Chapter 2

Preliminary

"A weak background is a deadly thing, so make the strong one." – Robert Henri

This chapter presents background information related to software defect prediction techniques. We have sequentially illustrated various software metrics and various public projects in section 2.1 and section 2.1. Different classification and regression-based performance metrics in section 2.4. Learning methodologies used in this thesis in section 2.3.2 and baseline methods in section 2.5.

2.1 Software Metrics

Software fault prediction is a bridge between software bugs and software metrics. There are diverse software metrics used for SBP, McCabe [160], Halsted [87] and Chidamber & Kemerer's (CK) [39] were most widely used for SBP for an object-oriented software system. Some another metric are also employed like code smell metrics [86], line of comments/codes [10], context package cohesion metrics [293], web metrics [19], change metrics [123], mutation-based metrics [22], network metrics [149], modularization metrics [294] and cascading style metrics [20]. Twenty-one software metrics which are essential for the operation are described in table 2.1; this kind of metrics are mostly employed in classification based SDP. The basis Halsted metrics, Derived Halsted metrics, and McCabe metrics are the various software metrics used in experiments. These metrics consist of some parameter related to

software reliability decried in table 2.1. Few CK metrics more useful for CVDP and CPDP are listed in Table 2.2.

TABLE 2.1: Method level metrics description.

Types	**Metrics**	**Definition**
Basic Halsted	LOCode	Count lines of code
	LOBlank	Count number of blank lines
	LOCodeAndComment	Count of blank and code lines
	uniqueOp	Total number of unique operators
	uniqueOpnd	Total number of unique operands
totalOp	Total number of operators	
	totalOpnd	Total number of operands
	branchCount	Total number of branch count
	n	Total numbers of operator and operands
Derived halsted	t	Time estimator
	b	Effort estimation
	e	Program write effort estimation
	i	Intelligence
	d	Difficulty
	l	Program length(v/n)
	v	Volume
McCabe	loc	Total lines of code
	v(g)	Cyclomatic complexity
	ev(g)	Essential complexity
	lv(g)	Design complexity

2.2 Dataset Description

This thesis mainly considered datasets from NASA, and PROMISE repository software projects. NASA is the most widely used datasets in SFP. The list of all datasets that are employed in various classes of SDP is shown in Fig. 2.1. Table 2.3, and Table

TABLE 2.2: Object oriented level metrics description.

Metric class	**Metric name**	**Explanation**
Abstraction	DIT	Depth of inheritance tree
	NOC	Number of children
	MFA	Measure of functional abstraction
Cohesion	LCOM	Lack of cohesion methods
	LCOM3	Lack of cohesion in methods
	CAM	Cohesion among methods of class
Coupling	CBO	Coupling between object classes
	RFC	Response for class
	CA	Afferent couplings
	CE	Efferent couplings
	IC	Inheritance coupling
Complexity	LOC	Line of codes
	WMC	Weighted methods per classes
	NPM	Number of public methods
	AMC	Average methods complexity
	Max_cc	Max McCabe's cyclomatic complexity
	Avg_cc	Average McCabe's cyclomatic complexity
	MOA	Measure of aggregation
Encapsulation	DAM	Data access metric

2.4 illustrated various software project dataset which is from NASA and PROMISE repository, respectively. List of all SDP datasets is listed below.

- *NASA dataset:* It is the most widely used dataset in SFP; it is freeware dataset and can be downloaded from the link given below:
  `https://github.com/klainfo/NASADefectDataset/tree/master/OriginalData/MDP`.
- *PROMISE dataset:* It is also frequently employed datasets in the SFP domain. The datasets are freely available in PROMISE repository [216]. It can be downloaded from `http://promise.site.uottawa.ca/SERepository/`.
- *Eclipse dataset:* Most of its versions are freely available at `http://bug.inf.usi.ch/download.php`.
- *Student dataset:* This is mainly for academic study, which is developed by students. Few of them can be downloaded from `http://bug.inf.usi.ch/index.php`.

