A Distance-Based Feature Selection Approach for Software Anomaly Detection

Suravi Akhter[1], Afia Sajeeda[2] and Ahmedul Kabir[2]
[1]*Department of Computer Science and Engineering, University of Liberal Arts Bangladesh, Dhaka, Bangladesh*
[2]*Institute of Information Technology, University of Dhaka, Dhaka, Bangladesh*

Keywords: Software Defect Prediction, Bug Severity Classification, Feature Selection.

Abstract: An anomaly of software refers to a bug or defect or anything that causes the software to deviate from its normal behavior. Anomalies should be identified properly to make more stable and error-free software systems. There are various machine learning-based approaches for anomaly detection. For proper anomaly detection, feature selection is a necessary step that helps to remove noisy and irrelevant features and thus reduces the dimensionality of the given feature vector. Most of the existing feature selection methods rank the given features using different selection criteria, such as mutual information (MI) and distance. Furthermore, these, especially MI-based methods fail to capture feature interaction during the ranking/selection process in case of larger feature dimensions which degrades the discrimination ability of the selected feature set. Moreover, it becomes problematic to make a decision about the appropriate number of features from the ranked feature set to get acceptable performance. To solve these problems, in this paper we propose anomaly detection for software data (ADSD), which is a feature subset selection method and is able to capture interactive and relevant feature subsets. Experimental results on 15 benchmark software defect datasets and two bug severity classification datasets demonstrate the performance of ADSD in comparison to four state-of-the-art methods.

1 INTRODUCTION

Unexpected behaviour (anomaly) in software hampers software quality and introduces defects into the system. The defects may appear in different forms such as syntax errors, wrong program statements and specification errors which negatively impact user experience. However, identification of software defects for addressing the problem needs a considerable amount of time, effort and cost for the software development and maintenance team (Erlikh, 2000). Therefore, one of the highest priorities of software industry is the production of defect-free software at a lower cost.

Traditional machine learning algorithms such as support vector machine (SVM), k-nearest neighbour (kNN) and decision tree (DT) can be applied for the identification/prediction of defects from the vast amount of high dimensional data. However, higher dimensions make the prediction model more complex, thus requiring additional computation cost and time. Hence, unnecessary features should be excluded to improve the classification performance. For the removal of unnecessary features, feature selection methods can be used (Jimoh et al., 2018; Akintola et al., 2018; Agarwal et al., 2014).

The traditional feature selection methods can be broadly divided into two categories: feature ranking (Igor, 1994; Peng et al., 2005; Akhter et al., 2021; Moody and Yang, 1999; Vinh et al., 2016; Yang and Hu, 2012; Greene et al., 2009a; Urbanowicz R. J. Olson and H., 2018; Wang et al., 2012; Tarek et al., 2021) and feature subset selection (Sharmin et al., 2019; Khan et al., 2019; Ruiz et al., 2012; Sharmin et al., 2017b; Laradji et al., 2015) methods. Feature ranking methods produce the rank of provided feature set on the basis of some performance criteria. In contrast, feature subset selection approaches select a subset (S) of features from the original set of features (F).

The use of feature subset selection methods over ranking methods are more beneficial due to the automatic selection of desired number of features whereas finding the optimal number of features in ranking methods are problematic. There are various types of criteria for feature ranking/subset selection, such as correlation coefficient (Goh et al., 2004; Jo et al., 2019), mutual information (MI) (Sharmin

Akhter, S., Sajeeda, A. and Kabir, A.
A Distance-Based Feature Selection Approach for Software Anomaly Detection.
DOI: 10.5220/0011859500003464
In *Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2023)*, pages 149-157
ISBN: 978-989-758-647-7; ISSN: 2184-4895

et al., 2019; Lewis, 1992; Peng et al., 2005; Nguyen Xuan Vinh and Bailey, 2016), pareto optimality (PO) (Sharmin et al., 2017a) and distance-based (Kira et al., 1992; Igor, 1994; Greene et al., 2009a; Pisheh and Vilalta, 2019; Yang and Hu, 2012; Sun, 2007) methods.

At the beginning, correlation coefficient based feature selection method was used in software defect prediction (Laradji et al., 2015). Apart from this, selection of attribute with log filtering (SAL) is proposed which is dependent on a specific classifier and wants to maximize the performance measure metric value (Sharmin et al., 2015). However, the methods that are classifier-independent and built on the intrinsic properties of the features have gained more popularity (Senawi et al., 2017). Later, BAT Search meta heuristic algorithm is applied for the relevant feature selection process and random forest classifier is applied for the defect prediction (Ibrahim et al., 2017). However, it starts the feature selection process from a random point, which might not exhibit the desired performance.

