Optimality of Naïve Bayes Classifier in Comparison to Other Complex Classifiers

Sandeep Katypally
NC State University
Centennial Campus
Raleigh
skatypa@ncsu.edu

Michael Herzog
NC State University
Centennial Campus
Raleigh
mherzog@ncsu.edu

Tim Menzies
NC State University
Centennial Campus
Raleigh
tjmenzie@ncsu.edu

ABSTRACT

The simple Bayes classifier is known to be optimal when the attributes are independent in the data given the classes. However, Pazzani et al [1] suggest that Naïve Bayes performs well in many domains which contain attribute dependencies and they also suggest that this classifier often outperforms more powerful classifiers. We reproduce these results and show that Naïve Bayes does not have a consistent performance as suggested by Pazzani et al [1]. We also find the effects of SMOTE on Naïve Bayes and other complex classifiers.

In this paper, we compare Naïve Bayes classifier with 9 other classifiers under different conditions and give an analysis of how Naïve Bayes compares against the other classifiers.

CCS Concepts

Computing methodologies → Supervised learning by classification • Computing methodologies → Cross-validation;

Keywords

SMOTE, Sci-kit learn, OOD Design Metrics, SK metrics, weighted metrics

1. INTRODUCTION

In the field of machine learning, supervised algorithms are those where classification models learn the model from a set of training examples and their corresponding class labels, then outputs a classifier. Now, the classifier takes unlabeled data and assigns it to the class label.

Over the years, many supervised learning algorithms came into existence. But, there has been no one algorithms which could be used for all kinds of datasets. Many studies have been done to produce the best supervised(classification) algorithms, but there is no one algorithm which performs better than every other algorithm as per the literature.

Related work on defect prediction has been done which include many detection profile methods and statistical approaches Q. Song et al. [16]. Many varieties of methods have been proposed to tackle defect prediction problems such as prediction using previous defects[14], code metrics in [9][10][11][12].

Some methods have also been proposed where defects can be predicted by using component's development history and structure of dependencies [7].

2. DATA

In this paper, we consider the software engineering defect prediction datasets which are taken from the PROMISE repository[2]. These datasets are used to classify the software components into being defective or non-defective.

The datasets contain the CK metrics which aim at measuring whether a piece of code follows OO principles or not.

Some of the Object-Oriented Design(OOD) attributes are:

• **Weighted Methods for Class (WMC):** The sum of the complexities of each method in a class. If all the method complexities are considered equal and have the value of 1, then WMC equals the number of methods in a class.

• **Depth of Inheritance Tree (DIT):** Number of classes that this class inherits from.

• **Number of Children (NOC):** The count of immediate subclasses of a class.

• **Response for Class (RFC):** The number of elements in the response set of a class. The response set of a class is the number of methods that can potentially be executed in response to a message received by an object of that class.

• **Lack of Cohesion of Methods (LCOM):** For a class C, LCOM is the number of method pairs that have no common references to instance variables minus the number of method pairs that share references to instance variables.

• **Coupling Between Objects (CBO):** For a class C, CBO is the number of classes that are coupled to C.

There are around 20 attributes in the CK metric datasets taken from the PROMISE repository. We will use the following datasets to perform classifier comparison:

- ANT
- CAMEL
- IVY
- JEDIT
- LOG4J
- LUCENE
- VELOCITY
- SYNAPSE
- XALAN
- XERCES

3. NAÏVE BAYES AND OTHER CLASSIFIERS

10 classifiers including Naïve Bayes have been used to learn the binomial CK metric datasets and compare their performance.

3.1 Naïve Bayes

Naïve Bayes(NB) is a probabilistic classifier which takes advantage of features being independent. It uses the Bayes rule to estimate the probability of a vector instance and labels it as a particular class.

$$p(C_k|\mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x}|C_k)}{p(\mathbf{x})}$$

Figure 1. Bayes Rule to find the posterior probability given the prior probabilities.

