# Software Fault Prediction using Machine Learning Algorithm

**Pooja Garg[1] Mr. Bhushan Dua[2]**
[1,2]Department of Computer Science and Engineering
[1,2]DIET, Karnal, Haryana, India

*Abstract*— Software fault prediction indicates the likelihood of software fault at an early stage of software development process and hence it will be easier to identify and correct them and also reduce faults that would occur at later stages. This will improve the overall quality of the software product. In the recent years, several machine learning techniques which uses examples of faulty and non-faulty modules to build prediction models. Software metric have been used as input to these machine learning techniques to represent the software modules. Support Vector Machines (SVM) is the main algorithm which has been used for classification of faulty and non-faulty modules. But prior to using SVM, A few data pre-processing methods has been proposed such as bootstrapping, clustering and random projection. These methods are needed because of certain problems with metric datasets such as class imbalance, noise, small dataset size and high dimension. It has showed by experiments that when software metrics dataset is pre-processed and transformed using above techniques, the performance of SVM in predicting faulty and non-faulty modules is better. The accuracy measure used for comparing performances of different models was F-measure. F-measure has been used because of its robustness to class imbalance. For some models F-measure has increased by about 40%, which is very encouraging. Experimental results shows that the proposed approach worked better than existing approach using MATLAB and Weka programming.

***Key words:*** Support Vector Machines (SVM) Bootstrapping, Clustering, Random Projection, Software Metrics, Metrics datasets

## I. INTRODUCTION

### A. *Software Metrics:*

Software metrics are various measurements of software. Measurements are useful because they are repeatable which means same measurement value will be reported by different persons for a particular object. Metrics are being increasingly used for quantitative and qualitative analysis of software. They give software professionals the ability to evaluate software process. One of the simplest and earliest metric is Lines of Code (LOC). LOC as the name implies means number of text lines in a software program. It gives a good measure of size of software. LOC metrics can be different depending on what they count. Other metrics are Halstead metrics, McCabe's metric suit, Node Count, Condition count, Edge count, Branch count, Call pairs, Error count Etc.

### B. *Metrics Datasets:*

A major challenge in developing software quality prediction models is to find sources of metrics data. Datasets which contains these metrics and fault information are not available widely. Although there are many organizations where this kind of data is available internally for its management, but these are not publicly available. NASA MDP datasets are one such publicly available datasets which is widely used in software quality modelling.

### C. *NASA MDP Metrics Datasets*

NASA MDP datasets are available from a total of 12 different NASA projects (Promise Data, 2012).Cleaned versions of datasets denoted by D' are used (Shepperd et al., 2013). Dataset from each project is given a name. The name, description and programming language for each of these 12 datasets is given below (Jianget al., 2008):

- JM1: Real-time predictive ground system written in C.
- CM1: NASA spacecraft instrument written in C.
- KC1: Storage management for receiving and processing ground data written in C++.
- PC1: Flight software for earth orbiting satellite written in C.
- KC3: Storage management for ground data written in Java.
- MW1: Zero gravity experiment related to combustion written in C.
- MC2: Video guidance system written in C.
- MC1: Combustion experiment of a space shuttle written in both C and C++.
- PC2: Dynamic simulator for attitude control systems written in C.
- PC3: Flight software for earth orbiting satellite written in C.
- PC4: Flight software for earth orbiting satellite written in C.
- PC5: Flight software for earth orbiting satellite written in C.

### D. *Support Vector Machines:*

Support Vector Machines (SVM) is the one of the most widely used supervised machine learning algorithm (Hsuet al., 2003). SVM tries to map the data so that data of different class is separated by a hyper plane that has maximum distance from nearest training data point of all classes. SVM always chooses a hyper plane that maximizes this margin. It is mathematically proved that by maximizing the margin of separation on training data, it reduces the complexity of SVM and gives better performance on unseen data. The optimized hyper plane is the one that minimize straining error and has maximum margin of separation.