Though the aforementioned methods (Sharmin et al., 2015; Ibrahim et al., 2017) perform reasonably well in software defect/bug prediction, it cannot capture non-linear relationship among the features and cannot remove the similar (redundant) types of features. MI based methods addresses these issues and a method namely bug feature selection (BFSp) is proposed for bug severity classification (Sharmin et al., 2017a). Moreover, min-redundancy and max-dependency (MRMD) is proposed where a new feature similarity measurement term is introduced for better approximation of feature ranking (Gao et al., 2020). However, MI fails to capture feature-feature interaction in case of larger feature dimensional data which is necessary for output prediction. The distance based methods can capture interaction among the features irrespective of feature dimension by considering all features in kNN estimation. For this reason, in this paper, we use distance based criterion.

Relief (Kira et al., 1992) is a distance-based method which is also called a statistical approach for feature selection where the feature is ranked based on the class separation capability of the feature. It takes $1NN$ for the class separation estimation and for this reason, the decision may be biased. Moreover, in case of noise, Relief cannot score the features properly. Later, ReliefF is proposed to address the aforementioned issues and handle multiple class data whereas its predecessor Relief was designed for binary classification (Igor, 1994). Besides this, ReliefF takes k nearest neighbour instead of $1NN$ that leads to a better approximation. Sometimes, it is required for ReliefF to take instances to meet the condition of k that are unnecessary for feature performance measurement.

Later, Spatially Uniform ReliefF (SURF) (Greene et al., 2009b) is proposed that uses a threshold distance value for kNN from a random instance. However, the instances close to the boundary instance are confusing instances. MultiSURF (Urbanowicz R. J. Olson and H., 2018) avoids these confusing instances during ranking through the introduction of dead-band zone. From the aforementioned distance-based ranking methods, finding the optimal number of features is quite difficult by adjusting the selected feature quantity when the feature dimension grows. Previously, in (Akhter et al., 2021), they proposed a distance-based feature subset selection method namely mRelief which is a modified version of Relief. However, mRelief has to choose/adjust the value of k for kNN to measure feature performance and sometimes it might take/discard the unnecessary/necessary data points for feature performance evaluation. To address this, we utilize the advantage of MultiSURF where the value of k is avoided by defining a threshold distance from data points. Moreover, to the best of our knowledge, there is no feature subset selection method that is grounded on MultiSURF and has not been applied in software defect prediction or bug severity classification.

In this paper, we propose a feature selection method namely anomaly detection for software data (ADSD) that selects of a subset of features for the purpose of detecting anomalies (defected code and/or bug severity) from software data. ADSD provides a feature subset selection mechanism, instead of ranking features (that traditional feature selection methods perform). The performance of ADSD is measured against four existing state-of-the-art methods on 18 benchmark datasets, and it is seen that ADSD performs comparitively better.

2 BACKGROUND

The five methods namely SAL, Relief, ReliefF, MRMD and SURF are discussed in this section related to understanding the proposed ADSD and the performance of ADSD over the existing methods.

2.1 SAL

SAL is a classifier dependent method that focuses to maximize the balance score. For feature f_k, the balance score ($b[f_k]$) defined in Eqn (1).

$$b[f_k] = 1 - \frac{\sqrt{(1-pd_{f_k})^2 + (0-pf_{f_k})^2}}{\sqrt{2}} \quad (1)$$

here, in Eqn (1), probability of detection (pd_{f_k}) and probability of false alarm (pf_{f_k}) are defined in Eqn (2) and Eqn (3) respectively.

$$pd_{f_k} = \frac{TP}{TP+FN} \quad (2)$$

$$pf_{f_k} = \frac{FP}{FP+TN} \quad (3)$$

here, TP is true positive, FN is false negative, FP is false negative and TN is true negative. To find the feature subset, SAL computes the pair-wise combinations of features and iteratively optimizes the combined balance score. Therefore, when the feature dimension grows finding the all possible feature combination is very expensive.

2.2 MRMD

MRMD gives more priority in capturing dependency between the features and class which maximizes the given optimization defined in Eqn (4).

$$J_{rel} = I(f_k;c) - \frac{2}{|F|}\sum_{f_i \in S} I(f_k;f_i) + \sum_{f_i \in S} I(f_k;c|f_i) \quad (4)$$

here, $|F|$ is the number of features, $I(f_k;c)$ is the relevancy between feature (f_k) and class (c), $I(f_k;f_i)$ is the redundancy between (f_k) and (f_i); and $I(f_k;c|f_i)$ is the complementary information provided by f_i for f_k to predict c.

2.3 Relief

Relief, a distance based method captures interaction among the features while ranking the features and it utilizes score ($s[f_k]$) for N number of instances defined in Eqn (5).

$$s[f_k] = \sum_{i=1}^{N} \frac{\Delta(I_i, miss(I_i)) - \Delta(I_i, hit(I_i))}{N} \quad (5)$$

Here, I_i is the i^{th} target instance, Δ is distance of hit (H_i) and miss (M_i) from I_i.

