A Robust Tuned K-Nearest Neighbours Classifier for Software Defect Prediction

A Robust Tuned K-Nearest Neighbours Classifier for Software Defect Prediction

Abdullah B. Nasser[1], Waheed Ghanem[2], Antar Shaddad Hamed Abdul-Qawy [3], Mohammed A. H. Ali [4], Abdul-Malik Saad[5], Sanaa A. A. Ghaleb [6] and Nayef Alduais[7]

[1] School of Technology and Innovation, University of Vaasa, 65200 Vaasa, Finland
[2] Faculty of Ocean Engineering Technology and Informatics, Universiti Malaysia Terengganu, Kuala Terengganu, Malaysia
[3] Department of Science & Information Technology, Faculty of Science SUMAIT University Zanzibar, Tanzania eng.antar2007@gmail.com
[4] University of Malaya, Faculty of Engineering, Department of Mechanical Engineering
[5] Division of Electronic and Computer Engineering, School of Electrical Engineering, Faculty of Engineering, Universiti Teknologi Malaysia, 81310, Johor, Malaysia abdul-malik@utm.my
[6] Faculty of Informatics and Computing Universiti Sultan Zainal Abidin Kuala Terengganu, Malaysia
[7] Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, Johor, Malaysia nayef@uthm.edu.my

Nasser.abdullah@uwasa.fi

Abstract. If the software fails to perform its function, serious consequences may result. Software defect prediction is one of the most useful tasks in the Software Development Life Cycle (SDLC) process where it can determine which modules of the software are prone to defects and need to be tested. Owing to its efficiency, machine learning techniques are growing rapidly in software defect prediction. K-Nearest Neighbors (KNN) classifier, a supervised classification technique, has been widely used for this problem. The number of neighbors, which measure by calculating the distance between the new data and its neighbors, has a significant impact on KNN performance. Therefore, the KNN's classifier will perform better if the k hyperparameters are properly tuned and the independent inputs are re-scaled. In order to improve the performance of KNN, this paper aims to presents a robust tuned machine learning approach based on K-Nearest Neighbors classifier for software defect prediction, called Robust-Tuned-KNN(RT-KNN). The RT-KNN aims to address the two abovementioned problems by 1) tuning KNN and finding the optimal value for k in both the training and testing phases that can lead to good prediction results, and 2) using the Robust scaler to rescale the different independent inputs. The experiment results demonstrate that RT-KNN is able to give sufficiently competitive results compared with original KNN and other existing works.

Keywords: Software Defect Prediction, Machine Learning, K-Nearest Neighbors classifier, Hyper-parameters Tuning.

1. Introduction

Software development has been growing rapidly, and it's getting more complex in terms of software size and functionalities. As a result, software reliability is an effect, which plays a critical role in the Software Development Life Cycle (SDLC) process. Catching software defects is one of the most difficult tasks in the software development life cycle. As a result of that, many software testing approaches [1, 2] have been proposed to reduce software defects, such as static testing, which includes reviewing and analysing the software's documents in order to prevent any software defects, while dynamic testing focuses on examining the software's behaviour to identify any software defects.

Machine learning is a well-known technology that is used in many fields. Machine learning's popularity stems from its advantages in learning the machine from historical data and making predictions on future data. Machine learning in software defect prediction or software testing, in general, is still quite new, However, similar to other areas of artificial intelligence, it is growing rapidly.

The code is a small unit in any application or program in which each portion of code is designed to do a certain task. In most cases, these codes contain some defects, which may result in a consequence disaster such as losing data, losing fortune or even losing lives. Software defect prediction is one of the most useful tasks in the SDLC's where it can determine which modules are prone to defect and need to be tested. It also allows the test manager to use the resources effectively while still adhering to the limits.

The purpose of software defect prediction includes classifying the software components into not defect-prone and defect-prone classes. It is also used for identifying any associations between defects. Another purpose of software defect prediction is to estimate the number of faults remaining in software systems.

