Software Defect Prediction Using Fuzzy Logic

by

Kripa Shankar Muthu Kumar

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
December 12, 2020

Keywords: Software Engineering, Fuzzy Logic, Defect Prediction Model

Abstract

Finding software defects in software project modules is a complex process and highly uncertain in nature. Even though multiple intensive machine learning and deep learning models are available to predict defects, it is important to define and construct a simple model that applies the domain expert's knowledge and handle uncertainty in measurement of features. We developed a Mamdani Fuzzy Logic-based Software Defect Prediction model that accepts both traditional membership functions (Triangular, Trapezoidal, etc) and domain expert's custom membership function to predict software defects. To improve upon the Mamdani system, we implemented a simple Takagi Sugeno model that provided better predictions. We evaluated our fuzzy logic models using popular regression models like Multiple Linear Regression and Random Forest Regression.

Acknowledgments

I want to express my sincere gratitude to Dr. Tung Nguyen for his invaluable guidance, support, and role as my major advisor. Dr. Tung had educated me on the subject of Fuzzy Logic and provided continuous support in all forms.

My sincere thanks to Dr. Cheryl Seals and Dr. Kai Chang for being my committee members, sharing their knowledge, and providing suggestions for the improvement of my research work.

I want to thank my parents, my brother and his family, and cousins for their love and support. I appreciate my colleague Tam Nguyen for helping me with my research, and all my friends who supported and encouraged me throughout my master's journey.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software fault prediction (SFP) is a process that helps us to predict software faults or bugs prior to the deployment of a software product. A bug or fault is defined as a characteristic of a software system that can lead to a system defect and ends up as a software failure. As a result, the system's behavior gets altered at the system users' end [1]. The resultant defects reduce the software's reliability and increase the developers' efforts to fix the issues.

Artificial intelligence (AI) techniques like Machine learning (ML) and Fuzzy logic are actively used in the area of SFP. Software source code metrics, object-oriented metrics, CK metrics [16], and others that are derived from the software systems act as features to help predict the defects. In this thesis, we will discuss SFP using fuzzy logic techniques. Fuzzy logic is conceived through the theory of fuzzy sets, introduced by Zadeh (1965)[3]. A fuzzy set assigns a degree of membership, a real number ranges from the interval [0,1] to the features (Antecedents) and the target (consequent).

The primary cause of software bugs is due to errors or misconceptions in any of the software development stages (Requirements, Design, Development, Testing, Deployment, and review). The other causes result from Arithmetic miscalculations, Logic or Syntactic errors, Resource leaks, Hardware interfacing issues, and more.

However, the source of software bugs can be classified as (1) Intrinsic bugs; and (2) Extrinsic bugs. Intrinsic bugs occur when code changes happen in the source code, and the extrinsic bugs happen due to external dependencies (3rd party APIs) [2]. In critical systems, the software bugs could produce disastrous effects: examples include various nuclear power and air traffic controller related crashes. In non-critical systems, the software bugs lead to revenue loss, increase in operational costs, time of the resources and legal risks [43].

1.1 Machine Learning/Deep Learning based Defection prediction models

Gathering from [4], [5], [6], we can discern that many researchers perform feature selections on software metrics like Line Of Code (LOC), Mccabe Complexity [30], Coupling Between Objects (CBO) [16], Code Churn (Process Metrics), supply them to models like Random forest regression, Artificial Neural Networks (ANN), and validated the models using Root Mean Square Error, Mean Magnitude relative error, and more. Even though the software metrics have a direct correlation with the defects, these models prove to be computationally expensive and don't generalize well/handle imprecise measurements across different systems.

1.2 Popular Defection prediction tools

In 2011, using Rahman et al.[9], Google built a defect prediction tool that predicts the bug hot spots–the files that contain bug fixes–and developed a scoring algorithm to prioritize newer commits [13]. In 2018, [11] Facebook developed a tool named Getafix that predicts the bugs at the method level and recommends auto-fixes. Using the abstract-syntax-tree-based differencer, they generate a data-set of concrete edits made between files associated with the bug-fix. Then, they produce abstract fix patterns–using hierarchical clustering, anti-unification–that provides the appropriate fixes with context.

1.3 Fuzzy Logic based Defect prediction models

A few researchers utilize fuzzy logic approach and techniques for Software Quality Prediction (SQP) and SFP [8,12,10].Dilip Kumar Yadav et al.[10] had designed a fuzzy model that accepts Experience of requirement team (ERT), Requirement Defect Density (RDD), Requirement Stability (RS) as features to predict the software residual defects in comparison to Bayesian Belief Networks Fenton et. al [7]. They used triangular membership functions to capture the features, fuzzify them, create domain expert rules to aggregate the fuzzy sets and defuzzify to predict early software defects. From this approach, we can understand that fuzzy logic has performed well than Fenton et al. [10] because fuzzy logic handles uncertainty in measurements and generalize well; however, it becomes more computationally intensive with defuzzification if there is an increase in the number of rules,and distribution of parameters in data sets affects the generalization.

To alleviate the above problems, initially, we built a Mamdani fuzzy system using the Scikit Fuzzy library and applied to data sets of software metrics obtained from open source systems like Apache Lucene, Eclipse JDT, and Eclipse Mylyn [14] to understand the applicability of Fuzzy logic to SFP. Then, we converted the system with traditional membership functions (Triangular, Trapezoidal) to a combination of data distribution based membership functions and custom membership (domain expert) functions.This modified FL system significantly improved the traditional system's results by more than 50 percentage, but still poorer than traditional ML models.

Then, we built a Takagi-Sugeno model to improve the accuracy and capability compared to ML-based regression models (Linear Regression, Binomial Regression), Random Forest Regression. The details of all of our methodologies and experiments are available in chapter 4.

The remainder of our work is structured as follows. Chapter 2 will discuss related software defection prediction researches and background. Chapter 3 will discuss the fuzzy logic theory and other technologies used in our work. Chapter 4 will discuss the data set, and Chapter 5 for Methodologies used to build our systems. Chapter 6 will provide an overall summary, the conclusion of our work, and plan for future improvements to our work.

Chapter 2

Literature Review

To understand the applications of the fuzzy logic in software engineering, we first learned the fuzzy methodologies through the Software Estimation Effort by Martín, C.L [36] et al. With the help of correlation, they selected features like LOC, Mccabe Complexity, Dhama Coupling to predict Development Time, and they extracted rule-sets from a data set that contain software metrics from 41-pascal modules. The fuzzy logic model outperformed a multiple linear regression model as it is difficult to construct a precise mathematical model for this domain and the metrics are uncertain.

Henrik Madsen, Grigore Albeanu, et al. [39] argued that modeling debugging activities in the form of fuzzy linguistic membership functions is critical since the discovery of bug is uncertain. Applying Monte Carlo sampling via operational profile distribution, they derived the probability conversion function and membership functions to predict the fault regions. Based on that, they compose rules to anticipate conditions and predict faults. They further extend this to use for bug detect-ability and degree of risk that is similar to Mutation Testing.

A software bug could occur if there's a misconception in any stage of the software development life cycle. Harikhesh Bahadur Yadav, Dilip Kumar Yadav et al. [38] selected features with Domain Expert's knowledge from each phase of a software development cycle. The phases are Requirements (Requirement Stability, Requirement Fault Density, etc), Design (Cyclomatic Complexity, Design Review Effectiveness), Coding (Programmer Capability, Process Maturity), and Testing (Staff Experience, Quality of Documented Test Cases). Using trapezoidal, triangular membership functions, and the Center of Gravity as the defuzzification method, they composed rules with the above metrics and formed a model that performed well than Fenton et al.[7] and Dilip Kumar et al.[10].

Many fuzzy logic approaches declare membership functions with domain expert parameters; however, Harikesh Bahadur Yadav et.al [37] concluded that predefined antecedent and consequent membership functions couldn't help us get better membership functions. Using the PROMISE repository, they selected 13 best static code metrics and generated input membership function values through K means clustering. Using the cluster centers, similarity values, and other parameters, they constructed the membership functions for the software features.

Focusing on the fact that an increased set of rules and redundancy make a fuzzy model inefficient, O. Dehzangi et al. came up with a data mining approach, mining frequent item-set, to consolidate and optimize the rule-sets in fuzzy rule-based classification systems. By filtering the n-dimensional rules using the support evaluation measure and adjusting the rule weights, they came up with an algorithm that preserves the rule patterns that give efficient accuracy values across the UCI ML repository projects [41].

Most of the fuzzy logic approaches adopt Mamdani System rule-sets and a suitable defuzzification method to derive a crisp output. However, Hamdi A. Al-Jamimi [8] utilized a Takagi Sugeno model with filtered and top K software metrics in the NASA repository to classify the defects [21]. They reported the accuracy values are better, but the consequent linear function in the Takagi Sugeno model is not discussed in their studies.

Adaptive Neuro Inference Fuzzy system (ANFIS) possesses a neural network architecture with a fuzzification layer (1st layer) that takes input values to construct membership functions, the second layer that takes in the fuzzy rules for the features, the third layer that calculates the firing strengths for the rules, the fourth layer that normalizes the strength values, and the fifth layer performs the defuzzification to produce the predicted consequent values [40]. To learn model parameters, Jang used a gradient descent algorithm and least-squares estimator [29].

Using the ANFIS, Riyadh A.K. Mehdi performed defect prediction activities on the PROMISE repository data sets and found that ANFIS gives better accuracy than Radial basis function and Probabilistic neural networks [42]; however, the ANFIS produced a low probability of detecting software defects. He further studied the ANFIS model to understand the explanatory power of each feature through sensitivity analysis. He discovered that Halstead metrics (LOC) [29] and Cyclomatic complexity [30] are important predictor variables.

Shruthi et al. proposed a linear multiple regression model with an optimized set of metrics with the weight allocation by computing marginal R square values (coefficient of determination). With the metrics and their respective weights, they calculate the normalized bug proneness index values that are between 0 to 1 [15]. The closer the index value to 1, the more buggy is the class. They tested their model using the Eclipse JDT Core dataset [14], and the optimized metrics were the number of past bugs, number of past versions, LOC, and entropy.

Deepak and Pravin also developed a Multiple Regression model (MLR) to predict software faults; however, they identified important predictors among Object-Oriented metrics and CK metrics [16] using factor analysis [18]. They used data sets from [14] to identify latent variables and factors to build their MLR, evaluated the model using R squared and adjusted R squared methods, and concluded that an MLR with factor analysis is better at finding the fault proneness rather than a simple MLR.

Euyseok Hong presented a Random Forest (RF) model with WEKA's attribute selection process–an ensemble of randomized decision trees that get the best output by taking the majority vote–generalized well on data sets [19] with architectural design phase metrics [17]. E Hong stated that due to RF's randomization that reduces the generalization error and its ability to perform well with noisy data sets, it is able to outperform Support Vector Machine (SVM) and MultiLayer perceptron irrespective of normalization of data.

Kalai Magal. R and Shomona Gracia Jacob created an improvised random forest model with Correlation-based Feature Subset Selection methods (CFS) to classify the defects–unlike regression [22]. This improvised random forest model produced a better accuracy, sensitivity, and specificity on NASA data sets compared with regular random forest, Naive Bayes Classifier, K-Nearest Neighbour, and Instance-Based Classifier [21]. However, there was no modification presented to extend their improvised RF to utilize on regression problems (the precise number of defects).

Yogesh, Arvinder Kaur, and Ruchika claimed that the Support Vector Machine model (SVM) with CK metrics [16] as features is better at predicting fault proneness based on its

Receiver Operating Characteristic Analysis (ROC) [20]. They developed the model using only the NASA KC1 [21] data set. However, the model's performance on cross projects was not discussed.

David Gray et.al [15] also applied SVM on several NASA data sets [21] to perform SFP; however, they focused more on Data Quality and Data balancing techniques to improve the reliability of their SFP. In the data pre-processing steps, they removed the repeated set of modules that had similar properties (lines, comments, operands, operators, conditional statements, etc.) and balanced the data using Random oversampling. They achieved 70 percent accuracy on the testing sets.

Zhen Yan et.al [24] applied fuzzy membership functions to help SVM (Fuzzy Support Vector regression) to reduce the effects of noises and imbalanced data and to improve prediction for modules that contain a large number of defects. They developed their model using MIS (Medical Imaging System) [25] and RSDIMU (Redundant Strapped Down Inertial Measurement Unit) [26] data sets that predominantly contain Halstead [29] and Mccabe complexity metrics [30] from languages like Pascal and C.

Liguo Yu [27] demonstrated the applicability of Negative Binomial Regression (NBR) and concept drift to predict the defects count in the Apache Ant system (Versions 1.3 to 1.7) and evaluated the model using self-assessment – testing a version's model on the same version data set (for example, 1.3 version model); and using forward assessment – testing a version's model on different version data sets with accuracy, precision, and recall. Liguo Yu proved that a Binary Logistic Regression gives a better recall than NBR; however, NBR is helpful in predicting multiples bugs in a system.

Jian Li et al.[28] devised a different approach, somewhat similar to techniques used in [11], to classify defects in tera-PROMISE Repository [31] of Java Projects. They argued that the traditional defect prediction methods – applying CK, Halstead, Object-oriented metrics to the statistical/machine learning models–wouldn't capture the syntactic information of the source code. Thus, they utilized the Convolutional Neural Network to capture the syntactic features with the help of Abstract Syntax Tree and word embedding and traditional CK metrics to classify the buggy files.

