Proceedings of the Second International Conference on Intelligent Computing and Control Systems (ICICCS 2018)
IEEE Xplore Compliant Part Number: CFP18K74-ART; ISBN:978-1-5386-2842-3

A comparative study of unsupervised learning algorithms for software fault prediction

R. Jothi
Department of Computer Engineering,
School of Technology,
Pandit Deendayal Petroleum University, Gandhinagar, India
Email: jothi.r@sot.pdpu.ac.in

***Abstract*—Software fault prediction is an important task in software development process which enables software practitioners to easily detect and rectify the errors in modules or classes. Various fault prediction techniques have been studied in the past and unsupervised learning methods such as clustering techniques are drawing much attention in the recent years. K-means is a well known clustering algorithm which is applied on various exploratory analysis including software fault prediction. This paper provides a comparative study on software fault prediction using K-means clustering algorithm and its variants. We use five software fault prediction datasets taken from PROMISE repository to evaluate the prediction accuracy of the clustering algorithms. Experimental results indicate that proper initial seed selection enables K-means algorithm to effectively group the faulty modules.**

***Keywords*—Software fault prediction;Unsupervised learning;K-means clustering; K-means++.**

I. Introduction

With rapid increase in the complexity of software products, quality assurance has become a pivotal activity in the software development process. Determining fault-proneness of software modules during earlier stages of software development process saves cost and time. In order to identify the fault prone modules, software practitioners build fault prediction models using software metrics such as lines of code (LOC), cyclomatic complexity, etc. There are various techniques for software fault prediction such as discriminant analysis [1], logistic regression [2], artificial neural networks [3], etc. A systematic review on fault prediction techniques can be seen in [4], [5].

Machine learning techniques such as supervised and unsupervised learning methods have been widely used for fault prediction. While supervised methods use labeled data for fault analysis, unsupervised methods predict the faulty modules in the absence of fault labels [5], [6], [7]. Clustering is an unsupervised learning method and finds its application in various exploratory analysis including software fault prediction. Goal of clustering is to discover meaningful patterns inherent in a dataset without any prior knowledge about the data items [8]. Different clustering methods such as fuzzy clustering [9], Self Organizing Map (SOM) [10] and K-means clustering [7] have been employed for software fault prediction.

K-means clustering is rich in literature [11], [12], [13]. Being a partitional clustering algorithm, K-means partitions a set of items into k clusters such that the intra-cluster separation minimized. Due to its simple implementation, K-means has been extensively used in software fault prediction [6]. Another well-known partitioning based clustering algorithm that is also widely applied in software fault analysis is Fuzzy C-means (FCM). While K-means algorithm tries to obtain hard partitioning, where each item belongs to only one cluster, FCM algorithm obtains a soft partitioning, where each item may belong to more than one cluster. Core of the FCM is to obtain a cluster membership matrix M, where M_{ij} is the degree to which an item x_i forms closeness with a cluster S_j [22]. Item x_i will be assigned to the cluster S_j for which membership value M_{ij} is maximum.

The fault prediction accuracy of K-means based defect clustering depends on the way the initial centers are chosen. K-means algorithm easily gets trapped with local optimum if the initial centers are randomly chosen [11]. Moreover K-means is highly sensitive to noise and outliers. Despite its inherent drawbacks, K-means clustering still remains as a popular method for software fault prediction due to its simplicity and high performance. It is widely used as a bench-marking algorithm to compare the results of other clustering techniques. This motivated us to carry out a comparative study of software fault prediction using K-means algorithm and its variants. We compare the results K-means with few of its variants such as Fuzzy c-means (FCM), K-means++ [14] and Quad Tree K-means (QDK) [6]. Performance of these algorithms have been demonstrated on five software fault prediction datasets taken from NASA PROMISE repository.

Remainder of the paper is organized as follows. In Section II, a brief description of K-means algorithm and its variants are briefly described. Section III presents an overview of K-means based fault prediction methods. Experimental setup for the present study is described in IV. The results of experimental validation are reported and discussed in Section V, and finally the conclusion is given in VI.

II. K-means algorithm

Let $X = \{x_1, x_2, \cdots, x_n\}$ be the set of items to be clustered into k groups, where k is the number of classes of items. Core of K-means algorithm is to minimize the intra-cluster distance which is defined through sum of squared error (SSE) criterion as follows [11].