K-Nearest Neighbors (KNN) classifier, a supervised classification technique, has been widely used for this problem. A set of parameters, including the numbers of neighbors k, in the KNN classifier, must be tuned. The performance of KNN is highly dependent on the number of neighbors and the problem at hand. Therefore, the classifier will perform better if these hyperparameters are properly tuned [3]. Another related issue in KNN is that KNN is a distance-based method and the independent inputs or features of the dataset often come in different scale/units therefore calculating the distance will not functions at its best due to some outliers in the dataset [4].

Therefore, this research presents a robust tuned machine learning approach based on K-Nearest Neighbors classifier for software defect prediction, called Robust-Tuned-KNN(RT-KNN). The RT-KNN aims to address the two abovementioned problems by 1) tuning KNN to select the optimal value for k in both the training and testing phases that can lead to good prediction results and 2) using the robust scaler to rescale the different independent inputs. First, to find the optimum values of hyperparameters, which are the initial values of the model's parameters that are set by the data analysts before training the model, we investigate the number of neighbors manually and using Grid Search. In the tuning process, the number of neighbors is varied while the other

parameters remain constant based on the recommended values. Then compare their results and find the best number of neighbors. Then, the optimal results of the hyperparameters obtained are used in the second step which uses the robust scaler to scale the model's features.

The rest of the paper is organized as follows: Section 2 explains how the K-Nearest Neighbors Algorithm works, Section 3 reviews the related works, while Section 3 presents the methodology of RT-KNN for software defect prediction. The results and discussion are presented in Section 4, and finally, the conclusion is in Section 5.

2. The k-nearest Neighbors Algorithm

The K-Nearest Neighbors Algorithm (KNN) is a supervised learning approach proposed by Evelyn Fix and Joseph Hodges [5]. It is used for both the classification and regression of data. In both cases, the nearest k neighbors are used for testing the models. The idea behind KNN is a very simple algorithm. The algorithm defines k nearest neighbors of the new data, then it is classified based on a majority vote of its neighbors.

Suppose we have two classes, A with a blue circle and B with a green triangle, and we want to classify the red pentagon (it is called the test vector), as shown in Fig 1. The algorithm calculates the distance between the test vector and k neighbors. Suppose k=4, then the algorithm will calculate the distance between the test vector and its four neighbors, then the test vector is classified (also called labelled) as class A, since there are 3 vectors out of the 4 nearest neighbor vectors from class A and only 1 from class B.

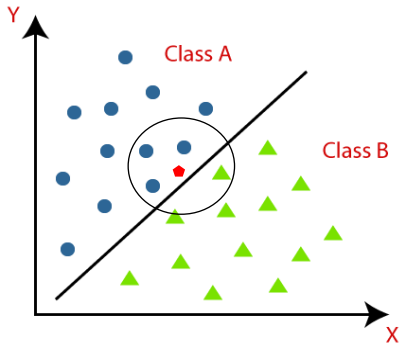

Fig 1. illustration of KNN's Classification

3. Related works

In general, software defect prediction can be classified into two categories: manual classification methods and automatic classification methods. In manual classification methods, analysts utilize their own experience to classify software defects into distinct groups [6]. The analyst begins by establishing the category of defects. Then, based on their experience, they locate the defect in the matching category. However, this method of classification is difficult and requires a huge number of human resources [7].

In 1992, IBM introduced Orthogonal Defect Classification (ODC). ODC is a manual classification tool that extracts some information from the software to provide insight and diagnostics. ODC defines eight defect attributes in its newest version [8], and it employs several attributes to describe defect features. Software defects are classified into eight categories based on the characteristics of these attributes, such as function, verification, algorithm, assignment, timing, association, interface, and documentation. Similar to ODC, IEEE introduces IEEE standard classification for Anomalies. The classification using IEEE stander is based on user experience. Therefore, engineers must first classify the anomalies as defects or failures and then use the standard. Another manual classification is called Thayer classification where the tester fills out error reports during testing and comments from users to categorise errors according to their nature [6]: calculation errors, logic errors, I/O errors, data processing errors, operating system's errors, and so on.