Tanvi et al. [32] compared the fuzzy modeling approach with Artificial Neural Networks (MATLAB ANN TOOLBOX) with real-time software data sets that contain features/process metrics like Requirement Stability (RS), Requirement Fault Density (RFD), process maturity, Quality of documented test cases (QDT), etc. They validated the fuzzy model and ANN model based on Mean Magnitude Relative Error (MMRE) and Balanced Mean Magnitude Relative Error (BMMRE). However, the authors didn't discuss how ANN performed better than fuzzy modeling and the features' importance in ANN modeling.

According to Amirabbas Majd et al. [35], statement-level predictions with a Deep Learning Long Short Term Memory (LSTM) approach can help Software Defect Prediction (SDP) systems to discover bug proneness. Similar to the process and CK metrics, they employed statement-level metrics that are categorized into external-linear: Function, Blocks Count, FOR Block, Recursive Blocks Count, etc; and Internal-linear: statement-level metrics like Literal Count, Variable Count, Pointer Count, etc. With minimal parameter tuning and feature selection strategies, they trained their LSTM model with Code4Bench data set [44] and their model's recall is 70 percent better than the Random forest model's recall. They attributed the model's better performance to their diverse data sets and LSTM's ability to retain/memorize the code structure for generalization purposes.

Peng He et al. [33] conducted a empirical study on Software Defect Prediction using models like Naive bayes, Logistic Regression, Bayesian Networks, Support Vector Machine, and Decision Tree with 34 data sets (Ant, Camel, Ivy, Jedit, Lucene, Velocity, etc.). They introduced a concept called coverage index – measure the degree of coverage between two groups of metrics – that helped them to present simplified metrics through the top 5 metrics (CBO, LOC, Lack of Cohesion Methods, etc) and applied them to the above-mentioned models. They discovered that NB and Bayesian networks perform well with simplified metrics when the recall or the F-measure (false negatives) of a system is important. The logistic regression and SVM models performed well when precision is important.

Since the ensemble approach involves extracting insights from multiple modeling results, Ran Li et al.[34] performed an empirical study of ensemble approaches like Ada boost, bagging, Random subspace method (RSM), Random Forest (RF), Vote using NASA MDP data set, and selected RF as their model based on F-measure and Area Under Curve (AUC). Also, they had undertaken the SMOTE and Resample under-sampling method to balance and optimize the data. The improvised RF model with sampling's AUC and F-measure is 25 percent better than a normal RF model.

Chapter 3

Fuzzy Logic Theory

3.1 Classical logic and Fuzzy logic

Classical logic means binary inference of elements in the universe: yes or no; true or false. Timothy[50] states that for a binary/classical logic, $T(P)$, where T stands for truth and P stands for proposition, maps a binary value (either 0 (false) or 1(truth)) for elements in the universe of all propositions.

$$T : u \in U \rightarrow (0, 1) \tag{3.1}$$

All elements u in the universe U that are true for proposition P defined as the truth set of P, denoted $T(P)$. And, those elements that are false for proposition P are called the falsity set of P [50].

Fuzzy logic is an extension of multi-valued logic that takes any real number between 0 to 1 (both inclusive) to handle the partial truth [51]. [50] A fuzzy logic proposition, $P\sim$ is a statement that doesn't entail precisely defined boundaries.

[50] Fuzzy helps us to describe vague and imprecise terms. Fuzzy propositions used to describe a person's height or a current/ambient temperature (slightly hot, hot, cold, very cold, etc) or movie quality (bad, okay, good, very good, etc). Fuzzy logic captures the human mind's way of reasoning.

Fuzzy propositions are assigned to fuzzy sets. Suppose proposition $\underset{\sim}{P}$ is assigned to fuzzy set $\underset{\sim}{A}$; then, the truth value of a proposition, denoted $T(\underset{\sim}{P})$, is given by

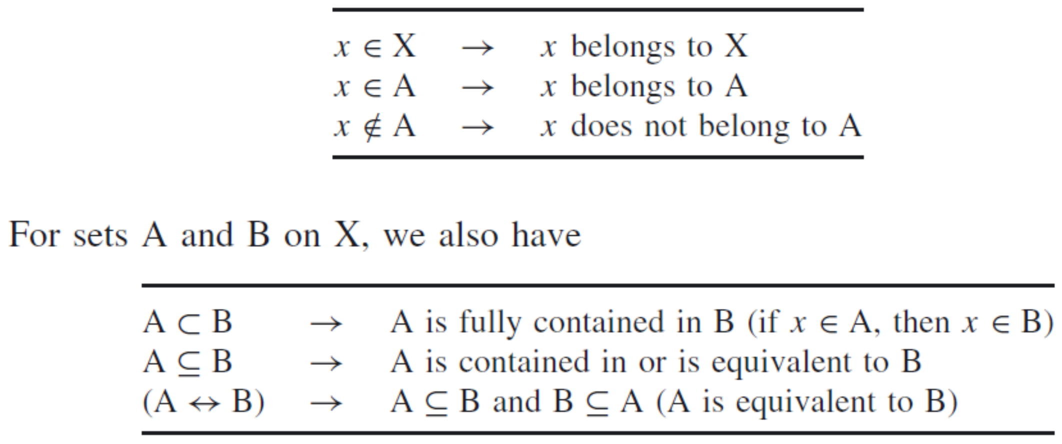
$$T(\underset{\sim}{P}) = \mu_{\underset{\sim}{A}}(x), where\, 0 \leq \mu_{\underset{\sim}{A}}(x) \leq 1 \tag{3.2}$$

3.2 Fuzzy sets and Classical sets

3.2.1 Classical sets and Operations

A set is defined as a mathematical abstraction of universe (universe of discourse elements) and the events that are associated with it. The universe of discourse consists of elements with the same characteristics.

A classical set or a crisp set is a collection of distinct objects that belong to the universe of discourse. For example, a car that belongs to a set of vehicles (Motorbike, Tractor, etc). However, there are two types of universes: discrete and continuous universe. The discrete universe contains a finite collection of elements and cardinal number (count of elements). The continuous consists of infinite set of elements. We can define the relation between crisp sets A and B and universe as follows [50]:



$x \in$ X	$\rightarrow$	x belongs to X
$x \in$ A	$\rightarrow$	x belongs to A
$x \notin$ A	$\rightarrow$	x does not belong to A

For sets A and B on X, we also have

A $\subset$ B	$\rightarrow$	A is fully contained in B (if $x \in$ A, then $x \in$ B)
A $\subseteq$ B	$\rightarrow$	A is contained in or is equivalent to B
(A $\leftrightarrow$ B)	$\rightarrow$	A $\subseteq$ B and B $\subseteq$ A (A is equivalent to B)

Figure 3.1: Crisp sets and Universe

Classical Sets Operations comprise Union, Intersection, Complement, and Difference [50]. For example, if we have sets A and B on the Universe X, then the operations will be performed as below.

Union	$\mathrm{A} \cup \mathrm{B} = \{x \mid x \in \mathrm{A} \text{ or } x \in \mathrm{B}\}.$
Intersection	$\mathrm{A} \cap \mathrm{B} = \{x \mid x \in \mathrm{A} \text{ and } x \in \mathrm{B}\}.$
Complement	$\overline{\mathrm{A}} = \{x \mid x \notin \mathrm{A}, x \in \mathrm{X}\}.$
Difference	$\mathrm{A} \mid \mathrm{B} = \{x \mid x \in \mathrm{A} \text{ and } x \notin \mathrm{B}\}.$

Figure 3.2: Classical sets operations

3.2.2 Fuzzy sets and operations

The fuzzy sets contain elements with no precise boundaries–they are vague and ambiguous. A particular element can be present in one or many fuzzy sets in the same universe with varying membership degree values (a real number in the range [0,1]). Hence, membership of an element (universe) in the fuzzy set is measured by a function that attempts to describe vagueness and ambiguity.

For example, element on the universe, say x, is a member of fuzzy set $\underset{\sim}{A}$, then the mapping equation is as follows ($0 \leq \mu_{\underset{\sim}{A}}(x) \leq 1$):

$$\underset{\sim}{\mathrm{A}} = \left\{ \frac{\mu_{\underset{\sim}{\mathrm{A}}}(x_1)}{x_1} + \frac{\mu_{\underset{\sim}{\mathrm{A}}}(x_2)}{x_2} + \cdots \right\} = \left\{ \sum_i \frac{\mu_{\underset{\sim}{\mathrm{A}}}(x_i)}{x_i} \right\}.$$

When the universe, X, is continuous and infinite, the fuzzy set $\underset{\sim}{\mathrm{A}}$ is denoted as

$$\underset{\sim}{\mathrm{A}} = \left\{ \int \frac{\mu_{\underset{\sim}{\mathrm{A}}}(x)}{x} \right\}.$$

Figure 3.3: Fuzzy sets

Consider three fuzzy sets $\underset{\sim}{A}$, $\underset{\sim}{B}$, $\underset{\sim}{C}$ on the Universe X. For them, the available **fuzzy sets operations** are as follows:Union, Intersection, and Complement.

Union $\mu_{\underset{\sim}{A} \cup \underset{\sim}{B}}(x) = \mu_{\underset{\sim}{A}}(x) \vee \mu_{\underset{\sim}{B}}(x).$

Intersection $\mu_{\underset{\sim}{A} \cap \underset{\sim}{B}}(x) = \mu_{\underset{\sim}{A}}(x) \wedge \mu_{\underset{\sim}{B}}(x).$

Complement $\mu_{\overline{\underset{\sim}{A}}}(x) = 1 - \mu_{\underset{\sim}{A}}(x).$

Figure 3.4: Fuzzy sets Operations

3.3 Membership Functions

As discussed earlier, membership functions represent the degree of truth for elements in the fuzzy sets [45]. We will discuss the features and types of membership functions.

3.3.1 Features

[50] The core of a membership function for some fuzzy set $\underset{\sim}{A}$ is defined as the region of the universe that possesses complete and full membership in the set $(\mu_{\underset{\sim}{A}}(x) = 1)$.

For the same set $\underset{\sim}{A}$, the support of a membership function is defined as the region of the universe has a nonzero membership $(\mu_{\underset{\sim}{A}}(x) > 0)$.

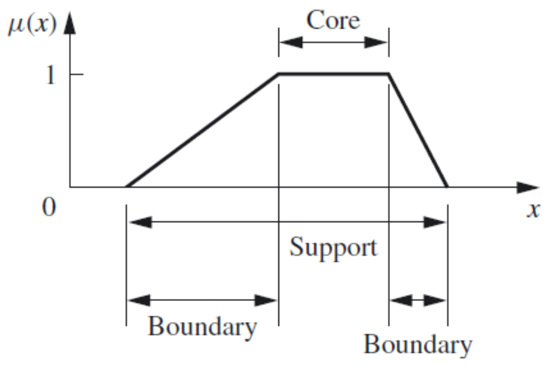


Figure 3.5: Features of membership functions

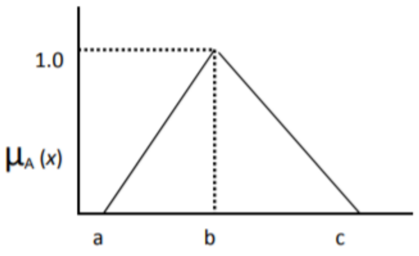
The boundaries of a membership function belong to the region of the universe containing elements that have a nonzero membership but not complete membership $((0 \leq \mu_{\underset{\sim}{A}}(x) \leq 1))$[50].

The crossover points of a membership function for a particular fuzzy set $\underset{\sim}{A}$ has values equal to 0.5 $(\mu_{\underset{\sim}{A}}(x) = 0.5)$.

The height of a fuzzy set $\underset{\sim}{A}$ is the maximum value of the membership function : hgt($\underset{\sim}{A}$) = max $(\mu_{\underset{\sim}{A}}(x))$. If the hgt($\underset{\sim}{A}$) < 1, the fuzzy set is defined as subnormal.

3.3.2 Types of membership functions

Triangular Membership function [54]: Let a, b and c represent the x coordinates of the three vertices of $(\mu_{\underset{\sim}{A}}(x))$ in a fuzzy set A (a: lower boundary and c: upper boundary where membership degree is zero, b: the centre where membership degree is 1).



$$\mu_A(x) = \begin{cases} 0 & \text{if } x \le a \\ \dfrac{x-a}{b-a} & \text{if } a \le x \le b \\ \dfrac{c-x}{c-b} & \text{if } b \le x \le c \\ 0 & \text{if } x \ge c \end{cases}$$

Figure 3.6: Triangular membership function

Trapezoidal Membership function is defined by a lower limit a, an upper limit d, a lower support limit b, and an upper support limit c, where a < b < c < d.

$$\mu_A(x) = \begin{cases} 0, & (x < a) \text{ or } (x > d) \\ \dfrac{x-a}{b-a}, & a \le x \le b \\ 1, & b \le x \le c \\ \dfrac{d-x}{d-c}, & c \le x \le d \end{cases}$$

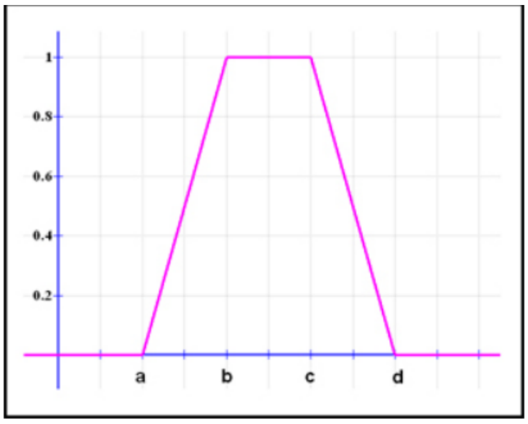


Figure 3.7: Trapezoidal membership function

Gaussian membership function is defined as Gaussian(x:c,s) where c, s represents the mean and standard deviation.

$$\mu_A(x,c,s,m) = \exp\left[-\frac{1}{2}\left|\frac{x-c}{s}\right|^m\right]$$

Here c represents centre, s represents width and m represents fuzzification factor.