Algorithm 1: K-means Algorithm [11]

Input: Dataset X and the number of clusters k.
Output: k clusters of X.

1) Randomly select k data items from the dataset as initial cluster centers for the set of partitions $S = \{S_1, S_2, \cdots\}$, where the center of the partition S_i is denoted as μ_i, $1 \leq i \leq k$.
2) For each data item x_j in X, compute distance between x_j and μ_i, $1 \leq i \leq k$.
3) Assign x_j to the partition S_i, such that the distance between x_j and μ_i is minimum.
4) Recompute centers; repeat steps 2 and 3 if there is change in centers. Else stop.

$$SSE = \sum_{i=1}^{k} \sum_{x_j \epsilon s_i} d(x_j, \mu_i)^2. \quad (1)$$

where $d(x_j, \mu_i)$ denotes the distance between the data item x_j and the cluster center μ_i. K-means algorithm starts with k arbitrary centers and iteratively recomputes the centers and reassign the items to the nearest centers. If there is no change in centers, then the algorithm stops. The steps involved in K-means algorithm is described in Algorithm 1.

As the initial centers for K-means partition are chosen randomly, two or more centers may collide in a nearby region. As a consequence, the items are forced to be assigned to one of the nearest centers, leading to poor results. This is illustrated in Fig. 1. It is clear from the figure that, K-means gets trapped with local optimum if the initial centers are not chosen properly. Numerous researches have taken place to devise initialization methods for K-means, to name a few [14], [15], [13].

III. Related work

K-means clustering algorithm has been widely used for software fault prediction. Zhong et al. [7] applied K-Means and Neural-Gas techniques for predicting the fault proneness of software modules. The prediction results are then refined by a domain expert to classify the modules as either fault-prone or non fault-prone using certain statistical data. Experimental study on real datasets showed that K-means was much faster than Neural-Gas technique but with reduced accuracy as compared to Neural-Gas technique. A constrained based semi-supervised clustering scheme using K-means was proposed by Seliya and Khoshgoftaar for software fault prediction [16]. Their approach uses expert's domain knowledge as a constraint to iteratively label clusters as fault-prone or not. Fuzzy version of K-means clustering was used by [9] for software fault prediction.

The accuracy of defect prediction using K-means algorithm depends on the way the initial centers are chosen. Recently, a number of initialization approaches have been proposed to overcome the drawbacks of K-means algorithm. To name a few, sampling-based seed selection [17], global k-means [18], principal dimension analysis [19] [20], K-means++ [14]. A comparative study of K-means initialization methods can be seen in [15].

K-means++ is one of popular variants of K-means. K-means++ chooses the first center c_1 randomly from the dataset and other centers c_i, $2 \leq i \leq k$ are chosen such that distance between c_i and the previously chosen centers is maximum. Muhammed Maruf et al. [21] presented a case study on the use of K-means++ algorithm for estimating the defects in a webpage source code. According to the experimental analysis, the accuracy of K-means++ for detecting the defects in webpages was significantly better. Bishnu and Bhattacherjee[6] applied quad tree-based K-means algorithm for software fault prediction. Quad tree on d-dimensional space is a 2^d-way branching tree which is built by recursive decomposition of space using separators parallel to the coordinate axes. With the use of quad tree, they identify the initial cluster centers for K-means algorithm. Experimental results showed that clusters obtained by this algorithm achieved maximum gain values and reduced error rates. However, the performance of the algorithm relies on the parameters such as user defined threshold for minimum and maximum number of data points (MIN, MAX) and distance threshold for nearest neighbors δ.

While there are numerous cluster center initialization approaches exist to tackle the local optimum problem of K-means, their application to software fault prediction is not studied in depth. This paper provides a comparative study of software fault analysis using K-means, K-means++ [14] and Quad Tree k-means (QDK) [6].

IV. Experimental Setup

We consider classical K-means algorithm and its variants namely K-means++ [14] and QDK [6] for the task of software fault prediction. For the Quad Tree-based algorithm, we set the input parameters as follows: the minimum number of data points in a node $MIN = 0.05$, maximum number of points in a node $MAX = 0.95$, nearest neighbor distance $\delta = 60$.