In literature, there are many works similar to ODC, IEEE standard classification, and Thayer classification such as Roger classification, Putnam classification and Michael classification, to name a few.

On the other hand, automatic classification methods use algorithms such as machine learning to classify the defects. In the literature, several machine learning approaches have been proposed to address the software defect prediction problem, such as logistic regression[9], neural network[10], support vector machines[11], and random forests [12].

Based on the information extracted from ODC, LiGuo Huang et al. proposed new classification methods using a support vector machine (SVM) called ODC automatic classification. Before using the SVM classifier, the method divides the defects manually into six classes and then creates vectors to be used in SVM[11].

Close to ODC automatic classification method, Thung et al. proposed another automatic classification based on SVM. However, they believe that the reported bugs and the source code are very important for the classification process. Xia Xin et al. proposed an automatic defect classification model that uses fuzzy set feature selection and a 10-fold cross-validation method. The method starts extracting the information from the bug report, then extracting the subset of features then evaluating the model using cross-validation. Another automatic classification based on trigger mode was proposed by Xia Xin et al. By analyzing the code, the method classifies the defect into Bohrbug, which is a simple defect and easy to isolate, or Mandelbug defect which is more complex than the Bohrbug defect. The method uses a fuzzy set to select the important features that can help to classify the defects [6].

4. Robust Tuned K-Nearest Neighbors Classifier for Software Defect Prediction

This section describes the methodology of Robust Tuned K-Nearest Neighbors Classifier for Software Defect (RT-KNN) prediction including describing and analyzing the dataset and describing the steps that use in tuning and rescaling K-Nearest Neighbors Classifier.

A. Dataset

The JM1 dataset was used as the basis for our research. The dataset is one of NASA Metrics Data Program which is real-time predictive ground system. The dataset consists of 10885 instants, each instant represents a module, and 22 attributes including the target attribute that label the software as DEFECTED class 1 or UNDEFEATED or CLEAN class 0, as Fig 2 shows. The dataset contains some statistical measurements of the software such as count of lines (LOC), McCabe Cyclomatic Complexity v(G), McCabe Essential Complexity ev(G), McCabe Design Complexity iv(G) and some Halstead's Complexity Measures such as Halstead total operators (n) Halstead "volume" (v), Halstead "program length" (l), Halstead "difficulty" (d), Halstead "intelligence" (i) so on. More information about the dataset can be found in [13].

	count	mean	std	min	25%	50%	75%	max
loc	10885.0	42.016178	76.593332	1.0	11.00	23.00	46.00	3442.00
v(g)	10885.0	6.348590	13.019695	1.0	2.00	3.00	7.00	470.00
ev(g)	10885.0	3.401047	6.771869	1.0	1.00	1.00	3.00	165.00
iv(g)	10885.0	4.001599	9.116889	1.0	1.00	2.00	4.00	402.00
n	10885.0	114.389738	249.502091	0.0	14.00	49.00	119.00	8441.00
v	10885.0	673.758017	1938.856196	0.0	48.43	217.13	621.48	80843.08
l	10885.0	0.135335	0.160538	0.0	0.03	0.08	0.16	1.30
d	10885.0	14.177237	18.709900	0.0	3.00	9.09	18.90	418.20
i	10885.0	29.439544	34.418313	0.0	11.86	21.93	36.78	569.78
e	10885.0	36836.365343	434367.801255	0.0	161.94	2031.02	11416.43	31079782.27
b	10885.0	0.224766	0.646408	0.0	0.02	0.07	0.21	26.95
t	10885.0	2046.464876	24131.544463	0.0	9.00	112.83	634.25	1726654.57
lOCode	10885.0	26.252274	59.611201	0.0	4.00	13.00	28.00	2824.00
lOComment	10885.0	2.737529	9.008608	0.0	0.00	0.00	2.00	344.00
lOBlank	10885.0	4.625540	9.968130	0.0	0.00	2.00	5.00	447.00
locCodeAndComment	10885.0	0.370785	1.907969	0.0	0.00	0.00	0.00	108.00
uniq_Op	10885.0	11.172458	10.045803	0.0	5.00	11.00	16.00	411.00
uniq_Opnd	10885.0	16.744162	26.664174	0.0	4.00	11.00	21.00	1026.00
total_Op	10885.0	68.079302	151.486061	0.0	8.00	29.00	71.00	5420.00
total_Opnd	10885.0	46.367680	100.333717	0.0	6.00	19.00	48.00	3021.00
branchCount	10885.0	11.287129	22.593722	0.0	3.00	5.00	13.00	826.00
defects	10885.0	0.193477	0.395042	0.0	0.00	0.00	0.00	1.00

