A New Learning to Rank Approach for Software Defect Prediction

Sara Al-omari
Department of Computer Science
Applied Science Private University
Amman, Jordan

Yousef Elsheikh
Department of Computer Science
Applied Science Private University
Amman, Jordan

Mohammed Azzeh
Department of Data Science
Princess Sumaya University for Technology
Amman, Jordan

***Abstract*—Software defect prediction is one of the most active research fields in software development. The outcome of defect prediction models provides a list of the most likely defect-prone modules that need a huge effort from quality assurance teams. It can also help project managers to effectively allocate limited resources to validating software products and invest more effort in defect-prone modules. As the size of software projects grows, error prediction models can play an important role in assisting developers and shortening the time it takes to create more reliable software products by ranking software modules based on their defects. Therefore, there is need a learning-to-rank approach that can prioritize and rank defective modules to reduce testing effort, cost, and time. In this paper, a new learning to rank approach was developed to help the QA team rank the most defect-prone modules using different regression models. The proposed approach was evaluated on a set of standardized datasets using well-known evaluation measures such as Fault-Percentile Average (FPA), Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and the Cumulative Lift Chart (CLC). Also, our proposed approach was compared with some other regression models that are used for software defect prediction, such as Random Forest (RF), Logistic Regression (LR), Support Vector Regression (SVR), Zero Inflated Regression (ZIR), Zero Inflated Poisson (ZIP), and Negative Polynomial Regression (NPR). Based on the results, the measurement criteria were different than each other as there was a gap in the accuracy obtained for defects prediction due to the nature of the random data, and thus was higher for RF and SVR, as well as FPA achieved better results than MAE and RMSE in this research paper.**

Keywords*—*Software engineering; software testing; software defect prediction; learning to rank approach

I. Introduction

Predicting the exact and precise defect number is the best and most accurate option for software engineers, but because of the difficulty of achieving this task in real scenarios, it is not enough to rely on classifying modules into defects or not, so there is a need for another solution that can improve the defect prediction performance and increase the quality assurance of confidence in defect prediction [1]. This solution can be achieved by using a learning to rank approach that supports defect prediction models to rank and prioritize modules based on certain factors [1].

The importance of SDP models for predicting software defects has been discussed by using the LTR approach to rank a program according to the number of defects. The new model is supposed to improve the performance of the existing defect prediction models. Predicting the number of defects in software modules using machine learning regression models. However, this paper proposes a new learning to rank approach that supports defect prediction models for ranking and prioritization.

Most of the research in the past decade has focused on proposing new indicators for constructing predictive models [1]. The most studied indicators are the source code and process metrics [2]. Source code metrics measure the complexity of the source code [2]. Process metrics are derived from software documents, such as version control systems and issue trackers, which regulate the entire development process. Process metrics quantify many aspects of the software development process, such as source code changes, ownership of source code files, and developer interaction. The process metrics used to predict errors have been validated in many studies [2].

Defect prediction research is generally based on machine learning [3]. Predictive models built using machine learning approaches can predict the probability of errors in the source code (classification) or the number of errors in the source code (regression). Some studies have proposed the latest machine learning techniques, such as: improving active learning and prediction. The researchers also focused on determining the accuracy of predictions. Failure prediction models attempt to identify failures at the system, component, package, or file/class level. According to recent research, errors in modules or methods can also be identified and changed to different levels. Better accuracy can help developers by limiting the scope of source code reviews ensure quality. Suggesting a pre-processing method for predictive model is also an important research put forward in error prediction research. Before building the model, the following methods can be used for prediction: function selection, normalization, and noise protection [3]. Through the proposed pre-processing method, the predictive characteristics of actions in related research can be improved [3]. The researchers also proposed methods to predict defects in software projects [3]. The majority of the above representative studies were performed and verified within an internal prognostic framework, and the predictive model was developed and tested within the same project [4]. However, this is difficult for new projects that lack development history information. Create a predictive model. Typical methods for predicting crossover errors are metric compensation [4], nearest neighbor (NN) filters, naive transfer Bayes (TNB), and TCA+ (state-of-the-art transfer learning approach). Adjust the predictive model by selecting similar instances, transforming data values, or developing new models [4].

Another important topic for defect prediction between items is to study the possibility of cross-prediction. Several studies have confirmed that cross-prediction is difficult to achieve; only a few cross-prediction combinations are effective [5]. Determining cross-prediction capabilities will play a major role in predicting errors between projects. There are many studies on the possibility of cross-prediction based on decision trees [5]. However, their decision tree has only been tested on certain software datasets and has not been studied.

The purpose of the SDP for the classification task is to predict which modules are likely to contain the most defects in order to allocate efforts to improve software quality, which means relative prediction and extraction of the exact number of defects, but this requires many conditions and accurate data to give the exact number of defects, which becomes difficult when the data is too large. However, the Learning to Rank (LTR) method provides a linear model by directly improving classification performance. It has been verified that it is useful to make forward adjustments to the classification performance metrics of the SDP model for constructing classification problems [6].

In this paper, a new learning to rank approach was developed to help the QA team with the ranking of the most defect-prone modules. The proposed approach was evaluated on a set of benchmark datasets using known measures such as fault percentile average (FPA), cumulative lift chart (CLC), mean absolute error (MAE), and root mean square error (RMSE). Our proposed approach will be compared with the current learning to rank approaches used in defect prediction.

