Tackling the Class Imbalance Challenge for Software Defect Type Classification using Knowledge Augmented Method

Sangameshwar Patil
TCS Research and Dept. of CSE, IIT Madras
India
sangameshwar.patil@tcs.com,sangam@cse.iitm.ac.in

B. Ravindran
Dept. of CSE and RBC DSAI, IIT Madras
India
ravi@cse.iitm.ac.in

ABSTRACT

Automatically predicting the defect type from a software defect report is an important task for improving the software defect management process. State-of-the-art solutions for this task are based on supervised machine learning and its variants. These algorithms are susceptible to the imbalanced learning challenge (also known as the class imbalance problem). When the label distribution in training data is skewed, they are prone to perform poorly on the minority classes. This leads to imbalanced learning of the classifier model.

In this paper, we evaluate the feasibility of using the concept-based classification (CBC) approach to tackle this imbalanced learning challenge in the automated software defect type prediction task. CBC approach leverages a knowledge augmented representation of the documents to be classified and labels using concept-based semantic representation. As the CBC approach does not need labelled training data, it is able to treat each software defect type label without getting biased by the dominant classes in case of class-imbalanced datasets. Using experimental evaluation on real-life datasets, we show that the CBC approach is able to tackle the class imbalance better than the supervised machine learning algorithms for software defect type prediction task and provides a viable, knowledge augmented learning approach for the imbalanced learning problem.

CCS CONCEPTS

• Software and its engineering → Software defect analysis; **Maintaining software**; **• Computing methodologies → Machine learning**; **Natural language processing**.

KEYWORDS

Imbalanced Learning, Class Imbalance Problem, Software Defect Analysis, Concept-based Classification, Knowledge Augmented Learning

ACM Reference Format:
Sangameshwar Patil and B. Ravindran. 2023. Tackling the Class Imbalance Challenge for Software Defect Type Classification using Knowledge Augmented Method. In *Proceedings of Second Workshop on Knowledge Augmented Methods for Natural Language Processing, in conjunction with KDD 2023 (KNLP at KDD'23).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Software defect type prediction is an important problem in the software defect management process. The ability to automatically predict the software defect type from a software defect report can significantly help in improving the efficiency and reducing the cost of software defect management process. Different machine learning techniques such as supervised learning, semi-supervised learning, active learning, etc. have been proposed for this task [14, 15]. These techniques need training data labeled by human experts to learn the classification models. In many real-life software projects, such labeled training datasets suffer from the *class imbalance* in their defect datasets. For instance, Table 1 shows the defect type label distributions in two datasets classified using two different defect classification schemes. (Details of these datasets are elaborated in the Section 3.) In case of a class-imbalanced dataset, the instances belonging to one or a very few classes significantly outnumber the instances of other classes.

One of the limitations of the supervised learning algorithms is that they are susceptible to the *imbalanced learning problem* (i.e., the *class imbalance problem*). In the context of machine learning, this problem [2, 9] refers to the undesirable effects of skewed label distributions in training data on the classification algorithms. These algorithms tend to perform well when the number of instances belonging to each class in the training data are roughly similar to each other. The class-imbalance can potentially bias the classification algorithms to focus on the classes having dominant majority. As a result, performance of these algorithms can suffer on the classes having very few instances in the training data.

Concept-based classification (CBC) of software defect reports [11] has been proposed as an alternative to the traditional supervised learning based approaches for software defect type prediction. CBC approach leverages a *knowledge-augmented representation* of the documents to be classified and labels using concept-based semantic representation. CBC approach does not need labeled training data; instead it uses the text description of the software defect report and a set of keywords describing the software defect types. CBC leverages the human-curated knowledge in Wikipedia by treating each Wikipedia article as a concept. CBC projects these text snippets in a concept-space spanned by the Wikipedia articles. Similarity

Accepted to Second Workshop on Knowledge Augmented Methods for Natural Language Processing, in conjunction with KDD 2023.

Table 1: Dataset statistics and class label distribution.

Defect Classification Scheme	Defect type family	Data Set	
		Apache-Libs	Roundcube
ODC-based scheme (Table I of [14])	Control and Data flow	287	394
	Structural	98	76
	Non Code	115	55
	Total	500	525
IEEE-based scheme (Table 2)	Interface	98	102
	Logic and Data	288	347
	Description	35	4
	Build-Config-Install	57	48
	Standards	16	6
	Syntax	6	18
	Total	500	525

between the concepts shared by the projects of the software defect report and the defect type class labels is used to make the classification decisions. This allows the CBC approach to avoid the need of labeled training data to carry out software defect type classification by relying solely on the knowledge augmented representation.