Fig 2. JM1 dataset information

The classification of JM1 is challenging because there are many modules with the same identical attribute values having different defect labels [14]. The correlation between the defect class and the other features of the modules is not strong as shown in Fig 3. The correlation in the figure is scaled from 0 to 1. The last column and row of the figure, which present the correlation between the defect classes and other features, are low for all features. This issue may spur research on selecting the best combinations of features for better classification.

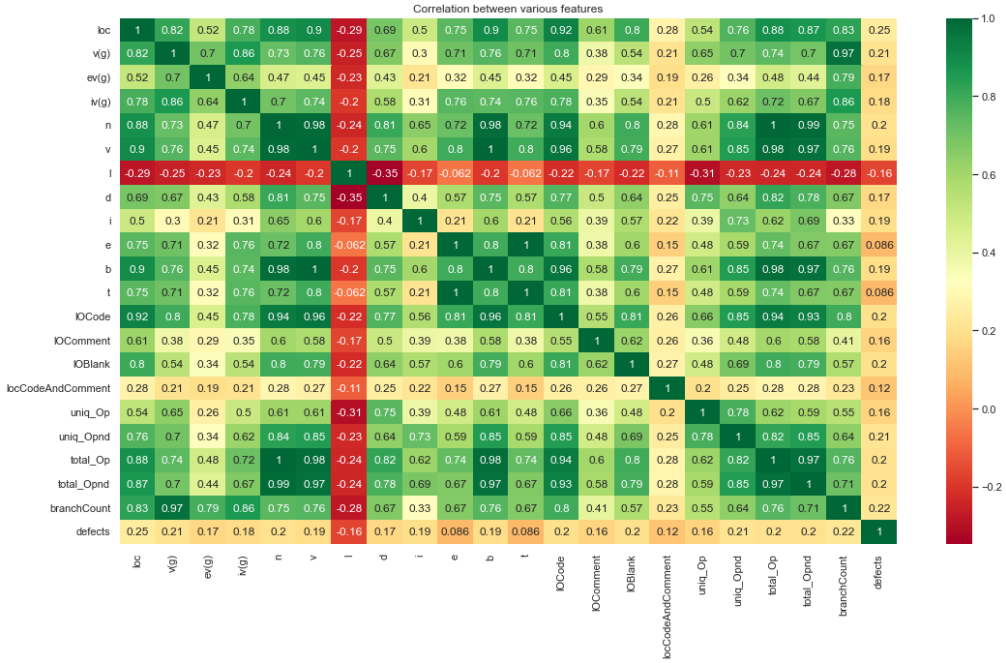

Fig 3. Correlation between the features of JM1 dataset

The dataset also has another issue which is unbalancing since it contains 8779 defected modules and 2106 undefeated modules. There is also overlapping between the two classes as the pairwise relations of the attributes of the dataset are shown in Fig 4.