2.4 ReliefF

In ReliefF, k-nearest neighbor is considered for better performance ($s[f_k]$) approximation of feature (f_k) defined in Eqn (6).

$$s[f_k] = \sum_{i=1}^{N} \sum_{c_n \neq y_n} \frac{P(y)}{1-P(c)} \frac{\Delta(I_i, M_i)}{N*k} - \frac{\Delta(I_i, H_i)}{N*k} \quad (6)$$

here, $P(y)$ and $P(c)$ are the probability of hit (same class data instances of I_i) and miss (different class data instances of I_i) class respectively. The choice of the value for k is sometimes troublesome- whether it can choose/discard unnecessary/necessary nearest neighbour(s) to meet the condition of k.

2.5 SURF

SURF defines a threshold distance value without fixing the value of k. The threshold distance (T) is determined by Eqn (7).

$$T = \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} \frac{\Delta(I_i, I_j)}{N*N} \quad (7)$$

T is same for all the instances and it is not practical that having the same T for all instances would be appropriate. MultiSURF defines the appropriate threshold distance value for each data instances that has been shown in proposed method section. The previously mentioned methods: Relief, ReliefF and SURF are feature ranking methods, and deciding the appropriate number of feature is a major concern to achieve better classification performance. To address this issue, we propose a distance-based feature selection approach for software anomaly detection (ADSD).

3 PROPOSED METHOD

Our proposed ADSD method encompasses two steps: ranking of features based on performance and a subset selection process. In this section, we briefly describe the steps for ADSD.

3.1 Ranking of Features

Individual feature performance is measured using the scoring mechanism of MultiSURF. As mentioned before, MultiSURF takes instances inside a threshold distance. For calculating threshold distance of i^{th} data instance (I_i), first, we compute a distance array D_i from the taken instance (I_i) to all the remaining instances (I_j) in Eqn (8).

$$D_i = \Delta(I_i, I_j) \quad (8)$$

To avoid the confusing instances, the dead-band zone is defined from the standard deviation of D_i, i.e., $\sigma(D_i)$. Finally, the threshold distance T_i of I_i instance is computed using Eqn (9).

$$T_i = D_i - \sigma(D_i) \quad (9)$$

Algorithm 1: Ranking the features.

```
Input: Instances $I = \{I_1, I_2, \cdots, I_N\}$ of dataset
  $D$ and features, $F=\{f_1, f_2, \cdots, f_m\}$
Output:Rank of features $(R)$, hit $(H_i)$ and
  miss $(M_i)$ instances for all instances

 1: Initialize all feature scores $s[f_k] := 0$
 2: for all i=1 to N do
 3:     $hit_c \leftarrow 0$ ; $miss_c \leftarrow 0$
 4:     Compute $T_i$ using Eqn (9)
 5:     for j=1 to N do
 6:         if $\Delta(I_i, I_j) < T_i$ then
 7:             if y(i)==y(j) then
 8:                 $hit_c + = 1$
 9:                 Append $j^{th}$ instance in $H_i$
10:             else
11:                 $miss_c + = 1$
12:                 Append $j^t h$ instance in $M_i$
13:             end if
14:         end if
15:         for k=1:F do
16:             $s[f_k] := s[f_k] - \frac{\Delta(f, I_i, H_i)}{N * hit_c} + \frac{\Delta(f, I_i, M_i)}{N * miss_c}$
17:         end for
18:     end for
19: end for
20: R = sort(s) in descending order
21: return R, H and M
```

The instances inside T_i are taken to measure the feature performance $(s[f_k])$ of f_k as shown in Eqn (10).

$$s[f_k] := s[f_k] - \frac{\Delta(f_k, I_i, H_i)}{N * hit_c} + \frac{\Delta(f_k, I_i, M_i)}{N * miss_c} \quad (10)$$

here, f_k is the k^{th} feature, (H_i, M_i) is the hit and miss instances of I_i instance inside T_i. And the same procedure is repeated for all the instances to define $s[f_k]$. The detailed process of ranking the features is given in Algorithm.1.

3.2 Feature Subset Selection

In order to find the feature subset, we first select the top-rank feature in the selected subset (S) from the ranked feature set, and the second top feature is taken as a candidate feature to measure their joint performance which is measured using combined score (S_c) defined in Eqn (11).

$$S_c = \sum_{i=1}^{N} \frac{\Delta(I_i, M_i, F_c)}{N * miss_c} - \frac{\Delta(I_i, H_i, F_c)}{N * hit_c} \quad (11)$$

here F_c is the combined feature set, S_c is the score of the combined feature set. The eligible candidate feature will increase the class separation (S_c) and thus when the value of S_c is larger than a threshold value the candidate feature is chosen to the selected subset. The detailed process of feature subset selection is provided in Algorithm.2.