The basis of SVM is linear discriminant functions. SVM classification function like linear discriminant functions is of form $w^t x + b$. Parameters to be learnt are w and b. If there are only 2 classes and patterns are linearly separable then in linear discriminant functions, function that is learn is such that $w^t x + b > 0$ for patterns of one class and $w^t x + b < 0$ for patterns of other class. Parameters have to be learned in such a way that above inequalities is followed for all training examples in case of linearly separable case. Now, consider two hyperplane at equal margins from$w^t x + b = 0$; $w^t x + b = 1$and $w^t x + b = -1$. These planes are chosen such that training

examples that are nearest to $w^t x + b = 0$ hyperplane falls on above 2 hyperplanes only (Duda et al., 2012). Figure 4 gives a clear illustration how hyperplane will look like in 2-d plane.

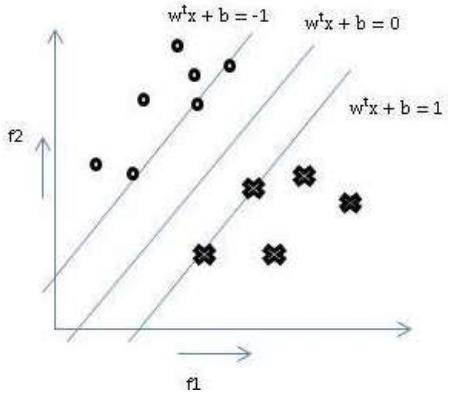



Fig. 1: Support Lines Characterizing the Margin

Figure 1 shows training patterns from two classes and labeled as 'O' and 'X' respectively. The two lines on either side of middle line are called as support hyperplanes (in this case of 2-d data, hyperplane is a line). The distance between hyperplane $w^t x + b = 1$ and decision boundary $w^t x + b = 0$ is given by:

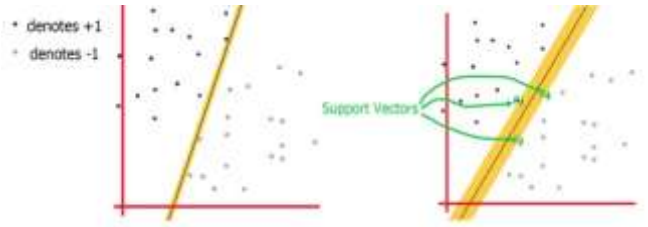
$$\frac{f(x)}{||X||} = \frac{1}{||w||} \text{where } w^t x + b = 1 \text{ , } f(x) = 1.$$

Similarly distance between hyperplane $w^t x + b = -1$ and decision boundary$w^t x + b = 0$ is also$\frac{1}{||w||}$. So total distance between the 2 supporting hyperplanes is$\frac{2}{||w||}$. SVM algorithm is nothing but to maximize this distance i.e. maximizing the margin between supporting hyperplanes. In other words, task is to find a w that minimizes$\frac{||w||}{2}$. For simplicity in calculation, $\frac{||w||}{2}$ is replaced by $\frac{||w||^2}{2}$. Constraint is that all 'X' class examples should satisfy $w^t x + b \geq 1$condition and all 'O' class examples should satisfy $w^t x + b \leq -1$. In other words, minimum margin of all examples should be at least $\frac{1}{||w||}$ from decision boundary. Lets $y_i$denotes the output lable where $y_i = 1$ for all 'X' class examples and $y_i = -1$ for all 'O' class examples and m denoted number of examples. Then the SVM problem can be summarized as follows:

$$\text{Minimize } \frac{||w||^2}{2} \qquad (1)$$

Subject to: $y_i(w^t x + b) \geq 1; i = 1,2, \ldots\ldots, m$

SVM needs only those examples that fall on supporting hyperplanes to learn the function. These training examples are called as supporting vectors. These examples (or vectors) are sufficient to learn both w and b. If data is not linearly separable, then decision boundary could be non-linear. Figure 4 showed an example of SVM in linearly separable case. Training examples of positive class is denoted by solid circle and negative class by hollow circle.
Margin of a linear classifier is the width of boundary (yellow strip) that can be increased before hitting a point. A linear classifier for above example and its margin is shown in Figure 5(a).