Fig 4. The pairwise relations of the attributes of the dataset

B. Method and implementation

A good correlation between the independent variables and the target variables is a good sign for the prediction. The performance of KNN is highly dependent on the number of neighbors. Selecting the optimal value for k in both the training and testing phases can lead to good prediction results. In KNN, The training phase is just sorting the training data thus the main computation of the algorithm is at the time of prediction rather than at the time of fitting. In the prediction phase, the algorithm finds the k neighbors of the new point by calculating the distance between the new point and the neighbors, and uses these values to classify the new point. Therefore, the accuracy of the model is highly dependent on the number of neighbors. Additionally, due to the nature of and being a distance-based method therefore outliers' values in the training data may affect the performance of the method, consequently, it is necessary to reduce the effect of the outliers by rescaling the training data. Accordingly, this section aims to propose a robust tuned machine learning approach based on K-Nearest Neighbors (RT-TKK) classifier for software defect prediction.

The RT-KNN model has been implemented from the screech using python. Following are the main steps of the implementation:

1. Reading and Pre-processing the dataset: The JM1 dataset is used in this work. The data in the dataset is clean and does not require preprocessing.
2. The dataset is split into training (80%) and testing (20%) subsets.
3. In order to find the optimal value of k, the value of k is varied from 1 to 25 based on the recommendation values. Since the goal is to find the optimal value of k, the other parameters have been fixed.
4. The KNN model is trained and tested. The accuracy results of both training and testing are recorded. We can simply select the optimal value of k based on the testing accuracy results, however, we prefer to use another method to select the optimal value based on both training and testing results. Here the Pareto frontier method, which is multi-objective optimization, is used. Pareto frontier takes two equally-sized lists (training and testing accuracy) and returns the line of points (a.k.a Pareto curve) that is fitted to the points, in order.
5. The optimal value of returning k by Pareto frontier which is the efficient points of both the training and testing phase is used for prediction.
6. For comparison purposes, the GridSearchCV is used also for finding the optimal values, and then the results of our method and GridSearchCV are compared.
7. Steps 3 through 5 are repeated, but this time after rescaling the dataset using the Robust Scaler. Robust Scaler is useful for removing outliers.

In case some input variables have relatively big values compared to other input variables, these big values, may affect the accuracy of the model, are rescaled using the robust scaler. To calculate the robust scale, the value is subtracted from the median, and then it is divided by the interquartile range (75% -25%) value. As a result, the scaler focuses on the large values while ignoring the variables with lesser values.

5. Results and discussion

As mentioned earlier, the JM1 dataset was used in the experiments. The prediction accuracy is used to assess the performance of the tuning model. The prediction accuracy can be calculated using the formula:

$$\text{Accuracy} = \frac{\text{number of correct predicted test modules}}{\text{the total number of test modules}}$$

The following experiments have been carried out in order to assess the accuracy of RT-KNN:

Experiment 1: Tuning the K Value manually and using GirdSerachCV

In this experiment, the number of neighbors is varied from 1 to 25, while the other parameters of KNN are fixed. Fig 5 shows the accuracy scores of both the training and testing phases.

The figure shows that when the number of neighbors is less than SIX (6), the accuracy of training data is high while the accuracy of test data is low which is called underfitting, and when the number of neighbors is greater than or equal to SIX (6), the model obtains good and stable results. Even though it is recommended to use an odd number for k, the best point that got the best results for both training and testing is when k = 22.