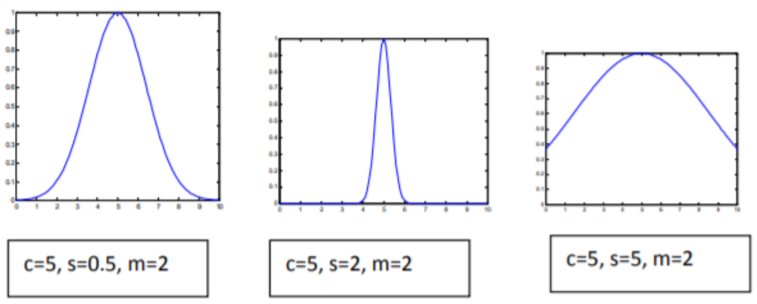


Figure 3.8: Gaussian membership function

A Generalized bell membership function membership function contains three parameters: a corresponds to width, c corresponds to center and b corresponds to slope.

$$gbellmf(x;a,b,c) = \frac{1}{1+\left|\frac{x-c}{b}\right|^{2b}}$$

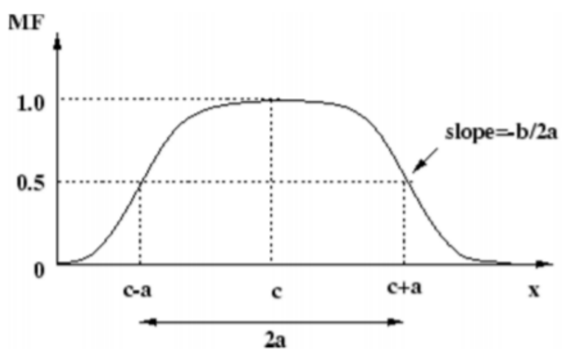


Figure 3.9: Generalized bell membership function

A **Sigmoid membership function** can be defined as Sigmf (x:a, c), the magnitude of a controls the width of the transition area, and c defines the center of the transition area (52).

$$f(x; a, c) = \frac{1}{1 + e^{-a(x-c)}}$$

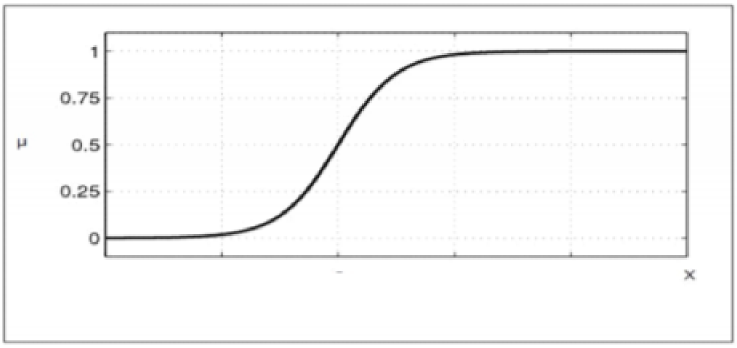


Figure 3.10: Sigmoid membership function

3.4 Fuzzification

Fuzzification is a process of transforming a crisp quantity to a fuzzy variable. Since crisp values also have considerable uncertainty, we perform fuzzification as crisp values are not purely deterministic. If the form of uncertainty happens to arise because of imprecision, ambiguity, or vagueness, then the variable can be represented through fuzzy membership functions [50].

For example, measuring water consumption with a water meter (crisp) or measuring voltage with the voltmeter could generate imprecise data due to experimental error.

To explain fuzzification of variables, let's consider the famous tipping problem example [49]. In this problem, we have food quality and service as inputs or Antecedents and tip percentage as your output. When we provide a crisp value for quality and service on a scale of 0 to 10, each antecedent and consequent will be transformed to fuzzy sets (poor, average, good).

If we choose a triangular membership function and apply parameters, we will get a membership function for the antecedents.

As mentioned in section 3.3.2, if trimf(x, abc) (triangular membership function) is a function, where x (array) takes crisp values, and "abc" are three parameters of trimf function, where $(a \leq b \leq c)$. The resultant fuzzy values are calculated according to the trimf formula.

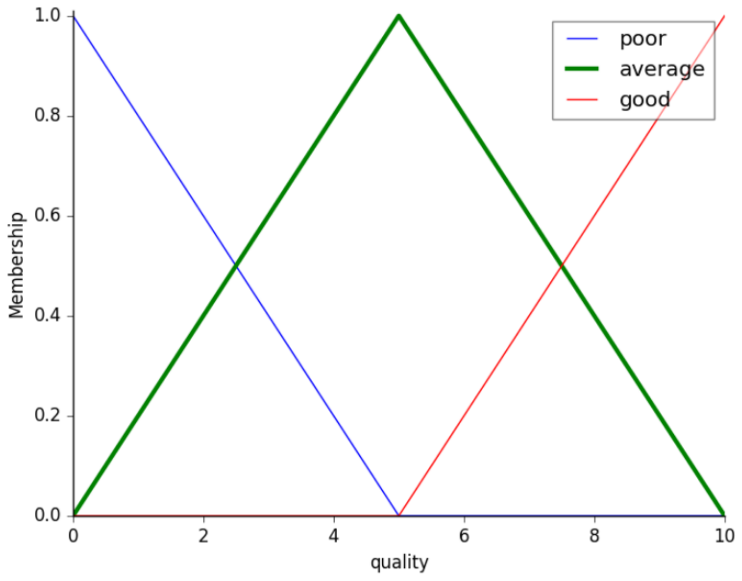


Figure 3.11: fuzzy quality - tipping problem

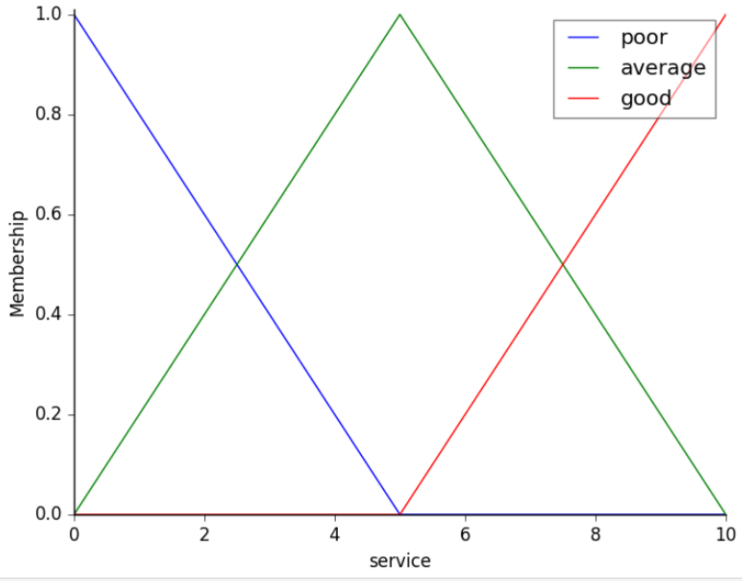


Figure 3.12: fuzzy service - tipping problem

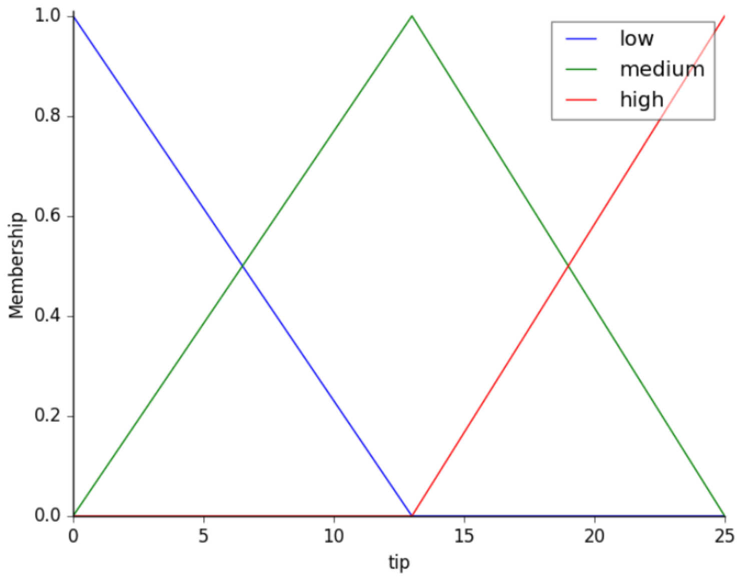


Figure 3.13: fuzzy tip - tipping problem

3.5 Fuzzy Systems with rules/inference

The fuzzy rule-based systems contain rules that help translate human knowledge to action. The rules are IF-THEN constructs that take antecedents and consequent as a part of the rule statements [50].

$$IF\, premise\ (antecedent), THEN\ conclusion\ (consequent). \tag{3.3}$$

The fuzzy rule-based system is most suitable in modeling complex systems that can be observed by humans because of the usage of linguistic variables as their antecedents and consequents. These variables are represented by fuzzy sets and logical connectives of these sets.

Rule 1:	IF condition C^1, THEN restriction R^1
Rule 2:	IF condition C^2, THEN restriction R^2
$\vdots$	
Rule r:	IF condition C^r, THEN restriction R^r

Figure 3.14: The canonical form for a rule based system.

Restrictions present in the canonical form in Figure 3.14 are generally modeled by fuzzy sets and relations. These restriction statements are connected by linguistic connectives such as “AND,” “OR,” or “ELSE” There could be multiple rules in a system, and in a rule statement, there could be multiple rule statements with fuzzy antecedents connected by linguistic connectives ”AND” and ”OR” (Conjunctive Antecedents and Disjunctive Antecedents).

3.5.1 Aggregation of rules

As there could be multiple rules for a fuzzy system, it is essential to have a strategy to combine the rules and produce a final membership function for a consequent. The strategies for the aggregation are as follows.

Conjunctive system of rules means the rule consequents are combined using a fuzzy intersection. For example, if y is the consequent, then the aggregation y^i, where i = 1,2..r as follows:

$$y = y^1 \text{ and } y^2 \text{and} \ldots \text{and } y^r$$

or

$$y = y^1 \cap y^2 \cap \cdots \cap y^r,$$

$$\mu_y(y) = \min(\mu_{y^1}(y), \mu_{y^2}(y), \ldots, \mu_{y^r}(y)), \text{ for } y \in \mathrm{Y}.$$

Figure 3.15: Fuzzy Intersection

Disjunctive system of rules is guided by union operation of all the rule consequents.

$$y = y^1 \text{ or } y^2 \text{ or} \ldots \text{or } y^r$$

or

$$y = y^1 \cup y^2 \cup \cdots \cup y^r,$$

$$\mu_y(y) = \max(\mu_{y^1}(y), \mu_{y^2}(y), \ldots, \mu_{y^r}(y)), \text{for} y \in \mathrm{Y}.$$

Figure 3.16: Fuzzy Union

We can discuss techniques of inference under the context of different fuzzy systems.

3.5.2 Fuzzy Systems

There are different types of fuzzy systems:

1. Mamdani System
2. Takagi Sugeno Model
3. Tsukamoto models.

The Mamdani System [47] is a common and popular fuzzy system in both academic literature and industries [47]. To discuss the inference technique, let's consider a two rule system with two antecedents and a consequent.

$$\text{IF } x_1 \text{ is } \underset{\sim}{\mathrm{A}}{}_1^k \text{ and } x_2 \text{ is } \underset{\sim}{\mathrm{A}}{}_2^k \text{ THEN } y^k \text{ is } \underset{\sim}{\mathrm{B}}{}^k, \qquad \text{for } k = 1, 2, \ldots, r,$$

Figure 3.17: Mamdani system rules

$\underset{\sim 1}{A}{}^k$ and $\underset{\sim 2}{A}{}^k$ are fuzzy sets representing the antecedents. $\underset{\sim}{B}{}^k$ represents consequent.

There are two scenarios we might handle with this system: (1) the inputs are scalar values, and we utilize a max-min inference method, and (2) For the same inputs, and we use a max–product inference method.