Algorithm 2: Feature subset selection.

```
Input: Instances, $I = \{I_1, I_2, \cdots, I_N\}$ and
  features, $F=\{f_1, f_2, \cdots, f_k\}$, rank$(R)$,
  Hits$(H)$ and Misses$(M)$
Output: Selected subset $(S)$ where $S \subseteq F$
 1: $S \leftarrow R(1)$
 2: for k = 2: F do
 3:     $f_c = S \cup R(k)$
 4:     for i=1:N do
 5:         Compute $S_c$ using Eqn (11)
 6:     end for
 7:     if $S_c > T$ then
 8:         $S \leftarrow S \cup R(k)$
 9:         $T = S_c$
10:     end if
11: end for
12: return S
```

4 EXPERIMENTAL RESULTS

In this section, we describe the datasets, methods and metrics used in our research, present the results of comparing ADSD with the existing methods.

4.1 Dataset Description

In software defect classification, the datasets available in NASA MDP repository and PROMISE repository are used in many research papers (Menzies et al., 2006; Lessmann et al., 2008). For bug severity classification, we choose two open source projects namely, Mozilla and GCC from bugzilla sources of data[1]. The details of the datasets, such as software project type, number of instances, and number of attributes of the dataset, are given in Table 1. In bug severity classification dataset, we have used five severity labels of bugs namely Blocker, Critical, Major, Minor, Trivial in accordance with the procedure followed by (Zhang et al., 2016; Lamkanfi et al., 2010; Sharmin et al., 2017a). Moreover, as these datasets are text dataset, a Term Document Matrix (TDM) is generated which is the result of a set of fundamental NLP techniques (tokenization, stop word removal, and stemming) as described in (Sharmin et al., 2017a).

[1]https://github.com/ProgrammerCJC/SPFR

Table 1: Dataset Description.

Dataset	Software Type	# of feature	# of instance
	Defect Dataset		
ar1	Embedded Software	30	121
ar3	Embedded Software (DishWasher)	30	63
ar4	Embedded Software (Refrigerator)	30	107
ar5	Embedded Software (Washing Machine)	30	36
cm1	NASA Space Craft Instrument	22	498
jm1	Real-time predictive ground system	22	10885
kc1	Storage Management	22	2109
kc2	Storage Management for ground data	22	522
kc3	Storage management for ground data	39	458
mc1	Storage management for ground data	39	9466
mc2	Video guidance system	39	161
mw1	A zero gravity experiment related to combustion	37	403
pc1	Flight software for earth orbiting satellite	22	1109
pc2	Flight software for earth orbiting satellite	37	1585
pc3	Flight software for earth orbiting satellite	38	1125
pc4	Flight software for earth orbiting satellite	38	1399
pc5	Flight software for earth orbiting satellite	39	17001
	Bug Severity Dataset		
GCC_sum	C compiler	2422	2725
Mozilla_sum	Browser	3072	2701

4.2 Methods & Metrics

To measure the performance, we conduct 10-fold (class-wise) cross-validation for training and testing. To measure the performance of ADSD against state-of-the-art methods using two classifiers: decision tree (DT) and $k-$ nearest neighbour (kNN), and calculate three metrics: accuracy, balance (defined in Eqn .1) and F_{score}. Accuracy is a very well-known metric. Besides this, inspired from previous work (Song et al., 2010; Menzies et al., 2006) we compute balance metric. The F_{score} metric is used to measure the model performance that is formed using the selected feature subset. Therefore, it measures the selected performance in appropriate class prediction which is defined in Eqn (12).

$$F_{score} = \frac{TP}{TP + .5 * (FP + FN)} \tag{12}$$

4.3 Result Discussion

For evaluating the performance of ADSD, we report the accuracy, balance and F_{score} metric in Table 2 and Table 3 for four methods namely, SAL, MRMD, MultiSURF and ReliefF over 15 defect and 2 bug severity classification datasets. Summary of these performance metric is represented using win/tie/loss which means the number of datasets for which ADSD performs better/equally-well/worse than other compared methods. As SAL is a feature subset selection method that was basically designed for software defect prediction, we choose SAL. Further, to show the superiority of the distance-based method over the MI-based method for capturing feature interaction in higher dimensions. ReliefF and MultiSURF are the traditional well-known distance-based feature ranking methods and therefore to show the impact of feature subset selection over ranking method we choose them. The impacts of our proposed ADSD method compared to the existing methods are discussed below.

Impact of Using Classifier Independent Method: Comparing the performance metric values for SAL and ADSD in Table 2 and Table 3, it is seen that SAL's performance is not consistent in terms of *balance* and accuracy metric as it is classifier dependent. It only optimizes the balance metric, and for this reason, it does not always guarantee an increase in accuracy and F_{score}. However, ADSD method is classifier independent, and therefore its performance is not impacted by the change of classifiers. Note that, SAL is excluded in our analysis of bug severity classification dataset as it needs a considerable amount of time due to large feature dimensions.

Impact of Feature Selection: ADSD maximizes class separability during the feature selection process which results in higher classification accuracy compared to the ranking method MultiSURF. In contrast, the degraded performance of MultiSURF is due to having the similarly behaved features in the top-ranked list that do not increase their class separability as well as classification performance. To understand the impact of feature selection, the difference of accuracy of MultiSURF (similar to our

Table 2: Performance analysis using KNN classifier.

