Carnegie Mellon University

Machine Learning Algorithms for Software Bug Prediction

Zhenjie Jia, Xinyi He

Carnegie Mellon University

Nikolaos Ploskas

University of Western Macedonia

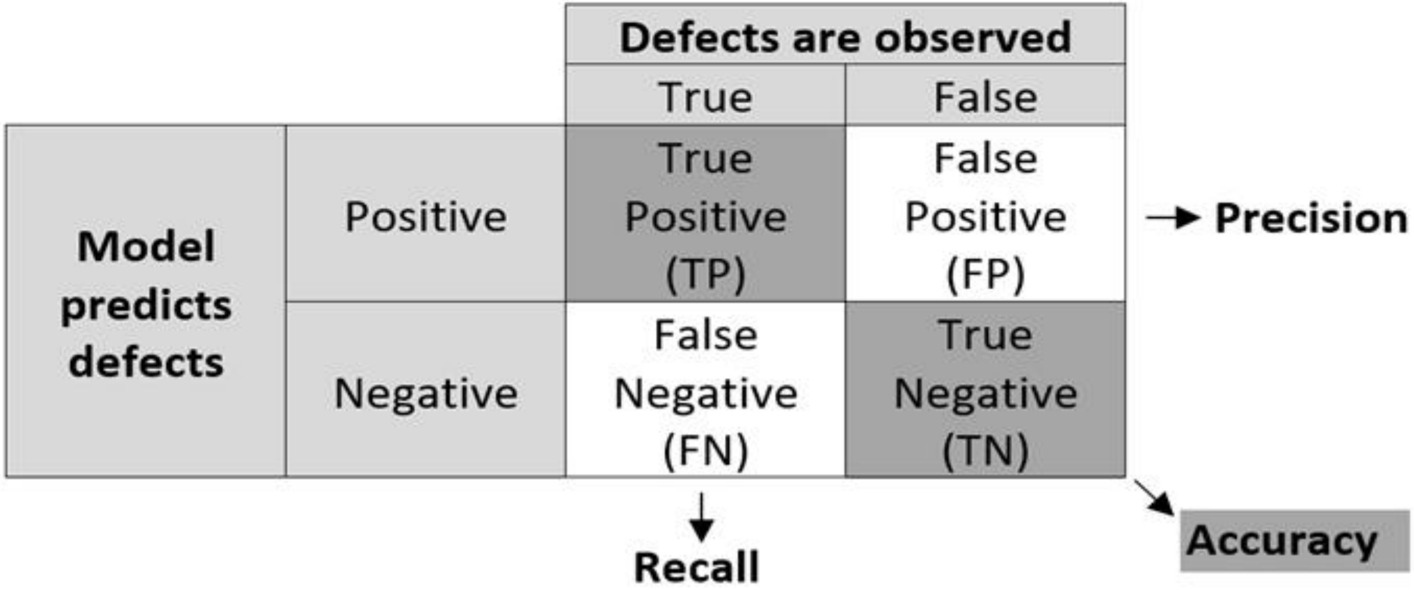
Nick Sahinidis

Carnegie Mellon University

MOTIVATION

- The effort in finding and fixing bugs in a software will consume near 80% of the budget of a software development
- Large open-source software have many independent features which make it become a good target in machine learning
- Version control tools and website like github makes it possible to extract features and form the dataset

RESEARCH GOAL



Previous work

1. D'Ambros's data set (D'Ambros et al., 2010)

Software: Eclipse JDT Core, Eclipse PDE UI, Equinox Framework, Lucene, Mylyn

Features: 1) change features: NR, NREF, NFIX, NAUTH, LINES…

2) code features: DIT, NOC, FanIn, FanOut,NOM,NOA...

2. Zimmermann's data set (Zimmermann et al., 2007)

Software: Eclipse release version: 2.0, 2.1, 3.0

Features: 1) complexity metrics: FOUT, PAR, NSF, MLOC…

2) structure of abstract syntax tree(s): frequency of each of nodes

Carnegie Mellon University

Previous Work

Couto et al. (2014) used multilayer perceptron with different activation functions to achieve an average recall ranging from 13% (Equinox)to 31% (Lucene). The max recall rate they got for the Eclipse JDT, Eclipse PDE UI, Equinox,and Lucene systems were 68%, 44%,31%, and 52%, respectively on D’Ambros’s dataset (D'Ambros et al., 2010).

Zimmermann et al. (2007) used logistic regression on their own dataset and achieved Accuracy ranging from 86.4% to 90% with Recall rate ranging from 17.1% to 27.7%.

