees was instantiated and trained on the training data. Predictions were made on the test data, and the Mean Absolute Error (MAE) was calculated to evaluate the model's performance.

Additionally, feature importance was analyzed to identify key contributors to the model's predictions. The two most important features, 'LongestLine' and 'RectOrth,' were extracted to train a simplified model for comparison.

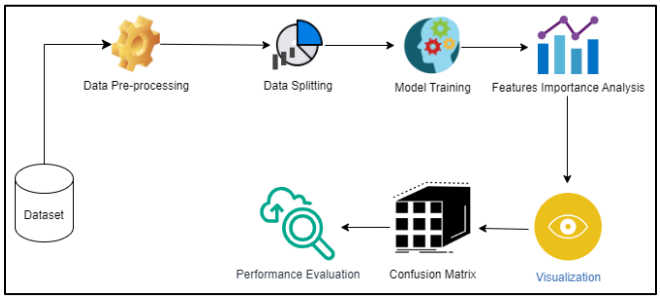

Fig. 1. Framework Diagram

B. *Performance of the Machine Learning*

The baseline predictions, using the mean of 'RectOrth2,' resulted in a baseline error of 2.88 degrees. The Random Forest Regressor model achieved a significantly reduced Mean Absolute Error of 0.59 degrees on the test set, with an accuracy of 77.59%.

The feature importance analysis revealed that 'LongestLine' and 'RectOrth' were the most influential variables in predicting layout quality, emphasizing their significance in the evaluation process.

The simplified model, considering only 'LongestLine' and 'RectOrth,' demonstrated a slightly higher Mean Absolute Error of 0.76 degrees and an accuracy of 70.21%. This suggests that while the model's predictive performance was reduced with fewer features, it still provided valuable insights into layout quality.

C. *Figures*

In Figure 2, the shapes of the feature matrices are illustrated, providing a visual representation of the dataset used in the machine learning analysis. The original dataset comprises 654 instances, each characterized by 19 features. Following the preprocessing steps, the dataset is divided into training and testing sets, with the training features consisting of 490 instances and 18 features. Correspondingly, the training labels form a one-dimensional array of shapes (490,) representing the quality labels associated with the training instances. The testing features and labels are structured similarly, with 164 instances in the testing set and labels conforming to the shape (164,). This figure serves as a valuable overview of the dataset dimensions, crucial for understanding the composition and scale of the data utilized in the subsequent machine learning experiments.

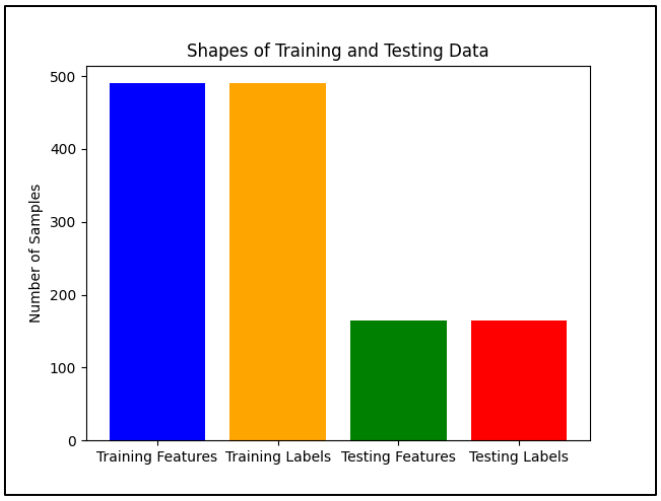

Fig. 2. Shape of Features

Figure 3 provide a comprehensive overview of the feature importances derived from the Random Forest model, shedding light on the influential factors contributing to the evaluation of layout quality in UML class diagrams. The importance values are calculated based on the contribution of each feature to the predictive performance of the model. Among the key features, "LongestLine" stands out with the highest importance score of 0.20, indicating its significant role in influencing the model predictions. Following closely, "RectOrth" and "RectOrth2" exhibit notable importance values of 0.17 and 0.10, respectively, emphasizing the significance of these metrics in assessing layout quality. The table further highlights the relevance of various features such as "AvgCrossingAngle," "id," and "AvgLineAngle," each contributing distinct insights into the machine learning model's decision-making process. This comprehensive breakdown in Figure 3 serves as a valuable reference for understanding the relative importance of individual features in the context of layout quality evaluation for UML class diagrams.