We also compare the results of above said algorithm with Fuzzy C-means algorithm (FCM).

A. Datasets

For experimental analysis we consider five software fault prediction datasets (AR1, AR3, AR4, AR5 and AR6) taken from online public repository PROMISE (http://www.promisedata.org/). The number of instances in the datasets in AR1, AR3, AR4, AR5 and AR6 are 121, 23, 107,36 and 109 respectively. These datasets are embedded software products implemented in C language. Each dataset contains the measurements of 29 static code attributes (complexity metrics) and 1 defect information (false/true).

B. Evaluation Parameters

The results of the clustering is evaluated with the performance measures false positive rate (FPR), false negative

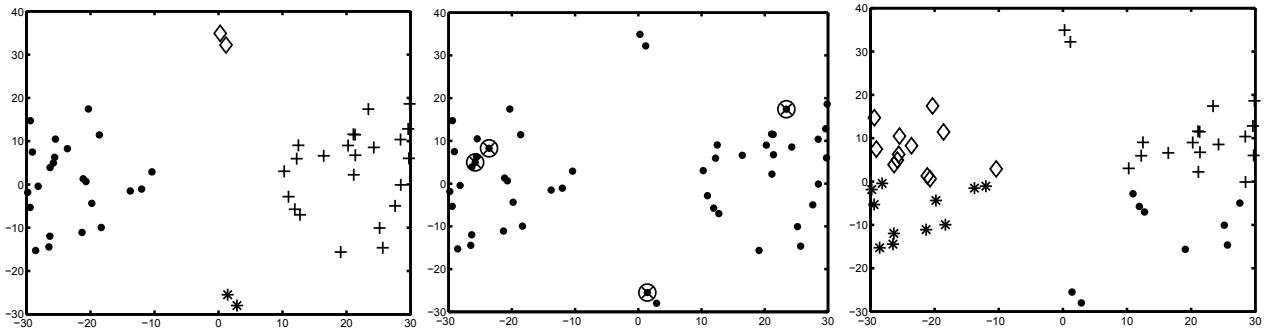

Fig. 1: K-means converging to a local optimum with random center initialization: (a) A given dataset with four clusters. (b) Randomly chosen initial centers. (c) The result of K-means with centers chosen in (b), K-means converges in 6 iterations with $SSE = 288.2418$.

rate (FNR) and error rate. False positive rate is the percentage of non-faulty modules misclassified as fault-prone. Similarly false negative rate is the percentage of faulty modules misclassified as non-faulty one. Error rate is the percentage of misclassified modules. The confusion matrix is constructed from the actual and predicted classifications done by a classification system. The matrix is formed by recording the values True positive (FP), false positive (FP), false negative (FN) and true negative (TN) as shown in Table I. The measures FPR, FNR and Error rates are computed from the confusion matrix as follows [23].

TABLE I: Confusion matrix.

Predicted	Actual		
		Yes	No
	Yes	TP	FP
	No	FN	TN

$$FPR = \frac{FP}{FP + TN} \tag{2}$$

$$FNR = \frac{FN}{FN + TP} \tag{3}$$

$$Error = \frac{FN + FP}{TP + FP + FN + TN} \tag{4}$$

The prediction accuracy of the algorithms are evaluated using FPR, FNR and Error rates and the lower values of error rates indicate good classification ratio [23].

V. Results

A. Fault Prediction Accuracy Analysis

We run all the algorithms for 10 times and record the average value of FPR, FNR and Error rates. Table II show the values of FPR obtained by different algorithms. On an average, K-means produces 19.53 % false positive alarms. As compared other algorithms, it attains slightly reduced false positive rate. Table II show the rate false negative values

TABLE II: Comparison of Fault Positive Rate (FPR).

Dataset	K-means %	K-means++ %	QDK %	FCM %
AR1	28.93	34.82	27.75	31.55
AR3	17.27	29.09	34.54	18.95
AR4	11.15	14.94	4.59	15.65
AR5	12.86	14.29	14.28	22.55
AR6	27.45	14.49	25.00	20.34
Avg	**19.53**	**21.53**	**21.23**	**21.81**

TABLE III: Comparison of Fault Negative Rate (FNR).