In this paper, we focus on following two research questions:

- RQ_1: Do the existing benchmark algorithms for software defect type classification [15] based on the supervised machine learning (ML) paradigm suffer from the *class imbalance problem*?
- RQ_2: Can concept-based classification (CBC) provide a feasible option for tackling the class imbalance problem, as the CBC approach does not depend on the labeled training data?

Rest of the paper is organized as follows: In Section 2, we give a brief overview of the concept-based classification approach. Details of the datasets and experimental setup are provided in the Section 3. The evaluation of the results and their analysis with respect to the research questions posed is provided in Section 4. Finally, we conclude in Section 5.

2 CONCEPT-BASED CLASSIFICATION OF SOFTWARE DEFECT REPORTS

Concept-based Classification (CBC) has been motivated by the observation that unlike the supervised learning algorithms, the human experts do not need a large number of labeled examples to carry out classification of software defects. Humans seem to make use of the inherent conceptual knowledge about the software defect reports and the defect type class labels to estimate the similarity between them. This intuition has been formalized in the notion of explicit semantic analysis (ESA) [5]. Concept-based classification approach for software defect type identification [11] is based on ESA. An article in the Wikipedia is considered to be equivalent to a concept grounded in human coginition. The Wikipedia is treated as a surrogate knowledge representation of the human concept-space.

Let $\mathcal{X}$ be the set of software defect reports, i.e., $\mathcal{X} = \{x_i | x_i$ is the textual description of i^{th} software defect$\}$. $\mathcal{Y}_d$ denotes the set of software defect type labels along with their descriptions to be used in classification, i.e., $\mathcal{Y}_d = \{(y_i, d_i) \mid y_i$ denotes the software defect type label and d_i is the set of keywords and phrases from textual description of y_i in the software defect classification scheme $\}$. Let $\mathcal{Y} = \{y_1, y_2, \ldots, y_m\}$ be the set of software defect type labels used for classification. Note that each defect type label $y_i \in \mathcal{Y}$ has a corresponding entry $(y_i, d_i) \in \mathcal{Y}_d$. Let N be the maximum number of concepts (i.e., Wikipedia articles) to be used in the concept-based representations of a defect report or a defect type label.

Then, the concept-based representations of the defect reports and the defect type labels are computed using ESA as follows – Let E_x denote the sparse vector representation of a defect report x computed using ESA. $E_x = \, < (c_j, p_{jx}) >$. It is composed of at most N tuples in the decreasing order of the concept importance score, p_{jx}. In a tuple entry $(c_j, p_{jx}) \in E_x$, the score p_{jx} quantifies the strength of association between the concept (i.e., Wikipedia article) c_j and the defect report x. It is computed as $p_{jx} = \sum_{t \in x} \tau(t, c_j)$; where t is a term in the defect report x and $\tau(t, c_j)$ is the term-weight of t in the Wikipedia article c_j, computed using the term-weighting function τ. We use BM25 [13] as the term-weighting function τ. Similarly, for a software defect type label y_i and its description d_i, the concept-based representation computed by ESA is $E_{y_i} = \, < (c_j, q_{jy_i}) >$. The importance score q_{jy_i} gives the degree of relevance between concept c_j and the defect type label y_i. The importance score is computed as $q_{jy_i} = \sum_{t \in d_i} \tau(t, c_j)$; where t is a term in d_i, the description of software defect type label y_i and $\tau(t, c_j)$ is the term-weight of t in concept (i.e., Wikipedia article) c_j.