(a) (b)

Fig. 2: (a) A linear classifier and its margin in yellow. (b) Linear classifier with the maximum margin – Linear SVM (Moore, 2001).

SVM is the maximum margin classifier as discussed above. The maximum margin will be something like as shown in Figure 2(b). SVM tries to make margin between supporting vectors and decision boundary as large as possible. For above example, there are three support vectors for the maximum margin decision boundary as shown in Figure 5(b). The boundaries of the yellow strip are the supporting hyperplanes (or lines in this case).

*E. Resampling and Bootstrapping:*

Resampling is a technique of generating new patterns from the available set of patterns of a class (Wolter, 2007). This is a popular way of reducing sparseness of data. If the training set size is small due to reasons like unavailability of enough patterns or high cost of collecting patterns then Pattern Synthesis is used to generate new patterns to add to the training set.

Resampling is broadly divided into the following two types:

- Model based
- Instance based

Model based resampling derives a model from the training set of patterns and uses this model for generating new patterns. The derived model could be a Hidden Markov Model (HMM) and or it could be a probability distribution model (Viswanathet al., 2005). The advantage is that artificial patterns can easily be generated once a model is in hand. But if during the derivation of the model some wrong assumptions were made then the patterns generated may be erroneous. Computational time for deriving the model can be prohibitive. Moreover once the model is derived this can be used to classify the patterns directly instead of using it to generate more patterns and then use some other technique to classify.

In Instance based resampling, there is no need to derive any model. By using the existing pattern, new patterns are derived instantaneously using some properties of the data. The advantage of instance based resampling is that there is no need to any assumptions and it's very fast compared to generating a probability model first (Viswanathet al., 2005). Unlike probability distribution model, number of generated patterns is finite. Bootstrapping is one such technique and is popularly used in statistics for resampling (Davison, 1997).

In a popular implementation of bootstrapping, for every pattern its r-nearest neighbors are found from among the training set patterns of the same class and a weighted average of the neighbors is taken(Hamamoto et al., 1997). Such an average pattern is called a bootstrap pattern. This procedure can be repeated many times to generate more patterns. There are many variations of this basic

bootstrapping technique like avoiding the same pattern's use more than once and taking weights differently etc. Bootstrapping has been used in conjunction with nearest neighbor classifiers only. In this dissertation the researcher uses a novel method which is to use patterns that are generated using bootstrapping instead of original software metrics training set and investigates whether this could increase prediction accuracy of SVM for the software fault proneness prediction problem.

*F. Clustering:*

Clustering is an unsupervised learning technique where the given patterns are clustered based on some criteria like distance between points (Duda et al., 2012). It is the process of partitioning a set of patterns. For example, consider the following collection of nine characters:

C .a 'A , e c * E

The three clusters after portioning are:

' , *aec C A E

In the above example, size is used as the basis for clustering. The three clusters consist of smallest sized, lower case and upper case letters. Thus clustering is a technique to divide data into smaller groups of data based on some similarity features. Most commonly used similarity measure in clustering is distance. Given a set of points the aim is to divide the points into clusters based such that points are close to each other in a cluster. A popular algorithm based on inter-cluster similarity is K-means clustering algorithm in which data points are divided into k clusters. Each point is closer to the mean of its cluster than the mean of other clusters (Kanungoet al., 2000). An example is given below.

