An approach for an accurate Fault Prediction Model using the MDP Dataset

Mr. Amrit Pal

Parth Pathak

GLA University, Mathura

Abstract:

Software fault prediction is the method of automatically detecting the faulty software from a bunch of software. It is necessary to detect all the software instances which are faulty to decrease the chances of faults in real-life implementations. We are using the MDP software defect dataset provided by NASA for building the prediction model. First, we compared the best suitable classification models on the basis of accuracy and sensitivity and then chose the best accurate among them. After that, we improved the model by removing the class imbalance using a synthetic re-sampling technique. We compared the technique with its variants and get the recall and F2-score of the predicted results after oversampling. Finally, we identified the most accurate approach for software fault prediction using the MDP dataset based on the experimental observations. This approach minimizes the type I and type II error in prediction results. This approach can be used as the basis of building more accurate and precise fault prediction models using the NASA MDP dataset.

1. INTRODUCTION

Faults can occur in software even when the development is finished due to various human errors and shortcomings of the system used for software development. These faults can cause failure in the actual implementation of the software at the user's end. The existing software fault prediction systems have done a helpful job in making error-free software. The systems are still unable to achieve high accuracy and robust quality check. Due to the ever-expanding scope of software, a system which can predict the faultiness of software with maximum accuracy and minimum manual interaction is mandatorily required.

The selection of dataset is the first important step in building a prediction model. We have used the Metrics Data Program (MDP) datasets for prediction of software faults. The MDP dataset is a standard dataset prepared by NASA through collection of software data from various sources. It is a binary dataset which describes software based on various numerical measures. In other words, it contains the attributes or features which describe software, and a binary class which labels any instance of software as faulty or not faulty. This dataset is widely used all over the world for making software fault prediction systems and conducting research in this field.

The dataset for any machine learning model needs to be suitable for a precise prediction. But, there are issues due to noisy data points and class imbalances. The ability of a model to differentiate the classes depends upon how well the model generalizes to the given data. This generalization does not apply properly if there is a large difference between the numbers of instances of each class. Thus, re-sampling of the minority class becomes necessary before applying any machine learning algorithm. The classification models can achieve a high sensitivity on a balanced data. There are various predictive models developed for predicting software faults on this data, majorly using Support Vector Machines, Decision trees, etc. The performance of these models is measured by how well the models are able to identify the positive classes (or the minority classes). In our case, we have to identify whether a test instance is faulty or not. In real-life situations, the faulty data points are much less in number than the non-faulty data points. Hence, the problem of detecting faults in software reduces to identifying all the positive classes correctly, by minimizing the type II error.

In this paper, we compare the performance of the existing prediction models on the cleaned MDP dataset and decide on which technique minimizes the type I and type II errors. After that, we apply the over-sampling techniques - Synthetic Minority Over-sampling Technique(SMOTE), Bordeline-SMOTE1, Borderline-SMOTE2, and cluster-based SMOTE – to increase the performance of our model. We compare their performance to get the best model to apply to the MDP dataset.

2. DATASET

Before applying prediction models on the dataset, it is important to do a quality check. As the original dataset, which was made available by NASA, struggled with its quality due to a number of issues, such as duplicate data points, missing values, etc, it was necessary for researchers to build preprocessing algorithms for cleaning of data. Due to its heavy use and demand of the researchers, the MDP dataset evolved through time and it is now available in a preprocessed and cleaned state, which we have used in our experiments. The cleaned MDP dataset is described in Table 1.

The validity of the preprocessing algorithm used to clean the dataset is highly important. It helps us in deciding if the data is suitable for using with various prediction models or not. We have used the data based on the quality issues described in the paper "Data Quality: Some Comments on the NASA Software Defect Datasets." It removes the duplicate data points from the dataset and adds the missing data values.