- *Open-Source dataset:* It includes some other open-source software projects, such as Xylan, Lucene, Ant, Apache, KDE, Gnome, Mozilla, Openoffice, Klac, Kpdf, Kspread, and Firefox, etc. It can be downloaded from `http://bug.inf.usi.ch/download.php`.

- *Others:* This is some private dataset, or industrial datasets, such as commercial Java application or commercial banking dataset.

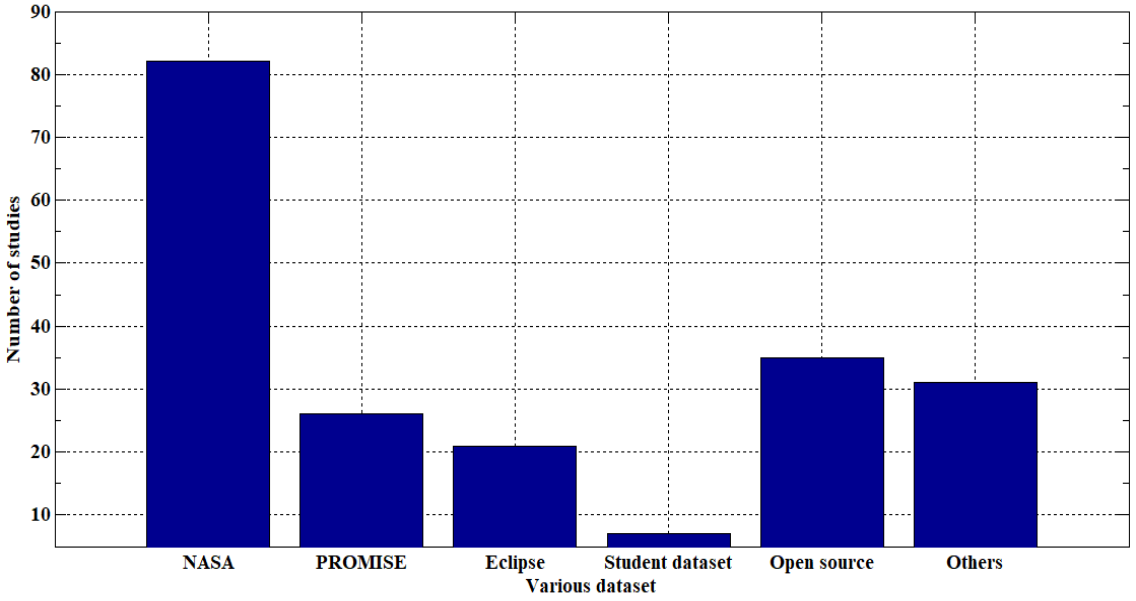

FIGURE 2.1: Percentage of studies for different dataset that are used in SFP.

TABLE 2.3: Dataset description before and after preprocessing.

Dataset	No. of metrics	No. of instance		No. of non-faulty instances		Fault instances(%)	
		Before	After	Before	After	Before	After
CM1	22	498	442	449	394	9.85	10.85
KC1	22	2109	1212	1783	897	15.45	25.98
KC2	22	522	375	415	270	20.50	28
KC3	40	194	194	158	158	18.55	18.55
MC1	39	1988	1988	1942	1942	2.30	2.30
MC2	40	125	125	81	81	35.20	35.20
PC1	22	1109	954	1032	884	6.95	7.35
JM1	22	10885	8912	8779	6905	19.35	22.5
MW1	38	253	253	226	226	10.67	10.67
PC2	37	745	745	729	729	2.15	2.15
PC3	38	1077	1077	943	943	12.44	12.44
PC4	38	1458	1344	1280	1167	12.21	13.17

TABLE 2.4: PROMISE Dataset description.