The rule cuts for one of the scenarios are as follows:

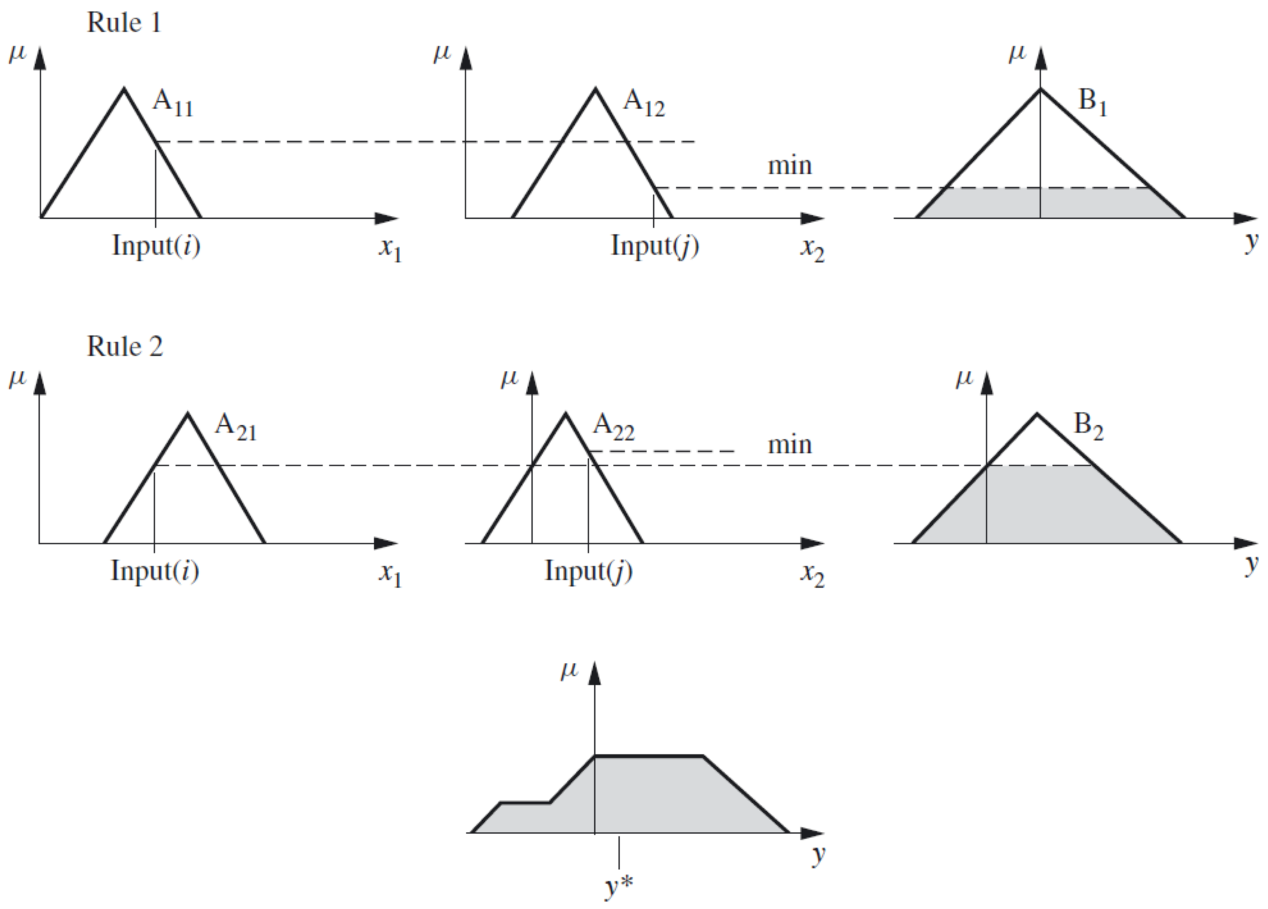


Figure 3.18: Mamdani rule inference (max-min)

After attaining the final consequent, we will apply some defuzzification methods to extract the crisp output. The defuzzification methods will be discussed in the next section.

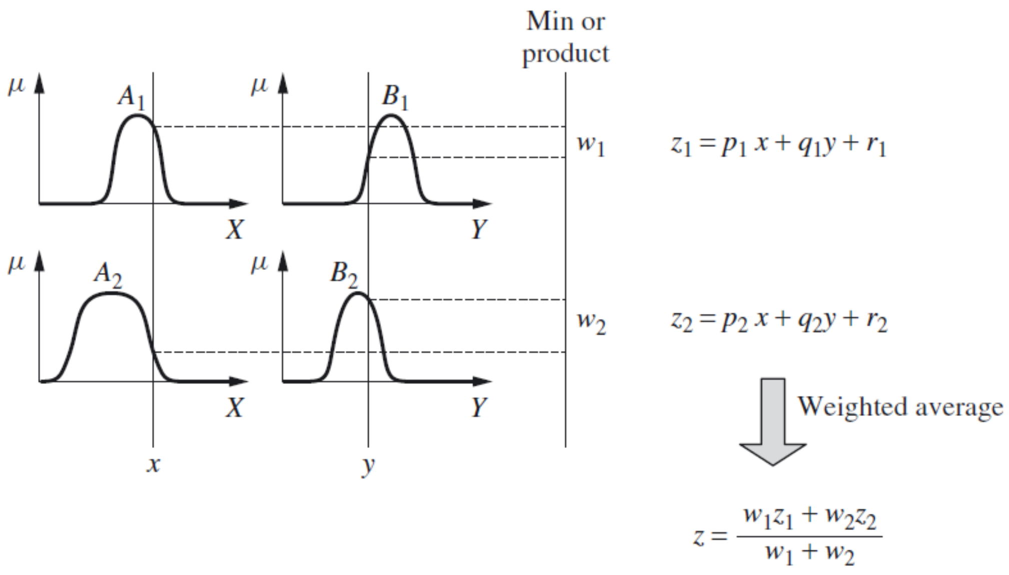
The **Sugeno model**[48] contains the same antecedent structure as similar to Mamdani, but the consequent is handled by a polynomial function. An example of a rule structure in the Sugeno model is as follows:

$$\text{IF } x \text{ is } \underset{\sim}{\mathrm{A}} \textit{ and } y \text{ is } \underset{\sim}{\mathrm{B}},\ \text{THEN } z \text{ is } z = f(x, y)$$

Figure 3.19: Sugeno rule

where z = f (x, y) is a crisp function in the consequent. f (x, y) is a polynomial function that takes inputs x and y, but it could be any kind of function (linear or non-linear).

[50]In a Sugeno model, each rule produces a crisp output with the help of a consequent function. After collecting the fuzzified values, we use the weighted average defuzzification method to retrieve the crisp output. This is a computationally low-intensive process than Mamdani's defuzzification process.



Min or product
μ
A_1
B_1
w_1
$z_1 = p_1 x + q_1 y + r_1$
X
Y
A_2
B_2
w_2
$z_2 = p_2 x + q_2 y + r_2$
Weighted average
x
y
$z = \frac{w_1 z_1 + w_2 z_2}{w_1 + w_2}$

Figure 3.20: Sugeno rule inference with the Weighted average defuzzification.

Tsukamoto model [53] has the same rule structure as the Mamdani system (premise and consequent are fuzzy), but the consequent part is a monotonically increasing or decreasing membership function (shoulder function) [50]. It uses the weighted defuzzification method and produces a crisp output. However, this model is useful for only specific situations or experiments.

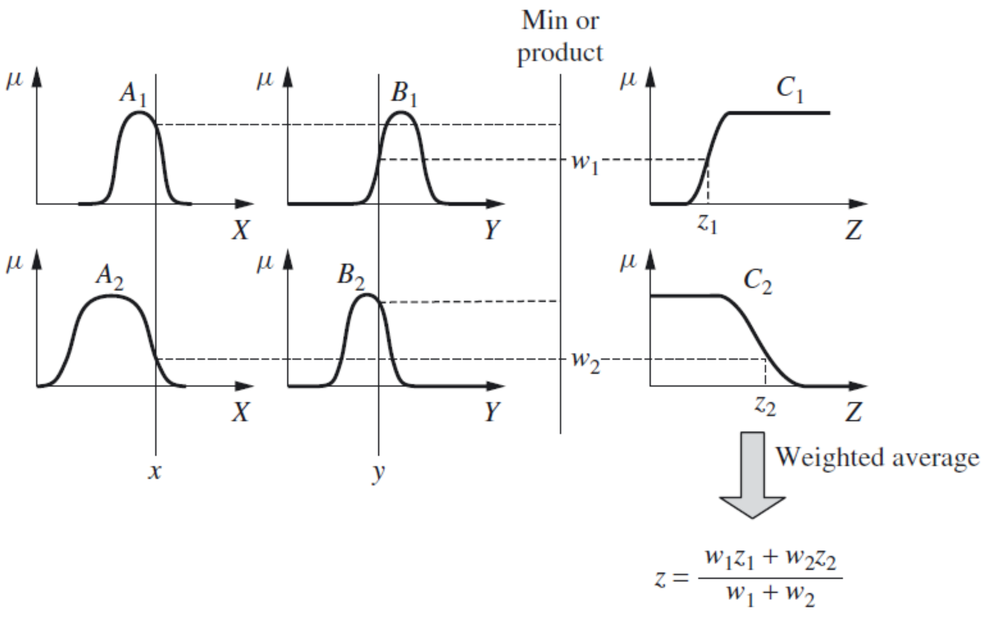


Figure 3.21: Tsukamoto fuzzy model inference with Weighted average defuzzification.

3.6 Defuzzification methods

After processing the rules, we need a defuzzification method to defuzzify the fuzzy membership function values to crisp outputs. Please note that most of these methods are actively applied in the Mamdani system and the other models (Sugeno and Tsukamoto) use the weighted average method.

3.6.1 Centroid method

[52]Centroid defuzzification calculates the center of gravity of the fuzzy set membership function. The centroid (crisp output) is derived using the below formula, where $\mu(x_i)$ is the membership value for point x_i in the universe of discourse [52]. The resultant value is the fuzzy generated scalar output.

$$\text{xCentroid} = \frac{\sum_i \mu(x_i) x_i}{\sum_i \mu(x_i)}$$

Figure 3.22: Centroid formula

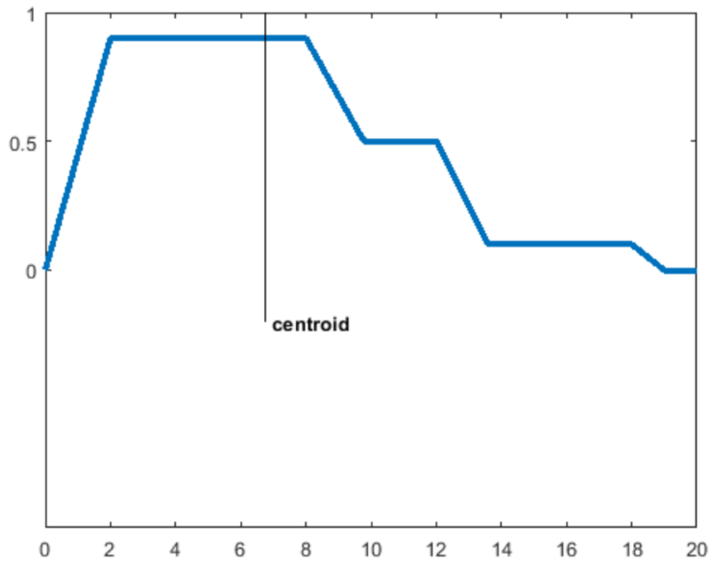


Figure 3.23: Centroid defuzzification

3.6.2 Bisector method

The bisector method finds a point that divides the fuzzy set membership into two halves of equal area. The defuzzified value is almost similar to value computed by Centroid defuzzification [52].

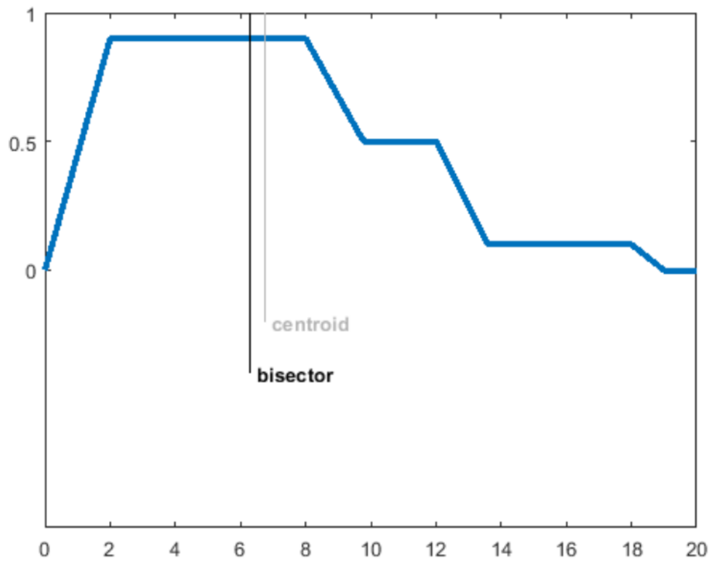


Figure 3.24: Centroid defuzzification

3.6.3 Mean of maximum (MOM), Largest of maximum (LOM), and Smallest of maximum (SOM)

All the three defuzzification methods locate the maximum membership locations (plateau) and extract the crisp output based on their nature.

MOM extracts the mean of the resultant set of maximum membership locations (MML) of crisp inputs.

SOM returns the minimum of MML of crisp inputs, and LOM returns the maximum of MML of inputs.