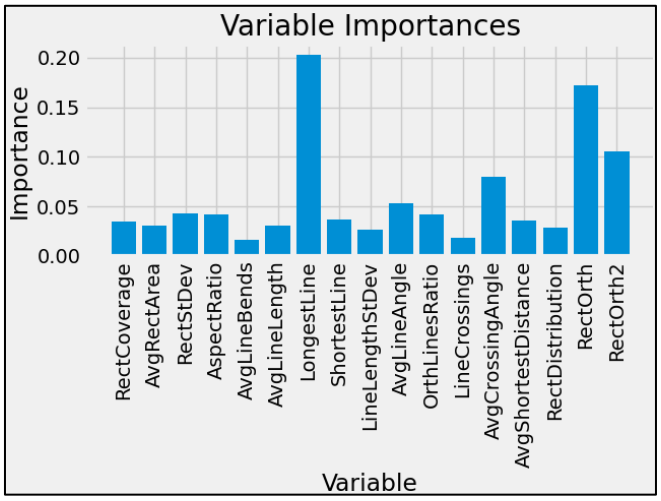

Fig. 3. Variable Importance

Figure 4 presents the Confusion Matrix, a pivotal visualization illustrating the performance of the machine learning model in classifying UML class diagrams based on their layout quality.

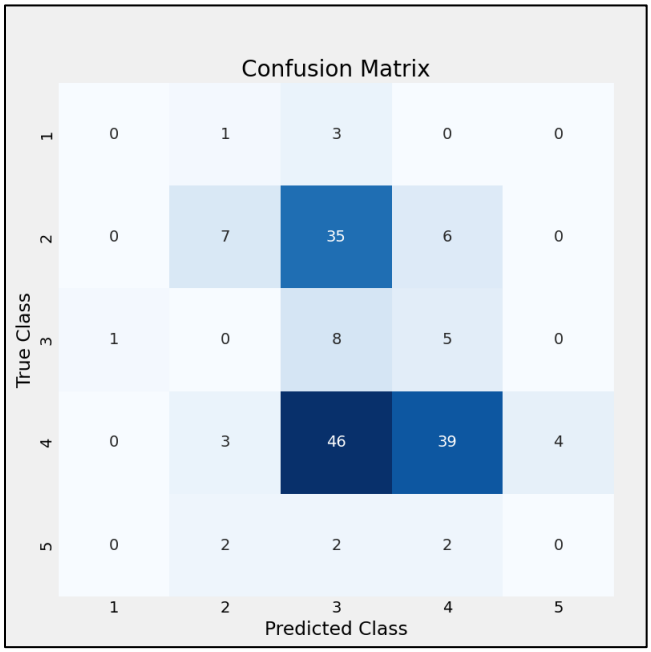

Fig. 4. Confusion Matrix

The Random Forest tool we used has a neat feature – it allows us to take a look at individual trees within the "forest" it creates. Think of these trees as decision-makers, each contributing to the final decision. In our study, we've singled out one of these trees to show you exactly how it makes decisions.

Figure 5 presents a detailed view of a decision tree, showcasing the first three levels of the tree's structure. Decision trees are like a flowchart of decision-making, and in this illustration, we delve into the initial stages of how the model processes information. Each node in the tree represents a decision based on a specific feature, guiding the model through the data. As we move down the tree, we can observe the criteria it uses to classify UML class diagrams. This visual insight allows us to grasp the early steps of the decision-making process, providing a clearer understanding of which features play a crucial role in determining the layout quality of the diagrams.

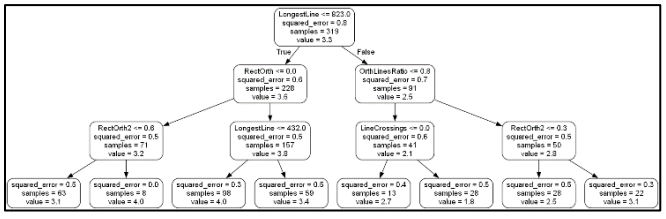

Fig. 5. Reduced Tree with 3 Levels

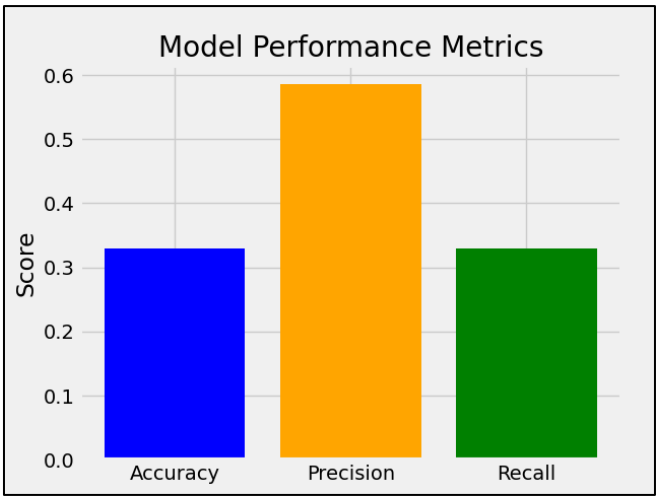

Fig. 6. Evaluation Measures

D. Evaluation

The confusion matrix visually depicted the model's classification performance, showcasing the number of true positive, true negative, false positive, and false negative instances. Accuracy, precision, and recall were calculated, providing a comprehensive view of the model's overall performance, ability to avoid false positives, and ability to capture true positives, respectively.