3.2 Logistic Regression

Logistic regression is a regression model where the dependent variable is categorical. Logistic regression is a specific case of a linear regression. It is used to solve the problem where the dependent variable pass/fail is represented by "1" and "0".

$$F(x) = \frac{1}{1 + e^{-(\beta_0+\beta_1 x)}}$$

F(x) is is interpreted as the probability of the dependent variable equaling a "success" or “failure”

3.3 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a method used to find a linear combination of features that characterizes or separates two or more classes of objects or events. LDA is closely related to various methods including ANOVA, regression analysis PCA and factor analysis.

3.4 Quadratic Discriminant Analysis

Quadratic discriminant analysis (QDA) is closely related to linear discriminant analysis(LDA). QDA assumes the measurements from each class are normally distributed. It is used when a linear model will not suffice and two or more classes must be separated by a quadratic surface.

3.5 Multi-Layer Perceptron or Artificial Neural Network

Artificial Neural Network (ANN) is an information processing algorithm that is inspired by the way biological nervous systems process information. An ANN comprises of large number of nodes connected to each other than solve a problem in unison. Each individual node may have a summation function which combines the weighted values of all its inputs together.

3.6 Random Forests

Random forests is an ensemble classification algorithm that operates by constructing a multitude of decision trees. Decision trees are a popular method for various machine learning tasks. Random forests can also be used to rank the importance of variables in a regression or classification problems.

3.7 K-Nearest Neighbors

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure. KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry.

3.8 Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning algorithm that is used to analyzed data for classification and regression analysis. It is a non-probabilistic binary linear classifier. SVM can be used as a non-linear classifier by using something called a ‘kernel trick’

3.9 Decision Tree

Decision tree learning uses a decision tree and makes a predictive model by mapping observations about an item to the conclusions about the item’s target or label. It is one of the predictive modelling approach used in machine learning and data mining.

3.10 Perceptron

Perceptron is a kind of linear classifier. This classifier allows for online learning, which means it trains training set elements one at a time. It maps its input x to and output value y through a function f(x).

4. DATA PREPROCESSING

4.1 Discretization

We discretize the datasets, where numbers are converted into finite number of bins, using the following discretization schemes:

Equal Frequency Discretization(Nbins):

Divides every attribute data into n bins which have equal number of elements.

Equal Width Discretization (Histogram discretization):

Divides every attribute data into n bins such that range or every bin is equal

These simple discretization schemes are also very effective on performance of the classifier as per Dougherty95[3] . The plots shown in the results section are constructed after performing Equal Frequency Discretization on the datasets.

4.2 Synthetic Minority Over-sampling Technique(SMOTE):

SMOTE is a technique described in Chawla, Nitesh V., et al [4] which balances the number of positive and negative examples in the data set by oversampling the minority class.

This technique is used on the training dataset since the datasets are skewed (i.e positive to negative ratio of the labels was much less than 1).

Note: Cases were SMOTE worked and did not work are illustrated clearly in the following sections.

5. EVALUATION METRICS

Classifiers are evaluated based on some standard performance measures which are defined as follows:

	p' (Predicted)	n' (Predicted)
p (Actual)	True Positive	False Negative
n (Actual)	False Positive	True Negative

Figure 2. Confusion matrix illustrating the true positives, false positives, true negative and false negatives

Recall is a fraction of relevant instances that are correctly predicted

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

Precision is a fraction of correctly predicted instances that are relevant

$$\text{Precision} = \frac{TP}{TP + FP}$$

F Beta Score(beta=1) is the harmonic mean of precision and recall

$$\text{F} = \frac{2}{\left(\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}\right)}$$

Accuracy is the fraction of total correct prediction to total number of predictions.

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

Metrics for each label are calculated and then they are averaged, weighted by support (the number of true instances for each label). This method accounts for label imbalance.

Note: it can result in an F-score that is not between precision and recall.

6. EXPERIMENT

6.1 Setup

Python with Numpy and Sci-Kit Learn libraries been used to perform the experiment on a Macintosh operating system.

10 Classifiers as mentioned in the previous sections including Naïve Bayes has been used on 10 software datasets. The datasets are taken from the Promise repository[2].