Software	Version	No. of modules	No. of buggy modules	% Buggy of modules	Max bug count
ant	ant-1.3	125	33	16.00%	3
	ant-1.4	178	47	22.47%	3
	ant-1.5	293	35	10.92%	2
	ant-1.6	351	184	26.21%	10
	ant-1.7	745	338	22.28%	10
camel	camel-1.0	339	14	3.83%	2
	camel-1.2	608	216	35.53%	28
	camel-1.4	872	146	16.63%	17
	camel-1.6	965	118	19.48%	28
log4j	log4j-1.0	135	25	18.5%	9
	log4j-1.1	109	37	33.9%	9
	lof4j-1.2	205	188	91.7%	10
lucene	lucene-2.0	195	91	46.6%	22
	lucene-2.2	247	144	58.2%	47
	lucene-2.4	340	200	58.5%	30
jedit	jedit-3.2	272	90	31.01%	45
	jedit-4.0	306	75	24.51%	23
	jedit-4.1	312	78	25.32%	23
	jedit-4.2	367	48	13.08%	10
xerces	xerces-1.2	440	115	16.14%	4
	xerces-1.3	453	68	15.23%	30
	xerces-1.4	588	429	72.95%	62
	xerces-init	163	78	47.85%	11
velocity	velocity-1.4	198	49	24.7%	7
	velocity-1.5	214	141	65.8%	10
	velocity-1.6	229	78	34.0%	12
poi	poi-1.5	237	139	58.64%	20
	poi-2.0	314	37	11.78%	2
	poi-2.5	385	248	64.44%	11
	poi-3.0	442	281	63.57%	19

2.2.1 Challenges Over Datasets

Shepperd et al. [220] claims over the quality of the NASA datasets. They found a high rate of duplicate and noisy instances. They recommended that 1) reflects the provenance of the datasets they used, 2) report any preprocessing is inadequate detail to enable relevant replication, 3) effort required to understand data before applying learning technique. [124] claimed that collected data contain noise because the current fault collection practices are based on optional keywords, and faulty instances are not correctly linked with changelogs. Since the quality of changelogs varies across the projects, fault information can also vary from different projects; Kumar et al. [129] claimed that in some projects, the fault information is not available in the source. However, the cost parameter can be incorrectly collected. Wagner et al. [256] suggested a few techniques to handle cost parameter issues. Software projects that are developed within the organization (banking) may contain

different fault patterns. The SFP technique results can be varied if the model is directly applied without analyzing the fault pattern [8].

2.3 Methodologies

In this section we will illustrates principal methods that are employed in our work. First we will explain the methods involve in preprocessing and then learning techniques.

2.3.1 Techniques Involve in Preprocessing

Petric et al. [194], and Gray et al. [80] have questioned the standard of defect prediction dataset. After analyzing their suggested solution to deal with problem-related to the standard of the dataset. A recent study about the quality of NASA dataset and suggestion to handle issues in NASA datasets were given by [220]. Duplicate instances need to be deleted [81]. Missing data points handled by mean corresponding values [257] as there will be no big mean difference after applying it.

Duplicate instance deletion: The software modules which have the same set of software metrics and same class labels referred to as duplicate instances. Such kind of problem can occur in the real world; even repeated instances can lead to the negative impact on machine learning algorithms, it can perform over-optimistic when it is correctly classified as a part of test data. Moreover, they cause to make training phase more time consuming and also reduces the performance of a learning model. So these are the reasons to remove the duplicate instances. We have removed the duplicate data instances from the dataset to avoid such problems in our experiments.