	Accuracy				Balance				F_{score}			
Dataset	**SAL**	**MSURF**	**ReliefF**	**ADSD**	**SAL**	**MSURF**	**ReliefF**	**ADSD**	**SAL**	**MSURF**	**ReliefF**	**ADSD**
ar1	91.30(20)	91.31	92.30	**92.31(1)**	29.25	29.22	29.21	**29.29**	48.34	48.23	48.54	**48.75**
ar3	71.43(24)	85.11	85.23	**85.71(2)**	19.21	29.29	29.29	**60.91**	41.67	46.15	46.15	**78.79**
ar4	54.55(23)	71.82	72.33	**72.73(2)**	31.39	29.29	43.21	**64.64**	47.62	45.00	61.18	**80.70**
cm1	**78.33(30)**	66.67	75.00	77.00(5)	38.16	16.32	31.57	**45.16**	57.81	40.00	51.71	**62.44**
jm1	75.83(21)	81.27	81.46	**81.67(2)**	**37.44**	32.97	35.29	37.23	**57.14**	50.88	56.95	57.11
kc1	83.41(20)	84.21	84.22	**84.83(3)**	**45.91**	48.78	34.85	36.1	**64.50**	67.06	63.73	63.43
kc2	**88.68(21)**	**88.68**	**88.68**	**88.68(14)**	57.43	56.44	57.43	**58.51**	82.49	81.51	**82.59**	82.35
kc3	71.43(30)	76.19	85.21	**85.71(6)**	21.85	25.36	40.97	**41.83**	41.67	43.24	65.95	**66.11**
mc1	**99.25(36)**	**99.25**	**99.25**	**99.25(2)**	**49.28**	29.29	29.29	29.21	**50.99**	49.81	49.81	49.78
mc2	71.43(31)	85.71	92.38	**92.86(27)**	46.62	55.56	**57.61**	57.57	65.00	85.42	90.51	**91.43**
mw1	78.48(27)	78.86	77.78	**86.78(4)**	23.85	41.74	38.76	**43.85**	44.90	52.97	**55.98**	44.90
pc1	85.71(32)	89.42	89.42	**89.61(9)**	25.47	37.11	37.11	**37.43**	46.15	**57.22**	**57.22**	57.15
pc2	**98.11(30)**	97.48	96.86	96.86(1)	62.90	28.40	27.96	**29.29**	48.52	49.36	49.20	**49.68**
pc3	81.42(33)	84.24	85.32	**85.84(1)**	32.61	32.11	32.11	**45.71**	52.78	51.71	51.71	**64.75**
pc5	96.88(37)	96.88	96.46	**97.24(2)**	**64.20**	59.29	44.16	48.84	64.43	72.43	65.24	**66.38**
win/tie/loss	**11/2/3**	**12/2/1**	**12/3/0**	-	**10/0/5**	**12/0/3**	**13/0/2**	-	**9/1/5**	**10/0/5**	**10/0/5**	-

Table 3: Performance analysis using DT classifier.

	Accuracy				Balance				F_{score}			
Dataset	**SAL**	**MSURF**	**ReliefF**	**ADSD**	**SAL**	**MSURF**	**ReliefF**	**ADSD**	**SAL**	**MSURF**	**ReliefF**	**ADSD**
ar1	91.21(20)	84.62	92.23	**92.31(1)**	29.23	23.85	29.25	**29.29**	**48.00**	45.83	**48.00**	**48.00**
ar3	85.71(24)	85.71	85.71	**100.0(2)**	29.29	29.29	29.29	**100.0**	46.15	46.15	46.15	**100.0**
ar4	90.91(23)	81.82	72.73	**100.0(2)**	67.61	29.29	43.21	**100.0**	77.08	45.00	61.18	**100.0**
cm1	83.33(30)	77.78	72.22	**86.11(5)**	**47.84**	33.61	29.29	38.33	52.44	53.55	46.27	**55.67**
jm1	76.77(21)	80.15	80.24	**81.67(1)**	**41.32**	34.87	29.29	29.29	40.42	53.72	44.95	**44.95**
kc1	84.36(20)	82.94	85.78	**85.89(3)**	**50.88**	44.03	42.33	40.48	**62.38**	62.94	61.94	60.04
kc2	82.79(21)	80.57	79.25	**83.02(14)**	**58.49**	71.07	38.26	53.35	**75.48**	75.15	57.29	71.14
kc3	80.95(30)	80.95	76.19	**85.71(6)**	41.83	41.83	25.36	**46.97**	61.11	61.11	43.24	**65.95**
mc1	99.46(36)	99.25	99.25	**99.83(1)**	**49.49**	29.29	29.29	29.14	40.09	49.81	49.81	**49.76**
mc2	57.14(31)	**71.43**	**71.43**	**71.43(27)**	32.34	43.43	43.43	**47.32**	53.33	57.58	57.58	**68.89**
mw1	85.19(27)	81.48	88.89	**92.59(4)**	45.04	41.74	52.20	**52.86**	62.50	58.97	72.38	**73.00**
pc1	88.31(32)	**89.61**	**89.61**	**89.61(9)**	27.33	28.29	37.1	**37.19**	46.9	47.26	57.22	**57.24**
pc2	98.74(30)	98.11	98.22	**98.74(1)**	**29.29**	28.84	**29.29**	29.24	**49.68**	49.52	**49.68**	**49.68**
pc3	85.84(33)	87.41	87.22	**87.61(1)**	**47.57**	41.82	41.82	41.82	60.23	**61.60**	**61.60**	**61.60**
pc5	97.00(37)	96.77	97.12	**97.47(2)**	**60.68**	56.64	46.69	52.51	66.47	66.82	66.60	**67.84**
win/tie/loss	**14/1/0**	**13/2/0**	**13/2/0**	**-**	**7/0/8**	**9/1/5**	**10/2/3**	**-**	**11/2/2**	**10/1/4**	**9/4/2**	**-**