6.2 Results

Naïve Bayes classifier is compared to other classifiers for every dataset. A plot per performance metric i.e. for accuracy, precision, recall, F beta score and run time have been plotted as below.

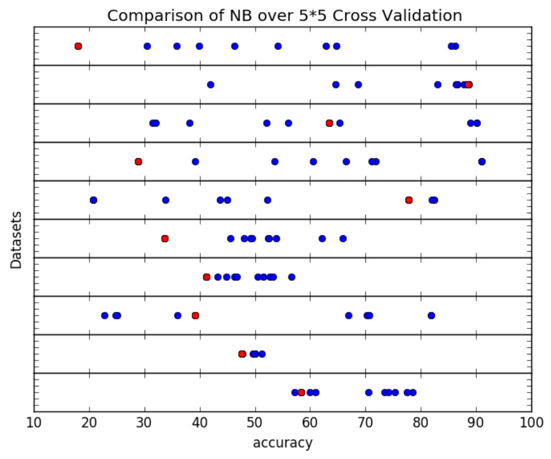

Figure 3. Accuracy comparison of 10 datasets on 10 classifiers 5×5 cross-validation. Red dot represents the Naïve Bayes classifier; blue dots represents other 9 classifiers. SMOTE has been performed on the training dataset only.

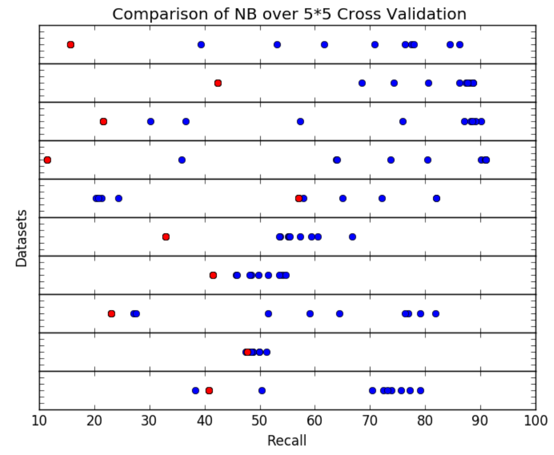

Figure 2. Recall comparison of 10 datasets on 10 classifiers 5×5 cross-validation. Red dot represents the Naïve Bayes classifier; blue dots represents other 9 classifiers. SMOTE has been performed on the training dataset only.

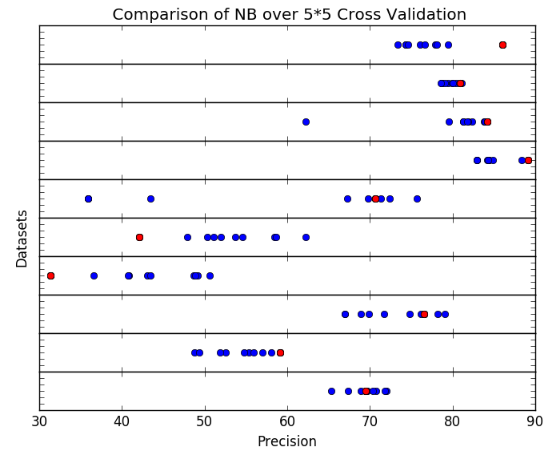

Figure 4. Precision comparison of 10 datasets on 10 classifiers 5×5 cross-validation. Red dot represents the Naïve Bayes classifier; blue dots represent other 9 classifiers.

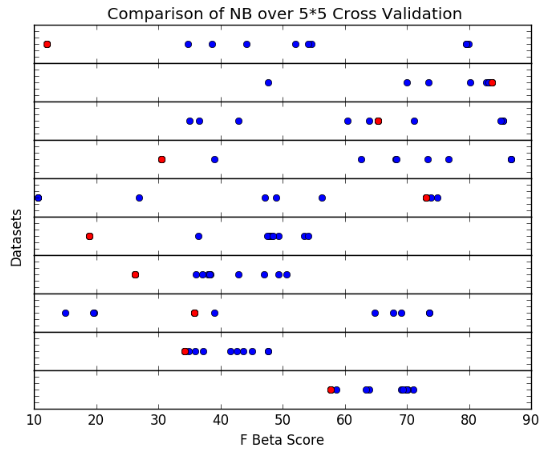

Figure 5. F Beta Score comparison of 10 datasets on 10 classifiers 5×5 cross-validation. Red dot represents the Naïve Bayes classifier; blue dots represent other 9 classifiers.