Serial No.	Dataset	#Features	No. of Instances		No. of Classes			
			Training	Testing	Training		Testing	
					Y	N	Y	N
1	CM1	38	261	66	37	224	5	61
2	JM1	22	6225	1557	1320	4905	352	1205
3	KC1	22	946	237	239	707	75	162
4	KC3	40	155	39	31	124	5	34
5	MC1	39	1590	398	36	1554	10	388
6	MC2	40	100	25	37	63	7	18
7	MW1	38	202	51	20	182	7	44
8	PC1	38	564	141	52	512	9	132
9	PC2	37	596	149	14	582	2	147
10	PC3	38	861	216	113	748	21	195
11	PC4	38	1029	258	142	887	35	223
12	PC5	39	1368	343	379	989	92	251

Table 1: *The table shows the description of the cleaned and preprocessed dataset from the NASA Metrics Data Program. The dataset have been divided into two parts- for training and testing.*

As the above table shows, we have divided the data into two parts – training data and testing data. The ratio we have used is 4:1. Since the dataset is a binary dataset, it labels the instances with two classes, i.e. either 'Y' or 'N'. The number of each class corresponding to the training and testing data is also helpful if we know it in advance. These formulations of data in a table format are necessary for choosing the prediction methods and also implement them. After knowing our dataset, we should choose the prediction methods to apply from existing machine learning techniques.

3. LITERATURE REVIEW

A classifier can be used with the MDP data, which will identify some pattern to label the new data. There are several classifiers which can be applied. The selection of a classifier depends on the efficiency and total training and testing time. The oversampling techniques are also important in this context. The following techniques can be used with our data for developing a model.

i. Random Forest

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Basic parameters to Random Forest Classifier can be total number of trees to be generated and decision tree related parameters like minimum split, split criteria etc. This technique is efficient in building multi-class models. While, the specificity rate is high in models developed using this algorithm, but struggles to have a high recall value. Random forests can be used to rank the importance of variables in a regression or classification problem in a natural way. Referenced from *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html*.

ii. K-Nearest Neighbours

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant, or vary based on the local density of points. The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice. The neighbors are taken from a set of objects for which the class (for *k*-NN classification) or the object property value (for *k*-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.
Referenced from
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

iii. AdaBoost

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases. It can be used in conjunction with many other types of learning algorithms to improve performance. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner.
Referenced from
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html.

iv. Gaussian Naïve Bayes

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution. Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. Naïve Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem.
Referenced from
https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html.

v. Linear Support Vector

In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot). Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line). In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes.
Referenced from
https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html.

Confusion Matrix and Performance Measures

The performance measures are computed from the confusion matrix. We have used the following format for confusion matrix. Since, we have to minimize the FNR (False Negative Rate), we have used recall or sensitivity to measure the performance and maximize its value.

		Predicted Values	
Actual Values	**N**	True Negative (TP)	False Positive (FP)
	Y	False Negative (FN)	True Positive (TP)

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\textit{Recall} = \frac{TP}{TP + FN}$$

The sensitivity or recall is a standard way of measuring performance of a classifier. We use it in the Results section to show comparison between performance of a model using different oversampling techniques. The recall value indicates that how well the classifier is able to predict the positive classes.

4. EXPERIMENT

We considered all of the above mentioned algorithms and implemented those using Python. The classes 'Y' and 'N' were converted to floating point values, i.e. 1.0 and 0.0 respectively, in order to make the classes suitable for use with the library based machine learning classifiers. We checked the accuracy of these algorithms on each dataset to find a relation between type of algorithm and the dataset. We also calculated the mean accuracy for each algorithm. The results of the experiment are shown in Table 2. The random forest classifier gave the best accuracy value. While, the Gaussian naïve bayes classifier was low in accuracy, its computation time for training and testing was significantly higher. AdaBoost and k-nearest neighbors also performed well on accuracy.