Missing value replacement: Usually, the instances contain various software metrics. There can be more reasons for missing instances, such as data collection done carelessly or data capturing carelessness. If any instance has a missing value, then it cannot satisfy the input criteria of our model. We have placed the missing value with the mean of the corresponding metrics values. For example, a metrics (m) contains 50 instances (m_1 to m_{50}), an instance m_{49} and m_{50} are missing, and both of them will replace by mean of m_1 to m_{48}.

$$m_{49} = m_{50} = \frac{1}{48} \sum_{n=1}^{48} m_n \tag{2.1}$$

Normalization: Various software metrics values have a different order of magnitude; then, there is a need to scale the data over these metrics. We employ simple min-max normalization technique [274] in chapter 3 to chapter 6. The transformation of normalized values done in between close interval of zero to one. Let a metric m, its minimum value is min(m), and the maximum value is max(m). For every value of m_i of a metric m, let $\bar{m}$ be its calculated normalized value, which is calculated as:

$$\bar{m} = \frac{m_i - min(m)}{max(m) - min(m)} \tag{2.2}$$

Sampling Techniques: Sampling techniques are the redistribution of data points in the dataset, which helps to avoid skewed distribution of positive or negative instances. As we have discussed in section 2.2.1, the datasets have a different bug label, and it has a skewed distribution, which causes class imbalance problems [100]. Many researchers have suggested a solution to this issue. When the data distribution of positive and negative instances is approximately equal, there will be no class imbalance problem. Two main sampling techniques are:
(a) Over-sampling techniques
(b) Under-sampling techniques
We have applied SMOTE [33] sampling technique in chapter 3. As after applied the SMOTE sampling method in those techniques, the performance significantly increased without any biases. SMOTE were widely applied in SBP domain [189, 266]. In few of our work we employed a random over-sampling method [3]. Multi-label random oversampling [31] that in a preprocessing step in chapter 4, chapter 5, and chapter 6. The dataset and the percentage of imbalance of a class p are as an input. Mean Imbalance Ratio (MIR) and Imbalance Ration per Label (IRL) calculated during the cloning of minority labels. Multi-label random oversampling clone the minority class according to their IRL and discard the label, which has high IRL; a complete explanation of this algorithm is given by Charte et al. [31].

2.3.2 Learning Techniques

In this section, we will discuss the various learning models that we have utilized; we mainly discuss the long short term memory, attention layer, and sequence to sequence model. The rest of the detailed content will be illustrated in the respective

chapter.

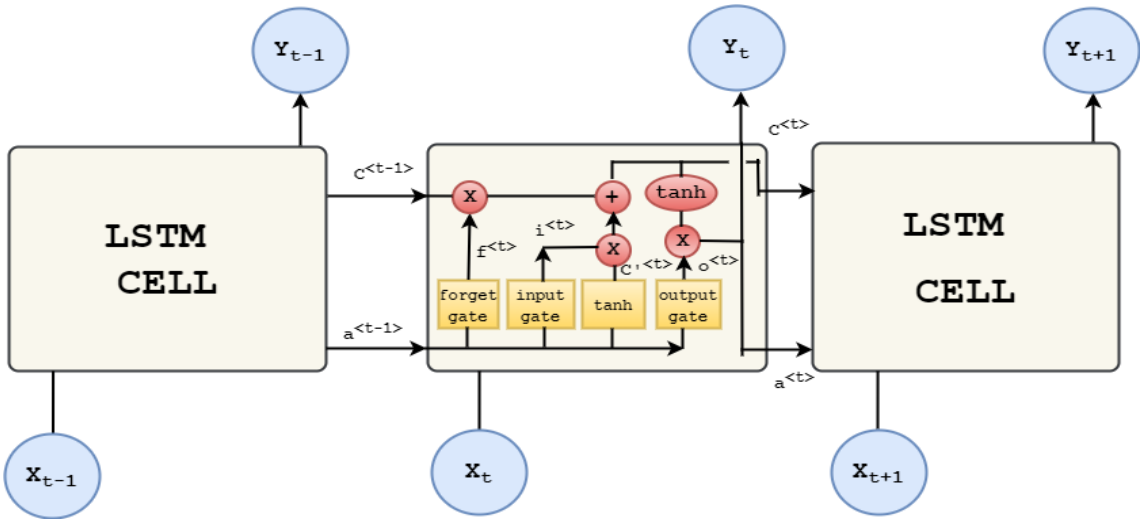

FIGURE 2.2: Underlying architecture of LSTM.