From the above plots, in most of the datasets accuracy of the NB classifier is worse than most classifiers. Likewise, recall of NB is also worse compared to most other classifiers. However, precision of NB is better than most of the classifiers. F Beta score(Beta=1) is worse for NB classifier. In some cases, F Beta score of NB is the worst compared to other classifiers. The performance of the NB classifier is inconsistent throughout 10 datasets but mostly performing worse than other classifiers.

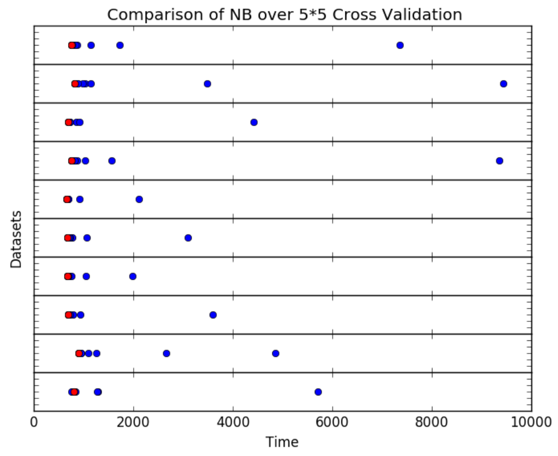

Figure 6. Run-time comparison of 10 datasets on 10 classifiers 5×5 cross-validation. Red dot represents the Naïve Bayes classifier; blue dots represent other 9 classifiers.

From the above run-time graph, NB classifier runs faster than always when compared to other classifiers. This is the only consistency we have seen for NB so far. We also plotted the performance metrics of the classifiers on datasets without applying SMOTE on them so that we could compared the performance of NB and other classifiers due to SMOTE.

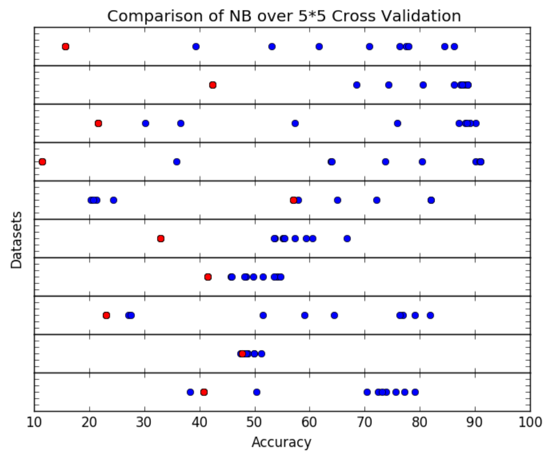

Figure 7. Accuracy comparison of 10 datasets on 10 classifiers 5×5 cross-validation with NO Smote.

Accuracy is much worse for NB compared to when we did SMOTE.

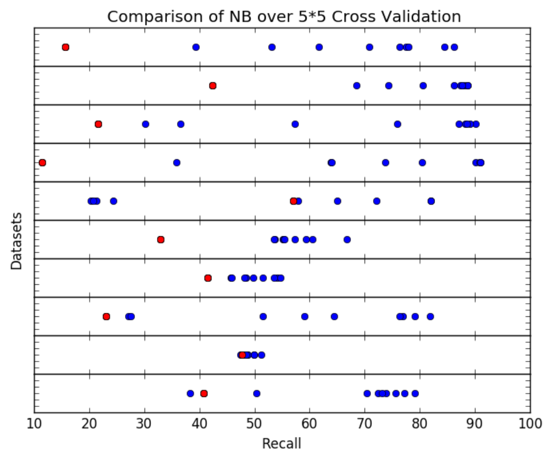

Figure 8. Recall comparison of 10 datasets on 10 classifiers 5×5 cross-validation with NO Smote.