We also compared these algorithms on the basis of sensitivity as shown in Table 3. These values give us more insight on the performance of the models. All models performed low on sensitivity which indicates that the models are unable to generalize, basically because of the class imbalances. The positive classes are very less in number as compared to the negative classes. Due to this, the classifier cannot predict the positive classes correct and thus the recall or sensitivity value is very low. These results are also in contrast to the accuracy values as the models having high accuracy have low sensitivity and vice versa. This gives us an essential information that for an imbalanced data, accuracy is not a good performance measure. Hence, for all the further steps we have used sensitivity as the performance metric.

Serial No.	Dataset	Accuracy of different Classifiers				
		Random Forest	k-NN (k=7)	AdaBoost	Gaussian NB	Linear SVC
1	CM1	0.9242	0.9242	0.8485	0.8788	0.8939
2	JM1	0.7803	0.7553	0.7771	0.7675	0.754
3	KC1	0.7004	0.6413	0.6962	0.6751	0.6582
4	KC3	0.8205	0.7692	0.7436	0.7436	0.7179
5	MC1	0.9748	0.9748	0.9648	0.7965	0.7261
6	MC2	0.64	0.64	0.64	0.64	0.6
7	MW1	0.8627	0.8627	0.8431	0.8235	0.8431
8	PC1	0.9362	0.9149	0.8794	0.8865	0.8936
9	PC2	0.9866	0.9866	0.9798	0.953	0.9664
10	PC3	0.9028	0.875	0.8611	0.125	0.8843
11	PC4	0.8643	0.8604	0.8721	0.8527	0.8178
12	PC5	0.7522	0.7084	0.7201	0.7259	0.6968
Mean Accuracy		0.845416667	0.826066667	0.818816667	0.739008333	0.787675

Table 2: The table shows acuuracy values for each algorithm on different dataset, along with the mean accuracies.

Serial No.	Dataset	Calculation of Sensitivity				
		Random Forest	k-NN (k=7)	AdaBoost	Gaussian NB	LinearSVC
1	CM1	0	0	0.2	0.2	0
2	JM1	0.053977273	0.133522727	0.147727273	0.136363636	0.255681818
3	KC1	0.16	0.133333333	0.24	0.253333333	0
4	KC3	0	0	0	0	0.2
5	MC1	0	0	0.2	0.4	0.5
6	MC2	0.142857143	0.142857143	0.428571429	0.142857143	0.285714286
7	MW1	0	0	0	0.714285714	0.857142857
8	PC1	0	0	0.333333333	0.222222222	0.666666667
9	PC2	0	0	0	0	0.5
10	PC3	0	0	0.285714286	0.952380952	0.19047619
11	PC4	0	0.057142857	0.457142857	0.371428571	0
12	PC5	0.239130435	0.25	0.315217391	0.108695652	0.25
Mean		0.049663738	0.059738005	0.217308881	0.291797269	0.308806818

Table 3: The table shows sensitivity or recall values for each algorithm on different dataset, along with the mean.

After close observations of these comparisons, we got to know that Random Forest and k-Nearest Neighbors are not good approaches to solve our problem since they have large type II error. AdaBoost is the optimum approach which provides both sensitivity and accuracy. We are going to use AdaBoost for performing further experiments with oversampling.

Synthetic Minority Oversampling Technique (SMOTE)

Oversampling of the minority class instances is done to balance the training dataset. Synthetic instances are created which are not the exact copy of the existing instances and can help in balancing data without overfitting and noise. We first divided the majority and minority class instances from the training dataset. Then for each minority instance x_i we calculate k-nearest neighbours which belong to minority class and store them in a set P.

$$P = \{p_1, p_2, p_3, \ldots, p_k\}$$

$$k = \text{floor value of } (m_s / m_l)$$

Here, m_s is the number of majority instances and m_l is the number of minority instances. After that, for each p_i in the nearest neighbours of x_i, a synthetic data point is generated using the following formula and then all the synthetic point are combined with the initial dataset to produce a new balanced dataset.

$$s_i = x_i + \text{rand}(0, 1)*|x_i - p_i|$$

The rand(0, 1) function generates random real values in the range [0, 1).