Despite achieving high accuracy, it is crucial to interpret the results considering the domain-specific implications. The choice of the threshold for classification significantly influences the evaluation metrics. Further exploration and fine-tuning of the model, possibly through hyperparameter optimization, could enhance its performance.

V. Conclusion and Future Work

This section gives the conclusion and future directions of this study.

A. Conclusion

The dimensions of our feature set are as follows: (654 samples, 19 features). During the training phase, the dataset was configured to have 490 samples with 18 features, while the corresponding labels were shaped as (490,). For testing, we employed 164 samples with 18 features, and the labels were shaped accordingly as (164,). The baseline error for the RectOrth2 variable was 2.88, and the mean absolute error across all predictions was 0.59 degrees. The accuracy of our model reached 77.59%. Notably, key variables contributing to the model's decision-making included LongestLine (Importance: 0.2), RectOrth (Importance: 0.17), and RectOrth2 (Importance: 0.1). Other factors like AvgCrossingAngle, id, and AvgLineAngle also played significant roles, with respective importance of 0.08, 0.05, and 0.05. The model demonstrated robust performance, yielding an accuracy of 70.21%, a precision of 60.71%, and a recall of 53.13%.

The evaluator serves diverse purposes: In an industrial context, it can integrate into Quality Assurance tools to autonomously assess the quality of artifacts within a project. This integration could advance the quality management of modeling artifacts, addressing a current lag compared to code artifacts. In an educational setting, our evaluator becomes a valuable component for automatically grading UML class diagrams. This functionality aligns well with the growing trend of online learning environments. Additionally, our evaluator finds applications in algorithms aimed at automatically generating layouts for class diagrams. For instance, in reverse engineering scenarios where diagrams represent source code artifacts, our evaluator acts as an oracle and provides valuable feedback to machine-learning layout algorithms.

B. Future Directions:

Looking ahead, our research opens avenues for future exploration. While we concentrate on UML class diagrams in this work, extending our approach to other diagram types poses an intriguing challenge. Different diagram structures and elements may require tailored approaches, yet commonalities in layout aesthetics could allow for adaptable methodologies. The identified important aesthetics from our study can inform the development of new automatic layout algorithms, potentially incorporating reinforcement learning for improved outcomes. Moreover, our evaluator has the potential to contribute to the overall assessment of UML models, complementing existing methods that focus on design quality. Providing specific feedback on layout aspects could enhance the learning experience in diagram creation, supplementing the current grading system with more nuanced insights. Improvements in image recognition, especially regarding complex cases like dotted or curved lines, are essential for enhancing the robustness of our evaluator. Future studies should also explore additional text-related features within diagrams for a more comprehensive evaluation.

Acknowledgment

Special Thanks to all the teachers, colleagues, and supporters.

References

[1] G. Reggio, M. Leotta, F. Ricca, and D. Clerissi, “What are the used UML diagrams? A Preliminary Survey.” [Online]. Available: www.devx.com/architect/Article/45694

[2] G. Scanniello *et al.*, “Do software models based on the UML aid in source-code comprehensibility? Aggregating evidence from 12 controlled experiments,” *Empir Softw Eng*, vol. 23, no. 5, pp. 2695–2733, Oct. 2018, doi: 10.1007/s10664-017-9591-4.

[3] A. M. Fernández-Sáez, M. Genero, M. R. V Chaudron, D. Caivano, and I. Ramos, “Are Forward Designed or Reverse-Engineered UML diagrams more helpful for code maintenance?: A family of experiments,” *Inf Softw Technol*, vol. 57, pp. 644–663, 2015, doi: https://doi.org/10.1016/j.infsof.2014.05.014.

[4] M. H. Osman, T. Ho-Quang, and M. Chaudron, “An Automated Approach for Classifying Reverse-Engineered and Forward-Engineered UML Class

Diagrams," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2018, pp. 396–399. doi: 10.1109/SEAA.2018.00070.

[5] E. Fauzi, B. Hendradjaya, and W. D. Sunindyo, "Reverse engineering of source code to sequence diagram using abstract syntax tree," in *2016 International Conference on Data and Software Engineering (ICoDSE)*, 2016, pp. 1–6. doi: 10.1109/ICODSE.2016.7936137.

[6] M. J. Decker, K. Swartz, M. L. Collard, and J. I. Maletic, "A Tool for Efficiently Reverse Engineering Accurate UML Class Diagrams," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 607–609. doi: 10.1109/ICSME.2016.37.