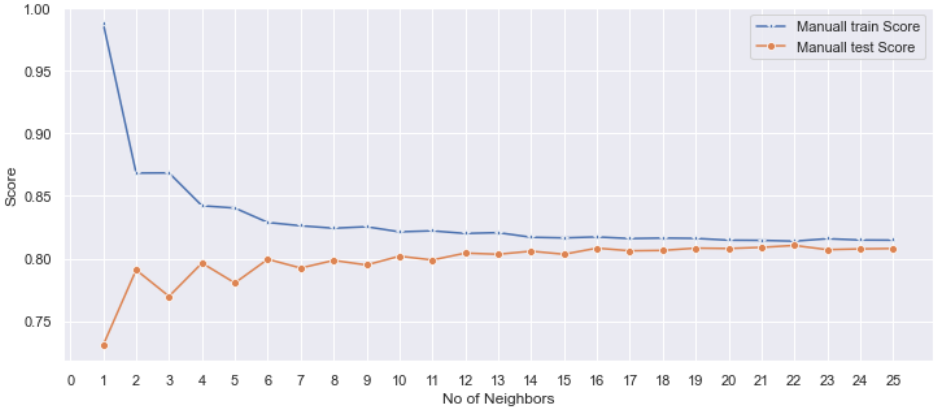

Fig 5. Train and test accuracy for KNN manually tuning

In the second part of this experiment, the KNN is tuned using GirdSerachCV. Similar to manual tuning, the value of k is varied from 1 to 25, and the other parameters have been fixed. Fig 6 shows the accuracy scores obtained using GridSearchCV. The results

show that overfitting when the number of neighbors is small then as expected the results are getting better with a large number of neighbors.

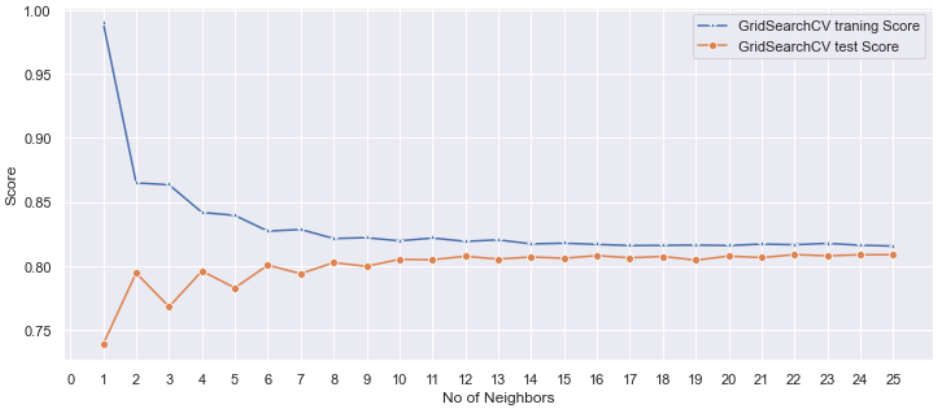

Fig 6. Tuning KNN using GridSearchCV

Experiment 2: Comparison of Manual and GridSearchCV tuning

The third part of experiment 1 is comparing the manual and GridSearcCV. The results of comparing manual and GridSearchCV tuning are presented in Fig 7 and 8. The figures show the comparison of accuracy scores between manual and GridSearchCV tuning. The results show that the performance of the two models start to obtain good and stable results when the number of neighbors (k) is greater or equal to 12. The best accuracy in both models is obtained by manual tuning is when k equal to 22.

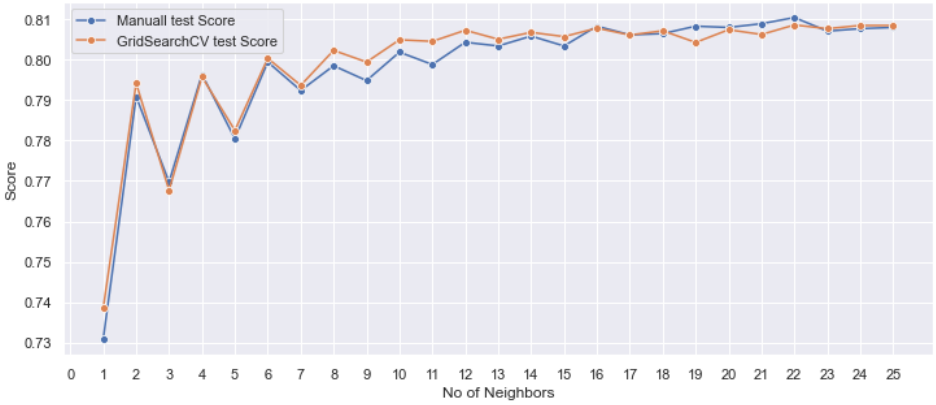

Fig 7. visual presentation of accuracy scores comparison between manual and GridSearchCV