Once the concept-based representations have been computed, the degree of relevance of a defect type label for a given sofware defect is computed using cosine similarity. To make the final classification decision, the defect type class label having the highest cosine similarity with the concept-based representation of the defect report x is used.

$$CBC(x, \mathcal{Y}, \mathcal{Y}_d) =_{y_i \in \mathcal{Y}} \frac{E_x \cdot E_{y_i}}{\|E_x\| \cdot \|E_{y_i}\|}$$
$$=_{y_i \in \mathcal{Y}} \frac{\sum\limits_{c_j \in E_x \,\wedge\, c_j \in E_{y_i}} p_{jx} \cdot q_{jy_i}}{\sqrt{\sum\limits_{c_j \in E_x} p_{jx}^2} \cdot \sqrt{\sum\limits_{c_j \in E_{y_i}} q_{jy_i}^2}} \tag{1}$$

3 EXPERIMENTAL SETUP

3.1 Datasets

We use two software defect datasets from real-life software projects and two different software defect classification schemes for experimental evaluation of the research questions. The datasets are available on request for research purpose. The first dataset, *Apache-Libs*, is the standard software defect dataset used by Thung et al. [14, 15] to establish and benchmark the state-of-the-art for the software defect type classification. The Apache-Libs dataset contains a total of 500 defects from Apache JIRA repositories of three open-source libraries. It consists of (i) 200 randomly sampled defects from Mahout, (ii) 200 randomly sampled defects from Lucene, and (iii) 100 randomly sampled defects from OpenNLP. These libraries are

https://issues.apache.org/jira/issues/
Mahout, the machine learning library, https://mahout.apache.org
Lucene, the search engine library https://lucene.apache.org/core
OpenNLP, the natural language processing library https://opennlp.apache.org

Table 2: The software defect type families based on Table A.1 (Annexure A) of IEEE 1044-2009 Standard [8]

Defect Type Family	Defect Type in IEEE 1044 Table A.1	Description
Logic and Data	Logic	"Defect in decision logic, branching, sequencing, or computational algorithm, as found in natural language specifications or in implementation language. Examples: Missing else clause; Incorrect sequencing of operations; Incorrect operator or operand in expression; Missing logic to test for or respond to an error condition (e.g., return code, end of file, null value, etc.); Input value not compared with valid range; Missing system response in sequence diagram; Ambiguous definition of business rule in specification ..."
	Data	"Defect in data definition, initialization, mapping, access, or use, as found in a model, specification, or implementation. Examples: Variable not assigned initial value or flag not set; Incorrect data type or column size; Incorrect variable name used; Valid range undefined; Incorrect relationship cardinality in data model; Missing or incorrect value in pick list ..."
Interface	Interface	"Defect in specification or implementation of an interface (e.g., between user and machine, between two internal software modules, between software module and database, between internal and external software components, between software and hardware, etc.). Examples: Incorrect module interface design or implementation; Incorrect report layout (design or implementation); Incorrect or insufficient parameters passed; Cryptic or unfamiliar label or message in user interface; Incomplete or incorrect message sent or displayed; Missing required field on data entry screen ..."
Description	Description	"Defect in description of software or its use, installation, or operation ..."
Syntax	Syntax	"Nonconformity with the defined rules of a language. ..."
Standards	Standards	"Nonconformity with a defined standard. ..."
Build-Config-Install	Others (Build/ Package/ Installation/ Config)	"(Other defects for which there is no defined type in IEEE 1044-2009 Table A.1). Problems encountered during the build process, in library systems, or with management of change or version control ...", or "Problems in the configuration files or parameters", or "Problems in the installation process"

developed using Java programming language. The second defect report dataset has been collected from the Roundcube webmail software. Roundcube is a public webmail solution with a desktop-like GUI. It is developed using PHP, a database, TinyMCE rich-text editor, an IMAP library, etc.

3.2 Software Defect Classification Schemes

We use two different software defect type classification schemes derived from well-known standards. The first defect classification scheme has been used by [14, 15] to establish the state-of-the-art. It is based on the well-known IBM Orthogonal Defect Classification (ODC) [6, 7]. Please refer Table I of [14] for the details of this scheme. It uses three high-level defect type families to classify the defects: (i) Control and Data Flow, (ii) Structural, and (iii) Non-code.

The second defect classification scheme as shown in Table 2 is based on the sample defect classification scheme in Table A.1 (Annexure A) of IEEE 1044-2009 Standard [8]. It has six high-level defect type families, viz., (i) Interface, (ii) Logic and Data, (iii) Description, (iv) Standards, (v) Syntax, and (vi) Build-Config-Install.

https://github.com/roundcube/roundcubemail/issues
https://roundcube.net/about/

The dataset statistics and the class label distribution of software defect type families annotated using these two classification schemes are summarized in the Table 1.