Suppose there are following 9 points in 3-D plane.

$$X = \begin{Bmatrix} (1,0,0), (0,1,2), (7,7,7), (2,1,1), (6,6,6), \\ (6,4,1), (6,6,7), (6,3,1) \end{Bmatrix}$$

Let the number of clusters be 3; $C_1$, $C_2$and $C_3$.

$$C_1 = \{(1,0,0), (0,1,2), (2,1,1)\}$$

$$C_2 = \{(6,4\,1), (6,3,1)\}$$

$$C_3 = \{(7,7,7), (6,6,6), (6,6,7)\}$$

In above example distance between any two points in a cluster is less than distance between any two points in different clusters. For example, distance between $(1,0,0)$and $(0,1,2)$ is 2.5 unit and distance between $(1,0,0)$and $(6,3,1)$is around 6 units.

*G. Random Projection*

If we reduce the dimensionality of the input data by some technique that preserves certain properties in the input space like distance (dissimilarity measure) or angles (similarity measure) between data points, then we will be handling lower dimensional data. Moreover the minimum number of examples 'm' required is also lower in the lower dimensional space. Random projection is widely used technique to reduce dimension. Random projection preserves all pair wise distances to within a factor of $(1+\varepsilon)$ where

$$k = O\left(\frac{\log n}{\varepsilon^2}\right)$$

Let R be a random matrix used for projecting d-dimensional vectors onto k-dimensional space. If we have n d-dimensional patterns then A will be $(d \times n)$ and B will be $(k \times n)$ matrix

$$B^T = A^T \times R$$

It can be shown that the time taken to construct the random matrix is $O(nk)$ and performing projection takes $O(mnk)$ time.

## II. LITERATURE SURVEY

Choosing the appropriate machine learning algorithm is the most important task before building any software quality model. A study to analyze various machine learning algorithms for software quality prediction on NASA MDP data sets has been done by few authors in 2008 (Vandecruyset al., 2008). They have used 70% data for training and 30% for testing. SVM has also been tried. All machine learning algorithms that they analyzed were found to give close results. They also proposed a new algorithm AntiMiner+ which was giving slightly better results (Vandecruyset al., 2008).

One major issue in the software metrics datasets is the class imbalance (Japkowicz, 2000). Class imbalance means instances of one class in much more than instances of other class. In case of software metrics datasets, most of the modules are non-faulty. Hence, there is large imbalance in number of non-faulty and faulty modules. Class imbalance is found in many datasets and not just software metrics datasets (Guo et al., 2008). In metrics dataset most of the software modules are not defective. Because of this imbalance, problems arise in learning instance-based machine learning classifiers (Seiffertet al., 2007). Since there is imbalance in number of software modules instances with defects and number of instances with no defects, accuracy of the classifier is not the best possible measure due to bias. Because of the bias, if classifier tries to increase accuracy it will also increase false alarm rate. Increase in false alarm rate will mean software modules which are not defective would be predicted as defective. Companies then have to again test the modules for defects when there is none. This will increase time and cost. A research was done to use resampling techniques to overcome class imbalance (Pelayo& Dick, 2007). A resampling technique known as SMOTE which oversamples the minority class data was used. Results of the research were encouraging. For few selected datasets, there was around 25% improvement in accuracy. In one another research, empirical evaluation of five resampling techniques was done to overcome class imbalance (Afzal et al., 2012). Bootstrapping gave better performance compared to other resampling methods for four datasets but overall there was no significant improvement.

Combination of different classifiers for predicting software quality was also tried to improve performance (Tosun et al., 2008). Authors have used three classifiers and decision was taken on the majority vote. This means if two of the three classifiers predict one thing then that will be the overall result. The classifiers authors used were simple Naïve Bayes, Artificial Neural Networks (ANN) and Voting feature intervals (VFI). He applied the combination of three classifiers on NASA MDP datasets. There was no significant improvement in performance. But on non-NASA datasets, performance improved slightly. The algorithm was able to predict 75% of defected modules while using Naïve Bayes would have given only 70% detection rate (Tosun et al., 2008).