The paper is organized as follows: Section II presents work related to software defect prediction as well as learning to ranking methods. Section III presents the proposed model including the data sets as well as the evaluation metrics used. Section IV presents the implementations made in the paper and finally Section V presents the findings and discussions about them before ending with the paper's conclusion.

II. Related Work

A. Software Defect Prediction

There are many studies that address the issue of predicting software defects. Among them, for example, X. Huo and M. Li, in [4] who proposed a new perspective for software defect prediction. This clearly articulates the "pair-wise" relationship between the bad module and the clean module to better prioritize the modules that are prone to failure, thus using benchmark dataset to ensure software reliability. X. Jing et al. in [8] attempt to systematically summarize all the typical work on predicting software failures in recent years. Based on the results of this work, this paper will help software researchers and professionals to better understand previous failure prediction studies based on datasets, software indicators, scores, and technical modeling perspectives in a simple and effective way. A. Okutan and O. Yıldız in [9] used Bayesian networks to study the relationship between software performance and error propensity. They used 9 records in the Promise data repository and showed that RFC, LOC, and LOCQ are the most error prone. On the other hand, the effect of NOC and DIT on defects is limited and unreliable. Y. Ma et al. [10] looked at a cross-company defect prediction scenario in which the source and target data came from different companies. They presented a novel technique called Transfer Naive Bayes (TNB), which uses the information of all the proper features in training data to select training data that is similar to the test data. J. Zheng in [11] studied three cost-sensitive impulse approaches for driving neural networks to predict software failures using four datasets related to a single action from the NASA project. Experimental results show that threshold shift is the best choice for cost-effective prediction of software failures using neural network models from the three approaches studied, especially for project datasets developed in object-oriented languages. X. Jing et al. in [12] used vocabulary learning methods to predict software errors. They used the characteristics of open-source software measurement to study various vocabularies (including error-free modules and damaged modules and sub-words of general vocabulary) and sparse representation co-efficients. The dataset from the NASA project is used as a benchmark for evaluating the performance of all comparison methods. Experimental results show that CDDL is superior to several typical current error prediction methods. G. Czibula et al. in [13] proposed a classification model based on the mining of relational association rules. It is a discovery of relational association rules that can be used to predict whether a software module is flawed or not. On the open-source NASA datasets, an experimental evaluation of the proposed model. The results reveal that the classifier outperforms existing machine learning-based defect prediction approaches for the majority of the assessment measures studied. I. Laradji et al. in [14] introduced a two-variant (with and without feature selection) ensemble learning technique that is robust to both data imbalance and feature redundancy. Poor characteristics do not affect ensemble learners like random forests and the proposed technique, average probability ensemble (APE), as much as they do weighted support vector machines (W-SVMs). Furthermore, for the NASA datasets PC2, PC4, and MC1, the APE model paired with greedy forward selection (improved APE) attained AUC values of roughly 1.0. S. Liu et al. [15] employed the FECAR feature selection framework with Feature Clustering and Feature Ranking to forecast software defects. Using the FF-Correlation metric, this framework divides original features into k clusters. Then, using the FC-Relevance measure, it selects relevant features from each cluster. The data is based on real-world projects such as Eclipse and NASA. P. Krause and N. Fenton in [16] Focuses on a model developed for the Philips Software Center (PSC) using the expertise of the Philips Research Laboratory, which is specifically designed to predict the number of errors in various testing and operational phases. Seven of the 28 projects can obtain comprehensive data (completed questionnaires, more project data, and more error data). The study was not as successful as expected, and the authors confirmed that more investigations will be conducted in the future. The research is still in progress.

Li, M. Shepperd, and Y. Guo in [17] investigated the use and performance of unsupervised learning techniques in predicting software defect by conducting a systematic literature review that identified 49 studies with 2456 individual experimental results that met our inclusion criteria and were published between January 2000 and March 2018. Everything is in order. In this study, unsupervised classifiers did not appear to perform worse than supervised classifiers.

T. M. Khoshgoftaar and colleagues in [18] proposed a methodology that incorporates a feature selection approach for picking relevant qualities and a data sample approach for resolving class imbalance. They used nine software measurement datasets from the PROMISE software project repository. Experimental results show that feature selection based on sample data performs significantly better than feature selection based on raw data, and the fault prediction model can achieve the same effect whether the training data is sample data or raw data.

L. Son et al. in [19] proposed a methodological mapping where they dealt with nine studies questions similar to distinctive stages of improvement of a DeP model. They explored every issue related to the method from collecting records; Preprocess records, strategies used to build a DeP fashions for the metrics used to evaluate the overall performance of the model and statistical evaluation plans used to mathematically validate the results of the DeP model. Out of the full 156 research, they decided on ninety-eight research for addressing 9 studies questions fashioned for this systematic mapping.

M. Sohan et al. in [20] used a lot of project data to prepare a balance and unbalanced dataset to build a prediction of software defects. Experimental results show that no significant changes are observed between balanced and unbalanced learning models. For a balanced learning model with an unbalanced test dataset, only the AUC value (area under the curve) increases exponentially. X. Cai et al. in [21] proposed a hybrid multi-purpose dynamic local search Cuckoo Search (HMOCS) to simultaneously identify health solutions. The problem of class mismatch in the dataset and the selection of SVM (support vector machine) parameters is critical to the prediction software defect. Eight datasets were selected from the Promise database to verify the proposed model for predicting software failures. Compared with the results of 8 prediction models, this method effectively solves the problem of predicting software failure. W. Li et al. in [22] proposed a two-step classification method and a two-step classification method based on three-way decision-making to predict cost-sensitive software failures by using NASA data. On the same direction Abu-Alhija et al. [23] studied the impact of kernels and SVM on the performance of defect predictions. they found that RBF is more Superior than other kernels.