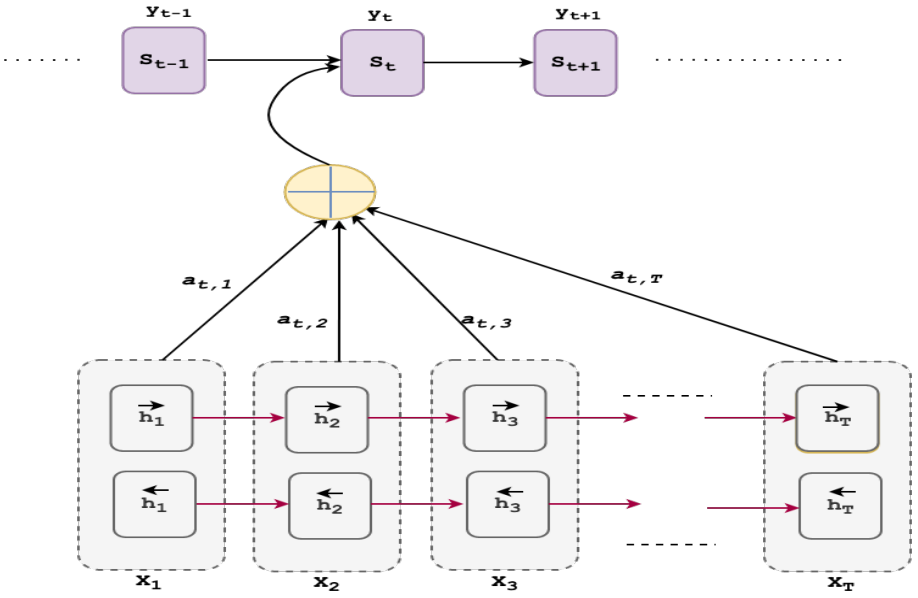

FIGURE 2.3: Attention layer architecture.

1. **Long short term memory:** In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sequential sentence, the previous words are needed, and hence there is a need to remember the previous words. It works fine when there are short sequences to train and find challenging when the sequence

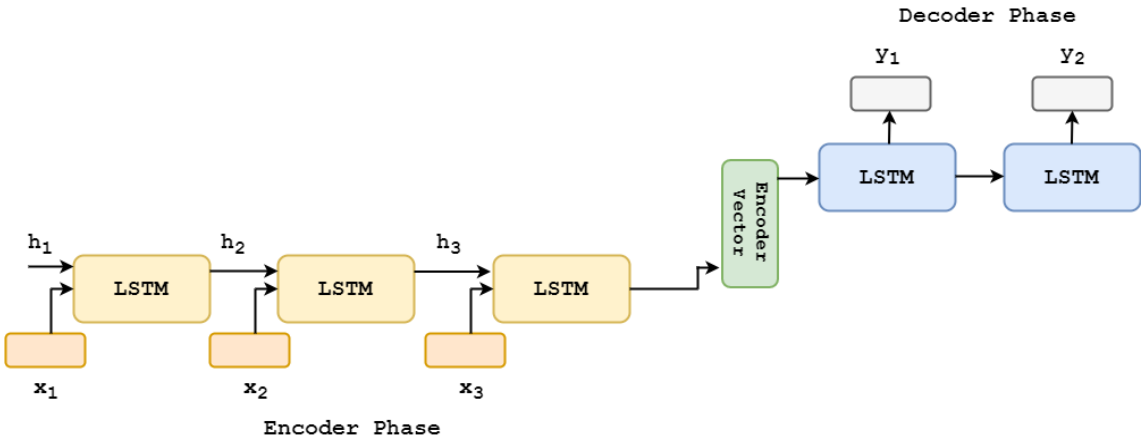

FIGURE 2.4: The architecture of Sequence to Sequence model using LSTM cell.