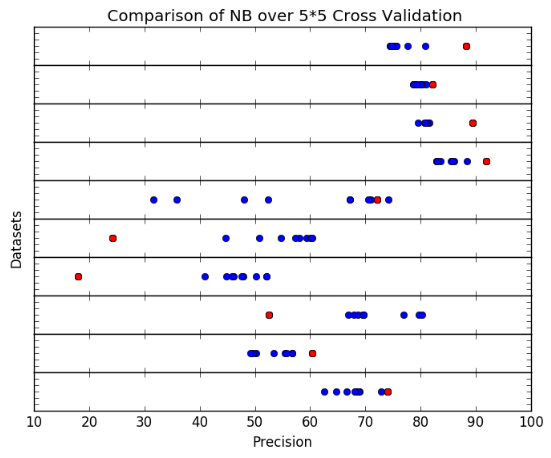

Figure 9. Precision comparison of 10 datasets on 10 classifiers 5×5 cross-validation with NO Smote.

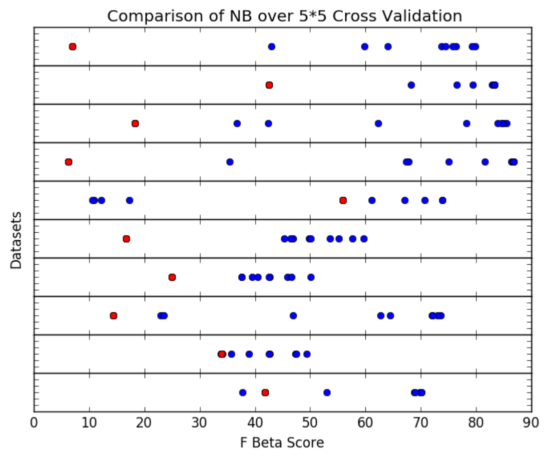

Figure 10. F beta score comparison of 10 datasets on 10 classifiers 5×5 cross-validation with NO Smote.

To understand the effect of Smote better, following plots showing the difference in the performance metrics of the classifiers for same datasets with and without Smote have been plotted. Where x-axis = Accuracy (Smote) - Accuracy (NO Smote) and, y-axis = different datasets.

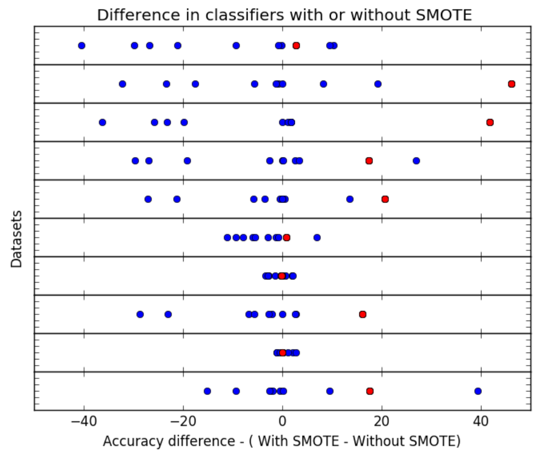

Figure 11. Accuracy difference comparisons of classifier with and without Smote on 10 datasets on 10 classifiers performing 5×5 cross-validation.

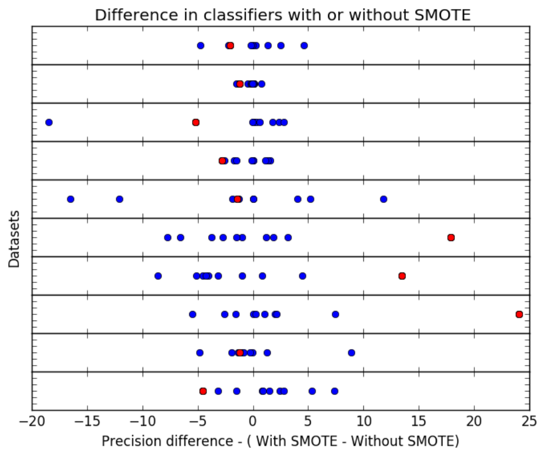

Figure 13. Precision difference comparisons of classifier with and without Smote on 10 datasets on 10 classifiers performing 5×5 cross-validation.