3.3 Algorithm Implementations

3.3.1 Supervised Learning: To implement the current state-of-the-art solutions for software defect type classification, we used scikit-learn [12] library. According to Thung et al. [15], following the standard fully supervised learning approach, the Support Vector Machine (SVM) algorithm had given the best results. We implemented it using the scikit-learn (version 0.20.1). Since, as per Table 1, the least number of examples for a class are 4 (in case of `Description` class), we use *stratified* 4-fold cross-validation to ensure that there is at least one instance for each class while splitting the dataset into training and test subsets.

3.3.2 SMOTE and Other Standard Approaches for Imbalanced Learning: An established approach to tackle the imbalanced learning problem has been either to oversample the minority class(es) or to undersample majority class(es) so that number of training examples in the input are comparable for each class. Random OverSampling

Table 3: Class imbalance results: (i) CBC is the Concept-based classification approach, (ii) Supervised ML results are for SVM classifier (the best performing supervised algorithm for this task as per [15]) with 4-fold cross-validation (i.e., 75% labeled training data and 25% test data), (iii) Class-imbalanced learning in supervised ML setting: SMOTE algorithm [1] with the SVM classifier. (Numbers in 'boldface' font highlight the high recall values for the majority class and the numbers with underline highlight the adverse effect on recall for the minority class, emphasizing the challenge of class imbalance for supervised ML setting.)

Dataset	Classification Scheme	Defect type family	Algorithm								
			CBC			Supervised ML (SVM)			SMOTE		
			P	R	F1	P	R	F1	P	R	F1
Apache Libs	ODC	Control & Data flow	73.44	77.78	75.55	67.53	**96.49**	79.43	73.09	81.52	76.98
		Structural	50.0	27.88	35.80	54.43	14.61	22.44	41.84	36.44	38.53
		Non Code	48.15	60.19	53.50	89.62	48.70	62.59	68.66	54.26	60.15
	IEEE	Interface	33.06	41.84	36.94	49.75	13.32	20.60	38.21	35.93	36.55
		Logic & Data	70.98	70.49	70.73	64.96	**97.42**	77.94	71.58	**82.48**	76.59
		Description	56.0	40.0	46.67	84.42	28.99	41.87	58.81	37.40	44.56
		Build-Config-Install	76.92	17.54	28.57	81.37	44.65	56.47	61.98	52.22	54.93
		Standards	40.0	12.50	19.05	0.00	0	0.00	14.58	4.62	6.76
		Syntax	6.52	50.00	11.54	0.00	0	0.00	0	0	0
Round cube	ODC	Control & Data flow	89.19	67.01	76.52	79.12	**99.18**	88.02	83.35	**91.29**	87.09
		Structural	42.86	47.37	45.00	53.33	5.92	10.47	41.95	28.55	33.49
		Non Code	25.17	65.45	36.36	85.70	35.05	48.67	71.51	50.31	57.49
	IEEE	Interface	34.50	67.65	45.70	57.38	14.97	23.39	46.40	37.62	41.35
		Logic & Data	82.61	49.28	61.73	70.96	**97.26**	82.04	75.62	**86.83**	80.81
		Description	4.76	25.00	8.00	0	0	0	0	0	0
		Build-Config-Install	66.67	33.33	44.44	85.52	35.83	48.89	73.72	51.04	59.12
		Standards	25.0	50.0	33.33	0	0	0	23.75	17.50	19.17
		Syntax	10.91	33.33	16.44	21.25	5.75	8.88	57.92	32.88	38.44

technique expands the training dataset by adding more examples to the minority class(es) by oversampling them. A complementary approach is taken by Random UnderSampling technique in which the training dataset size is shrunk by undersampling the majority class so as to remove the skewness in the label distribution and make the training dataset balanced with respect the number of instances belonging to each class.

Synthetic Minority Oversampling TEchnique, (or more popularly known as SMOTE) [1, 3] is an advanced version of the basic oversampling technique that creates new, synthetic training instances by interpolating the instances of the minority class in $\mathcal{X}$, the given input data. While iterating over an instance $x \in \mathcal{X}$ of the minority class, SMOTE first randomly selects x_{nn}, one of the k-nearest neighbors belonging to the same class. Then, to create a new artificial data instance, x_{SMOTE} to be added to the training set, the feature values of x and x_{nn} are linearly interpolated, i.e., $x_{SMOTE} = x + \alpha.(x_{nn} - x)$, where $\alpha \in (0, 1)$.