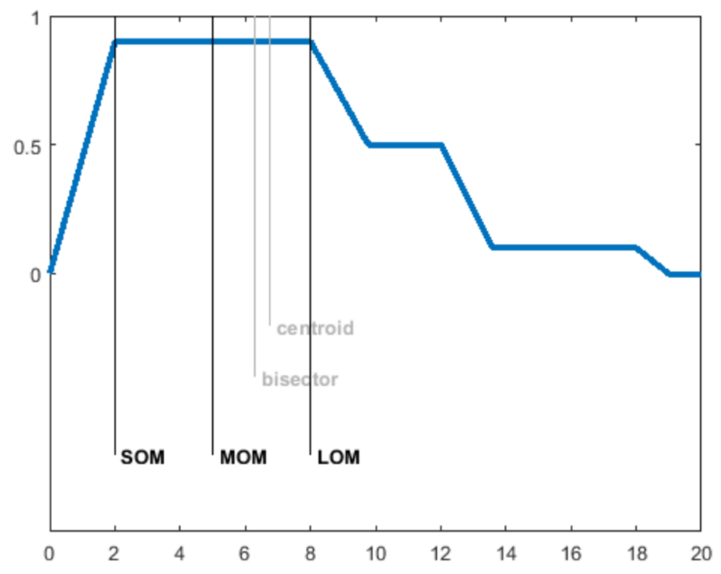


Figure 3.25: MOM, SOM, and LOM defuzzification methods

3.6.4 Weighted average defuzzification method

The weighted average method is the most commonly used in fuzzy applications as it is computationally efficient.

$$z^* = \frac{\sum \mu_{\underset{\sim}{C}}(\overline{z}) \cdot \overline{z}}{\sum \mu_{\underset{\sim}{C}}(\overline{z})},$$

Figure 3.26: Weighted average formula

$\sum$ denotes the algebraic sum and $\bar{z}$ stands for the centroid of each symmetric membership function. The weighted average method's weighting in output is performed by selecting its respective maximum membership value. However, as we discussed in the Sugeno model (Figure 3.20), this is applied to various unsymmetrical functions [50].

Chapter 4

Software Defect Dataset

We utilized a dataset provided by M. D'Ambros et al. [14] that has a collection of software metrics and histories derived from large software systems: Eclipse JDT Core, Eclipse PDE UI, Equinox Framework, Lucene, and Mylyn. Each and every system consists of software metrics (features): Change metrics, CK metrics, Object-oriented metrics, Complexity of code change, Code churn metrics. Software metrics type and definition and their members are available in table 4.1. They collected the data with the changelog information, source code version information, and defect details through the SCM system (Subversion) and bug tracking systems (Bugzilla/ Jira/ Zoho Bug Tracker). They linked FAMIX (a meta-model of object-oriented code) classes with bug reports, performed pattern matching with bug ids and time stamps, and retrieved the bug count for each class.

As a part of data preprocessing, we applied a log transformation on a selected set of features (Eclipse JDT and Apache Lucene). This is because our fuzzy model learns parameters from data sets so we need to achieve a normal distribution. As for the model evaluation, we applied K-fold cross-validation on linear regression and random forest regression model using scikit learn [46]. This evaluation procedure is discussed in detail in chapter 5.

Metrics Type	Definition	Metrics members
Change metrics	Metrics that are related to file-level changes (especially, buggy changes), code refactoring and other version control related attributes.	Number of Revisions (NR), Number of times file involved in bug fixing (NFIX),Number of authors who committed that file (NAUTH), etc.
CK metrics	Chidamber and Kermer metrics entail concepts for object-oriented model software and design	Coupling Between Objects (CBO), Depth of Inheritance Tree (DIT), Lack of Cohesion in Methods (LCOM), etc.
Object oriented metrics	Metrics that measure the object-oriented related design components and concepts	Number of private methods (NPM), Number of Attributes Inherited (NOAA), Number of public attributes (NPA), etc.
Complexity of code change	Measurement and distribution of code changes over a time interval in a system.	History of Complexity Metric (HCM), ED-HCM (Exponentially Decayed HCM), etc.
Code churn metrics	It involves sampling of source code history and calculating the deltas of source code metrics.	Partial Churn(PCHU), Weighted Churn (WCHU), etc.

Table 4.1: Software Metrics

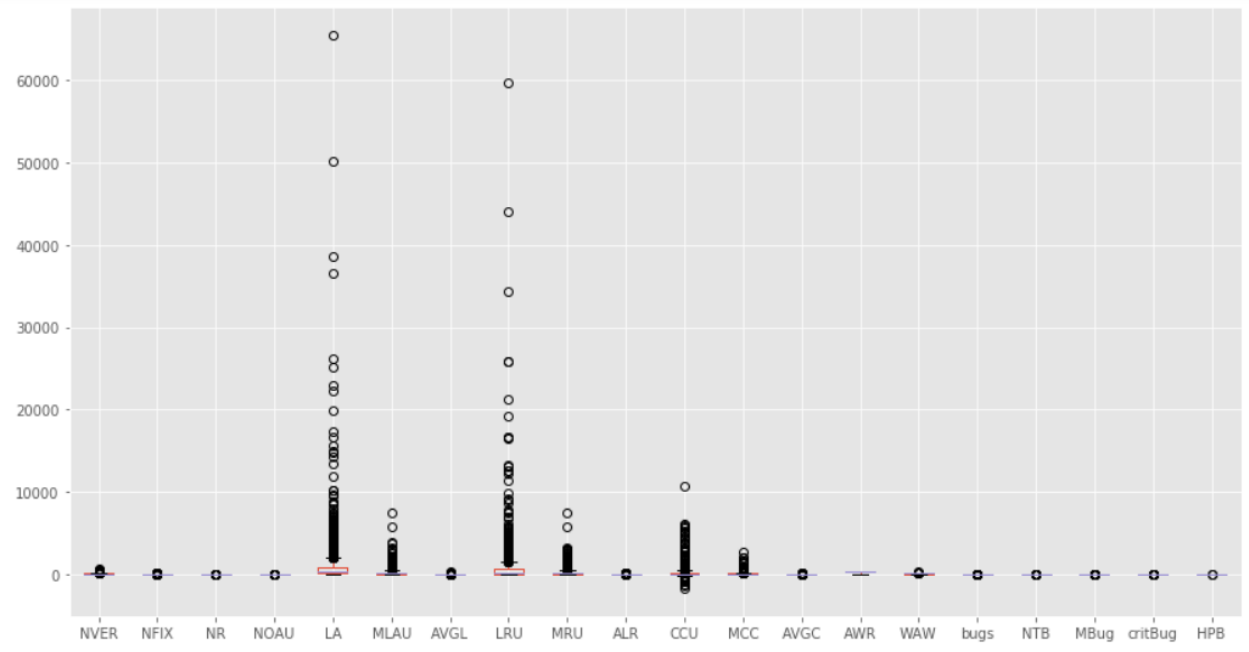


Figure 4.1: Change metrics data distribution (Eclipse JDT)

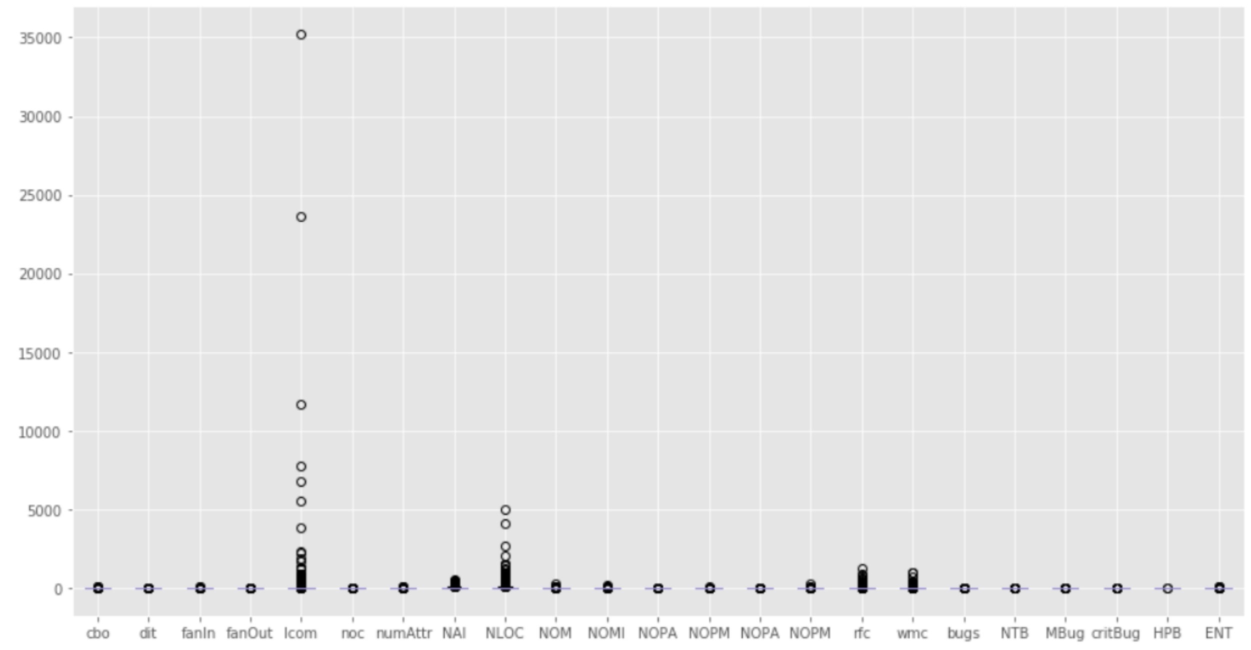


Figure 4.2: Churn metrics data distribution (Eclipse JDT)

The non-normalized data distribution has been provided above. The column names are in short form due to space limitations. Below is the table with the full form in the same order.

Churn Metrics	Classname, Coupling between Objects (CBO) , Depth of Inheritance Tree (DIT), fanIn, fanOut, lack of cohesion in methods (lcom), Number of Childre (noc), numberOfAttributes(NOA), numberOfAttributesInherited (NOAI), numberOfLinesOfCode (NLOC), numberOfMethods (NOM), numberOfMethodsInherited (NOMI), numberOfPrivateAttributes (NOPA), numberOfPrivateMethods (NOPM), numberOfPublicAttributes (NOPA), numberOfPublicMethods (NPM), Response for class (rfc), Weighted method count (wmc), bugs, nonTrivialBugs (NTB), majorBugs (MB), criticalBugs (critBug) , highPriorityBugs (HPB), CvsEntropy (ENT)
Change Metrics	Classname , numberOfVersions (NVER), numberOfFixes (NFIX), numberOfRefactorings (NR), numberOfAuthors (NOA), linesAdded (LA), maxLinesAdded (MLA), avgLinesAdded (AVGL), linesRemoved (LR), maxLinesRemovedUntil (MR), avgLinesRemoved (ALR), codeChurn (CC), maxCodeChurn (MCC), avgCodeChurn (AVGC), ageWithRespectTo (AWR), weightedAgeWithRespectTo (WAW), bugs, nonTrivialBugs (NTB), majorBugs (MB), criticalBugs (CB), highPriorityBugs (HPB).

Table 4.2: Data metrics full abbreviation

Here is the log transformation of a few domain-expert selected features (Change, Churn, and size).

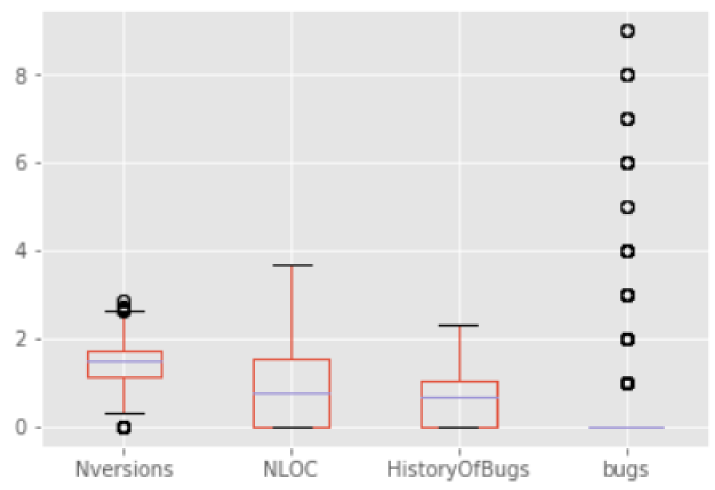


Figure 4.3: Log transformed features (Eclipse JDT)

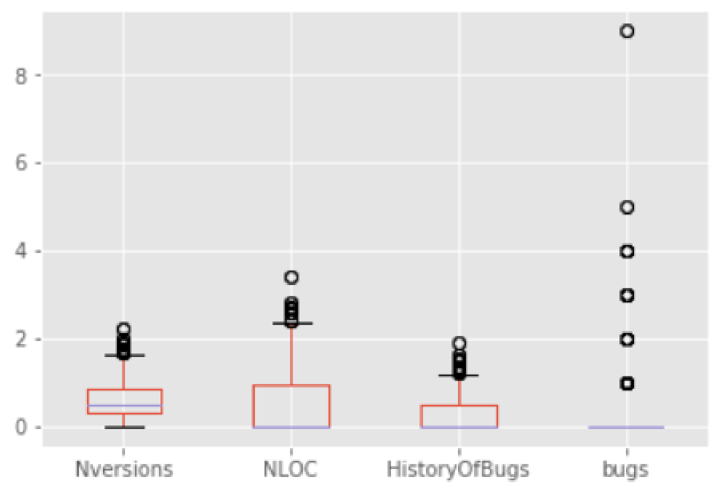


Figure 4.4: Log transformed features (Apache Lucene)

Chapter 5

Experimentation and Results

We developed a fuzzy logic based software defect prediction model with Mamdani and Takagi Sugeno (TS) systems. The algorithm is built using scikit-fuzzy logic library, numpy, and Pandas (Python) that takes in the defect prediction dataset [14] (as discussed in Chapter 4) with configuration parameters to predict discrete predictions (defects). We present the algorithm and explain all the steps in the fuzzy logic procedure.

5.1 Fuzzy logic with Mamdani system

For our Mamdani fuzzy logic based approach, we selected only three features from the dataset: LOC (Halstead or size metrics), Number of Revisions (Change metrics), and Number of Bugs Found (History of bugs) [47] . We chose minimal features to keep our fuzzy logic model simple and test its ability to predict the target with minimum information. Please note that this section is explained using Eclipse JDT, Apache Lucene, Eclipse Mylyn, and Equinox data sets.