SVM was also used for software defect prediction. One such paper evaluated SVM capability for fault

proneness prediction and compared its performance with other classifiers. (Elish&Elish, 2008). The results were at least as good as for other classifiers for all datasets with slightly better results for couple of datasets. One more study analyzed the empirically validate the relation between software metrics and fault proneness using SVM as the learning algorithm (Singh et al., 2009). The findings stated that there were few metrics which were found to have relation to fault proneness.

For machine learning applications, data transformation techniques could help to achieve better performance. For software quality models also, authors have tried to use these transformation techniques (Jiang et al., 2007). They have used log normalization, discretization and their combination for filtering. After filtering they used the data on various machining learning classifiers. Performances of Naïve Bayes improved after pre-processing using discretization. Other than that classifiers performance did not improve much by pre-processing data using log normalization and discretization. So the important finding is that this transformation did not succeed in improving the performance of classifiers. Other finding was that the random forest performed better overall than other classifier (Jiang et al., 2007).

Logistic regression was also used for fault prediction on a small data set (Denaroet al., 2002). The data set was from antenna network configuration. Performance of logistic regression was good on that data set with True Positive Rate (TPR) just below 90%. TPR is the fraction of examples belonging to positive class that are correctly classified.

There was also one study performed on the analysis of NASA MDP data sets (Koru &Liu, 2005). It was found that large sized software modules contain more defects. That is of course expected. They also noted that many data sets are small sized. Hence metrics values will also be small. This will result in small variance between data sets of two classes and hence classification becomes difficult. To prove this, they decided to divide sets of instances of modules into different sizes. Different classifiers were used on each set and the result was that modules with large size performed better than smaller sized modules(Koru &Liu, 2005).

Selecting best subset of metrics is a good way to eliminate unwanted and redundant metrics. This process is called as feature selection. Principle Component Analysis (PCA) is a popular algorithm to select best subset of features and has been tried for software metrics datasets also (Menzieset al.,2003). Authors have used PCA to remove inter-correlation among the metrics. This was done by transforming the metrics set to a smaller dimension set. Number of metrics was reduced to 5-6 from original 20 metrics. Results were good and better performance was achieved.

Another major issue in software metrics data is noise and outliers (Seiffertet al., 2007). In the context of machine learning, noise means corruption, redundancy or any other imperfection in data that may impact the learning algorithm's performance. Noise may appear because a faulty module was not detected in testing and hence during metrics collection it was labeled as non-faulty. If in a dataset, there are one or more software metrics that shows very little variation across the faulty and non-faulty models, then these would also be considered as noise. Another example of noise would be two or more metrics showing nearly identical variation across all instances (Khoshgoftaaret al., 2005). This would result in redundant information from some metrics. Next is the problem of outliers. Outliers are related to measurement errors. An instance in dataset is termed as outlier if values of its features are very different from other instances from the same class. A research paper based on Random Forests showed more robustness to noise and outliers (Guo et al., 2004).Authors have used trade off between accuracy and recall using a 'parameter called 'cut-off'. They also used Random Forests for feature selection and selected five best features. But that did not improve the performance of the classifier. The model's performance was also compared with other machine learning classifiers. It was found that Random Forests give slightly better performance (Guo et al., 2004).

## III. PROPOSED WORK

The proposed algorithm is described below:
Two variations of bootstrapping were used to pre-process the metrics data. In the first method nearest neighbors of each dataset example were found in the input space. In the second method nearest neighbors were found in the kernel space. Both the algorithms are given below although only algorithm 1 will be used in this dissertation.