B. Learning to Rank Approaches in Software Defect Prediction

X. Yang et al. in [1] used the LTR methodology for a wide range of real-world datasets and provided a full evaluation and comparison SDP for the ranking job, which included 10 construction approaches compared to other approaches on eleven real-world datasets. The relationship between CLC and FPA was also explored, as well as the need for metric selection over two sets of data for SDP for the ranking assignment. For the ranking job, the LTR technique to building SDP models yielded good accuracy and clarity of understanding. Also Xiaoxing Yang et al. in [2] used the learning-to-rank approaches to anticipate software defects. They presented the experimental results, which include a comparison of their approach to three other approaches from the literature, as well as five publicly available datasets. They employed the evolutionary optimization method to directly optimize the model performance measure, fault-percentile-average, which is not the same as the loss functions. For most datasets, the proposed learning-to-rank approach outperformed linear regression and logistic regression in terms of fault percentile-average models. Z. Cao et al. in [3] employed a learning to rank based approach to address the lack of legacy specifications that quantify the possibility of a candidate rule becoming a specification using 38 interesting measures. The benchmark dataset contains 28 classes from the Java 6 SDK that have been manually identified as having specification rules. These guidelines were derived from the completion of 14 projects. Experimental results using classes from the Java 6 SDK show that our learning to rank-based technique can enhance the best ranking performance using a single measure by up to 66 percent. X. Yu et al. [5] investigated the effect of 23 learning to rank approaches for EADP using 41 releases of 11 open source software projects taken from the PROMISE data repository to examine the impact of 23 learning to rank techniques for EADP. When the 23 approaches are trained on the original feature subset, the experimental findings demonstrate that BRR performs best in terms of FPA, while BRR and LTR perform best in terms of Norm (Popt) subset.

M. Buchari Yu et al. in [6] used two public benchmark datasets to create and assess the implementation of Chaotic Gaussian Particle Swarm Optimization on the Learning-to-Rank software defect prediction methodology for train model parameter. They conclude that using Chaotic Gaussian Particle Swarm Optimization in a Learning-to-Rank strategy can increase defect module ranking accuracy in datasets with high-dimensional characteristics. Y. Ma et al. in [7] used a top-k learning to rank (LTR) approach in the scenario of CPDP. The PROMISE dataset shows that SMOTE-PENN outperforms the other six competitive resampling approaches and Rank Net performs the best for the proposed.

III. The Proposed Model

A. Dataset

In this paper, a benchmark dataset was used from several types of versions, and the dataset was collected from the GitHub libraries and from previous research. The dataset applied to the developed regression models was 28 in total with different features, as shown in Table I below. The methodology on which the dataset was applied is to read the required data, then it was ensured that the data did not contain null values, and then the data was divided into x (features) and y (total defect). Finally, feature scaling technique was applied to make the output the same standard as it mentioned in the implementation section.

B. Developing Regression Model

In this paper, a model was proposed to predict the defects of the software modules and then rank the most defect-prone modules using six regression models such as (Random Forest, Logistic Regression, Support Vector Regression, Negative Binomial Regression, Zero Inflated Regression, and Zero Inflated Poisson). However, after we prepared the dataset, we applied the data to our modules. They are divided into two categories: variations of the Poisson regression model and regression trees:

TABLE I. BENCHMARK DATASETS

Datasets	Feature Number	Total Defect
ant-1.7	20	746
camel-1.0	20	339
camel-1.6	20	965
data_arc	20	225
data_ivy-2.0	20	352
data_prop-6	20	644
data_redaktor	20	175
JDT_R2_0	48	2397
JDT_R2_1	48	2743
JDT_R3_0	48	3420
JDT_R3_1	48	3883
JDT_R3_2	48	2234
jedit-3.2	20	272
jedit-4.2	20	367
log4j-1.1	20	109
lucene-2.0	20	195
PDE_R2_0	48	576
PDE_R2_1	48	761
PDE_R3_0	48	881
PDE_R3_1	48	1108
PDE_R3_2	48	1351
poi-2.0	20	314
synapse-1.0	20	157
synapse-1.2	20	256
velocity-1.6	20	229
xalan-2.4	20	723
xerces-1.2	20	440
xerces-1.3	20	454

1) Variations of Poisson Regression Model: NBR (Negative Binomial Regression) has been commonly used for SDP. ZIP, ZIR, and NPR are all variations of Poisson regression. When the response variable of the dataset contains a large number of zeros, the Poisson regression model will reduce the probability of zeros. Zero-inflated models can explicitly model the excessive occurrence of zero faults. Zero-inflated models assume that zero-defect modules come from two distinct sources.
2) Regression Trees: SVR, LR, and RF are different types of regression trees. RF is an ensemble classifier consisting of many trees, and outputs the average of individual trees .LR Random Forests is a set of decision trees that have been combined to form an ensemble. It is an approach for Supervised Learning. Several decision trees are used to process the input data. It is powered by constructing a variable number of decision trees at training time and displaying the class that is the mode of the classes or mean prediction (for regression) of the individual trees. SVR attempts to predict actual values. To separate the data, this technique employs hyperplanes. If this separation is not achievable, the kernel trick is used, in which the dimension is increased, and the data points become separable by a hyperplane. Logistic regression is a data analysis technique that is used to define and explain the connection between one dependent binary variable and one or more nominal, ordinal, interval, or ratio-level independent variables.