ranking process) and ADSD for *mw*1 dataset is shown in Fig.1 considering both DT and kNN classifier. ADSD selects 4 features out of 37 and it's accuracy is better from the MultiSURF in both *k*NN and DT classifiers. Analyzing Fig.1, we can see that the

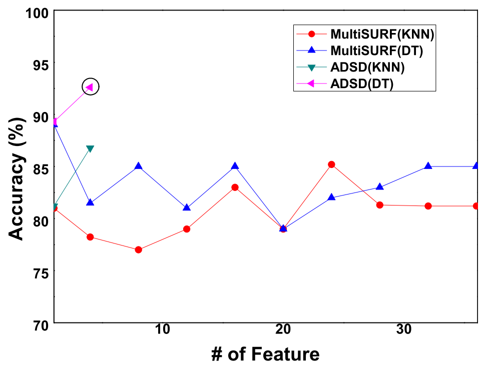
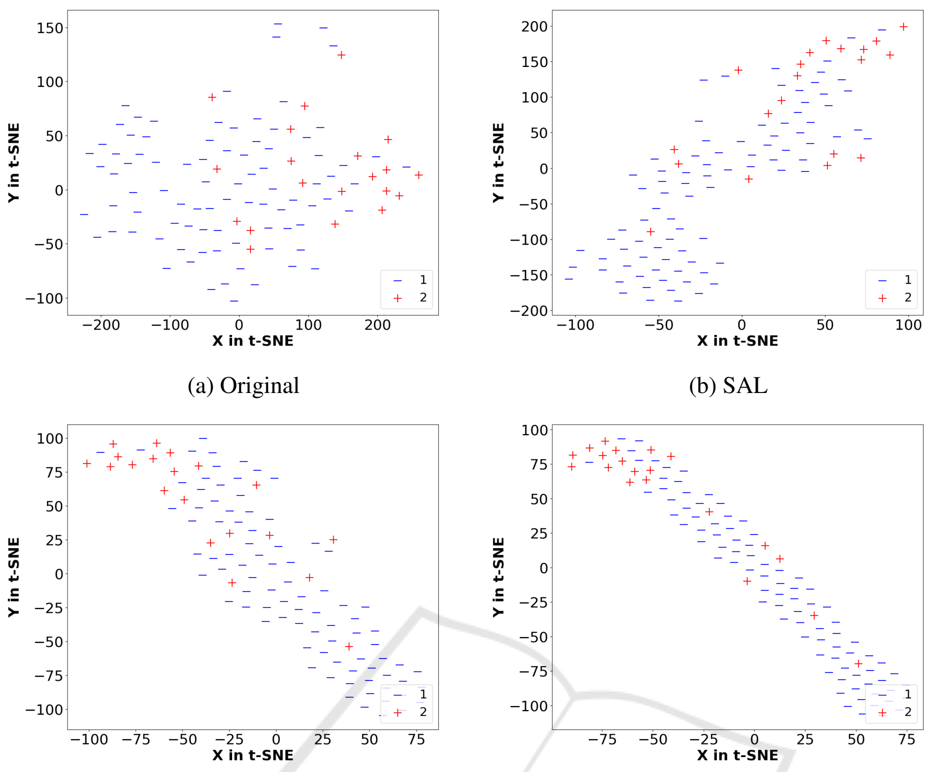


Figure 1: Comparison of MultiSURF and ADSD in terms of accuracy for mw1 dataset.

maximum accuracy is 92% (marked using a circle) achieved by ADSD with 4 selected feature using DT classifier. Though we increase the number of features for MultiSURF, it could not exceed the accuracy achieved by ADSD. Therefore, it is important to select an independent and interactive feature subset rather than increasing the number of features.

Impact of Selected Features Quality: The quality of the selected features is easily understandable from their clustering capability. In this regard, we use the tSNE visualization technique to see the quality of selected features. From Fig.2, we see that, ADSD has better clustering capability than the others, though it is not optimal.