### *A. Bootstrapping in Input Space*

Algorithm 1:Bootstrapping in Input Space
Input: metrics data set $x_i$ size m, no. ofneighborsk
for $i = 1$ tomdo
findknearestneigborsof$x_i$ininutspace
$\min_j \|x_i - x_i^j\|$
$\text{neighbors}(x_i) = \{x_i^1, x_i^2, \ldots\ldots, x_i^k\}$
$x_i = \frac{1}{k}\sum_{j=1}^{j=k} x_i^j$
endfor
Output: Bootstrapped metrics data set $x_i$
Bootstrapping in Kernel Space

Finding neighbors in the kernel space makes more sense. Using kernel function, faulty and non-faulty modules are linearly separated by projecting metrics into higher dimension space. Bootstrapping in the kernel space reduces the Dia and increases the Margin in the kernel space. Neighbors in the kernel space were found using kernel trick as follows: The distance between two points and $x_1$ is$\|\phi(x) - \phi(x_1)\| = k(x,x) + k(x_1,x_1) - 2k(x,x_1)$

Algorithm 2: Bootstrapping in Kernel Space
Input: datasetx$_i$sizem, no. ofneighborsk
for $i = 1$ tomdo
findknearestneigborsof$x_i$
ininutspace
$\min_j \|\phi(x_i) - \phi(x_i^j)\|$
$\text{neighbors}\ (x_i) = \{x_i^1, x_i^2, \ldots\ldots, x_i^k\}$
$x_i = \frac{1}{k}\sum_{j=1}^{j=k} x_i^j$

end for
Output: Bootstrapped metric data set $x_i$

### *B. Clustering*

Clustering can also be used as a data synthesis technique. Clustering is an unsupervised learning technique where the given training examples are clustered based on some criteria like distance between points. Clustering has been used in association with SVMs before (Finley&Joachims,2005). A notable example is CBSVM (Yu et al., 2003). But the impact of clustering to control the VC dimension of SVM has not been discussed in the literature earlier. The researcher presents an approach to reduce the VC dimension of SVM using clustering. Here cluster centers replace the original metrics data sets. Training of SVM takes place on these cluster centers instead of the original metrics data set. This can be thought of as a training data synthesis technique where the synthetic data examples are the cluster centers. It can be shown that clustering reduces the VC dimension of SVM. When the data examples belonging to a class are clustered and the entire dataset is replaced by these cluster centers, the Margin increases and the diameter decreases. The training examples responsible for decreasing the margin or increasing the diameter are those which lie at the boundaries. When metrics data set is clustered these points are shifted towards the centre of the metrics data set. They are not present themselves in the final data set (the cluster centres form the final data set)but the centre to whose cluster these points belong is influenced by each point. Thus if the cluster has a boundary data example, then it will pull the cluster centre towards the boundary but other inner data examples also pull the centre. Thus every data example has its influence on the centre, and the shape of the boundary is not lost due to clustering. The boundary just shrinks and all the data examples which lie outside this new boundary are effectively removed. Most probably these 'outside' data examples are noise. Thus clustering takes care of noisy data removal. Clustering brings out the true boundary of the data set by filtering out the noisy data. If required, we can eliminate points in small size clusters from the data sets. When the training data is clustered, the cluster centroid is average of all data examples belonging to that cluster. If the size of every cluster is more than one then this average pattern lies closer to the centroid of the entire training set that the farthest pattern in the training set. When the cluster centres replace the training set the new Dia and Margin are smaller than the corresponding values of the original training set. Let the number of clusters in each class after clustering be k. Let the cluster centres be$\{X_1, X_2, \ldots . X_k\}$ in class 1 and $\{Y_1, Y_2, \ldots . Y_k\}$ in class 2. Let the points belonging to cluster $X_1$ be $\{x_1^1, x_2^1, x_3^1 \ldots .\}$ and similarly for other clusters in class 2. Then we have clusters' centres.

$$X_i = \sum_{k=0}^{n} x_i^j \text{ where} x_i^j \epsilon \text{ cluster } X_i \; \forall \; i$$

$$Y_i = \sum_{k=0}^{n} y_i^j \text{ where} y_i^j \epsilon \text{ cluster } Y_i \; \forall \; i$$