C. Evaluation Measures

In this paper, different measures were used to evaluate the accuracy of our modules, the goal of accuracy evaluation is to make it easier to determine which modules we evaluate is good. The evaluation method is to obtain the percentage. The percentage of defects in the preceding modules of the ranking is commonly applied. To evaluate SDP models for the ranking task. The following are the evaluation measures we used:

- Fault-Percentile-Average: FPA is one of the evaluation measures which could reflect the effectiveness of different prediction models across all cuts off values as shown in equation 1. FPA is the average of the proportions of actual defects in the top m (m=1,2,..,k) modules to the whole defects, which is a more comprehensive performance measure than the percentage of defects in the top 20% modules. A higher FPA means a better ranking, where the modules with most defects come first [1].

$$\frac{1}{k}\sum_{m=1}^{k}\frac{1}{n}\sum_{i=k-m+1}^{k}\frac{1}{n}n_i \tag{1}$$

  where:
  - k is the number of software modules.
  - n is the total number of defects in all modules.
  - m is the modules to the whole defects.

- Root Mean Square Error:
  RMSE stands for Root Mean Squared Error. The standard deviation of the errors that occur when making a prediction on a dataset is known as the RMSE. This is the same as MSE (Mean Squared Error), but the root of the number is considered when calculating the model's accuracy. The errors are squared before being averaged in RMSE as shown in equation 2. This means that RMSE gives larger mistakes a higher weight. This suggests that RMSE is far more beneficial when substantial errors exist and have a significant impact on the model's performance. This characteristic is important in many mathematical calculations since it avoids taking the absolute value of the error. In this metric as well, the lower the value, the better the model's performance.

$$RMSE = \sqrt{\left[\sum\left(P_i - O_i\right)/n\right]} \tag{2}$$

  where P_i is the predicted value for the ith observation in the dataset. O_i is the observed value for the ith observation in the dataset. n is the sample size.

- Mean Absolute Error:
  The Mean Absolute Error (MAE) is a statistic that assesses the average magnitude of errors in a set of predictions without taking their direction into account as shown in equation 3. The Mean Absolute Error is the average of the absolute differences between prediction and actual observation over the test sample, assuming that all individual deviations are equally weighted. It is less susceptible to outliers than MSE because it does not penalize large errors. When performance is measured using continuous variable data, it is commonly employed. It produces a linear value that equalizes the weighted individual disparities. The model's performance improves as the value decreases.

$$MAE = \frac{1}{n} \times \sum_{i=1}^{n} |o_i - P_i| \tag{3}$$

- Cumulative Lift Chart:
  A lift chart graphically Represents the improvement provided by the mining model to random estimation and measures the change in the form of elevation estimation as shown in equation 4. Through comparing the elevation estimates of different models, you can determine which model is better.

$$CLC = FBA - \left(\frac{1}{2k}\right) \tag{4}$$

  where *k* is the number of software modules.

D. Research Methodology

In this paper, a new learning to rank (LTR) approach was developed to help the QA team rank the most defect-prone modules in the software and thus reduce testing efforts using various regression models. The datasets used were taken from the standard dataset, and the datasets are divided into training and test data. In the Software Defect Prediction Program (SDP), training data and test data were selected in two separate ways. First, in the same dataset, the training and test data were randomly selected (or may be sequential). In the second stage, the training will be taken from the dataset as the previous version, and the test data from another dataset will be taken as the next version. The first approach was adopted and used. We then evaluated the data using known evaluation measures such as Fault-Percentile-Average (FPA), Mean Absolute Error (MAE), Root-Mean-Square Error (RMSE) and Cumulative Lift Chart (CLC). Our proposed LTR approach was compared with the current LTR approaches used in software defects prediction. Fig. 1 illustrates the research methodology used in this paper.

E. Implementation

To prove the success of our proposed LTR approach, it is necessary to apply our work and show and compare the results. Various regression models were used in a separate way from previous studies, by applying the LTR approaches and the programming language that we will discuss. Python 3.6 and Spyder 3.2.6 were used to evaluate the accuracy of the ML regression models (SVM, RF, LR, ZIP, ZIR and NPR). In addition to the usage of Google Colab to run the existing LTR approaches to do comparison with the proposed LTR approach in this paper. Each model was developed separately from the others, but in this section, we collected the models to present the methodology in an obvious way. Each model was used from twenty-eight datasets. The databases were configured prior to use so that they were all applied in a uniform manner. We will go through the methodology in a clear manner by explaining the steps of the code.

IV. Result

In this paper, four evaluation measures were used to calculate the accuracy of our regression models, and we will present the findings in tables depending on the evaluation measures.

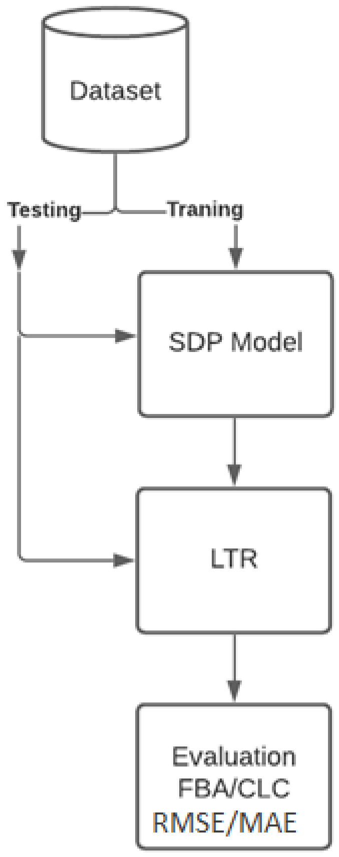

Fig. 1. Research Methodology.