Boderline SMOTE 1

In borderline SMOTE 1, the set of k-nearest neighbours *P* contain instances of both the classes and the number of majority instances in *P* is considered to be *n*. We assume that the size of set *P* is *m*. Now,

If $n==m$, x_i is noise instance

If $0<=n<m/2$, x_i is safe instance

If $m/2<=n<m$, x_i is borderline instance

The borderline instances are used to generate new synthetic points using the SMOTE formula.

Borderline SMOTE 2

There is a little but impactful difference between Borderline 1 and Borderline 2. Instead of just minority classes, borderline 2 oversamples the borderline instances of majority classe. This, in return, increases the hard-to-learn instances along the borderline and the ability of the model to differentiate the classes increases. It uses a range [0, 0.5) to produce random values for the SMOTE formula.

Cluster-based SMOTE

It is also similar to the borderline SMOTE algorithm, except that here we use the safe data points to create new data points between them, so that the minority class forms its own clusters away from the majority class boundary. It helps in creating a separate domain for the minority class instances so that the precision of the prediction model increases.

There are more oversampling algorithms such as ADASYN, etc. These algorithms are almost similar in performance to the SMOTE algorithm. We applied the SMOTE algorithm and its variants on the cleaned MDP dataset. We used the AdaBoost classifier for prediction as discussed before. The performance metrics used were recall value and F-measure. The F-measure was calculated using the following *Dice Similarity Coefficient* (DSC) with $\beta=2$.

$$F_\beta = \frac{(1+\beta)^2 \cdot TP}{(1+\beta)^2 \cdot TP + \beta^2 \cdot FN + FP}$$

The main objective to use this formula is to correctly measure the performance of the prediction model on imbalanced data. It enforces more importance on the false negatives. Hence, it will measure the amount of type II error. We must maximize the F-measure

value in order to minimize the type II error. The algorithms ran successfully on all data and the results are being shown in the next section.

5. RESULTS

We first numerically compute the sensitivity and f-measure of the datasets using AdaBoost by applying the oversampling techniques. The sensitivity values are given in Table 4, while the DSC values are given in Table 5. We can clearly see that the regular SMOTE and Borderline SMOTE1 give satisfying results. Borderline SMOTE2 method gives a negative result as it oversamples the majority class instances also. This leads to increased noise and overfitting, which is the reason for the decrease in value. The cluster-based SMOTE has also not performed well enough. These results give us a well defined model for software fault prediction with the MDP dataset providing high recall and accuracy. The increase in recall value indicates that the type II error is decreased and the increase in the F-measure value indicates that both type I and type II errors are decreased.

		Original	SMOTE	Borderline SMOTE1	Borderline SMOTE2	Cluster-based SMOTE
1	CM1	0.2	0.939393939	0.909090909	0.145833333	0.2
2	JM1	0.147727273	0.806826998	0.654152446	0.041666667	0.494339623
3	KC1	0.24	0.748691099	0.414141414	0.137614679	0.3
4	KC3	0	0.871794872	0.857142857	0.4	0
5	MC1	0.2	0.981770833	0.981012658	0	0.2
6	MC2	0.428571429	0.75	0.428571429	0.428571429	0.727272727
7	MW1	0	0.892857143	0.891304348	0.081632653	0
8	PC1	0.333333333	0.955882353	0.943925234	0.024390244	0.333333333
9	PC2	0	0.962264151	0.962264151	0	0
10	PC3	0.285714286	0.935779817	0.927083333	0.015306122	0.322580645
11	PC4	0.457142857	0.903614458	0.919047619	0.237209302	0.558823529
12	PC5	0.315217391	0.814946619	0.640522876	0.264900662	0.620967742
	Mean	0.217308881	0.880318523	0.794021606	0.148093758	0.3131098

Table 4: Sensitivity (Recall) values of the AdaBoost classifier after applying SMOTE and its variants for oversampling.