[7] T. Ziadi, M. A. A. da Silva, L. M. Hillah, and M. Ziane, "A Fully Dynamic Approach to the Reverse Engineering of UML Sequence Diagrams," in *2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, 2011, pp. 107–116. doi: 10.1109/ICECCS.2011.18.

[8] U. Sabir, F. Azam, S. U. Haq, M. W. Anwar, W. H. Butt, and A. Amjad, "A Model Driven Reverse Engineering Framework for Generating High Level UML Models From Java Source Code," *IEEE Access*, vol. 7, pp. 158931–158950, 2019, doi: 10.1109/ACCESS.2019.2950884.

[9] M. J. Mcgill, H. C. Purchase, M. Mcgill, L. Colpoys, and D. Carrington, "Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study," 2002. [Online]. Available: https://www.researchgate.net/publication/2531384

[10] H. Eichelberger, "Aesthetics and automatic layout of UML class diagrams," Universität Würzburg, 2005.

[11] C. F. J. Lange, "Assessing and improving the quality of modeling : a series of empirical studies about the UML," Mathematics and Computer Science, 2007. doi: 10.6100/IR629604.

[12] H. Störrle, "On the impact of layout quality to understanding UML diagrams.," *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 135–142, 2011.

[13] D. Rosca and L. Domingues, "A systematic comparison of roundtrip software engineering approaches applied to UML class diagram," in *Procedia Computer Science*, Elsevier B.V., 2021, pp. 861–868. doi: 10.1016/j.procs.2021.01.240.

[14] L. Khaled, "A Comparison between UML Tools," in *2009 Second International Conference on Environmental and Computer Science*, 2009, pp. 111–114. doi: 10.1109/ICECS.2009.38.

[15] G. Wang, B. McSkimming, Z. Marzec, J. Gardner, A. Decker, and C. Alphonce, "Green: A Flexible UML Class Diagramming Tool for Eclipse," in *Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion*, in OOPSLA '07. New York, NY, USA: Association for Computing Machinery, 2007, pp. 834–835. doi: 10.1145/1297846.1297913.

[16] H. C. Purchase, "Metrics for Graph Drawing Aesthetics," *J Vis Lang Comput*, vol. 13, no. 5, pp. 501–516, 2002, doi: https://doi.org/10.1006/jvlc.2002.0232.

[17] M. J. Mcgill, H. C. Purchase, M. Mcgill, L. Colpoys, and D. Carrington, "Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study," 2002. [Online]. Available: https://www.researchgate.net/publication/2531384

[18] H. Eichelberger and K. Schmid, "Guidelines on the aesthetic quality of UML class diagrams," *Inf Softw Technol*, vol. 51, no. 12, pp. 1686–1698, 2009, doi: https://doi.org/10.1016/j.infsof.2009.04.008.

[19] B. Gosala, S. R. Chowdhuri, J. Singh, M. Gupta, and A. Mishra, "Automatic classification of uml class diagrams using deep learning technique: Convolutional neural network," *Applied Sciences (Switzerland)*, vol. 11, no. 9, May 2021, doi: 10.3390/app11094267.

[20] "Dataset of the Paper 'Automatically Classifying UML Class Diagrams from Images using Deep Learning,'" Zenodo, Nov. 2020. doi: 10.5281/zenodo.4252890.

[21] S. Shcherban, P. Liang, Z. Li, and C. Yang, "Multiclass classification of four types of UML diagrams from images using deep learning," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, Knowledge Systems Institute Graduate School, 2021, pp. 57–62. doi: 10.18293/SEKE2021-185.

[22] J. Ahmed and M. Huang, "Classification of role stereotypes for classes in UML class diagrams using machine learning Master's thesis in Software Engineering."

[23] G. Bergström *et al.*, "Evaluating the layout quality of UML class diagrams using machine learning," *Journal of Systems and Software*, vol. 192, Oct. 2022, doi: 10.1016/j.jss.2022.111413.

[24] B. Karasneh and M. R. V Chaudron, "Img2UML: A System for Extracting UML Models from Images," in *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, 2013, pp. 134–137. doi: 10.1109/SEAA.2013.45.

[25] B. Karasneh and M. R. V Chaudron, "Extracting UML models from images," in *2013 5th International Conference on Computer Science and Information Technology*, 2013, pp. 169–178. doi: 10.1109/CSIT.2013.6588776.

[26] T. Ho-Quang, M. R. V Chaudron, I. Samúelsson, J. Hjaltason, B. Karasneh, and H. Osman, "Automatic Classification of UML Class Diagrams from Images," in *2014 21st Asia-Pacific Software Engineering Conference*, 2014, pp. 399–406. doi: 10.1109/APSEC.2014.65.