Twenty-eight datasets were used over six regression models in this paper. The goal of this paper is to present the accuracy of our software modules by applying 5-fold cross-validation technique. The reason to use 5-fold cross-validation is to get the best result based on 28 datasets with different features.

A. Fault Percentile Average

By calculating the average accuracy of Fault-Percentile-Average using 5-fold cross validation, we show the values in Table II.

The goal of building these regression models was achieved by finding the model that contains the largest number of errors and comparing the machine-trained model with the outputs in the data. The accuracy was measured to discover the model that predicts the number of errors, and the accuracy expresses the result of the prediction of the machine in how close it is to the original result. Here, the closer the result is to zero, the better the result. From our experience, it is difficult to determine which model is better because of the amount of disparate data, but we can determine the best model by comparing the models on one dataset only.

Fault-Percentile-Average (FPA) evaluation measure was used based on previous studies, which were studied based on classification models, thus showed satisfactory results, but in

TABLE II. COMPARISON OF THE LTR APPROACH WITH SIX EXISTING REGRESSION MODELS OVER 28 DATASETS WITH ALL METRICS USING FPA MEASURE

Datasets	**LR**	**RF**	**SVR**	**ZIP**	**ZFR**	**NBR**
ant-1.7	0.72154	0.77648	0.77122	0.76151	0.76784	0.81054
camel-1.0	0.60842	0.640408	0.66202	0.61714	0.57756	0.45306
camel-1.6	0.57958	0.71205	0.69675	0.71383	0.63459	0.72812
data_arc	0.53375	0.65012	0.62578	0.54251	0.5341	0.50761
data_ivy-2.0	0.52781	0.67103	0.65993	0.61565	0.54292	0.70769
data_prop-6	0.56486	0.64195	0.66541	0.66563	0.55886	0.71364
data_redaktor	0.647809	0.75695	0.74704	0.68828	0.64923	0.76476
JDT_R2_0	0.65196	0.72185	0.70785	0.69471	0.71048	0.67293
JDT_R2_1	0.63748	0.77034	0.74373	0.76109	0.67657	0.75939
JDT_R3_0	0.64215	0.784	0.77129	0.76294	0.73085	0.76901
JDT_R3_1	0.62876	0.77124	0.77164	0.74924	0.70828	0.75553
JDT_R3_2	0.70679	0.6923	0.768605	0.72612	0.76279	0.76752
jedit-3.2	0.7021	0.82434	0.79438	0.80392	0.79859	0.79549
jedit-4.2	0.68124	0.8172	0.78171	0.73489	0.71828	0.84774
log4j-1.1	0.77168	0.7857	0.77079	0.72377	0.76497	0.78366
lucene-2.0	0.73887	0.7384	0.75203	0.74142	0.74499	0.75741
PDE_R2_0	0.64553	0.79867	0.80697	0.66434	0.7267	0.783902
PDE_R2_1	0.7051	0.7989	0.78326	0.64177	0.69192	0.75939
PDE_R3_0	0.70525	0.7603	0.73343	0.72478	0.74422	0.72364
PDE_R3_1	0.72154	0.7458	0.755806	0.72328	0.73999	0.694006
PDE_R3_2	0.63911	0.6912	0.67414	0.618407	0.65225	0.60929
poi-2.0	0.53461	0.6831	0.66588	0.64228	0.5689	0.56535
synapse-1.0	0.72382	0.6905	0.54816	0.51767	0.64681	0.61529
synapse-1.2	0.68296	0.7114	0.70046	0.67401	0.68626	0.69261
velocity-1.6	0.63761	0.7448	0.73101	0.70212	0.614424	0.70044
xalan-2.4	0.53136	0.7831	0.73722	0.66721	0.56662	0.75826
xerces-1.2	0.5083	0.7058	0.67377	0.64692	0.54765	0.57162
xerces-1.3	0.65305	0.7982	0.78974	0.79185	0.63956	0.824109

TABLE III. COMPARISON OF THE LTR APPROACH WITH SIX EXISTING REGRESSION MODELS OVER 28 DATASETS WITH ALL METRICS USING MAE MEASURE

