AutoODC: Automated Generation of Orthogonal Defect Classifications

LiGuo Huang[1] Vincent Ng[2] Isaac Persing[2] Ruili Geng[1] Xu Bai[1] Jeff Tian[1]
Department of Computer Science and Engineering, Southern Methodist University[1]
Human Language Technology Research Institute, University of Texas at Dallas[2]
Dallas, TX, USA
{lghuang, rgeng, xbai, tian}@smu.edu[1], {vince, persingq}@hlt.utdallas.edu[2]

Abstract—**Orthogonal Defect Classification (ODC), the most influential framework for software defect classification and analysis, provides valuable in-process feedback to system development and maintenance. Conducting ODC classification on existing organizational defect reports is human intensive and requires experts' knowledge of both ODC and system domains. This paper presents AutoODC, an approach and tool for automating ODC classification by casting it as a supervised text classification problem. Rather than merely apply the standard machine learning framework to this task, we seek to acquire a better ODC classification system by integrating experts' ODC experience and domain knowledge into the learning process via proposing a novel Relevance Annotation Framework. We evaluated AutoODC on an industrial defect report from the social network domain. AutoODC is a promising approach: not only does it leverage minimal human effort beyond the human annotations typically required by standard machine learning approaches, but it achieves an overall accuracy of 80.2% when using manual classifications as a basis of comparison.**

Keywords-Orthogonal Defect Classification (ODC); text classification; natural language processing

I. INTRODUCTION

The systematic classification and analysis of defect data bridge the gap between causal analysis and statistical quality control, provide valuable in-process feedback to system development or maintenance, as well as help assure and improve system and software quality. The analysis results based on systematic defect classifications enable us to understand the major impact types of system defects and to pinpoint specific problematic areas for focused problem resolution and quality improvement. Orthogonal Defect Classification (ODC) [1], developed initially at IBM, is the most influential among general frameworks for software defect classification and analysis. ODC has been successfully used in various types of industrial and government organizations [2, 3].

Existing defect classification approaches and process are human intensive, requiring significant knowledge of both historical projects and ODC taxonomy from domain experts. In most cases, defect data are reported by users or developers during system development, operation and maintenance, and are stored in defect repositories with a customized classification scheme for an industrial organization or without being classified. These defect repositories are often called defect (bug) reports or defect (bug) trackers. Analysts need to read and understand the textual description of each reported defect in order to classify them into the ODC taxonomy. Manual ODC defect classification based on existing defect repositories is at best a mundane, mind numbing activity. Thus the types of follow-on ODC-based defect analyses are often restricted by the limited defect classification results.

To overcome these challenges in software system defect classification, we have developed AutoODC, an approach and tool for automatically generating the Orthogonal Defect Classification (ODC) from the defect report (tracker). This paper presents our research results to improve state-of-the-art of ODC generation. We cast the problem as a supervised text classification problem, where the goal is to train a classifier via machine learning techniques for automatically classifying a defect record in the defect report. While one may expect that classification accuracy consistently increases with the amount of training data, somewhat surprisingly we observed that this was not the case for our classification problem. We hypothesize that the reason could be attributed to the learning algorithm's inability to learn from the relevant portions of a defect report. Specifically, only a subset of the words in a report may be relevant for classification, and hence learning from all of the words in the report may introduce noise into the learning process and eventually deteriorate the performance of the resulting classification system. Consequently, we propose a new framework in which the learning algorithm learns from a richer kind of training data. Rather than simply determine the category to which a defect record belongs, a human annotator additionally annotate the segments in the record (e.g., words, phrases) that indicate why she believes the record should belong to a particular category. Our experiments demonstrate that learning from richer training data in our proposed annotation relevance framework yields a better defect classifier than one acquired via the standard text classification framework, with a 23% reduction in relative error.

We applied AutoODC to 403 defect records in an industrial defect report from Company P[1] and compared its performance with manual ODC classifications on the "impact" attribute[2]. Defect impact is an ODC attribute of a defect that shows the impact type (detailed in Section II) the defect would have had upon the customer and end users if it had escaped to the field.