Algorithms

Logistic regression

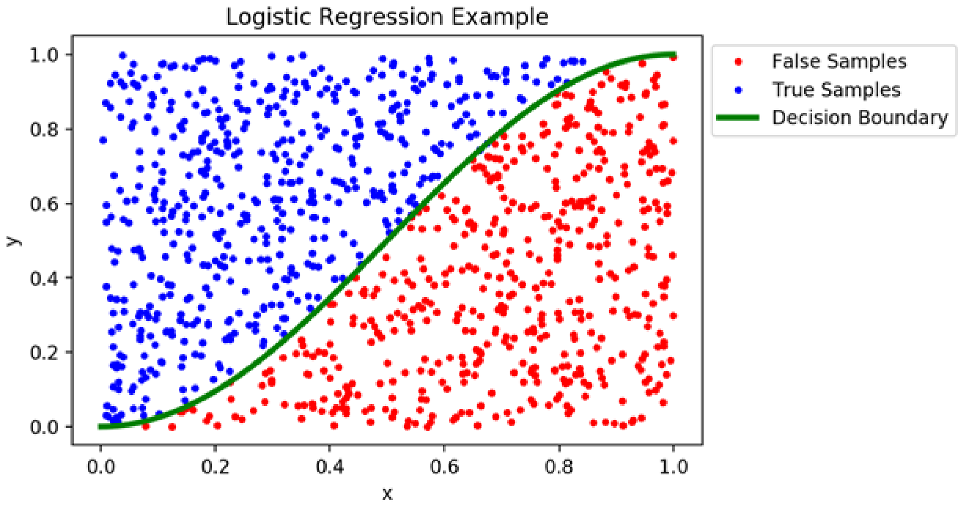
$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$Cost(h_\theta(x), y) = -log(h_\theta(x)) \quad if\ y = 1$$

$$Cost(h_\theta(x), y) = -log(1 - h_\theta(x)) \quad if\ y = 0$$

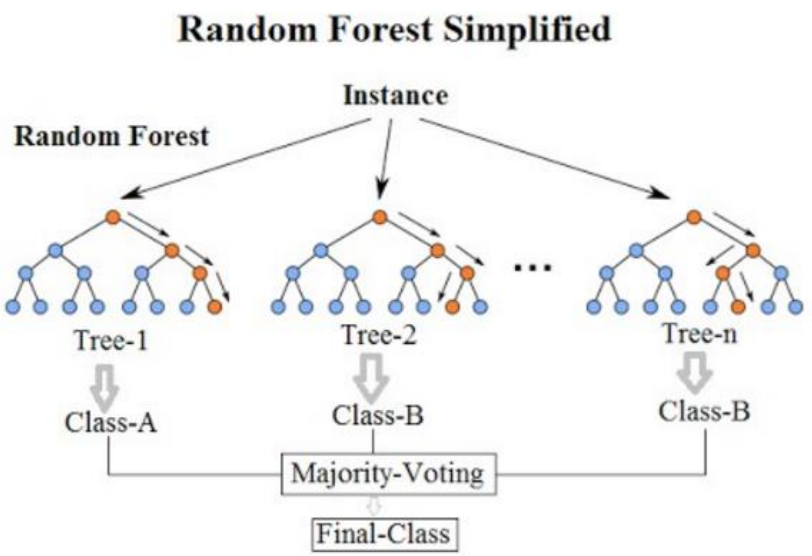
$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$