Datasets	**LR**	**RF**	**SVR**	**ZIP**	**ZFR**	**NBR**
ant-1.7	0.41879	0.6446	0.49934	1823.4	0.47655	1.17607
camel-1.0	0.05307	0.0962	0.11	96198	0.044205	1.0006
camel-1.6	0.53989	0.821	0.5827	0.9038	0.569209	1.8976
data_arc	0.14222	0.2289	0.2044	0.281	0.15111	8.3864
data_ivy-2.0	0.13062	0.2011	0.1915	0.2532	0.11931	0.93563
data_prop-6	0.10558	0.1989	0.1755	0.2164	0.103101	0.92528
data_redaktor	0.12	0.7564	0.1836	0.3213	0.12	0.94013
JDT_R2_0	1.60409	1.85684	1.5485	11.67	1.9147	12.9109
JDT_R2_1	0.8217	0.96404	0.8325	1.006	0.9085	4.59836
JDT_R3_0	1.3888	1.643	1.268	1.42	1.632	9.2229
JDT_R3_1	1.16981	1.65063	1.141	1.568	1.463	28.6809
JDT_R3_2	1.0069	1.037	1.003	1.194	1.111	5.76459
jedit-3.2	1.26168	1.5983	1.309	21.56	1.374	13.4099
jedit-4.2	0.30729	0.44233	0.357	66.56	0.3646	1.05466
log4j-1.1	0.67878	0.8246	0.7729	97.44	0.6958	1.2819
lucene-2.0	1.1846	1.4018	1.228	1.5778	1.344	1.7224
PDE_R2_0	0.41136	0.5771	0.4617	0.704	0.455	1.2279
PDE_R2_1	0.32592	0.45809	0.378	1.4671	0.3647	1.1646
PDE_R3_0	0.68745	0.9269	0.6769	1.103	0.7752	12.8876
PDE_R3_1	0.73639	1.025	0.7483	0.9479	0.9085	8.2126
PDE_R3_2	0.8815	1.016	0.8026	2.779	1.155	10.796
poi-2.0	0.15596	0.2296	0.2037	2.028	0.1356	1.412
synapse-1.0	0.15342	0.2271	0.2283	8.472	0.1489	1.117
synapse-1.2	0.49894	0.6402	0.536	0.7406	0.5739	1.01748
velocity-1.6	0.82647	1.468	0.8462	34.75	1.071	13.4941
xalan-2.4	0.23104	0.3194	0.2719	0.3883	0.2319	0.94382
xerces-1.2	0.28181	0.4473	0.3314	5.049	0.3331	1.0469
xerces-1.3	0.37284	0.5871	0.4693	0.6761	0.333	1.23983

this paper, we applied it to six regression models on a larger scale, so that we used all available databases in the field of rank learning, and we have obtained satisfactory results. In this paper, we demonstrated the success of the error-percentage-mean scale. All results were not shown over-fitting on the result.

As seen in Table II, the columns represent all the datasets we used and the rows represent the regression models that we created, for example row ant-1.7 represents the first dataset to which Fault-Percentile-Average has been applied to show the accuracy results for the regression model that was built To determine the model that contains the largest number of program errors, and this accuracy represents the proximity of the learned data to the test data and here we find that the best reading for it is 0.81054, which represents the negative binomial regression model, and this result does not mean that it is the best model because it may depend on the nature of the data and the evaluation measure.

B. *Mean Absolute Error*

By calculating the average accuracy of Mean-Absolute-Error using 5-fold cross validation, the values shown in Table III.

Because we are using regression models in this paper, it is necessary to mention the measurement criteria for the regression, such as Mean Absolute Error. We have used the same methodology in building defect models that contain the largest number of errors and measuring the average accuracy of the models by using 5-fold cross-valuation on a twenty-eight dataset with all features. As shown in Table III, the accuracy results from using the evaluation of the Mean Absolute Error of the regression. It is clear that some results have exceeded the relevance because most of the datasets are intended for classification. However, satisfactory results were shown in some datasets. This does not mean that other models failed to show accuracy in every way, but rather they showed satisfactory results according to the nature of the data.

C. *Root Mean Square Error*

Table IV shows the RMSE results based on different numbers of matrices. We applied 10 times 5-fold cross-validation over 28 datasets with all metrics. By calculating the average accuracy of Root-Mean-Square-Error using 5-fold cross validation, we show the values in Table IV.

This is another way to calculate mean precision with 5-fold validation using RMSE. As shown in Table IV, more than appropriate occurred in some of the data, and this is because the nature of the data is for classification and not for regression. However, we have achieved satisfactory results, and these results were mostly concentrated on two models (Linear Regression and support vector regression).

D. *Cumulative Lift Chart*

This is a way to evaluate the measure of our modules to show the relationship between two evaluation measures (FPA, MAE) in easy and effortless way by presenting the chart of all six modules we have used before as shown in Fig. 2. The charts show the performance of our regression models against other well known LTR methods algorithms [1-2] [3-10].

V. CONCLUSION

SDP models for ranking task manage testing resources more effectively by predicting which modules are likely to have more errors in the software program. SDP data is gathered by a variety of IT organizations and individuals, and it is noisy. As a result, estimating the number of errors per software