[1] The name of the industrial company is anonymous in this paper due to the proprietary rules. It is referred to as Company P in this paper.

[2] The definitions and taxonomy of ODC v5.11 attributes are accessible at http://www.research.ibm.com/softeng/ODC/DETODC.HTM .

978-1-4577-1639-3/11/$26.00 © 2011 IEEE ASE 2011, Lawrence, KS, USA

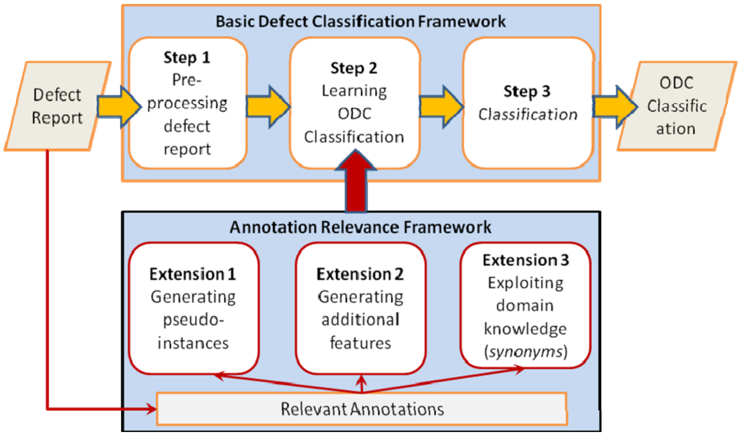
Basic Defect Classification Framework
Defect Report
Step 1
Pre-processing defect report
Step 2
Learning ODC Classification
Step 3
Classification
ODC Classific ation
Annotation Relevance Framework
Extension 1
Generating pseudo-instances
Extension 2
Generating additional features
Extension 3
Exploiting domain knowledge (synonyms)
Relevant Annotations

Figure 1. AutoODC Overview.

We evaluated AutoODC by calculating the overall accuracy. In summary, AutoODC makes the following contributions:

- AutoODC uses the annotation relevance framework, a novel framework that enables the acquisition of a defect classifier from a richer kind of training data by injecting experts' domain knowledge into the classifier so as to robustly analyze natural language defect descriptions.
- The AutoODC defect classifier achieves an accuracy of 80.2% when using manual defect classification as a basis of evaluation, where accuracy is computed as the percentage of defect records correctly classified by AutoODC.
- As a complement to the manual ODC classification, AutoODC improves the confidence of defect classification results by reducing the investigation set that a human analyst has to examine when performing ODC classification.

II. APPOACH

A. *Overview*

AutoODC accepts a semi-structured text defect report, where each defect record contains a defect "summary" (optional) and a "description" (required), and outputs the ODC classification and the confidence of classification for each record. It uses a SVM-based text classification system integrated with our annotation relevance framework to improve the accuracy of automated ODC classification. Figure 1 depicts an overview of the AutoODC approach which consists of two major components: (1) Basic Defect Classification System, and (2) Annotation Relevance Framework.

The **Basic Defect Classification System** is developed in three steps: 1) Pre-processing defect report; 2) Learning ODC; and 3) Classification. Aiming to improve classification accuracy, an **Annotation Relevance Framework** extends the Basic Classification System by exploiting relevant annotations in three ways: 1) generating additional instances (known as pseudo-instances) for training an SVM classifier; 2) generating additional features; and 3) adding domain knowledge.

B. *The Basic Defect Classification System*

Step 1: Pre-processing defect report

The input of AutoODC is a defect report consisting of defect records, each of which contains a textual "description" and an optional "summary". We construct a training set from defect records first by tokenizing the record and stemming each word using the stemmer that comes with the WordNet [6] lexical knowledge base[3]. We then represent each defect record as a binary-valued vector indicating the presence or absence of a word in the corresponding record, where each element corresponds to a distinct word that appears in the training set. Note that we did not remove stopwords from the vector, as preliminary experiments indicated that results deteriorated slightly with their removal. Also associated with the vector is a class value, which is the analyst-assigned ODC category of the corresponding record. Finally, to prevent long defect records, which presumably contain many 1's in their vectors, from having an undesirably larger influence on the learning process, we normalize each vector.