Carnegie Mellon University

Algorithms

Random forest

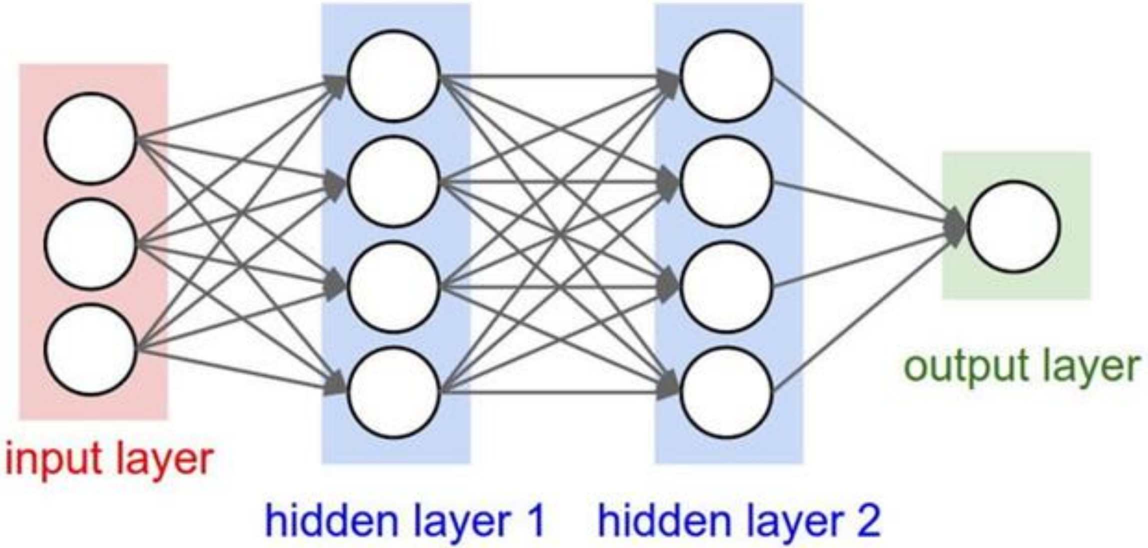


Random Forest (Koehrsen, 2017)

Carnegie Mellon University

Algorithms

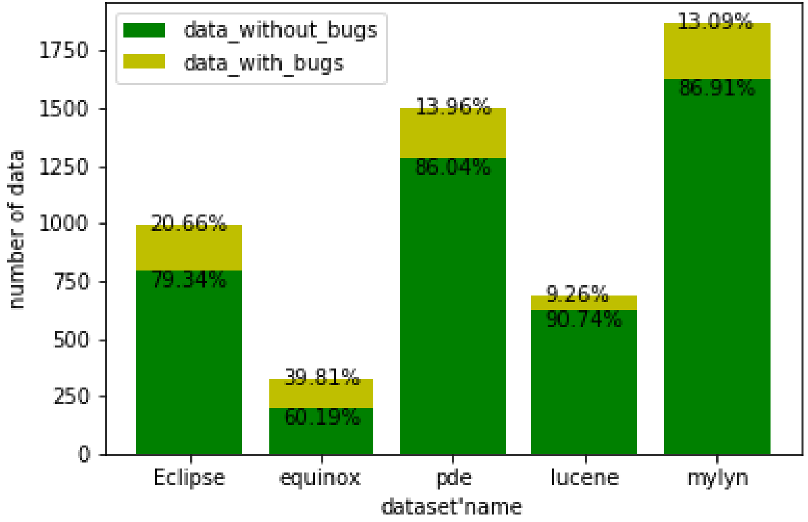
Multilayer perceptron (MLP)



Multilayer perceptron (Venelin, 2017)

Dataset

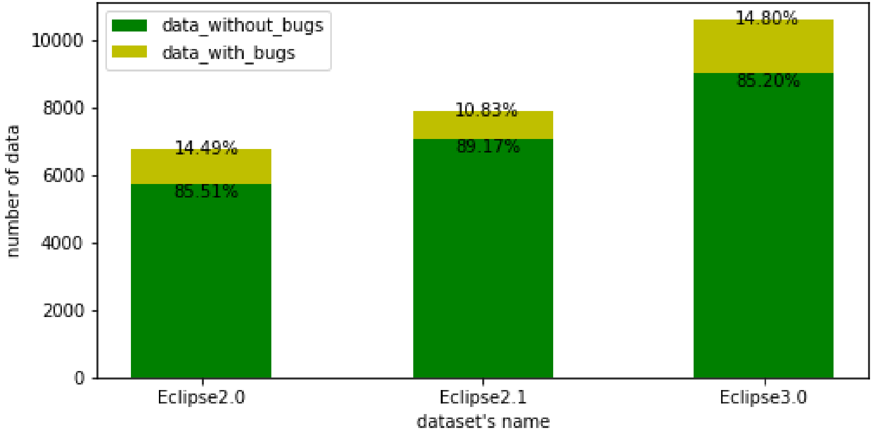
D'Ambros's data set (D'Ambros et al.. 2010)



Distribution of D'Ambros's dataset (D'Ambros et al., 2010)

Dataset

Zimmermann's data set (Zimmermann et al., 2007)



Distribution of Zimmermann's data set (Zimmermann et al., 2007)

Carnegie Mellon University

Data Preprocessing

Over sample: Add more copies of the minority class.