We use the *imbalanced-learn* python library [10] to implement the standard and well-known approaches for imbalanced learning:

(1) Synthetic Minority Oversampling TEchnique (SMOTE)
(2) Random Over-sampling (ROS)
(3) Random Under-sampling (RUS)

3.3.3 Concept-based Classification: To implement the CBC approach, we use the Wikipedia dump, the JWPL library and the Lucene library. Note that the CBC approach does not need labeled training data. Instead, CBC relies on the the human-curated knowledge in the Wikipedia as a computationally amenable approximation for the human concept-space. Since the defect reports and the software defect classification scheme are in English, we use the English Wikipedia as the knowledge-base. We remove *stub-like* Wikipedia articles which are very short (having less than 100 characters in content), or have less than 3 hyperlinks. We assume that such short articles do not constrain sufficient content or they are isolated/pendant nodes in the Wikipedia hyperlink graph. We use Wikipedia dump snapshot was taken from the Wikipedia website on 14^{th} April 2016. In order to parse the Wikipedia dump and extract plain text from the Wikitext markup language, we use the DKPro Java Wikipedia Library (JWPL) [4, 17]. We use Lucene (version 6.6) to index the Wikipedia pages parsed by JWPL.

3.4 Evaluation Measures

We use the standard evaluation metrics [16] of precision, recall and F_1 score to evaluate the accuracy of the proposed approach. *Precision* (P_i) for the i^{th} class label (i.e., defect type label $y_i \in \mathcal{Y}$) measures how many of the instances (i.e., defect reports) predicted as the i^{th} class actually belong to that class in the gold-standard

https://dumps.wikimedia.org
https://en.wikipedia.org/wiki/Help:Wikitext

dataset (i.e., the expert annotated ground-truth).

$$Precision(P_i) = \frac{TP_i}{TP_i + FP_i}$$

Recall (R_i) for i^{th} class label measures how many of the instances belonging to the i^{th} class in the gold dataset are correctly predicted by the classifier.

$$Recall(R_i) = \frac{TP_i}{TP_i + FN_i}$$

where TP_i denotes the true positives, FP_i denotes the false positives, and FN_i denotes the false negatives for the i^{th} class label. The $F1_i$ *score* for i^{th} class label is the harmonic mean of the corresponding precision (P_i) and recall (R_i).

$$F1_i = \frac{2 \cdot P_i \cdot R_i}{P_i + R_i}$$

4 RESULTS AND ANALYSIS

Table 3 summarizes the per-class accuracies in terms of the precision, recall and F1 measures for the different approaches to address the software defect type classification task.

As we can observe from Table 1, there is class imbalance in the datasets. For the IEEE-based classification scheme, the number of examples belonging to the *Logic and Data* class signifiantly dominate the minority classes such as *Description, Standards*, or *Syntax* for both the datasets. Similarly, when we use the ODC-based defect classification scheme, the *Control and Data flow* defect type dominates the other classes, though the degree of imbalance is relatively lesser than the IEEE based classification.

While using SMOTE and other standard approaches for imbalanced learning, we used SVM as the underlying classification algorithm as SVM has been reported [15] to perform the best for this task. Results of SMOTE and Random Over-sampling techniques were comparable. Random Under-sampling performed poorly than both SMOTE and Random Over-sampling. Due to space constraints, we report only the results for SMOTE based approach for imbalanced learning.

RQ_1: The impact of class imbalance in the dataset is discernible in the Table 3. The accuracy of the dominant classes such as *Logic and Data* or *Control and Data flow* is significantly higher than other classes for the supervised ML (SVM) as well as SMOTE. Note that, for the dominant classes the recall (R) is very high compared to other classes. This clearly indicates that the standard supervised learning approach tends to get biased by the classes having large number of training instances at the expense of the classes having smaller number of instances. This happens because they try to learn the dominant patterns in the labeled training data and in this process, the smaller classes get overshadowed by the patterns characterizing the classes with larger number of instances. Observe that the classes with less number of examples, such as *Syntax* or *Standards* get very poor accuracy.