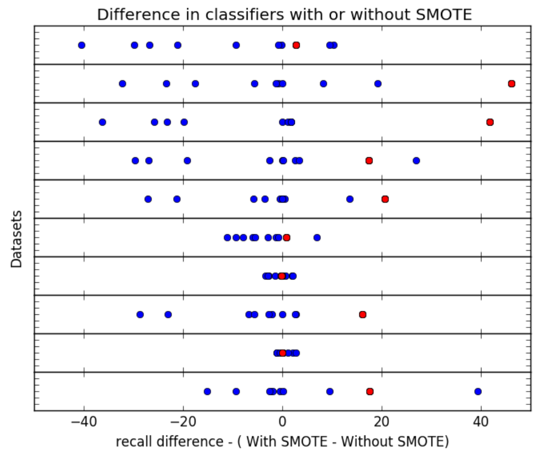

Figure 12. Recall difference comparisons of classifier with and without Smote on 10 datasets on 10 classifiers performing 5×5 cross-validation.

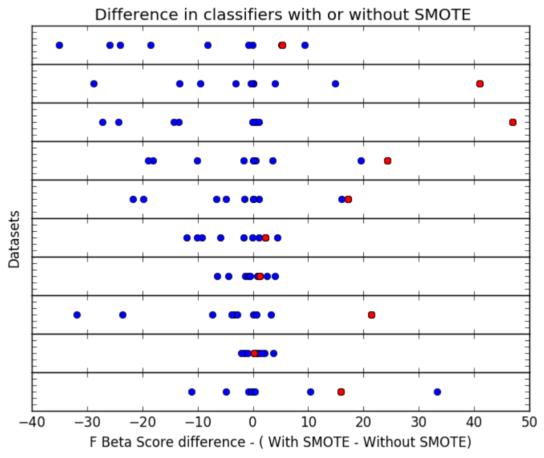

Figure 14, F beta score difference comparisons of classifier with and without Smote on 10 datasets on 10 classifiers performing 5×5 cross-validation.

From the performance difference plots of classifier on with and without Smote, performance of NB with Smote is either equal to and sometimes significantly greater performance of NB without Smote. However, precision different comparison in case of with Smote and without Smote should not be consider because precision is not so precise when Pos/Neg ratio is very less, (or data is skewed) as per Menzies, Tim, et al[5].

From the plots, it can be said that Smote improves performance of NB classifier. Hence Smote is a good preprocessing step for Naïve Bayes classifier. Taking this further and coloring some of the other classifiers which are performing worse with Smote.

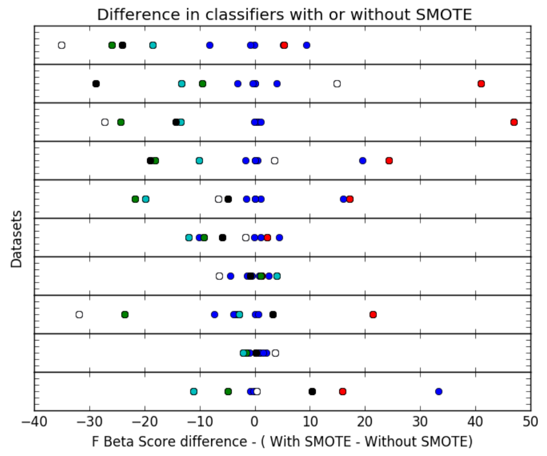

Figure 15. F Beta score difference comparisons of classifier with and without Smote on 10 datasets on 10 classifiers performing 5×5 cross-validation. Red=NB, Light Blue = Random Forests, Green= KNN, **Black**= CART, White=Perceptron, **Blue**=Others

Performance of all the other classifiers stays the same or deteriorates. Specifically, performance of KNN, CART, Random Forests and Perceptron deteriorate significantly. Remaining classifiers do not improve nor deteriorate significantly. Therefore, Smote is not suggested when applying classifiers like KNN, CART, Random Forests and Perceptron.