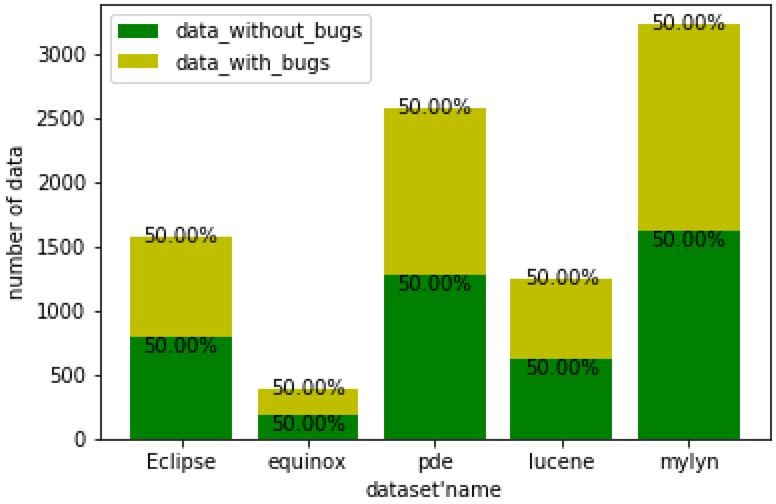
Under sample: Remove some observations of the majority class.

Syn sample: Create synthetic samples.

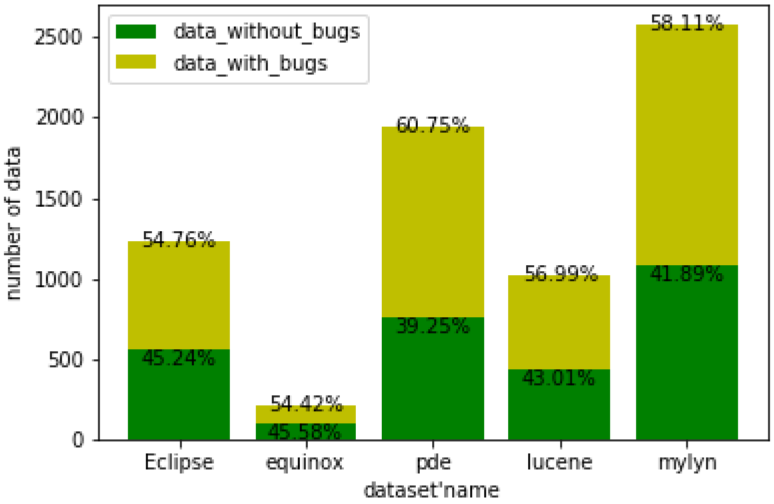
Standardization: Let the distribution of the data follow a normal distribution.

Data Preprocessing

Distribution of the dataset after Preprocessing



Distribution of D'Ambros's dataset (D'Ambros et al., 2010) after over sampling



Distribution of D'Ambros's dataset (D'Ambros et al., 2010) after syn sampling

Carnegie Mellon University

Results on Zimmermann's dataset

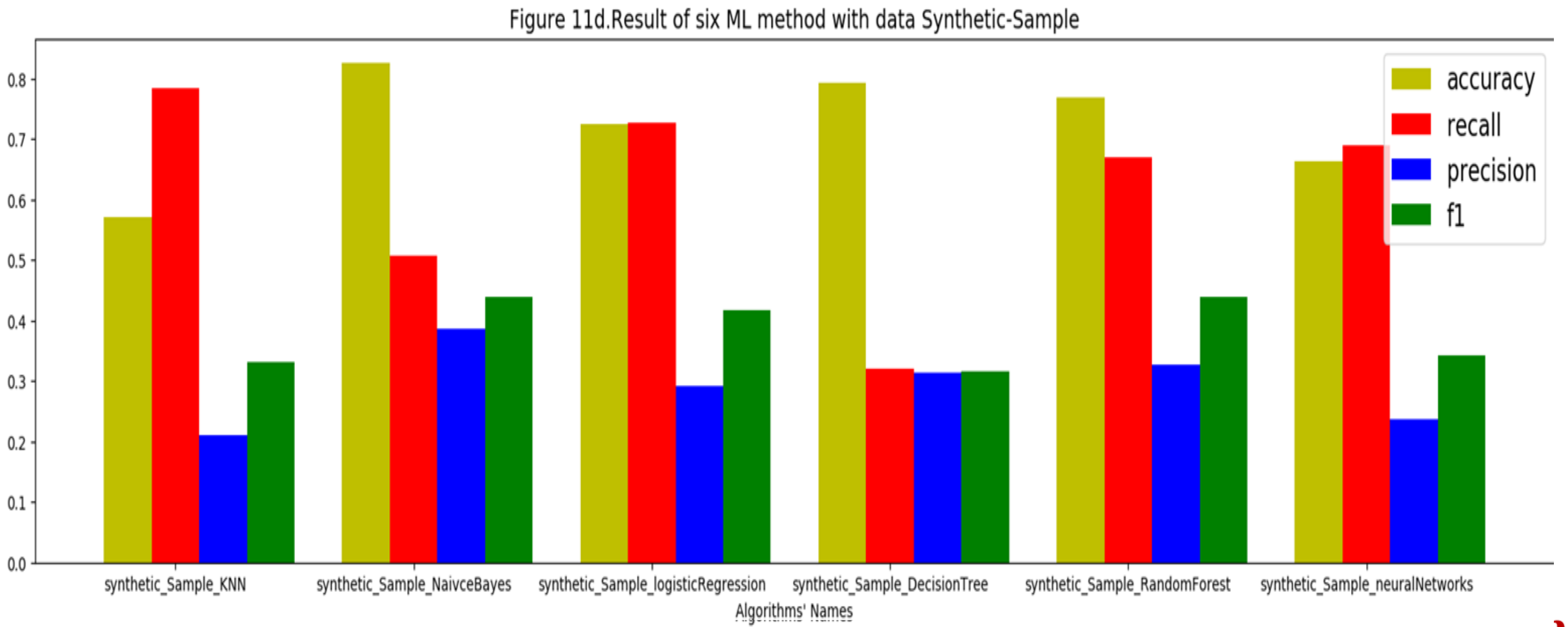
1) training and testing on the same project