Let the new Dia and Margin be
$Dia' = \| X_p - Y_q \square$
$Margin' = \| X_s - Y_t \square$
Now we show that

$Dia' \leq Dia$
$Margin' \leq Margin$

$$Dia' = \left\| \frac{1}{k_p} \sum_{i=1}^{k_p} x_p^i - \frac{1}{l_q} \sum_{j=1}^{l_q} y_q^j \right\|$$

$$= \left\| \frac{l_q \sum_{i=1}^{k_p} x_p^i - k_p \sum_{j=1}^{l_q} y_q^j}{k_p \times l_q} \right\|$$

$\leq Dia$

In the case of Margin too, the proof follows in similar fashion.

## IV. EXPERIMENT RESULT

The improved approach of using bootstrapping and clustering with SVM has many advantages compared to previous approach:

### *A. Experiment:*

First normalization is done on all datasets. Then outliers are removed using Weka Outlier removal filter. Then for 10-fold cross validation, dataset is broken into 10 equal subsets. 9 sets are used for training and rest one for testing. Before dividing datasets, randomize option is first selected in Weka to randomize the data. The training set's minority class, i.e. faulty modules are oversampled by using SMOTE.

Then bootstrapping is applied to the oversampled training set separately for each class. Bootstrapping will give output dataset for each class which is the combined to make a single set before applying SVM. After that six SVM models with RBF kernels are built by varying gamma and cost factor. Then models are testing on test data. Procedure is repeated for all 10 folds. An example of using SMOTE in Weka is shown in Figure 3. The dataset is one fold on CM1 after outlier removal. Before applying SMOTE, number of defective modules was 30 as shown in Figure 3(a). The effect of applying SMOTE on CM1 is shown in Figure 3(b). SMOTE percentage used was 300% which increase the number of defective modules to 120. Table 1 gives the SMOTE percentage for each dataset used in this experiment.

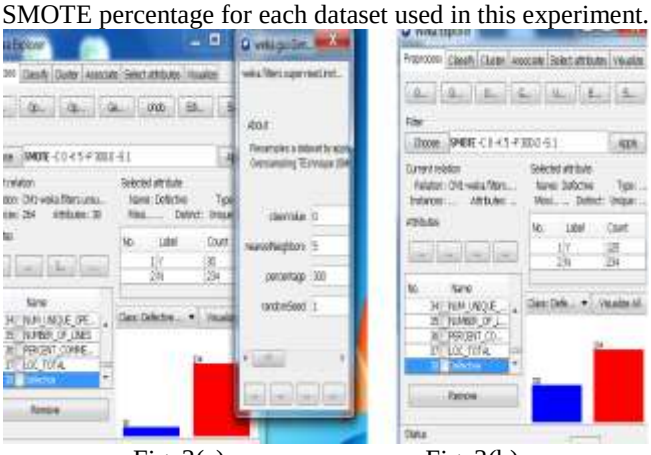


Fig. 3(a) Fig. 3(b)

Fig. 3: (a)SMOTE filter in Weka. (b) Effect of SMOTE on CM1 dataset with percentage parameter as 300.

| Dataset | SMOTE Percentage |
|---|---|
| CM1 | 300 |
| JM1 | 100 |
| KC1 | 200 |
| KC3 | 150 |

| | |
|---|---|
| MC1 | 300 |
| MC2 | 0 |
| MW1 | 300 |
| PC1 | 300 |
| PC2 | 300 |
| PC3 | 250 |
| PC4 | 300 |
| PC5 | 300 |

Table 1: SMOTE percentage for different datasets

Finally an example of comparison of Cyclomatic Density values before and after bootstrapping is shown below by Figure 4 for CM1 dataset.