RQ_2: Observe that under the CBC approach, the accurancy of minority classes is not as severely impacted due to the class imbalance as in the case of supervised ML approach. CBC approach does not need the labeled training data and instead it relies on the knowledge augmented representation using Wikipedia as a concept-space. As a result of this, CBC approach does not get biased by the patterns characterizing the classes that dominate the label distribution in training data. SMOTE does better than the standard supervised ML approach, but still the minority classes do suffer to an extent. Further, we need to note that SMOTE based approach does still require the labeled training data (75% labeled data in this case due to 4-fold cross-validation). CBC approach does not have this dependence on labeled training data. Hence, we believe that CBC approach does provide a feasible alternative to tackle the class imbalance problem which is faced by the traditional supervised learning approach and the over-sampling/under-sampling based approaches for imbalanced learning.

5 CONCLUSION

In this paper, we examined the feasibility of using the concept-based classification approach to tackle the class imbalance problem faced by the supervised learning paradigm. The CBC approach leverages a knowledge augmented representation of the documents to be classified and labels computed using the human-curated knowledge in Wikipedia. As the CBC approach does not need labeled training data, it is able to treat each software defect type label without getting biased by the skew in the label distributions in case of class imbalanced datasets. In case a dataset is class imbalanced, then the CBC approach has a natural advantage over the standard supervised machine learning algorithms. This helps the concept-based classification approach to overcome the important limitation, viz., dependence on labeled training data, when using the supervised machine learning paradigm as well standard imbalanced learning approaches such as SMOTE.

REFERENCES

[1] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[2] Nitesh V. Chawla, Nathalie Japkowicz, and Aleksander Kotcz. 2004. Edit.: Special Issue on Learning from Imbalanced Data Sets. *SIGKDD Explorations Newsletter* 6, 1 (June 2004), 1–6. https://doi.org/10.1145/1007730.1007733

[3] Alberto Fernández, Salvador Garcia, Francisco Herrera, and Nitesh V Chawla. 2018. SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. *Journal of artificial intelligence research* 61 (2018), 863–905.

[4] Oliver Ferschke, Torsten Zesch, and Iryna Gurevych. 2011. Wikipedia Revision Toolkit: Efficiently Accessing Wikipedia's Edit History. In *Proc. of the ACL-HLT 2011 System Demonstrations*. Association for Computational Linguistics, 97–102.

[5] Evgeniy Gabrilovich and Shaul Markovitch. 2007. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proc. of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, Vol. 7. 1606–1611.

[6] IBM. 2013. Orthogonal Defect Classification version 5.2 extensions for Defects in GUI, User Documentation, Build and National Language Support (NLS). https://researcher.watson.ibm.com/researcher/files/us-pasanth/ODC-5-2-Extensions.pdf. (URL accessibility verified on 9th Nov., 2018).

[7] IBM. 2013. Orthogonal Defect Classification version 5.2 For Software Design and Code. http://researcher.watson.ibm.com/researcher/files/us-pasanth/ODC-5-2.pdf. (URL accessibility verified on 9th Nov., 2018).

[8] IEEE. 2009. IEEE Standard 1044-2009 Classification for Software Anomalies.

[9] Nathalie Japkowicz and Shaju Stephen. 2002. The class imbalance problem: A systematic study. *Intelligent data analysis* 6, 5 (2002), 429–449.

[10] Guillaume Lemaître, Fernando Nogueira, and Christos K Aridas. 2017. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *The Journal of Machine Learning Research* 18, 1 (2017), 559–563.

[11] Sangameshwar Patil. 2017. Concept based Classification of Software Defect Reports. In *Proc. of 14th International Conference on Mining Software Repositories (MSR)*. IEEE/ACM.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine

Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[13] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

[14] Ferdian Thung, Xuan-Bach Le D., and David Lo. 2015. Active Semi-Supervised Defect Categorization. In *Proc. of IEEE 23rd International Conference on Program Comprehension (ICPC).* 60–70.

[15] Ferdian Thung, David Lo, and Lingxiao Jiang. 2012. Automatic defect categorization. In *Proc. of 19th Working Conference on Reverse Engineering (WCRE).* IEEE, 205–214.

[16] Mohammed J. Zaki and Jr. Wagner Meira. 2014. *Data Mining and Analysis: Fundamental Concepts and Algorithms.* Cambridge University Press.

[17] Torsten Zesch, Christof Müller, and Iryna Gurevych. 2008. Extracting Lexical Semantic Knowledge from Wikipedia and Wiktionary.. In *Proc. of 6th International Conference on Language Resources and Evaluation (LREC)*, Vol. 8. 1646–1652.