Algorithm name	Parameters	Dataset	Range of Parameters
K-nearest neighbors	n_neighbors=3, weights = 'distance', algorithm='auto', leaf_size=30, p=2, metric='minkowski'	Zimmermann's data set(Zimmermann, 2007)	n_neighbors = [1:9], weights = {'uniform','distance'}, eaf_size=[20:50], p = [1,2]
Naive Bayes	GaussianNB()	Zimmermann's data set(Zimmermann, 2007)	BernoulliNB() GaussianNB()
Logistic Regression	solver='liblinear', max_iter=1000, class_weight='balanced', penalty='l2', dual=False, tol=0.0001, C=1.0	Zimmermann's data set(Zimmermann, 2007)	solver = {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'} C = {0.1, 1.0, 10.0} max_iter = {100, 1000} class_weight = {'balanced',None}
Decision Tree	random_state=0, class_weight='balanced', criterion='gini', splitter='best'	Zimmermann's data set(Zimmermann, 2007)	criterion = {'gini', 'entropy'} class_weight = {'balanced',None}
Random Forest	n_estimators = 200, criterion = 'entropy'	Zimmermann's data set(Zimmermann, 2007)	n_estimators = {10, 20, 50, 100, 200} criterion = {'gini', 'entropy'}
Neural Networks	solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(100, 100), activation='relu', alpha=1e-05, batch_size='auto', learning_rate='constant', shuffle=True	Zimmermann's data set(Zimmermann, 2007)	hidden_layer_sizes = {(50,), (100,), (200,), (50, 50), (100, 100), (200, 200)} solver = {'lbfgs', 'sgd', 'adam'} alpha = {1e-3, 1e-4, 1e-5}