Dataset	K-means %	K-means++ %	QDK %	FCM %
AR1	55.55	65.55	65.55	50.00
AR3	62.50	25.00	25.00	45.95
AR4	95.00	65.00	45.00	85.00
AR5	25.00	20.00	12.50	28.55
AR6	86.23	56.67	50.55	60.75
Avg	**64.86**	**46.44**	**39.72**	**54.05**

generated by different algorithms. It is observed here that QDK produces lower rate of false negative alarms with average FNR rate of 39.72%. Table IV shows the total error rate of different algorithms. It can be seen from the table that, the average error rate of the K-means algorithm is quite low as compared to others.

Probability of defect prediction (PD) is defined as the probability of correct classification of a module that contains a fault. Fig. 2 show the comparison of algorithms with respect to probability of detection. The average defect prediction probability of the algorithms as follows: K-means (52.2%), K-means++ (34.6%), QDK (58.6%), FCM (43.01%). It is observed from the figure that QDK performs fairly well in predicting the faults.

B. Efficiency Analysis

In order to analyze the efficiency, we measure the number of iterations (I) and run time taken by different algorithms. We also record the sum of squared error SSE of the algorithms. Table V shows these results. It is observed that QDK takes fewer iterations than K-means and K-means++.

TABLE IV: Comparison of Total Error Rate.

Dataset	K-means %	K-means++ %	QDK %	FCM %
AR1	30.91	36.36	27.75	27.45
AR3	9.52	33.33	33.33	15.35
AR4	17.85	29.91	12.14	30.84
AR5	14.44	16.67	13.88	21.25
AR6	31.56	25.32	30.00	20.65
Avg	**20.85**	**28.32**	**23.42**	**23.11**

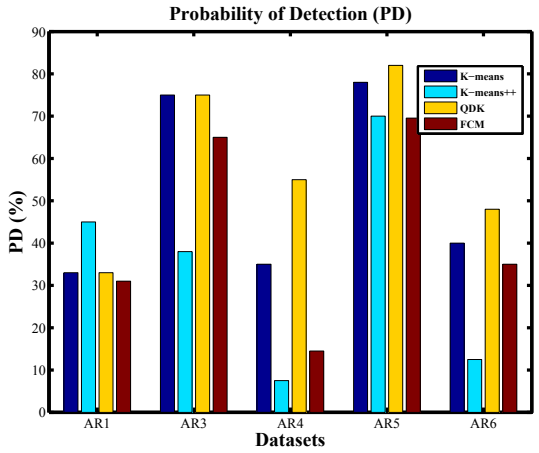

Fig. 2: Comparison of algorithms with respect to probability of detection

TABLE V: Comparison of Number of iterations (I), Runtime and SSE.

Dataset	Parameter	QDK	Kmeans	Kmeans++
	I	6	9	8
AR1	SSE	424.18	443.67	410.14
	Time (ms)	5.1	7.0	10.9
	I	4	7	7
AR3	SSE	221.99	221.99	204.33
	Time (ms)	5.7	8.4	5.8
	I	4	9	7
AR4	SSE	383.33	385.10	381.61
	Time (ms)	9.2	113.1	10.1
	I	5	4	4
AR5	SSE	134.45	123.76	111.82
	Time (ms)	4.8	5.6	4.1
	I	7	8	6
AR6	SSE	408.74	352.88	235.86
	Time (ms)	6.9	7.1	9.1

VI. Conclusion

K-means clustering has been a popular method for analyzing software fault prediction. In this paper, we have carried out a comparative study of software fault prediction using clustering algorithms namely K-means and its variants such as K-means++, QuadTree K-means (QDK) and FCM. Experiments were carried out on fault prediction datasets taken from PROMISE repository. Results of the algorithms were evaluated using the parameters such as false positive rate, false negative rate, total error rate and probability of defect prediction. QDK algorithm has shown a fairly better performance as compared to K-means, K-means++ and FCM. It is important to note here that the choice of parameters in QDK algorithm may influence the algorithm in getting global optimum solution. K-means++ still inherits the drawback of unstable partitioning results as in K-means, due to the random choice of the first center point. Thus the performance of K-means++ is not improved as compared to K-means. It is also worthwhile to note that K-means also performed equally well in terms of prediction rate. If the chance of getting trapped with local optimum is reduced by a careful selection of initial centers, K-means produces better clustering accuracy with improved computational performance. As a future work, we will carry out an extensive analysis of different K-means variants for the task of fault prediction.