No	accuracy
1	0.730863
2	0.790876
3	0.769749
4	0.796387
5	0.780465
6	0.799449
7	0.792407
8	0.79853
9	0.794856
10	0.801898
11	0.798836
12	0.804348
13	0.803429
14	0.805879
15	0.803429
16	0.808328
17	0.806185
18	0.806491
19	0.808328
20	0.808022
21	0.808941
22	0.810472
23	0.807103
24	0.807716
25	0.808022

Manual tuning

No	accuracy
1	0.738548
2	0.794199
3	0.767685
4	0.795774
5	0.782518
6	0.800367
7	0.793674
8	0.802336
9	0.799449
10	0.804961
11	0.804567
12	0.807323
13	0.805092
14	0.806798
15	0.805749
16	0.807849
17	0.806142
18	0.807192
19	0.804305
20	0.807455
21	0.806274
22	0.808636
23	0.807717
24	0.808505
25	0.808505

GridSearchCV

Figure 8: Accuracy scores comparison between manual and GridSearchCV

Experiment 3: Applying Robust Scaler on KNN

As stated earlier, outlier values in the dataset may affect the performance of the KNN classifier. In this experiment, the dataset was rescaled using the robust scaler to lessen the influence of outliers. It is clear how much the effect of rescaling the dataset on accuracy; the accuracy scores have been improved significantly. In fact, other two scalers have been tested in this data set, which are MaxMin Scaler and standard Scaler, only the robust scaler achieves better performance than the others.

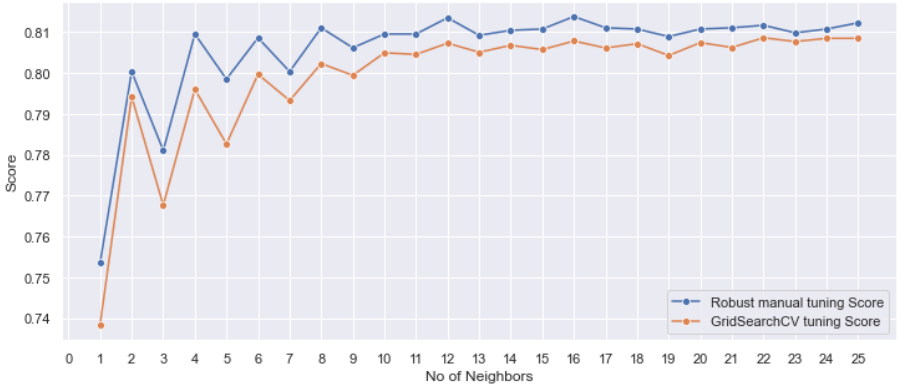

Figure 9: Comparison of RT-KNN with GridSearchCV after Applying Robust Scaler

Experiment 4: Comparison with Previous Works

In this experiment, the RT-KNN results are compared with the best-obtained results from the existing work. The optimal k obtained from the previous experiments that applied the robust scaler is used for predicting the same test data. Only previously published works that use the same dataset and have the same number of features are considered. The results are collected from[7, 12, 15]. The comparison includes different types of classifiers, some are distance-based similar to KNN while the others are based on SVM and decision tree. As Fig 10 shows, among the existing works, the tuning-KNN and Decision Tree (FR) achieved the highest accuracy.