5.1.1 Fuzzification of input features

When the fuzzy simulation gets triggered, firstly, the antecedents (inputs) are fed into the scikit-fuzzy's [49] fuzzification method where a membership function's fuzzy sets [low, medium, high] and the fuzzy universe are utilized to map a crisp input to a term (fuzzified value) on a scale of 0 to 1. In our experiment, we utilized a trapezoidal membership function for antecedents as it generalizes well for almost all the data domains. We capture the trapezoidal parameters (a,b,c,d) from the data distribution (Quantile) and configure the fuzzy sets accordingly. Below is the fuzzy set visualization for the antecedents [49].

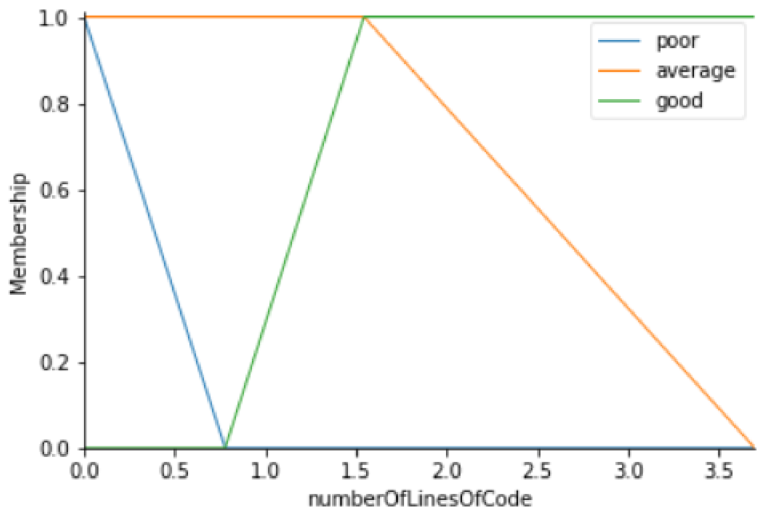


Figure 5.1: Line of code

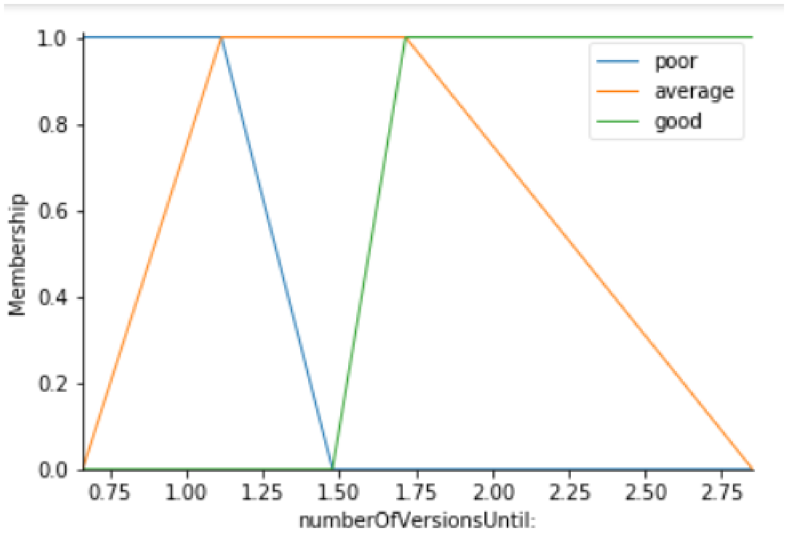


Figure 5.2: Number of Versions

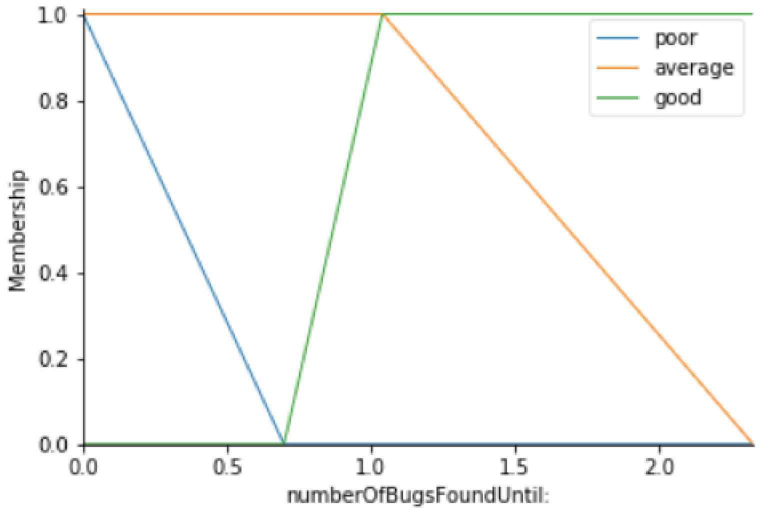


Figure 5.3: History of Bugs

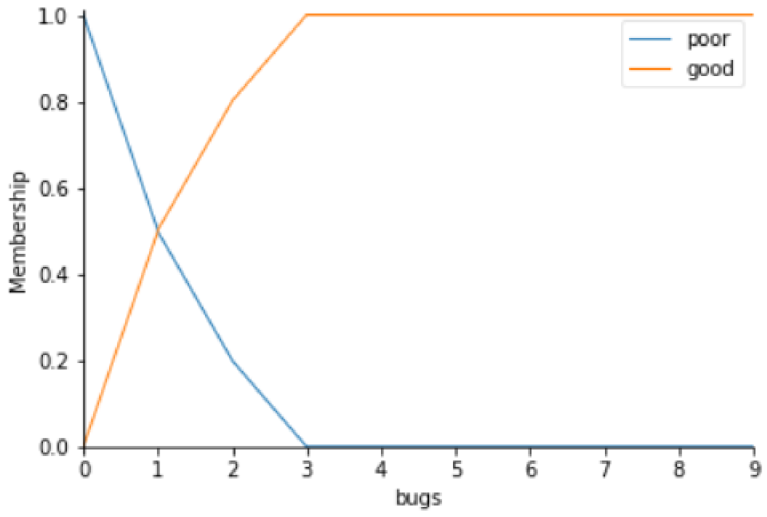


Figure 5.4: Bugs/Defects

Please note that due to difficulty in changing the fuzzy sets names in scikit-fuzzy, consider this mapping: **poor means low; average means medium; good means high**.

For consequent membership functions, we constructed a custom-defined membership function (refer figure 5.4) as the data distribution is skewed for defects: using the trapezoidal membership for consequent yielded an abysmal Root Mean Squared Error (RMSE) value. After examining the data distribution for all the software systems, we designed a paradigm that says for the classes (input Java Classes of the systems) that contain bugs in the range of 0 to 3 will be subjected to the varying degrees of membership values in fuzzy set low and fuzzy set high: in fuzzy set low, 0 will have the highest membership value and 3 will get the lowest membership value and vice versa in fuzzy set high. The classes with bugs more than 3 in fuzzy set high will have membership value 1.

5.1.2 Fuzzy rules design and inference

We designed a rule-set with fuzzy sets low and high to depict the relation between antecedents and consequent. Our simple rule-set definition with IF and THEN constructs is presented in Table 5.1.

IF numberOfLines is low THEN bugs is low
IF numberOfLines is high THEN bugs is high
IF numberOfVersions is low THEN bugs is low
IF numberOfVersions is high THEN bugs is high
IF historyOfBugs is low THEN bugs is low
IF historyOfBugs is high THEN bugs is high

Table 5.1: Mamdani Fuzzy rules for software defect prediction

In scikit-fuzzy [49], after fuzzification, we compute the rules provided by the domain expert to make necessary cuts on all fuzzy sets and calculate a final value using them. The rule computation comprises of three steps in scikit-fuzzy:

1. Aggregation

2. Activation

3. Accumulation

Aggregation refers to the net accomplishment of antecedents by AND or OR operations. If we have more than one antecedent, AND takes the minimum of membership values; OR takes the maximum of membership values. Activation checks if there is any weight (value) associated with consequent to apply before firing the rules. Accumulation keeps accumulating the aggregated values from various rule firings into one membership value.

After computing rules for all the classes, the defuzzification method will receive membership values for further processing of crisp outputs.

5.1.3 Defuzzification of membership values

Scikit-fuzzy logic provides 5 different modes of defuzzification: Centroid, Bisector, Middle of maximum, Smallest of maximum, and Largest of maximum. These methods are extensively explained in the fuzzy theory section (Chapter 3). The Scikit-fuzzy's defuzzification methods take in a composite membership function and process it to produce a crisp output. This will be a single number.

In our experiment, we initially applied Centroid and Bisector defuzzification methods. Since these methods estimated larger predictions than the actual, we employed the smallest of maximum (SOM) defuzzification method, and it resulted in better predictions. Since the SOM is conservative and picks a value agreed upon by all the rules, it works well for our experiments.

A comparison between different defuzzification methods' RMSE on Eclipse JDT is presented below:

Defuzzification Method	RMSE
Centroid/bisector	3.3
Mean of maximum	2.9
Smallest of maximum	1.2
Largest of maximum	4.1

Table 5.2: Different defuzzification methods' RMSE values

The composite membership function generated by all the rules is give below:

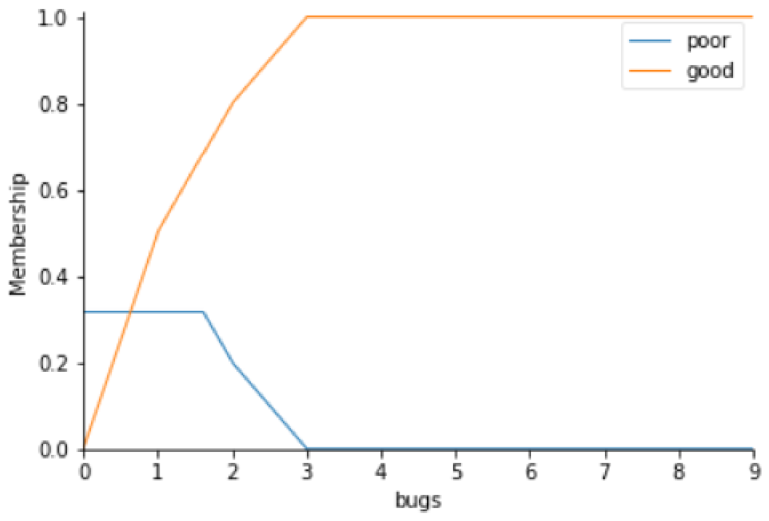


Figure 5.5: Defuzzification

5.2 Fuzzy logic with a Takagi Sugeno system

The TS model applies the same rules as Mamdani but the consequent is defined by a polynomial function. For example, IF x is A1 and y is B1 then Z is $f(x, y)$, where $f(x, y) = p1x+p2y+b1$. A1 and B1 are fuzzy sets. p1, p2, and b1 are the parameters of the polynomial function.

We developed a TS model with fuzzification support from Scikit-fuzzy. For polynomial function parameters, we set the domain- expert-recommended-values. After computing the rules, we send the membership values to TS's weighted average defuzzification method to obtain the crisp outputs.

The results are better than the Mamdani system, but still not better than traditional machine learning algorithms. However, more parameter tuning and optimization are currently in progress.

5.3 Model evaluation with ML algorithms

We tested our model's performance and explanatory power, we utilized Scikit-learn Linear Regression and Random Forest Regression with K-Fold cross-validation. The models with cross-validation training provide better accuracy than normal training. However, the time complexity of these ML models with CV is considerably higher than fuzzy logic models. In the results section, we compare the performances of all the models with the help of RMSE (Root Mean Squared Error) and Mean Absolute Error (MAE).

5.4 Results

Here are the evaluation results of all the models for various metrics using Eclipse JDT and Lucene dataset.