Parameters for different algorithms

Results on D'Ambros's dataset

1) training and testing on the same project: eclipse



Results on D'Ambros's dataset

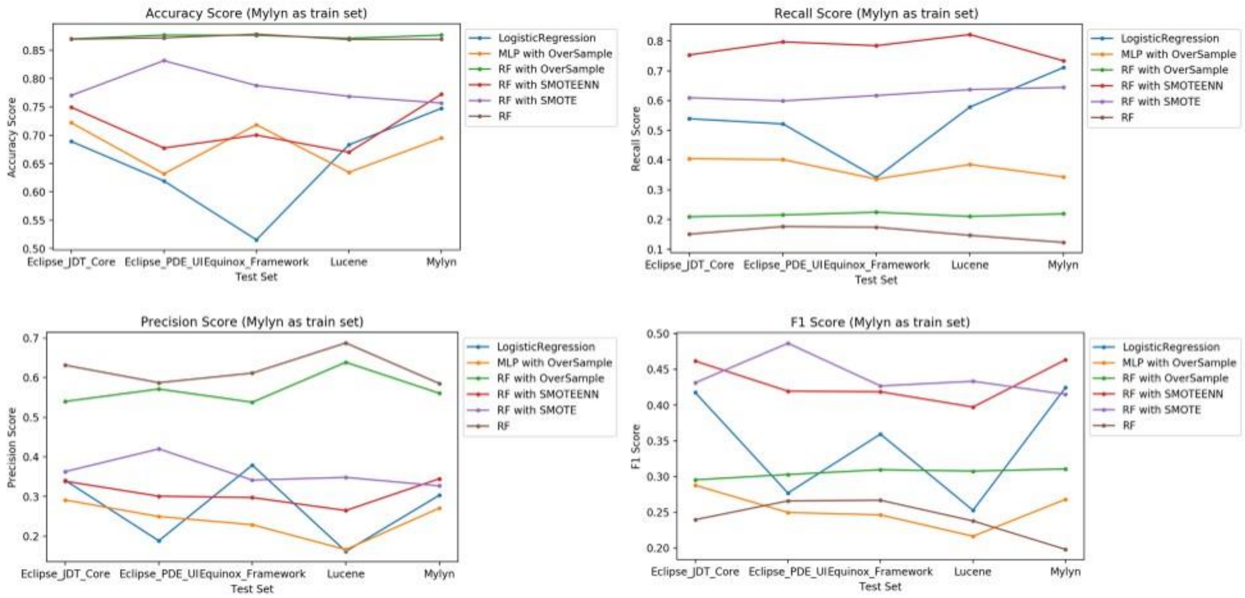
2) training and testing on the different projects

Algorithm name	Parameters	Dataset	Range of Parameters
LogisticRegression	solver = 'liblinear', C = 1.0,max_iter = 1000, class_weight = 'balanced'	D'Ambros's dataset (D'Ambros, 2010)	solver = {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'} C = {0.1, 1.0, 10.0} max_iter = {100, 1000}
MLP with OverSample	solver = 'lbfgs', alpha = 1e-5, hidden_layer_sizes = (100, 100)	D'Ambros's dataset (D'Ambros, 2010)	hidden_layer_sizes = {(50,), (100,), (200,), (50, 50), (100, 100), (200, 200)} solver = {'lbfgs', 'sgd', 'adam'} alpha = {1e-3, 1e-4, 1e-5}
RF	n_estimators = 200, criterion = 'entropy'	D'Ambros's dataset (D'Ambros, 2010)	n_estimators = {10, 20, 50, 100, 200} criterion = {'gini', 'entropy'}
RF with OverSample	n_estimators = 200, criterion = 'entropy'	D'Ambros's dataset (D'Ambros, 2010)	n_estimators = {10, 20, 50, 100, 200} criterion = {'gini', 'entropy'}
RF with SMOTEENN	n_estimators = 200, criterion = 'entropy'	D'Ambros's dataset (D'Ambros, 2010)	n_estimators = {10, 20, 50, 100, 200} criterion = {'gini', 'entropy'}
RF with SMOTE	n_estimators = 200, criterion = 'entropy'	D'Ambros's dataset (D'Ambros, 2010)	n_estimators = {10, 20, 50, 100, 200} criterion = {'gini', 'entropy'}

Parameters for different algorithms

Results on D'Ambros's dataset

2) training and testing on the different projects



Accuracy, Recall, Precision and F1 score with different algorithms using Mylyn as train set with different projects on D'Ambros's dataset (D'Ambros, 2010)