7. CONCLUSION

Naive Bayes classifier performs comparatively worse than most other complex classifiers on many of the CK metric datasets.

SMOTE helps improve performance of Naïve Bayes classifier on all CK metric datasets

SMOTE deteriorates the performance of KNN, CART, Random Forests and Perceptron significantly on all CK metric datasets.

SMOTE does not affect other classifiers (Artificial Neural Networks, LDA, QDA, Decision Tree, Support Vector Machine) significantly.

8. FUTURE WORK

- One or more of the following aspects can be considered for improving upon this work:
- Consider more classifiers and find out which classifiers to not use at all
- Consider more classifier and find out which of these should not be used with SMOTE
- Use tuned parameters for classifiers instead of default parameters

9. ACKNOWLEDGMENTS

We thank Dr. Tim Menzies for his encouragement to pursue this project. We would also like to thank Sushma Ravichandran for doing good work as a TA by helping us with homeworks during the semester.

10. REFERENCES

[1] Domingos, Pedro, and Michael Pazzani. "On the optimality of the simple Bayesian classifier under zero-one loss." *Machine learning* 29.2-3 (1997): 103-130.

[2] http://openscience.us/repo/defect/ck/

[3] Dougherty, James, Ron Kohavi, and Mehran Sahami. "Supervised and unsupervised discretization of continuous features." *Machine learning: proceedings of the twelfth international conference*. Vol. 12. 1995.

[4] Chawla, Nitesh V., et al. "SMOTE: synthetic minority over-sampling technique." *Journal of artificial intelligence research* 16 (2002): 321-357.

[5] Menzies, Tim, et al. "Problems with Precision: A Response to" Comments on'Data Mining Static Code Attributes to Learn Defect Predictors'"." *IEEE Transactions on Software Engineering* 33.9 (2007): 637.

[6] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." *Journal of Machine Learning Research* 12.Oct (2011): 2825-2830.

[7] Bird, Christian, et al. "Putting it all together: Using socio-technical networks to predict failures." *2009 20th International Symposium on Software Reliability Engineering* IEEE, 2009.

[8] D'Ambros, Marco, Michele Lanza, and Romain Robbes. "An extensive comparison of bug prediction approaches." *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 2010.

[9] Hall, Tracy, et al. "A systematic literature review on fault prediction performance in software engineering." *IEEE Transactions on Software Engineering* 38.6 (2012): 1276-1304.

[10] Nagappan, Nachiappan, Thomas Ball, and Andreas Zeller. "Mining metrics to predict component failures." *Proceedings of the 28th international conference on Software engineering*. ACM, 2006.

[11] Radjenović, Danijel, et al. "Software fault prediction metrics: A systematic literature review." *Information and Software Technology* 55.8 (2013): 1397-1418.

[12] Shepperd, Martin, David Bowes, and Tracy Hall. "Researcher bias: The use of machine learning in software defect prediction." *IEEE Transactions on Software Engineering* 40.6 (2014): 603-616.

[13] Chidamber, Shyam R., and Chris F. Kemerer. "A metrics suite for object oriented design." *IEEE Transactions on software engineering* 20.6 (1994): 476-493.

[14] Kim, Sunghun, et al. "Predicting faults from cached history." *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 2007.

[15] Wang, Shuo, and Xin Yao. "Using class imbalance learning for software defect prediction." *IEEE Transactions on Reliability* 62.2 (2013): 434-443.

[16] Catal, Cagatay, and Banu Diri. "A systematic review of software fault prediction studies." *Expert systems with applications* 36.4 (2009): 7346-7354.

[17] Song, Qinbao, et al. "A general software defect-proneness prediction framework." *IEEE Transactions on Software Engineering* 37.3 (2011): 356-370.