is large. It also suffers from a vanish gradient problem [93] and exploding gradient problem [187]. Long short term memory (LSTM) [94] overcomes these challenges. The heart of an LSTM network is it's a cell or say cell state which provides a bit of memory to the LSTM so that it can remember the past or present sequences. LSTM cell consists of three gates, "forget gate," "input gate," and "output gate" as given in Fig. 2.2. The gates of LSTM have a unique ability to add or remove information to the cell. x_t, and y_t are input and output sequence at t time step. The w_c, w_i, w_f, w_o are the weight matrix for candidate cell, input gate, forget gate, and output gate respectively, whereas b_c, b_i, b_f, and b_o are bias value for candidate cell, input gate, forget gate, and output gate, respectively. Tanh is an activation layer, its value lies between [-1, 1], σ is activation function layer lies between [0, 1].
Step by step process of the LSTM cell is explained below.

(I) The first step of LSTM cell is to decide which information needs to be flushed away from the cell state. This determines by the sigmoid function of forget gate (f^t). As it takes new input sequences from x_t and previous cell output sequence i.e., from $a^{<t-1>}$.

$$f^t = \sigma(w_f[a^{t-1}, x^t] + b_f) \tag{2.3}$$

(II) The next step decides which information needs to be stored in the cell state. It has two segments; the first sigmoid layer decides which values will

need to be updated, second tanh layer creates a vector for new candidates values, C'$^{<t>}$ can be added to the state. After that, it combines these two and updates (i$^{<t>}$) the state.

$$i^t = \sigma(w_i[a^{t-1}, x^t] + b_i) \tag{2.4}$$

$$C'^{<t>} = tanh(w_c[a^{<t-1>}, x^t] + b_c) \tag{2.5}$$

(III) After that, it update the old cell state, C$^{<t-1>}$, into the new cell state C$^{<t>}$.

$$C^{<t>} = i^t * C'^{<t>} + f^t * C^{<t-1>} \tag{2.6}$$

(IV) In the final step, it decides what output needs to be provided. It first runs the sigmoid layer that decides which output will be provided, and then it pulls a cell state to tanh. It triggers that output sequence, which had agreed to be output.

$$o^t = \sigma(w_o[a^{t-1}, x^t] + b_o) \tag{2.7}$$

$$a^{<t>} = o^t * tanh * C^{<t>} \tag{2.8}$$

2. **Attention Layer:** The attention layer [14] was initially introduced for neural language translation. Later it has been widely applied in many real-world applications such as multilingual machine translation [64], image captioning [278], image classification [258], cross-project defect prediction [34] etc,. We have utilized Bahdanau [14] attention model in our thesis work. Fig. 2.3 shows the basic architecture of Bahdanau's attention architecture. The attention layer mainly provides extra weights to the input sequences. The hidden state (s_i) is computed by using Eqn. 2.9. The attention layer has mainly three different stages; first, it generates attention weights to the input sequence using equation 2.10. Here amount of attention Y_t should pay to $\alpha_{t'}$. Second, it also generates the context vector using equation 2.11.

$$s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{2.9}$$

$$a_{t,s} = \frac{exp(score(Y_t, \hat{Y}_{t'}))}{\sum_{t'=1}^{T_x}(exp(score(Y_t, \hat{Y}_{t'})))} \tag{2.10}$$

$$c_t = \sum_{t'} a_{t,s} \hat{Y}_{t'} \tag{2.11}$$

$$a_t = f(c_t, h_t) = tanh(W_c[c_t; h_t]) \tag{2.12}$$

The context vector computes the probability distribution of the source sequence in every single sequence by holding all cell's output to input. The Eqn 2.12 refers to attention vector generation. The complete details Bahdanau attention layer is given in the article [14].