Step 2: Learning ODC

In our experiments, we use a support vector machine (SVM) [4] as the underlying learning algorithm for classifier training. Our choice of SVM is motivated in part by the fact that it has offered impressive performance on a variety of text classification tasks, and in part by the fact that our annotation relevance framework aims to improve SVM's learning procedure. In order to integrate our extensions into SVM, we adopt a one-versus-others training scheme, where we train one SVM classifier for each of the classes. For instance, we train one classifier for determining whether a defect record belongs to *Reliability*, another classifier for determining whether a record belongs to *Usability*, etc. In essence, each classifier represents exactly one ODC class, and the number of SVM classifiers we train is equal to the number of defect classes. The training set for training each of these classifiers is different from each other. Specifically, to train a classifier for determining whether a record belongs to class *i*, we take the pre-processed training set described above, and assign the class value of each training instance as follows. If the training instance belongs to class *i*, its class value is +1; otherwise, its class value is -1.

Step 3: Classification

After training, we apply the resulting classifiers to classify each defect record in the test set, where the test instances are created in the same way as the training instances (see Step 1). A test record is assigned the class whose classifier returns the highest absolute value among the set of values returned by all the classifiers. In addition to the ODC defect category, AutoODC will return a confidence value associated with the class. Confidence is measured by the data point's distance from the SVM hyperplane either from the "+" or "-" side.

C. *Exploiting Relevant Annotations*

This section describes our three relevant annotation based extensions to the basic defect classification system.

Extension 1: Generating pseudo-instances

In the basic defect classification system, we train for each ODC category *c* a classifier that identifies defect records belonging to *c*, and describe how to create the training set for

[3] Other stemmers, such as the Porter stemmer [5], can be used, but we found that the WordNet stemmer yields slightly better accuracy.

training each of these classifiers. Our first extension involves augmenting this training set with additional positive and negative training instances generated from the relevant annotations and modifying SVM's learning procedure to profitably exploit these additional training instances, referred to as pseudo-instances below.

A relevant annotation is a text fragment that motivates a human annotator to assign a particular class to a text document. In AutoODC, a relevant annotation can be a relevant word or phrase in the defect description that provides hints for analysts to decide which ODC category a defect should belong to.

To see how relevant annotations can be used to generate pseudo-instances, let us define some notation. Let $\mathbf{x}_i$ be the vector representation of a training defect record R_i. Without loss of generality, assume that R_i belongs to class c_i. Since we adopt a one-versus-others training scheme for generating training instances, $\mathbf{x}_i$ is a positive instance for training the classifier representing class c_i. Given $\mathbf{x}_i$, we construct one or more not-so-positive training instances $\mathbf{v}_{ij}$ (positive pseudo-instances**)**. Specifically, we create $\mathbf{v}_{ij}$ by removing relevant annotation r_{ij} (the relevant words/phrases extracted from the original defect summary and description in the report) from R_i. In other words, the number of pseudo-instances $\mathbf{v}_{ij}$ we create from each original instance $\mathbf{x}_i$ is equal to the number of relevant annotations present in R_i, and each $\mathbf{v}_{ij}$ is created by deleting exactly one relevant annotation substring from R_i. Since $\mathbf{v}_{ij}$ lacks a relevant annotation and therefore contains less evidence that a human annotator found relevant for classification than $\mathbf{x}_i$, the correct SVM classifier should be less confident of a positive classification on $\mathbf{v}_{ij}$.