Carnegie Mellon University

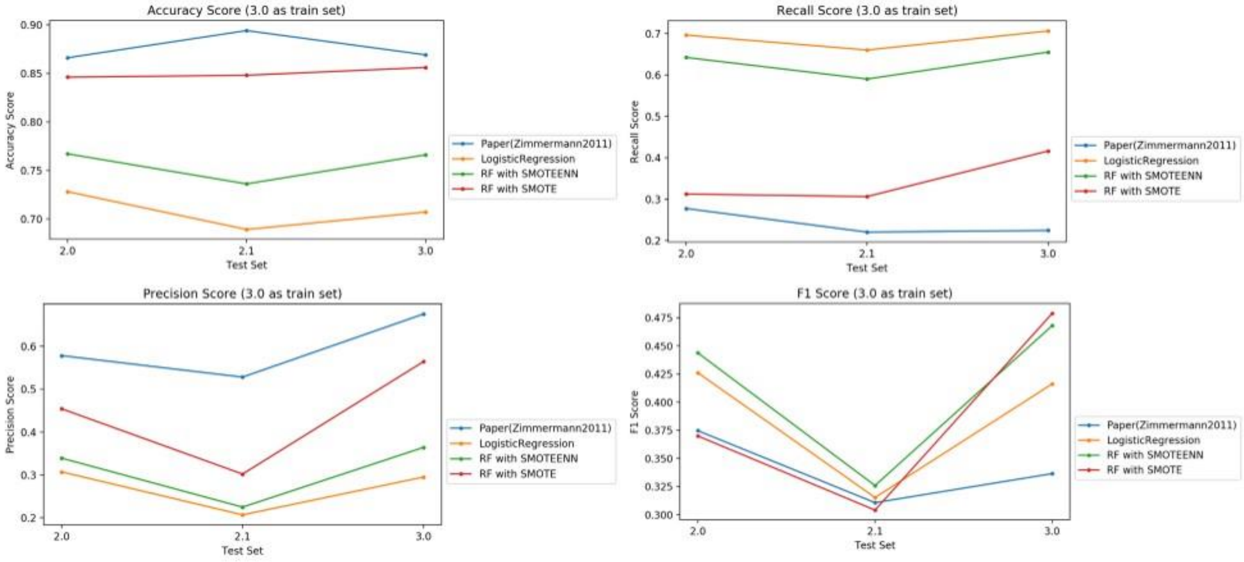
Result on Zimmermann’s data set

Algorithm name	Parameters	Dataset	Range of Parameters
LogisticRegression	solver = 'liblinear', C = 1.0,max_iter = 1000, class_weight = {0: 0.125, 1: 0.875}	Zimmermann’s data set(Zimmermann, 2007)	solver = {‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’} C = {0.1, 1.0, 10.0} max_iter = {100, 1000}
Random Forest with SMOTE	n_estimators = 200, criterion = 'entropy'	Zimmermann’s data set(Zimmermann, 2007)	n_estimators = {10, 20, 50, 100, 200} criterion = {‘gini’, 'entropy'}
Random Forest with SMOTEENN	n_estimators = 200, criterion = 'entropy'	Zimmermann’s data set(Zimmermann, 2007)	n_estimators = {10, 20, 50, 100, 200} criterion = {‘gini’, 'entropy'}

Parameters for different algorithms

Carnegie Mellon University

Result on Zimmermann’s data set



Accuracy, Recall, Precision and F1 score with different algorithms using Eclipse 3.0 as train set with different versions of Eclipse on Zimmermann’s data set (Zimmermann, 2007)

Training	Testing	Defects	Accuracy_score	Recall_score	Precision_score	F1_score
2	2	0.145	0.876	0.265	**0.692**	0.383
			0.785	0.707	0.416	0.524
			0.8	**0.787**	0.37	0.503
			0.884	0.52	0.654	**0.579**
	2.1	0.108	**0.89**	0.191	**0.478**	0.273
			0.718	**0.629**	0.22	0.326
			0.758	0.58	0.242	**0.342**
			0.844	0.36	0.31	0.333
	3	0.148	**0.861**	0.171	**0.613**	0.267
			0.724	**0.639**	0.298	0.406
			0.765	0.583	0.332	**0.423**
			0.838	0.368	0.442	0.402
2.1	2	0.145	**0.87**	0.203	**0.664**	0.311
			0.816	**0.554**	0.401	**0.465**
			0.8	0.535	0.361	0.431
			0.846	0.23	0.441	0.302
	2.1	0.108	**0.9**	0.16	**0.668**	0.258
			0.805	0.543	0.295	**0.382**
			0.781	**0.631**	0.251	0.359
			0.893	0.305	0.485	0.374
	3	0.148	**0.864**	0.139	**0.717**	0.233
			0.778	0.488	0.331	0.394
			0.771	**0.513**	0.326	**0.399**
			0.84	0.226	0.424	0.295
3	2	0.145	**0.866**	0.277	**0.578**	0.375
			0.728	**0.696**	0.307	0.426
			0.767	0.642	0.339	**0.444**
			0.846	0.312	0.454	0.370
	2.1	0.108	**0.894**	0.22	**0.528**	0.311
			0.689	**0.66**	0.207	0.315
			0.736	0.59	0.225	**0.326**
			0.848	0.306	0.302	0.304
	3	0.148	**0.869**	0.224	**0.675**	0.336
			0.707	**0.706**	0.295	0.416
			0.766	0.655	0.364	0.468
			0.856	0.416	0.564	**0.479**

Paper (Zimmermann2011)	LogisticRegression (solver = 'liblinear', max_iter = 1000, class_weight = {0: 0.125, 1: 0.875})
RandomForestClassifier (n_estimators = 200, criterion = 'entropy') with SMOTEENN	RandomForestClassifier (n_estimators = 200, criterion = 'entropy') with SMOTE

Carnegie Mellon University

Conclusion

- All machine learning algorithms can achieve 70% - 85% accuracy score without any data preprocessing methods.
- Logistic regression, random forest and neural networks behave better after syn sampling within the same open source software on D'Ambros' dataset.
- The f1 score cross the projects when using Eclipse_JDT_Core, Equinox_Framework as train set are better on D'Ambros' dataset.
- the recall score and f1 score after using random forest with data preprocessing on Zimmermann's data are much better than Zimmermann's result.