3. **Sequence to sequence model (Seq2Seq):** It is also known as encoder-decoder model [245]. It map fixed-length input (T_x) to a fixed-length output (T_y), where $T_x \neq T_y$. It consists of three sections, first is encoder, intermediate (encoder vector) and third is decoder. The underlying architecture of sequence to sequence model using the LSTM unit is shown in Fig. 2.4.

   Encoder: It is a stack of many recurrent units (GRU, LSTM), where every unit accepts a single element of an input sequence, then collects information from it, and finally propagates information in the forward direction. Each input vector is represented as x_i where i is the order of that input vector. The hidden state h_t is computed using equation 2.13

$$h_t = f(W^h * h_{t-1} + W^{hx} * x_t) \tag{2.13}$$

   Encoder vector: It is the final hidden layer of the encoder and acts as the decoder's initial hidden state. The main objective of this phase is to encapsulate the information of all input elements which help the decoder make accurate predictions.

   Decoder: It is a stack of many recurrent units where each unit predicts an output sequence y_t at a time step t. Every recurrent unit accepts a hidden state from the previous unit and produces an output as well as its hidden state. Hidden state h_i and output y_t are calculated using equations 2.14 and 2.15 respectively as:

$$h_t = f(W^h * h_{t-1}) \tag{2.14}$$

$$y_t = SOFTMAX(W^s * h_t) \tag{2.15}$$

2.4 Performance metrics

Our work employed both classification (chapter 3) and regression-based (chapter 4 to chapter 6) performance measures. In this section, we will briefly illustrate performance measures which are used in this thesis. First, we will discuss the classification, then regression-based performance measures.

Classification based performance measure: Area under the curve (AUC) of ROC (receiver operating characteristic curve), F-measure, Mathews correlation coefficient (MCC), and precision-recall area (PRC) are employed in chapter 3. The thesis considered a convention that the faulty module will be positive samples, and non-faulty modules will be negative samples. Evaluation of classifier more inclusively in class imbalance situation, the AUC and F-measure usually applied [196] in such condition. F-measure examine both precision and true positive rate (TPR) or recall, as it is the harmonic mean of precision and recall, which means it gives information about the number of defective instances that the model predicts in the total number of defective instances and the number of defective instances the model predicts in all instances. Whereas AUC lies between [0,1], the area under the ROC curve measures the comparative performance between true positive rate (TPR) and false positive rate (FPR). More is the area under the ROC curve, better will be the performance of the classifier.

$$F - measure = \frac{2 * TPR * Pricision}{TPR + Precision} \tag{2.16}$$

$$AUC = \frac{\sum rank(ins_{+class}) - \frac{X(X+1)}{2}}{X * Y} \tag{2.17}$$

$\sum rank(ins_{+class})$ is represented as the sum of the rank of every positive sample, X and Y are all positive and negative samples, respectively. Mathews co-relation Coefficient (MCC) [158] is also a classifier evaluation methodology; it considers all true positive and false positive and all true negative and false negative into an account. The value of MCC lies between [-1, +1], the higher the value of MCC than highly efficiently the model will predict toward imbalanced datasets. MCC is widely used in the SBP model and other predictive techniques; the formula of MCC

is showing below.

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.18}$$

TN, TP, FN, FP are true positive, true negative, false negative, and false positive, respectively. Precision-recall curve (PRC) [49], as from the name is made of precision and recall. Precision is the fraction of pertinent instances amongst all retrieve instances, whereas recall is a fraction of relevant instances that are extracted from a total number of related instances. When comparing precision and recall to each other, if precision increases, then recalls decreasing and vice-versa. To achieve the balance among both, the precision-recall curve comes to play the role. PRC can observe the trade-off between precision and recall.