Evaluation Metric	Model	Score
RMSE	LR	0.50
	RF	0.42
	TS Fuzzy	0.90
	Mamdani Fuzzy	1.20
MAE	LR	0.41
	RF	0.42
	TS Fuzzy	0.44
	Mamdani Fuzzy	0.70

Table 5.3: Evaluation Results with Eclipse JDT Dataset

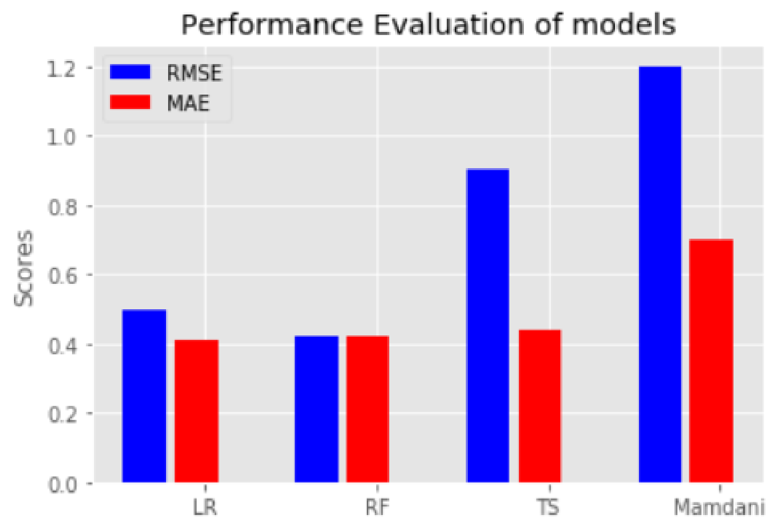


Figure 5.6: Evaluation with bar charts using Eclipse JDT Dataset

Evaluation Metric	Model	Score
RMSE	LR	0.25
	RF	0.17
	TS Fuzzy	0.55
	Mamdani Fuzzy	1.4
MAE	LR	0.17
	RF	0.17
	TS Fuzzy	0.19
	Mamdani Fuzzy	0.78

Table 5.4: Evaluation Results with Apache Lucene Dataset

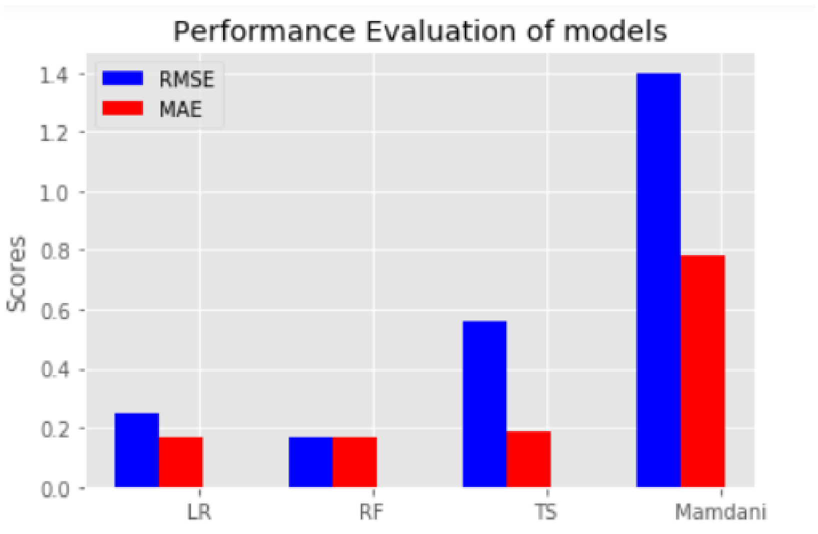


Figure 5.7: Evaluation with bar charts using Lucene Dataset

Evaluation Metric	Model	Score
RMSE	LR	0.28
	RF	0.28
	TS Fuzzy	0.68
	Mamdani Fuzzy	1.1
MAE	LR	0.18
	RF	0.29
	TS Fuzzy	0.33
	Mamdani Fuzzy	0.67

Table 5.5: Evaluation Results with Mylyn Dataset

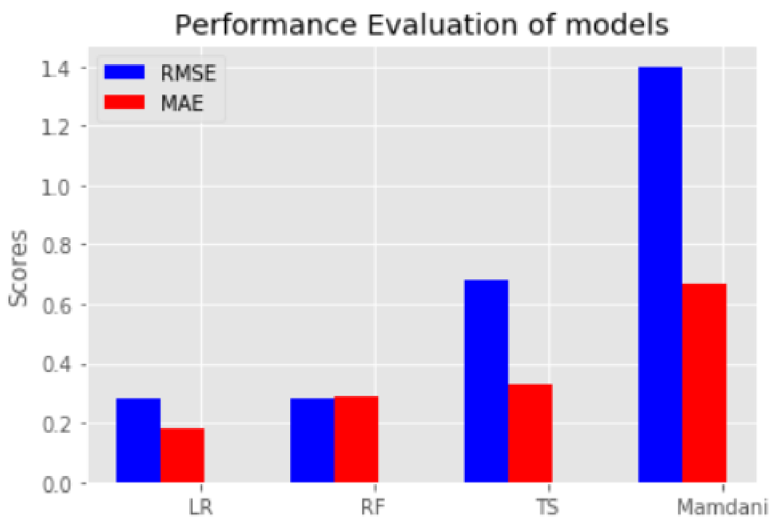


Figure 5.8: Evaluation with bar charts using Mylyn Dataset

Evaluation Metric	Model	Score
RMSE	LR	0.67
	RF	0.61
	TS Fuzzy	1.3
	Mamdani Fuzzy	1.6
MAE	LR	0.65
	RF	0.60
	TS Fuzzy	0.63
	Mamdani Fuzzy	1.04

Table 5.6: Evaluation Results with Equinox Dataset

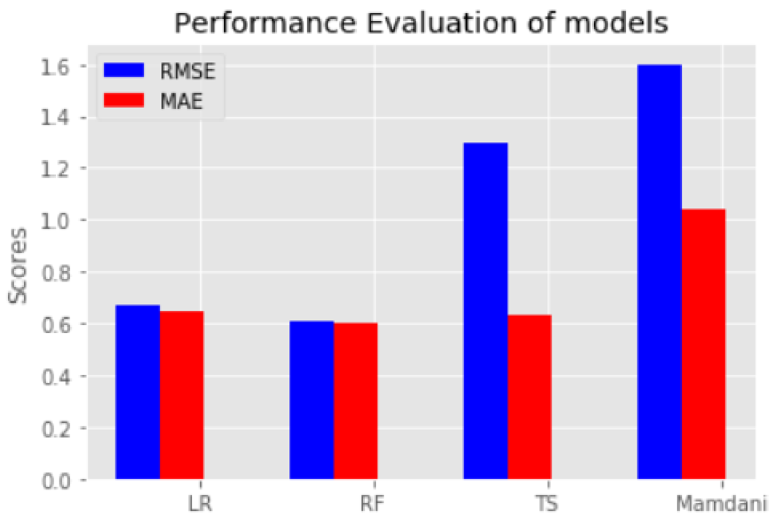


Figure 5.9: Evaluation with bar charts using Equinox Dataset

The results suggest that LR and RF perform well under all the metrics. The improvised Mamdani system's results might not look impressive when compared to ML, but it is 60 to 75 percent better than traditional Mamdani systems (refer table 5.2). However, the Takagi Sugeno model proves to be competitive, especially, with MAE. As the consequent takes both linear and non-linear functions, the final defuzzified value is shaped well for better predictions.

However, the Linear Regression Model with CV performed well since the features have good explanatory power and considerable linear relationships with the target. Also, the parameters are tuned/approximated using least squares.

The random forest regression model is an ensemble model with many estimators. For both regression and classification, there are sub-trees that are trained with different sets of features. The RF takes a final decision based on estimations from all learners that leads to a good performance. However, these models are inherently complex with the count of estimators and depth.

From the results and the nature of models, we can discern that the computationally low-intensive TS fuzzy logic model almost performs similarly to the ML models, and the Mamdani system has a lot of scope for improvement.

To understand the prediction power in terms of classification, we generated the class labels for defects (true/false) using binning and predicted using our fuzzy systems, and evaluated our results with Logistic Regression, and Random Forest classifier. Below are the results and discussions of the classification.

Evaluation Metric	Model	Score
Precision	LR	0.83
	RF	0.83
	TS Fuzzy	0.82
	Mamdani Fuzzy	0.81
Recall	LR	0.84
	RF	0.84
	TS Fuzzy	0.73
	Mamdani Fuzzy	0.71
F-measure	LR	0.83
	RF	0.83
	TS Fuzzy	0.75
	Mamdani Fuzzy	0.73

Table 5.7: Classification Results with JDT Dataset

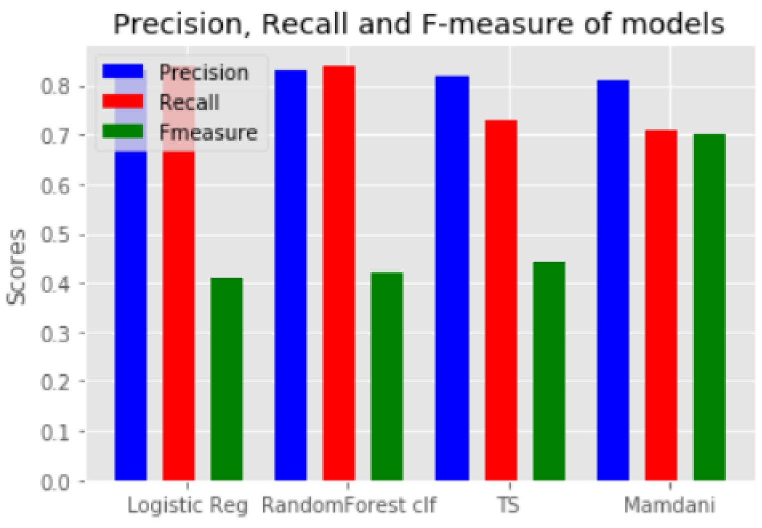


Figure 5.10: Classification with bar charts using Eclipse JDT Dataset

Evaluation Metric	Model	Score
Precision	LR	0.90
	RF	0.90
	TS Fuzzy	0.89
	Mamdani Fuzzy	0.89
Recall	LR	0.92
	RF	0.91
	TS Fuzzy	0.86
	Mamdani Fuzzy	0.75
F-measure	LR	0.90
	RF	0.90
	TS Fuzzy	0.88
	Mamdani Fuzzy	0.80

Table 5.8: Classification Results with Lucene Dataset

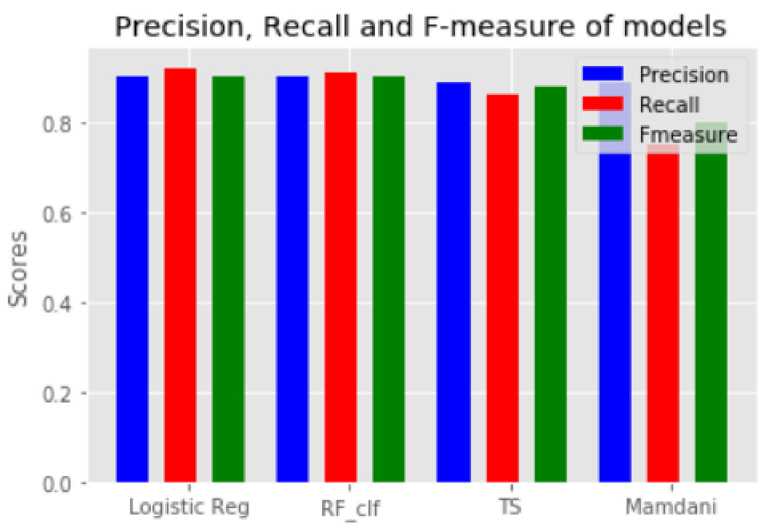


Figure 5.11: Classification with bar charts using Lucene Dataset

Evaluation Metric	Model	Score
Precision	LR	0.75
	RF	0.85
	TS Fuzzy	0.81
	Mamdani Fuzzy	0.81
Recall	LR	0.86
	RF	0.87
	TS Fuzzy	0.71
	Mamdani Fuzzy	0.65
F-measure	LR	0.80
	RF	0.82
	TS Fuzzy	0.75
	Mamdani Fuzzy	0.71

Table 5.9: Classification Results with Mylyn Dataset

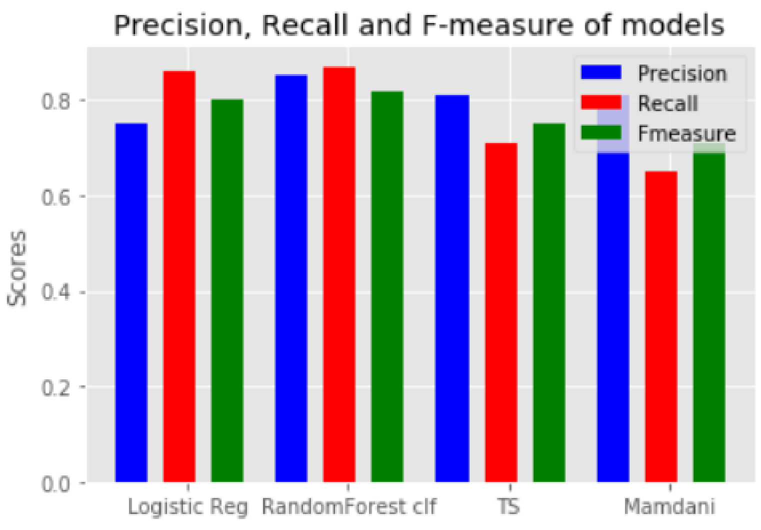


Figure 5.12: Classification with bar charts using Mylyn Dataset

Evaluation Metric	Model	Score
Precision	LR	0.66
	RF	0.72
	TS Fuzzy	0.69
	Mamdani Fuzzy	0.73
Recall	LR	0.65
	RF	0.70
	TS Fuzzy	0.70
	Mamdani Fuzzy	0.73
F-measure	LR	0.65
	RF	0.70
	TS Fuzzy	0.69
	Mamdani Fuzzy	0.73

Table 5.10: Classification Results with Equinox Dataset

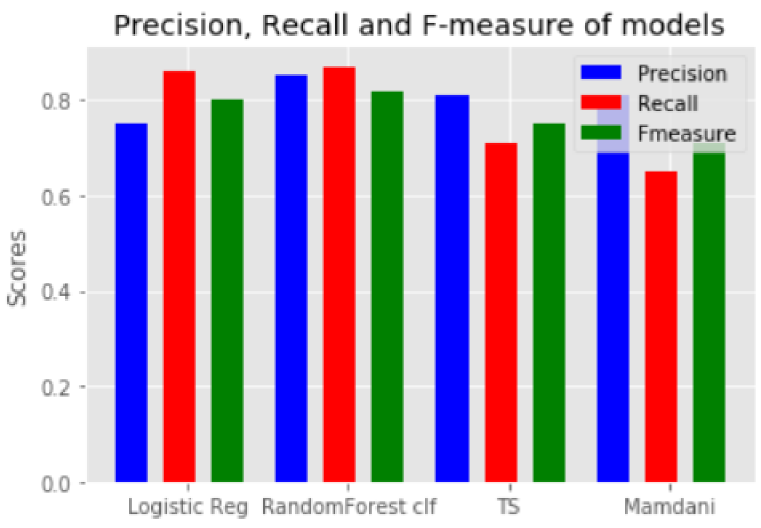


Figure 5.13: Classification with bar charts using Equinox Dataset

From the results, we can gather that the fuzzy systems are almost similar to the ML algorithms in all of the data sets in terms of Precision/Recall/F-measure. There is only a minimal difference that could change with the scale of data sets. We can understand that Fuzzy systems have the capacity to predict the presence of defects with fewer parameters. With more tuning and deciding threshold values, we can achieve better results entirely than traditional ML algorithms.