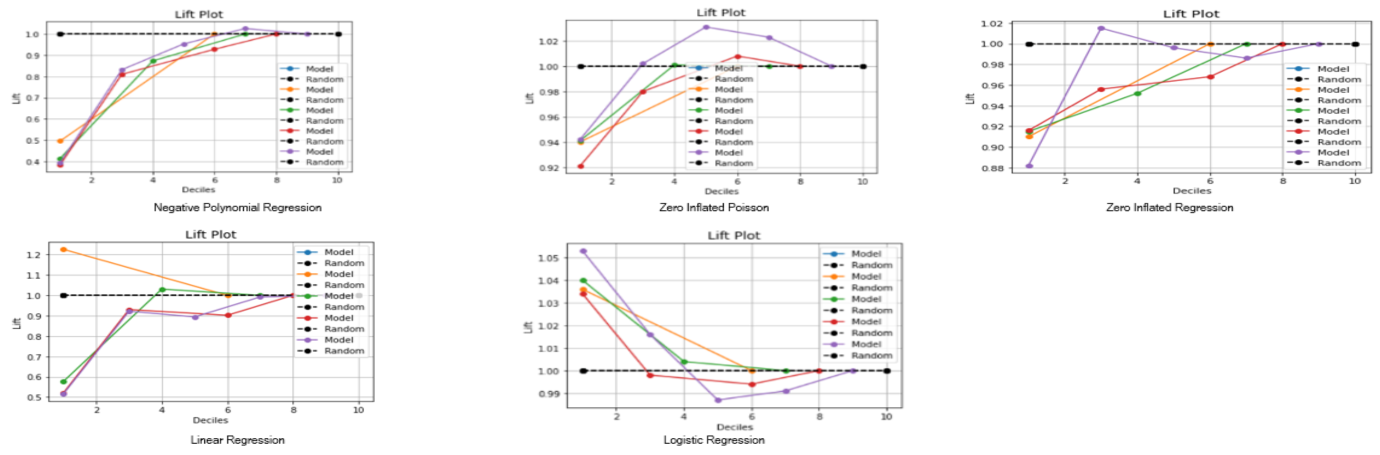

Fig. 2. Cumulative Lift Chart for Various Models.

TABLE IV. Comparison of the LTR Approach with Six Existing Regression Models over 28 Datasets with All Metrics using RMSE Measure

datasets	LR	RF	SVR	ZIP	ZFR	NBR
ant-1.7	0.6435	0.8016	0.70547	27.81	0.6898	1.0757
camel-1.0	0.2265	0.3097	0.3313	14872	0.2022	0
camel-1.6	0.728	0.9049	0.7573	0.936	0.7498	1.3337
data_arc	0.3754	0.478	0.4517	0.5301	0.3831	2.4535
data_ivy-2.0	0.3604	0.448	0.4373	0.5031	0.3447	0.9671
data_prop-6	0.3233	0.4459	0.4183	0.465	0.3201	0.9618
data_redaktor	0.339	0.1953	0.4281	0.5668	0.3431	0.9695
JDT_R2_0	1.2608	1.3495	1.243	2.465	1.38	3.244
JDT_R2_1	0.894	0.9771	0.905	0.9989	0.9497	1.7868
JDT_R3_0	1.151	1.2797	0.905	1.183	1.272	2.7136
JDT_R3_1	1.06	1.273	1.044	1.23	1.205	3.5289
JDT_R3_2	0.9956	1.09	0.9929	1.088	1.053	2.024
jedit-3.2	1.101	1.258	1.126	3.988	1.165	2.98
jedit-4.2	0.5475	0.66471	0.591	5.421	0.6002	1.026
log4j-1.1	0.8222	0.9069	0.8762	5.2293	0.8325	1.1205
lucene-2.0	1.085	1.1798	1.104	1.2416	1.1497	1.2979
PDE_R2_0	0.636	0.7588	0.675	0.8356	0.6715	1.1061
PDE_R2_1	0.5614	0.6731	0.6063	1.7129	0.5924	1.0776
PDE_R3_0	0.8213	0.9577	0.8175	1.035	0.876	2.4717
PDE_R3_1	0.8567	1.0085	0.8625	0.9721	0.9505	2.5657
PDE_R3_2	0.9381	1.004	0.8932	1.454	1.068	2.6754
poi-2.0	0.394	0.4767	0.4502	1.242	0.3654	1.1445
synapse-1.0	0.3737	0.4758	0.4744	2.232	0.3666	1.0521
synapse-1.2	0.6936	0.7973	0.7256	0.8583	0.7557	1.0084
velocity-1.6	0.9029	1.2028	0.9148	4.828	1.032	3.3796
xalan-2.4	0.4798	0.565	0.521	0.6228	0.48	0.9714
xerces-1.2	0.53	0.6662	0.5751	1.917	0.5754	1.022
xerces-1.3	0.6088	0.565	0.679	0.8178	0	1.1126

module is difficult, if not impossible, due to a lack of precise historical data. Some academics propose utilizing a ranking-based performance metric to assess SDP models such as CLC and FPA. However, contemporary SDP models have been enhanced to properly predict a specific number of errors. However, a decent model based on individual loss functions may fail to provide a satisfactory ranking. As a result, in this paper, we proposed a unique approach, distinct from earlier studies, for developing models by direct improvement of the ranking performance measurement. We applied the LTR approach to a wide range of real-world datasets in this paper and present a complete assessment and comparison of RF, SVR and LR with other approaches. We also estimated the error using FPA and MAE and then used CLC to explain the disparity between its results.

The following are the key findings from our research paper:

1) Employing the regression approach rather than the classification approach, as opposed to prior studies in the literature where the classification technique is employed. This is to highlight the contrast between the classification and regression models. Whether or not this model has mistaken, the data is divided into 0 and 1, with 0 containing no errors and 1 containing errors. This is known as classification. However, the regression models that we are working on estimate the number of mistakes in each website, which means that the first website based on the characteristics (x values) has a number of errors, and so the regression models train the model when the data enters it. The characteristics will predict mistakes that are either equal to or near to the amount of genuine errors. This is the point of using regression models.
2) Proposing a new LTR approach with scipy.stats and apply it to multiple models to compare and calculate accuracy. We discovered that the model produced using regression models accomplished what was expected of it in terms of identifying models with the highest number of errors, and the percentage of accuracy varied according to the type of data. And according to the comparison with the standard measures, we found that the model RF and SVR is better.

Based on the results and their comparison, we found that the measurement criteria differ from each other, so that we found a gap in calculating the accuracy in some measurements due to the nature of the random data, and FPA achieved better results than MAE, RMSE in this research paper.