Carnegie Mellon University

References

Thank you to Dr. Nick Sahinidis and Dr. Ploskas

[1] Tassey, G. (2002). The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology, RTI Project*, *7007*(011), 429-489.

[2] Akiyama, F. (1971, August). An Example of Software System Debugging. In *IFIP Congress (1)* (Vol. 71, pp. 353-359).

[3] Kamei, Y., & Shihab, E. (2016, March). Defect prediction: Accomplishments and future challenges. In *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)* (Vol. 5, pp. 33-45). IEEE.

[4] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2016, May). Automated parameter optimization of classification techniques for defect prediction models. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)* (pp. 321-332). IEEE.

[5] Tóth, Z., Gyimesi, P., & Ferenc, R. (2016, July). A public bug database of github projects and its application in bug prediction. In *International Conference on Computational Science and Its Applications* (pp. 625-638). Springer, Cham.

[6] Xia, X., Lo, D., Wang, X., Yang, X., Li, S., & Sun, J. (2013, March). A comparative study of supervised learning algorithms for re-opened bug prediction. In *2013 17th European Conference on Software Maintenance and Reengineering* (pp. 331-334). IEEE.

References

[7] Zhang, F., Zheng, Q., Zou, Y., & Hassan, A. E. (2016, May). Cross-project defect prediction using a connectivity-based unsupervised classifier. In *Proceedings of the 38th International Conference on Software Engineering* (pp. 309-320). ACM.

[8] Couto, C., Pires, P., Valente, M. T., Bigonha, R. S., & Anquetil, N. (2014). Predicting software defects with causality tests. *Journal of Systems and Software*, *93*, 24-41.

[9] Osman, H., Ghafari, M., & Nierstrasz, O. (2017, February). Hyperparameter optimization to improve bug prediction accuracy. In *2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)* (pp. 33-38). IEEE.

[10] Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K. I., Adams, B., & Hassan, A. E. (2010, September). Revisiting common bug prediction findings using effort-aware models. In *2010 IEEE International Conference on Software Maintenance* (pp. 1-10). IEEE.

[11] Shivaji, S., Whitehead Jr, E. J., Akella, R., & Kim, S. (2009, November). Reducing features to improve bug prediction. In *2009 IEEE/ACM International Conference on Automated Software Engineering* (pp. 600-604). IEEE.

[12] Zimmermann, T., Premraj, R., & Zeller, A. (2007, May). Predicting defects for eclipse. In *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)* (pp. 9-9). IEEE.

[13] Jureczko, M., & Madeyski, L. (2010, September). Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering* (p. 9). ACM.

References

[14] Kim, S., Zhang, H., Wu, R., & Gong, L. (2011, May). Dealing with noise in defect prediction. In *2011 33rd International Conference on Software Engineering (ICSE)* (pp. 481-490). IEEE.

[15] Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W. M., Ohira, M., Adams, B., ... & Matsumoto, K. I. (2010, October). Predicting re-opened bugs: A case study on the eclipse project. In *2010 17th Working Conference on Reverse Engineering* (pp. 249-258). IEEE.

[16] Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W. M., Ohira, M., Adams, B., ... & Matsumoto, K. I. (2013). Studying re-opened bugs in open source software. *Empirical Software Engineering*, *18*(5), 1005-1042.

[17] D'Ambros, M., Lanza, M., & Robbes, R. (2010, May). An extensive comparison of bug prediction approaches. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)* (pp. 31-41). IEEE.

[18] Shirabad, J. S., & Menzies, T. J. (2005). The PROMISE repository of software engineering databases. School of Information Technology and Engineering, University of Ottawa, Canada, 24.

[19] Koehrsen, W. (2017). Random Forest Simple Explanation. Published in www.medium.com. Last accessed on 10/23/2019.

[20] Venelin, V. (2017). Creating a Neural Network from Scratch — TensorFlow for Hackers (Part IV). Published in www.medium.com. Last accessed on 10/29/2019.

[21] Zohar, K., & Amir, S. (2018). Amazon SageMaker supports kNN classification and regression. Published in www.medium.com. Last accessed on 10/29/2019.

Carnegie Mellon University