		Original	SMOTE	Borderline SMOTE1	Borderline SMOTE2	Cluster-based SMOTE
1	CM1	0.185185185	0.94224924	0.905797101	0.162790698	0.185185185
2	JM1	0.172528202	0.804082264	0.673459827	0.051104972	0.526527331
3	KC1	0.27027027	0.758218452	0.437100213	0.160944206	0.335195531
4	KC3	0	0.885416667	0.841121495	0.412371134	0
5	MC1	0.208333333	0.983820459	0.980392157	0	0.208333333
6	MC2	0.416666667	0.78125	0.416666667	0.416666667	0.677966102
7	MW1	0	0.902527076	0.907079646	0.097087379	0
8	PC1	0.3	0.953079179	0.936920223	0.02994012	0.3
9	PC2	0	0.962264151	0.962264151	0	0
10	PC3	0.285714286	0.926430518	0.924195223	0.018773467	0.337837838
11	PC4	0.470588235	0.910194175	0.919047619	0.267576076	0.552325581
12	PC5	0.337209302	0.819027182	0.650730412	0.297619048	0.633223684
	Mean	0.22054129	0.88571328	0.796231228	0.159572814	0.313049549

Table 4: F-Measure values of the AdaBoost classifier after applying SMOTE and its variants for oversampling of data.

6. CONCLUSION

The NASA datasets have previously been used widely in studies of software defects. In 2013, Shepperd et al. presented a set of rules for removing erroneous data from the NASA datasets making this data more reliable to use. But, these rules did not remove the problem of class imbalances in the dataset. A balanced dataset is equally important to an error-free dataset for using with a machine learning model. The classification model requires to be trained rigorously in order to give good results. The ability of a model to generalize depends upon the quality and quantity of training and for this purpose it is necessary to have sufficient amount of data instances of each class so that the model can learn to clearly distinguish among classes. Due to the extensive use of NASA MDP datasets, it is necessary to build accurate oversampling techniques to maximize the true detection of positive instances in MDP dataset.

Our study shows that the standard SMOTE algorithm for synthetic re-sampling of positive data points provides great results when used with the AdaBoost Classification Model on the NASA MDP dataset. This combination minimizes the type I and type II errors in prediction results and gives a high recall value. But, the prediction results still have a lot of standard deviation and further improvements are still required to build the best software fault prediction system. The further research should seek to improve the results of this model to get the maximum accuracy possible.

7. REFERENCES

[1] Shepperd, M., Song, Q., Sun, Z. and Mair, C., 2013. "*Data quality: Some comments on the NASA software defect datasets.*" IEEE Transactions on Software Engineering, 39(9), pp.1208-1215.

[2] Petrić, J., Bowes, D., Hall, T., Christianson, B. and Baddoo, N., 2016, "*The jinx on the NASA software defect data sets.*" In Proceedings of the ACM 20th International Conference on Evaluation and Assessment in Software Engineering (EASE). Limerick, Ireland.

[3] Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP: "*SMOTE: synthetic minority over-sampling technique.*" J Artif Intell Res. 2002, 16: 341-378.

[4] H. Han, W. Wen-Yuan, M. Bing-Huan, "*Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning*" Advances in intelligent computing, 878–887, 2005.

[5] H. M. Nguyen, E. W. Cooper, K. Kamei, "*Borderline over-sampling for imbalanced data classification*" International Journal of Knowledge Engineering and Soft Data Paradigms, 3(1), pp.4–21, 2009.

[6] He, Haibo, Yang Bai, Edwardo A. Garcia, and Shutao Li. "*ADASYN: Adaptive synthetic sampling approach for imbalanced learning*," In IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), pp. 1322–1328, 2008.

[7] scikit-learn's documentation.