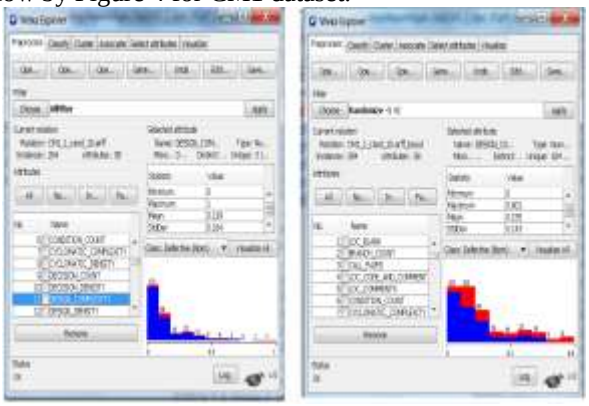


Fig. 4: Design Complexity Metric (a) before bootstrapping b) after bootstrapping

As can been seen by Figure 17 that standard deviation has come down which is expected while taking any kind of averages.

## V. CONCLUSION

By using different filtering techniques and various pre-processing and transformation techniques like bootstrapping, clustering , random projection, large number of fault prediction problem are solved using the SVM(software vector machine) Algorithm and the datasets used are the popular NASA Metric Data Program datasets.
From the result the conclusions are:

1) There is very significant improved in fault prediction of models built after using bootstrapping and clustering as data transformation.
2) Performance of fault proneness prediction model increases when a subset of metrics based on a correlation measure is selected instead of all metrics for building the model.

## REFERENCES

[1] Japkowicz,N.(2000). The class imbalance problem: Significance and strategies. Proc. of the Int'l Conf. on Artificial Intelligence.
[2] Denaro, G., Morasca, S., &Pezzè, M. (2002).Deriving models of software fault-proneness. Proceedings of the 14th international conference on Software engineering and knowledge engineering.ACM, pp. 361-368.
[3] Dijkstra, E. W. (2002). Go to statement considered harmful. Software pioneers. Springer Berlin Heidelberg, pp. 351-355.
[4] Menzies, T., Ammar, K., Nikora, A., & Stefano, S. (2003). How simple is software defect prediction?. Journal of Empirical Software Engineering, vol. 32, no. 2, pp. 1156-1161.
[5] Guo, L., Ma, Y., Cukic, B., & Singh, H. (2004).Robust prediction of fault-proneness by random forests. Software Reliability Engineering, 2004.ISSRE 2004. 15th International Symposium on IEEE, pp. 417-428..
[6] Koru, A. G., & Liu, H. (2005).An investigation of the effect of module size on defect prediction using static measures.ACM SIGSOFT Software Engineering Notes.ACM, vol. 30, no. 4, pp. 1-5.
[7] Lanza, M., Marinescu, R., &Ducasse, S. (2006). Object-oriented metrics in practice. Heidelberg: Springer, pp. 29-31.
[8] Jiang, Y., Cukic, B., & Menzies, T. (2007). Fault prediction using early lifecycle data. Software Reliability, 2007.ISSRE'07.The 18th IEEE International Symposium on. IEEE, pp. 237-246.
[9] Kotsiantis, S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. Informatica, vol. 31, pp. 249-268.
[10] Elish, K. O., &Elish, M. O. (2008). Predicting defect-prone software modules using support vector machines. Journal of Systems and Software, vol. 81, no. 5, pp. 649-660.
[11] Gondra, I. (2008). Applying machine learning to software fault-proneness prediction. Journal of Systems and Software, vol. 81, no. 2, pp. 186-195.
[12] Guo, X., Yin, Y., Dong, C., Yang, G., & Zhou, G. (2008).On the class imbalance problem. Natural Computation, 2008.ICNC'08.Fourth International Conference on. IEEE, vol. 4, pp. 192-201.
[13] Catal, C., &Diri, B. (2009). A systematic review of software fault prediction studies. Expert systems with applications, vol. 36, no. 4, pp. 7346-7354.
[14] Pressman, R. S. (2010) Software Engineering: A Practitioner's Approach. 8th Ed. New York: McGraw-Hill.