References

[1] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, and N. Goel, "Early quality prediction: A case study in telecommunications," *IEEE Software*, vol. 13, no. 1, p. 65, 1996.

[2] L. C. Briand, V. Brasili, and C. J. Hetmanski, "Developing interpretable models with optimized set reduction for identifying high-risk software components," *IEEE Transactions on Software Engineering*, vol. 19, no. 11, pp. 1028–1044, 1993.

[3] T. M. Khoshgoftaar, A. S. Pandya, and D. L. Lanning, "Application of neural networks for predicting program faults," *Annals of Software Engineering*, vol. 1, no. 1, pp. 141–154, 1995.

[4] C. Catal, "Software fault prediction: A literature review and current trends," *Expert systems with applications*, vol. 38, no. 4, pp. 4626–4636, 2011.

[5] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[6] P. S. Bishnu and V. Bhattacherjee, "Software fault prediction using quad tree-based k-means clustering algorithm," *IEEE Transactions on knowledge and data engineering*, vol. 24, no. 6, pp. 1146–1150, 2012.

[7] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Analyzing software measurement data with clustering techniques," *IEEE Intelligent Systems*, vol. 19, no. 2, pp. 20–27, 2004.

[8] R. Jothi, S. K. Mohanty, and A. Ojha, "Fast approximate minimum spanning tree based clustering algorithm," *Neurocomputing*, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092523121731295X

[9] X. Yuan, T. M. Khoshgoftaar, E. B. Allen, and K. Ganesan, "An application of fuzzy clustering to software quality prediction," in *Application-Specific Systems and Software Engineering Technology, 2000. Proceedings. 3rd IEEE Symposium on*. IEEE, 2000, pp. 85–90.

[10] A. Mahaweerawat, P. Sophatsathit, and C. Lursinsap, "Adaptive self-organizing map clustering for software fault prediction," in *Fourth International Joint Conference on Computer Science and Software Engineering, KhonKaen, Thailand*. Citeseer, 2007, pp. 35–41.

[11] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM computing surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[12] R. Xu and D. C. Wunsch, "Clustering algorithms in biomedical research: a review," *IEEE Reviews in Biomedical Engineering*, vol. 3, pp. 120–154, 2010.

[13] R. Jothi, S. K. Mohanty, and A. Ojha, "Dk-means: a deterministic k-means clustering algorithm for gene expression analysis," *Pattern Analysis and Applications*, Dec 2017. [Online]. Available: https://doi.org/10.1007/s10044-017-0673-0

[14] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[15] M. E. Celebi, H. A. Kingravi, and P. A. Vela, "A comparative study of efficient initialization methods for the k-means clustering algorithm," *Expert Systems with Applications*, vol. 40, no. 1, pp. 200–210, 2013.

[16] N. Seliya and T. M. Khoshgoftaar, "Software quality analysis of unlabeled program modules with semisupervised clustering," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 37, no. 2, pp. 201–211, 2007.
[17] P. S. Bradley and U. M. Fayyad, "Refining initial points for k-means clustering." in *Proceedings of 15th International Conference on Machine Learning (ICML)*, vol. 98, 1998, pp. 91–99.
[18] A. Likas, N. Vlassis, and J. J Verbeek, "The global k-means clustering algorithm," *Pattern Recognition*, vol. 36, no. 2, pp. 451–461, 2003.
[19] M. Erisoglu, N. Calis, and S. Sakallioglu, "A new algorithm for initial cluster centers in k-means algorithm." *Pattern Recognition Letters*, vol. 32, no. 14, pp. 1701 – 1705, 2011.
[20] S. Ting and D. Jennifer G., "In search of deterministic methods for initializing k-means and gaussian mixture clustering," *Intelligent Data Analysis*, vol. 11, no. 4, pp. 319–338, 2007.
[21] M. M. Öztürk, U. Cavusoglu, and A. Zengin, "A novel defect prediction method for web pages using k-means++," *Expert Systems with Applications*, vol. 42, no. 19, pp. 6496–6506, 2015.
[22] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. Morgan kaufmann, 2006.
[23] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE transactions on software engineering*, vol. 33, no. 1, pp. 2–13, 2007.