Sometimes the precision-recall curve is more useful than ROC because the ROC only gives an idea of how the classifier is performing in general because it considers the positive and negative class equally. If someone is interested in how the classifier is engaged in a particular class then PRC can be more informative. The MCC is more functional when classes are of different sizes, which means it is effective to address the class imbalance problem. It also measures the biased nature of binary classifiers. The PRC curve also targets to class imbalance, as PRC is not considered "True negative," it measures the balance between the two classes.

Regression based performance measure: To evaluate the performance of the regression based approaches, the brief illustration of all used performance matrices are given below.

1. **Mean Absolute Error (MAE)**: It measures the mean magnitude of the error in a prediction set without considering its direction of prediction. It is the mean over the testing sample of the absolute difference between the actual observation. Here every individual difference has equal weights. Thesis also considered MAE over validation set (Val_{mae}).

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \mid y_i - \hat{y_i} \mid \tag{2.19}$$

2. **Mean Squared Error (MSE):** It is the overall sum of the data points and the square of the difference between the predicted and actual target variables,

divided by the number of data points. Thesis also considered MSE over validation set (Val_{mae}).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.20}$$

3. **Accuracy:** It is the fraction of corrected predictions over total predictions. If more is the accuracy more accurately the model is predicting. Its value lies between 0 to 1, where 1 indicates all the correct predictions, 0 means no correct prediction, and 0.5 means random predictions occur. Thesis considered both, overall accuracy and accuracy over validation set as denoted by val_{acc}.

$$Accuracy = \frac{NumberofCorrectedPredictions}{TotalNumberofPredictions}. \tag{2.21}$$

2.5 Baseline Methods

This section briefly discusses state-of-the-art techniques for different domains, classification-based, cross-version defect count prediction, and cross-project defect number estimation. We compared the performance of our proposed techniques with these benchmark techniques.

Classification based SFP:

1. Tong et al. [249] employed stacked denoising autoencoders (SDA) together with ensemble learning to construct an SFP model; they used SDA for feature extraction of software metrics, and two layers of ensemble learning. They found deep features are more prominent in defect prediction compared with standard software features and found efficient results over existing methods.

2. There are few conventional classification model such as Naive Bayes (NB) [168], JRip or Ripper[165], Support Vector Machine (SVM) SVM is a discriminative classifier [237, 89], Multilayer perceptron (MLP) [Ruck et al.], and Instance-based learning (IBK) [5].

3. Three well know types of classical ensemble learning-based SBP models are: Bootstrap aggregating (Bagging) [24], AdaBoost [67], Breiman [24] and random forest [25]. EL techniques also successfully applied in SBP, through various models. [225] proposed an approach for SBP using cost-sensitive estimation

and voting algorithm and [295] also proposed using cost-sensitive estimation with boosting neural network. Ensemble learning was applied recently on imbalance data by [122].

Cross-version defect count prediction:

(I) As such few classical techniques are Linear Regression (LR), MLP [153], IBK [5], Additive Regression (AR) [272], M5rules [201, 56], and M5P [201].

(II) There are few regression based ensemble learning methods: Bagging [241], Gaussian Process (GP) [230], Decision Stump [96], Random Forrest [28], and Regression by Decentralization (RD) [65]. Most of these methods have been widely used in software bug count prediction and other estimation/prediction applications.

Cross-project defect number prediction:
AR [272], Bagging [241, 190], IBK [5], M5rules [201, 56], MLP [153, 121], Regression by Decentralization [65], RF [28, 29], and Zero-R [277]. There are saveral baseline approaches that are overlapping between CVDNP and CPDCP.
Hybrid regression analysis: Entire software prediction: As such there is no such state of the art methods so we mainly compared with the classical deep learning, and machine learning architectures. Detailed discussion is present in chapter 6.

1. Deep learning architectures are LSTM, Gated Recurrent Unit (GRU) [41], Convolution Neural Network (CNN) [109], and Recurrent Neural Network (RNN) [291].

2. Traditional machine learning techniques are RF [29, 28], AdaBoost [9], J48 [206], and Support Vector Machine (SVM) [229].