We could also note that despite Mamdani's poor performance in regression, it proved its usefulness in classification. This indicates that any fuzzy logic system with a relevant and simple rule base can help us discover the presence of defects. This factor is applicable in new software systems, too: even without the historic data, fuzzy logic can imply with minimum parameters and domain knowledge.

Chapter 6

Conclusion and Future directions

6.1 Summary

From our experiments, we conclude that fuzzy logic systems (TS model) have the capacity to match with the traditional machine learning algorithms in performance. It is evident from the performance measure metrics of the Sugeno Model from Table 5.3 to 5.10 (both classification and regression).

Despite Mamdani's poor performance in regression, we could discern how a rule-based system handles ambiguity and captures the domain experts' knowledge to predict crisp outputs and the presence of defects in classification problems. Since we discovered the potential in rule-based structures, we implemented a TS model to improvise our defect prediction system. The model performed well in both classification and regression based cross project predictions.

In this thesis, we started by presenting the importance of Software defect prediction systems and the implementation of such systems in literature and industries. After a careful literature review and theory presentation, we explored the dataset and studied briefly about our implementation.

For the implementation, we utilized the scikit-fuzzy logic [49] library to build our Mamdani and Fuzzy systems. We devised a custom membership function for our Mamdani to improve the traditional fuzzy models' accuracy values. To capture the relation between input features and the output better, we implemented a TS model that contains polynomial functions as consequents. With minimal feature selection and low time complexity, we propose the TS model to solve the defect prediction problem.

6.2 Future work

Even though the TS model showed huge potential in competing with traditional ML models, the model needs more features and efficient hyper-parameter tuning to predict better in both regression and classification problems. Also, we will conduct more cross project predictions: provide results to show that fuzzy systems work with the absence of features, doesn't require intense model training across systems, and provide better flexibility to add extra rules. We will continue our work by working with more defect prediction datasets and perform Domain-Expert-based feature selection, model optimization (TS and Mamdani), and explore other fuzzy models, too. Currently, we have a sci-kit learn type of an API/method that takes in dataset and parameters to return the predictions. We will improve our algorithm and build a UI interface to readily work with any dataset.

References

[1] Ian Sommerville (2016). Software Engineering, 10th Edition, Pearson Education Limited, England.

[2] Gema Rodríguez-Pérez, Gregorio Robles, Alexander Serebrenik, Andy Zaidman, Daniel M. Germán and Jesus M. Gonzalez-Barahona. How bugs are born: a model to identify how bugs are introduced in software components. Empirical Software Engineering, volume 25, pages 1294–1340 (2020)

[3] Stanford Encyclopedia of Philosophy. Fuzzy Logic. Retrieved 10/10/2020. https://plato.stanford.edu/entries/logic-fuzzy

[4] V. U. B. Challagulla, F. B. Bastani, I-Ling Yen and R. A. Paul, Empirical assessment of machine learning based software defect prediction techniques. 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, Sedona, Arizona, USA, 2005, pp. 263-270.

[5] N. E. Fenton and M. Neil, A critique of software defect prediction models, in IEEE Transactions on Software Engineering, vol. 25, no. 5, pp. 675-689, Sept.-Oct. 1999.

[6] Awni Hammouri, Mustafa Hammad, Mohammad M Alnabhan, Fathima Alsarayrah. Software Bug Prediction using Machine Learning Approach. (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 9, No. 2, 2018.

[7] Fenton N., M. Neil, W. Marsh, and P. Hearty, L. Radliński, and P. Krause. On the Effectiveness of Early Life Cycle Defect Prediction with Bayesian Nets. Empirical Software Engineering 2008; 13(5): 499-537

[8] H. A. Al-Jamimi. Toward comprehensible software defect prediction models using fuzzy logic. 7th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, 2016, pp. 127-130.

[9] Foyzur Rahman, Daryl Posnett, Abram Hindle, Earl Barr, and Premkumar Devanbu. 2011. BugCache for inspections: hit or miss? In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. Association for Computing Machinery, New York, NY, USA, 322–331.

[10] Yadav, D., Chaturvedi, S., and Misra, R. (2012). Early Software Defects Prediction Using Fuzzy Logic. International journal of performability engineering, 8, 399.

[11] Johannes Bader, Satish Chandra, Eric Lipert, Andrew Scott. Getafix: How Facebook tools learn to fix bugs automatically. Retrieved 10/11/2020. https://ai.facebook.com/blog/getafix-how-facebook-tools-learn-to-fix-bugs-automatically

[12] Moataz A. Ahmed and Hamdi A. Al-Jamimi. Machine learning approaches for predicting software maintainability: a fuzzy-based transparent model. IET Software, vol. 7, no. 6, pp. 317-326, December 2013.

[13] Chris Lewis and Rong Ou. Bug Prediction at Google. Retrieved 10/11/2020. http://google-engtools.blogspot.com/2011/12/bug-prediction-at-google.html.

[14] M. D'Ambros, M. Lanza and R. Robbes, An Extensive Comparison of Bug Prediction Approaches. MSR 2010, 7th IEEE Working Conference on Mining Software Repositories, pp. 31 - 41. IEEE CS Press.

[15] Shruthi Puranik, Pranav Deshpande, K Chandrasekaran. A Novel Machine Learning Approach for Bug Prediction. 6th International Conference On Advances In Computing and Communications, ICACC 2016, 6-8 September 2016, Cochin, India.

[16] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493, June 1994.

[17] Hong, E. Fault-proneness Prediction using Random Forest. International Journal of Smart Home, Vol. 6, No. 4, October, 2012.

[18] Sharma, D., Chandra, P. Identification of latent variables using, factor analysis and multiple linear regression for software fault prediction. Int J Syst Assur Eng Manag 10, 1453–1473 (2019).

[19] E.S. Hong. Early Software Quality Prediction using Support Vector Machine. Journal of the Korea Society of IT Services, vol. 10, no. 2, (2011), pp. 235-245

[20] Singh, Y., A. Kaur and R. Malhotra. Software Fault Proneness Prediction Using Support Vector Machines. Proceedings of the World Congress on Engineering 2009 Vol I WCE 2009, July 1 - 3, 2009, London, U.K.

[21] Software Defect Dataset, PROMISE REPOSITORY. http://promise.site.uottawa.ca/SERepository/datasets-page.html, (2013) December 4.

[22] Kalai Magal. R and Shomona Gracia Jacob. Improved Random Forest Algorithm for Software Defect Prediction through Data Mining Techniques. International Journal of Computer Applications (0975 – 8887) Volume 117 – No. 23, May 2015

[23] Gray D., Bowes D., Davey N., Sun Y., Christianson B. (2009) Using the Support Vector Machine as a Classification Method for Software Defect Prediction with Static Code Metrics. In: Palmer-Brown D., Draganova C., Pimenidis E., Mouratidis H. (eds) Engineering Applications of Neural Networks. EANN 2009. Communications in Computer and Information Science, vol 43. Springer, Berlin, Heidelberg.

[24] Yan Z., Chen X., Guo P. (2010) Software Defect Prediction Using Fuzzy Support Vector Regression. In: Zhang L., Lu BL., Kwok J. (eds) Advances in Neural Networks - ISNN 2010. ISNN 2010. Lecture Notes in Computer Science, vol 6064. Springer, Berlin, Heidelberg.

[25] Lyu, M.R. (ed.): Handbook of Software Reliability Engineering. IEEE Computer Society Press and McGraw-Hill Book Company (1996)

[26] Lyu, M.R., Huang, Z., Sze, K.S., Cai, X.: An empirical study on testing and fault tolerance for software reliability engineering. In: Proc. of the 14th IEEE International Symposium on Software Reliability Engineering (ISSRE 2003), November 2003, pp. 119–130 (2003).

[27] Liguo Yo. Using Negative Binomial Regression Analysis to Predict Software Faults: A Study of Apache Ant. I.J. Information Technology and Computer Science, 2012, 8, 63-70.

[28] J. Li, P. He, J. Zhu and M. R. Lyu, Software Defect Prediction via Convolutional Neural Network. 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), Prague, 2017, pp. 318-328.

[29] Halstead, Maurice H. (1977). Elements of Software Science. Amsterdam: Elsevier North-Holland, Inc. ISBN 0-444-00205-7.

[30] T. J. McCabe, A Complexity Measure. IEEE Transactions on Software Engineering, vol. SE-2, no. 4, pp. 308-320, Dec. 1976

[31] Tera Promise Repository. http://openscience.us/repo/defect/mccabehalsted/pc3.html, 2004.

[32] T. Sethi and Gagandeep. Improved approach for software defect prediction using artificial neural networks. 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, 2016, pp. 480-485.

[33] He, P., Li, B., Liu, X., Chen, J., and Ma, Y. (2015). An empirical study on software defect prediction with a simplified metric set. Information and Software Technology, 59:170-190, Feb 2014.

[34] Ran Li, Lijuan Zhou, Shudong Zhang, Hui Liu, Xiangyang Huang and Zhong Sun. Software Defect Prediction Based on Ensemble Learning Learning. In Proceedings of 2019 2nd International Conference on Data Science and Information Technology (DSIT'19). Seoul, Republic of Korea, 6 pages.

[35] Amirabbas Majd , Mojtaba Vahidi-Asl , Alireza Khalilian, Pooria Poorsarvi-Tehrani, Hassan Haghighi. SLDeep: Statement-level software defect prediction using deep-learning model on static code features. Expert Systems with Applications, Volume 147, 1 June 2020, 113156.

[36] Martín, C.L., Pasquier, J.L., Yáñez-Márquez, C., and Gutierrez-Tornes, A. (2005). Software development effort estimation using fuzzy logic: a case study. Sixth Mexican International Conference on Computer Science (ENC'05), 113-120.

[37] Yadav, H.B., Yadav, D.K (2015). Construction of Membership Function for Software Metrics. Procedia Computer Science 46: 933-940.

[38] Yadav, H.B., Yadav, D.K. A fuzzy logic based approach for phase-wise software defects prediction using software metrics. Information and Software Technology 2015, 63, 44–57.

[39] H. Madsen, P. Thyregod, B. Burtschy and G. Albeanu. A fuzzy logic approach to software testing and debugging. European Safety and Reliability Conference (ESREL 2006), Estoril-Portugal, vol. 1-3, pp. 1435-1442, September 18-22, 2006.

[40] J. S. R. Jang. ANFIS: adaptive-network-based fuzzy inference system. IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, No. 3, 1993, PP. 665- 685.

[41] O. Dehzangi, M. J. Zolghadri, S. Taheri and S. M. Fakhrahmad. Efficient Fuzzy Rule Generation: A New Approach Using Data Mining Principles and Rule Weighting. Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007), Haikou, 2007, pp. 134-139.

[42] Riyadh A.K. Mehdi. A Neuro-Fuzzy Model for Software Defects Prediction and Analysis. Intelligent Systems Conference (IntelliSys), 2020, Amsterdam, Netherlands.

[43] Martin Kleppmann. Designing Data Intensive Applications (2017). O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

[44] Majd, A. , Vahidi-Asl, M. , Khalilian, A. , Baraani-Dastjerdi, A. , and Zamani, B. (2019). Code4Bench: A multidimensional benchmark of codeforces data for different program analysis techniques. Journal of Computer Languages .

[45] L.A. Zadeh. Fuzzy Sets. Information and Control Volume 8, Issue 3, June 1965, Pages 338-353.

[46] Scikit-learn, machine learning in python. https://scikit-learn.org/stable. 2019 - 2020.

[47] E.H. Mamdani, S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. International Journal of Man-Machine Studies, Volume 7, Issue 1, January 1975, Pages 1-13

[48] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-15, no. 1, pp. 116-132, Jan.-Feb. 1985.

[49] Scikit-fuzzy. https://pythonhosted.org/scikit-fuzzy/overview.html. 2019 - 2020.

[50] Timothy J. Ross. Fuzzy Logic with Engineering applications. 3rd Edition, Wiley Publications, USA.

[51] Fuzzy Logic. Retrieved 10/25/2020. https://en.wikipedia.org/wiki/Fuzzy_logic.

[52] Defuzzification Methods. Retrieved 10/25/2020. https://www.mathworks.com/help/fuzzy/defuzzification methods.html.

[53] Tsukamoto, T. (1979) An approach to fuzzy reasoning method. In Gupta, M. M., Ragade, R. K., and Yager, R. R. (Eds) Advances in Fuzzy Set Theory and Applications (Amsterdam: North-Holland), pp. 137–149.

[54] Fuzzy Membership Function Formulation and Parameterization. Retrieved 10/25/2020.https://cse.iitkgp.ac.in/ dsamanta/courses/archive/sca/Archives/Chapter03FuzzyMembership