Impact of Distance-Based Method: To understand the impact of distance-based method over MI-based method, we evaluate the performance of MRMD with three distance based methods namely ReliefF, MultiSURF and our proposed ADSD in large feature dimensional dataset such as bug severity classification performance reported in Table 4. Analyzing Table 4, we can see that the classification performance of MRMD (MI-based) method is lower compared to ADSD, ReliefF and MultiSURF (MSURF). MRMD fails to accurately approximate the interaction among the features during feature ranking, resulting in degraded performance.



(a) Original (b) SAL

(c) MultiSURF (d) ADSD

Figure 2: Clustering ability of ADSD and compared methods visualized by the tSNE plot.

Table 4: Bug severity result.

	Bug severity result			
Dataset	**MRMD**	**ReliefF**	**MultiSURF**	**ADSD**
trainGCC_sum	57.03	59.12	58.39	**59.49**(576)
trainMozilla_sum	44.45	45.16	45.5	**46.95**(870)
win/loss	**2/0**	**2/0**	**2/0**	**0**

5 THREATS TO VALIDITY

This section discusses the potential threats that may degrade the acceptability of ADSD.

5.1 Internal validity

As the choice of classifiers has an impact on performance, to evaluate the performance of ADSD, we only use SVM and DT classifiers to estimate the accuracy, balance, and F-score performance metric. Therefore, change in classifiers might differ from the reported performance of ADSD and other compared methods.

5.2 External Validity

The datasets used for the experimentation are widely used in existing software defect and bug severity classification research work. However, we cannot provide guarantee that ADSD will be fitted to other software defect/bug data in case of the presence of noise in data. Sometimes, we may have to relax the feature selection threshold value to get the desired performance in new software bug/defect data.

5.3 Construct Validity

In this proposal, we use F_{score} measure to evaluate the performance of the model and there are other metrics like AUC and g-measure which could be considered for this evaluation.

6 CONCLUSION

In this paper, we a feature selection method namely Anomaly Detection for Software Data (ADSD). To get a reliable conclusion, we compared the performance of conventional approaches with ADSD on 15 benchmark software defect datasets and two bug severity classification datasets using three metrics- accuracy, balance and F_{score}. The combined feature performance maximization property of ADSD helped to achieve the highest accuracy among its contenders. However, the presence of outliers and imbalance problems in data might result in the degraded performance of ADSD. Moreover, when the number of samples grows very high ADSD cannot deal with it due to the huge computational complexity of kNN. To address this, we can take advantage of the sampling method which will be addressed in future.

ACKNOWLEDGEMENTS

This research is supported by the fellowship from ICT Division, Ministry of Posts, Telecommunications and Information Technology, Bangladesh. No - 56.00.0000.028.33.006.20-84; Dated 13.04.2021.

REFERENCES

Agarwal, S., Tomar, D., et al. (2014). Prediction of software defects using twin support vector machine. In *2014 International Conference on Information Systems and Computer Networks (ISCON)*, pages 128–132. IEEE.

Akhter, S., Sharmin, S., Ahmed, S., Sajib, A. A., and Shoyaib, M. (2021). mrelief: A reward penalty based feature subset selection considering data overlapping problem. In *Computational Science–ICCS 2021: 21st International Conference, Krakow, Poland, June 16–18, 2021, Proceedings, Part I*, pages 278–292. Springer.

Akintola, A. G., Balogun, A. O., Lafenwa-Balogun, F. B., and Mojeed, H. A. (2018). Comparative analysis of selected heterogeneous classifiers for software defects prediction using filter-based feature selection methods. *FUOYE Journal of Engineering and Technology*, 3(1):134–137.

Erlikh, L. (2000). Leveraging legacy system dollars for e-business. *IT professional*, 2(3):17–23.

Gao, W., Hu, L., and Zhang, P. (2020). Feature redundancy term variation for mutual information-based feature selection. *Applied Intelligence*, 50(4):1272–1288.

Goh, L., Song, Q., and Kasabov, N. (2004). A novel feature selection method to improve classification of gene expression data. In *Proceedings of the second conference on Asia-Pacific bioinformatics-Volume 29*, pages 161–166. Australian Computer Society, Inc.

Greene, C. S., Penrod, N. M., Kiralis, J., and Moore, J. H. (2009a). Spatially uniform relieff (surf) for computationally-efficient filtering of gene-gene interactions. *BioData mining*, 2(1):1–9.

Greene, C. S., Penrod, N. M., Kiralis, J., and Moore, J. H. (2009b). Spatially uniform relieff (surf) for computationally-efficient filtering of gene-gene interactions. *BioData mining*, 2(1):1–9.

Ibrahim, D. R., Ghnemat, R., and Hudaib, A. (2017). Software defect prediction using feature selection and random forest algorithm. In *2017 International Conference on New Trends in Computing Sciences (ICTCS)*, pages 252–257. IEEE.

Igor, K. (1994). Estimating attributes: analysis and extensions of relief. *European conference on machine learning. Springer, Berlin, Heidelberg*.