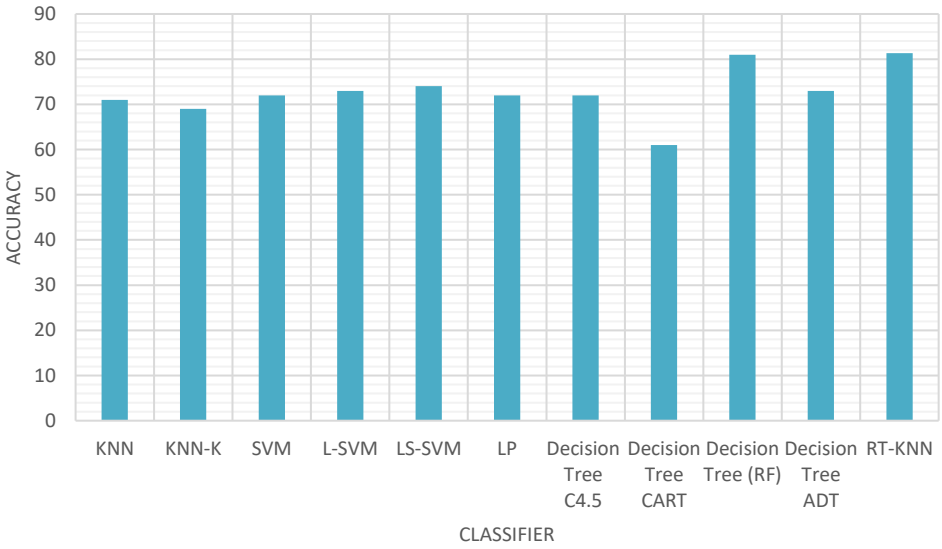

Fig 10. Comparison of Tuning-KNN with the existing classifier

6. Conclusion

This paper presents a machine learning software defect prediction approach based on the K-Nearest Neighbors classifier, called a robust tuned K-Nearest Neighbors (RT-KNN). The method attempts to address two KNN problems, which are finding the optimal value of the number of neighbors and rescaling outliers in the dataset. Manual and using GridSearchCV are applied to find the optimal value of k, and the robust scaler is used to rescale the big values. Several experiments have been carried out in order to assess the accuracy of RT-KNN, including tuning the value of K manually and using GirdSerachCV and comparing their results, then applying the robust scaler and finally comparing the RS-KNN with some published results. The results of the experiment

demonstrate that RT-KNN generates outcomes that are competitive with existing published works.

References

1. Pachouly, J., et al., A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools. Engineering Applications of Artificial Intelligence, 2022. **111**: p. 104773.
2. Akimova, E.N., et al., A survey on software defect prediction using deep learning. Mathematics, 2021. **9**(11): p. 1180.
3. Mabayoje, M.A., et al., Parameter tuning in KNN for software defect prediction: an empirical analysis. Jurnal Teknologi dan Sistem Komputer, 2019. **7**(4): p. 121-126.
4. Yu, C., et al. Indexing the distance: An efficient method to knn processing. in Vldb. 2001.
5. Fix, E. and J.L. Hodges, Discriminatory analysis. Nonparametric discrimination: Consistency properties. International Statistical Review/Revue Internationale de Statistique, 1989. **57**(3): p. 238-247.
6. Gao, J., et al. Research on software defect classification. in 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC). 2019. IEEE.
7. Lessmann, S., et al., Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Transactions on Software Engineering, 2008. **34**(4): p. 485-496.
8. Chillarege, R., Orthogonal defect classification. Handbook of software reliability engineering, 1996: p. 359-399.
9. Olague, H.M., et al., Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. IEEE Transactions on software Engineering, 2007. **33**(6): p. 402-419.
10. Arar, Ö.F. and K. Ayan, Software defect prediction using cost-sensitive neural network. Applied Soft Computing, 2015. **33**: p. 263-277.
11. Elish, K.O. and M.O. Elish, Predicting defect-prone software modules using support vector machines. Journal of Systems and Software, 2008. **81**(5): p. 649-660.
12. Guo, L., et al. Robust prediction of fault-proneness by random forests. in 15th international symposium on software reliability engineering. 2004. IEEE.
13. Mike, C. JM1/software defect prediction. 2004 22-June-2022]; Available from: https://www.openml.org/d/1053.
14. Zhong, S., T.M. Khoshgoftaar, and N. Seliya, Analyzing software measurement data with clustering techniques. IEEE Intelligent Systems, 2004. **19**(2): p. 20-27.
15. Cetiner, M. and O.K. Sahingoz. A comparative analysis for machine learning based software defect prediction systems. in 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT). 2020. IEEE.