Acknowledgment

Yousef Elsheikh and Sara Al-omari are grateful to the Applied Science Private University in Amman, Jordan, for the financial support granted to cover the publication fee of this research article.

Mohammad Azzeh thanks the Princess Sumaya University for Technology for supporting this research.

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] X. Yang, K. Tang and X. Yao, ”A Learning-to-Rank Approach to Software Defect Prediction”, IEEE Transactions on Reliability, vol. 64, no. 1, pp. 234-246, 2015. Available: https://www.ieee.org/publications/rights/index.html.

[3] Xiaoxing Yang, Ke Tang, and Xin Yao ,”A Learning-to-Rank Approach for Constructing Defect Prediction Models”, IEEE, vol. 1, no. 64, p. 9, 2021. Available: http://file:///C:/Users/pc/Desktop/Learning

[4] Z. Cao, Y. Tian, T. Le and D. Lo, ”Rule-based specification mining leveraging learning to rank”, Automated Software Engineering, vol. 25, no. 3, pp. 501-530, 2018. Available: https://ink.library.smu.edu.sg/sis_research/3988/.

[5] X. Huo and M. Li, ”On cost-effective software defect prediction: Classification or ranking?”, Neurocomputing, vol. 363, pp. 339-350, 2019. Available: https://www.journals.elsevier.com/neurocomputin.

[6] X. Yu, K. Ebo Bennin, J. Liu, J. Wai Keung, X. Yin and Z. Xu, ”An Empirical Study of Learning to Rank Techniques for Effort-Aware Defect Prediction”, IEEE, p. 12, 2021. Available: 2019.

[7] M. Buchari, S. Mardiyanto and B. Hendradjaya, ”Implementation of Chaotic Gaussian Particle Swarm Optimization for Optimize Learning-to-Rank Software Defect Prediction Model Construction”, Journal of Physics: Conference Series, vol. 978, p. 012079, 2018. Available: 10.1088/1742-6596/978/1/012079.

[8] Y. Ma, ”A Top-k Learning to Rank Approach to Cross-Project Software Defect Prediction”, IEEE, p. 11, 2021. Available: 2018.

[9] Z. Li, X. Jing and X. Zhu, ”Progress on approaches to software defect prediction”, IET Software, vol. 12, no. 3, pp. 161-175, 2018. Available: 10.1049/iet-sen.2017.0148.

[10] A. Okutan and O. Yıldız, ”Software defect prediction using Bayesian networks”, Empirical Software Engineering, vol. 19, no. 1, pp. 154-181, 2012. Available: 10.1007/s10664-012-9218-8.

[11] Y. Ma, G. Luo, X. Zeng and A. Chen, ”Transfer learning for cross-company software defect prediction”, Information and Software Technology, vol. 54, no. 3, pp. 248-256, 2012. Available: 10.1016/j.infsof.2011.09.007.

[12] J. Zheng, ”Cost-sensitive boosting neural networks for software defect prediction”, Expert Systems with Applications, vol. 37, no. 6, pp. 4537-4543, 2010. Available: 10.1016/j.eswa.2009.12.056.

[13] X. Jing, S. Ying, Z. Zhang, S. Wu and J. Liu, ”Dictionary Learning Based Software Defect Prediction”, p. 10, 2014.

[14] G. Czibula, Z. Marian and I. Czibula, ”Software defect prediction using relational association rule mining”, Information Sciences, vol. 264, pp. 260-278, 2014. Available: 10.1016/j.ins.2013.12.031.

[15] I. Laradji, M. Alshayeb and L. Ghouti, ”Software defect prediction using ensemble learning on selected features”, Information and Software Technology, vol. 58, pp. 388-402, 2015. Available: 10.1016/j.infsof.2014.07.005.

[16] S. Liu, X. Chen, W. Liu, J. Chen, Q. Gu and D. Chen, ”FECAR: A Feature Selection Framework for Software Defect Prediction”, IEEE, p. 10, 2014.

[17] P. Krause and N. Fenton, ”A probabilistic model for software Defect Prediction”, IEEE, p. 36, 2001.

[18] N. Li, M. Shepperd and Y. Guo, ”A systematic review of unsupervised learning techniques for software defect prediction”, Information and Software Technology, vol. 122, p. 106287, 2020. Available: 10.1016/j.infsof.2020.106287.

[19] T. M. Khoshgoftaar, K. Gao† and N. Seliya, ”Attribute Selection and Imbalanced Data: Problems in Software Defect Prediction”, IEEE, vol. 1, p. 8, 2010.

[20] L. Son, N. Pritam, M. Khari, R. Kumar, P. Phuong and P. Thong, ”Empirical Study of Software Defect Prediction: A Systematic Mapping”, Symmetry, vol. 11, no. 2, p. 212, 2019. Available: 10.3390/sym11020212.

[21] M. Sohan, M. Jabiullah, S. Motiur Rahman and S. Mahmud, ”Assessing the Effect of Imbalanced Learning on Cross-project Software Defect Prediction”, IEEE, 2019.

[22] X. Cai et al., ”An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search”, Concurrency and Computation: Practice and Experience, vol. 32, no. 5, 2019. Available: 10.1002/cpe.5478.

[23] Al-Haija, Haneen Abu, Mohammad Azzeh, and Fadi Almasalha. ”Software Defect Prediction Using Support Vector Machine.” International Journal of Systematic Innovation 7, no. 2 (2022): 37-47.