Jimoh, R., Balogun, A., Bajeh, A., and Ajayi, S. (2018). A promethee based evaluation of software defect predictors. *Journal of Computer Science and Its Application*, 25(1):106–119.

Jo, I., Lee, S., and Oh, S. (2019). Improved measures of redundancy and relevance for mrmr feature selection. *Computers*, 8(2):42.

Khan, M. S. H., Roy, P., Khanam, F., Hera, F. H., and Das, A. K. (2019). An efficient resource allocation mechanism for time-sensitive data in dew computing. In *2019 International Conference of Artificial Intelligence and Information Technology (ICAIIT)*, pages 506–510. IEEE.

Kira, Kenji, and Rendell, L. A. (1992). The feature selection problem: traditional method and a new algorithm. *AAAI*, 2:129–134.

Lamkanfi, A., Demeyer, S., Giger, E., and Goethals, B. (2010). Predicting the severity of a reported bug. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 1–10. IEEE.

Laradji, I. H., Alshayeb, M., and Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. *Information and Software Technology*, 58:388–402.

Lessmann, S., Baesens, B., Mues, C., and Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496.

Lewis, D. D. (1992). Feature selection and feature extraction for text categorization. In *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*.

Menzies, T., Greenwald, J., and Frank, A. (2006). Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering*, 33(1):2–13.

Moody, J. and Yang, H. (1999). Data visualization and feature selection: New algorithms for nongaussian data. *Advances in neural information processing systems*, 12:687–693.

Nguyen Xuan Vinh, Shuo Zhou, J. C. and Bailey, J. (2016). Can high-order dependencies improve mutual information based feature selection? *Pattern Recognition*, 53:46—-58.

Peng, H., Long, F., and Ding, C. (2005). Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 27(8):1226–1238.

Pisheh, F. and Vilalta, R. (2019). Filter-based information-theoretic feature selection. *Proceedings of the 2019 3rd International Conference on Advances in Artificial Intelligence*.

Ruiz, R., Riquelme, J. C., Aguilar-Ruiz, J. S., and García-Torres, M. (2012). Fast feature selection aimed at high-dimensional data via hybrid-sequential-ranked searches. *Expert Systems with Applications*, 39(12):11094–11102.

Senawi, A., Wei, H.-L., and Billings, S. A. (2017). A new maximum relevance-minimum multicollinearity (mrmmc) method for feature selection and ranking. *Pattern Recognition*, 67:47–61.

Sharmin, S., Aktar, F., Ali, A. A., Khan, M. A. H., and Shoyaib, M. (2017a). Bfsp: A feature selection method for bug severity classification. In *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 750–754. IEEE.

Sharmin, S., Ali, A. A., Khan, M. A. H., and Shoyaib, M. (2017b). Feature selection and discretization based on mutual information. In *2017 IEEE International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, pages 1–6. IEEE.

Sharmin, S., Arefin, M. R., Abdullah-Al Wadud, M., Nower, N., and Shoyaib, M. (2015). Sal: An effective method for software defect prediction. In *2015 18th International Conference on Computer and Information Technology (ICCIT)*, pages 184–189. IEEE.

Sharmin, S., Shoyaib, M., Ali, A. A., Khan, M. A. H., and Chae, O. (2019). Simultaneous feature selection and discretization based on mutual information. *Pattern Recognition*, 91:162–174.

Song, Q., Jia, Z., Shepperd, M., Ying, S., and Liu, J. (2010). A general software defect-proneness prediction framework. *IEEE transactions on software engineering*, 37(3):356–370.

Sun, Y. (2007). Iterative relief for feature weighting: algorithms, theories, and applications. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1035–1051.

Tarek, M. H., Kadir, M. E., Sharmin, S., Sajib, A. A., Ali, A. A., and Shoyaib, M. (2021). Feature subset selection based on redundancy maximized clusters. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 521–526. IEEE.

Urbanowicz R. J. Olson, R. S. Schmitt, P. M. M. M. and H., J. (2018). Multiple threshold spatially uniform relieff for the genetic analysis of complex human diseases. *Benchmarking relief-based feature selection methods for bioinformatics data mining. Journal of biomedical informatics*, 85:168–188.

Vinh, N. X., Zhou, S., Chan, J., and Bailey, J. (2016). Can high-order dependencies improve mutual information based feature selection? *Pattern Recognition*, 53:46–58.

Wang, H., Khoshgoftaar, T. M., and Napolitano, A. (2012). Software measurement data reduction using ensemble techniques. *Neurocomputing*, 92:124–132.

Yang, S. H. and Hu, B.-G. (2012). Discriminative feature selection by nonparametric bayes error minimization. *IEEE Transactions on knowledge and data engineering*, 24(8):1422–1434.

Zhang, T., Chen, J., Yang, G., Lee, B., and Luo, X. (2016). Towards more accurate severity prediction and fixer recommendation of software bugs. *Journal of Systems and Software*, 117:166–184.