This idea can be implemented by imposing additional constraints on a usual SVM classifier, which can be defined in terms of a weight vector $\mathbf{w}$. The usual SVM constraint on positive instance $\mathbf{x}_i$ is $(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1$, which ensures that $\mathbf{x}_i$ is on the positive side of the hyperplane. In addition to these usual constraints, we desire that for each j, $(\mathbf{w} \cdot \mathbf{x}_i - \mathbf{w} \cdot \mathbf{v}_{ij}) \geq \mu$, where $\mu \geq 0$. Intuitively, this constraint specifies that $\mathbf{x}_i$ should be classified as more positive than $\mathbf{v}_{ij}$ by a margin of at least μ.

We create negative pseudo-instances similarly from each negative training instance.

Extension 2: Generating additional features

In our second extension to the basic defect classification system, we exploit the relevant annotations to generate additional features for training an SVM classifier.

Recall that each defect record in the basic system is represented as a vector of words. Hence, each word in a defect report is a feature for the SVM classifier. Since the relevant annotations (relevant words/phrases) used in Extension 1 are supposed to be relevant for classification, we hypothesize that they can also serve as useful features for the SVM classifier. One way to exploit these relevant annotations as features is as follows. First, we collect all the relevant annotations from the defect records in the training set. Then we create one feature from each relevant annotation and augment the word-based feature set with these new features. In other words, each training/test instance that is not a pseudo-instance is now represented as a binary feature vector consisting of both the word-based features and the relevant annotation-based features. Given a defect record, the value of a relevant annotation-based feature is 1 if and only if the corresponding relevant word/phrase appears in the defect record.

However, some of these relevant annotation-based features, especially those long key phrases (e.g., "I would like to repair"), may not appear at all in the test set. This problem is commonly known as data sparseness. To combat the data sparseness, we increase the likelihood of seeing a relevant annotation-based feature in the test set as follows. Instead of using a relevant annotation directly as a feature, we create all possible bigrams (i.e., consecutive words of length two) from each relevant annotation and use them as additional features.

Extension 3: Exploiting domain knowledge

Another way to combat data sparseness is to apply domain knowledge to identify the relevant annotations that are synonymous. Specifically, we (1) collect all the relevant annotations from the training defect records, (2) have a human analyst partition them so that each cluster contains all and only synonymous relevant annotations, and (3) assign a unique id to each cluster. Given a training/test instance that is not a pseudo-instance, we exploit this domain knowledge as follows. We check whether a relevant annotation is present in the corresponding defect record. If so, we create an additional feature that corresponds to the id of the cluster containing the relevant annotation. An example of a synonym cluster from our defect report is {appear truncated, text cutoff}.

III. Evaluation

Our evaluation addresses the following research question: To what extent does our annotation relevance framework, which comprises the three aforementioned extensions to basic defect classification system, help improve ODC defect classification, and are the three extensions all contributing positively to overall performance?

A. Experimental Setup

AutoODC is experimented on classifying defect records under the ODC "impact" attribute. To optimize the performance of AutoODC, we have experimented with multiple variations of AutoODC by combining the basic defect classification system with the three extensions.

Data set. We acquired a defect report with 403 defect records in a social network project domain from the industrial Company P. To provide training/testing data for AutoODC, two expert analysts independently classified the 403 defects into 6 categories under the "impact" attribute with an initial agreement of 90% and then cross-validated their results to resolve the disagreements. The classified defects are distributed over the categories of Capability (284), Security (11), Performance (1), Reliability (8), Requirements (39), and Usability (60). The two analysts, who are co-authors of the paper, had six and three years of ODC classification and analysis experience in industry, respectively. One of them worked as an ODC expert at the IBM quality assurance group for six years. They both marked the words/phrases which they think are relevant for their assigning a defect record to a

TABLE I. FIVE-FOLD CROSS-VALIDATION RESULTS.

Experiment	Accuracy	Reliability			Capability			Integrity/Security			Usability			Requirements		
		P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
Basic	74.2	0.0	0.0	0.0	77.6	94.0	85.0	75.0	27.3	40.0	48.9	36.7	41.9	70.0	17.9	28.6
Basic+Ext 1	76.9	0.0	0.0	0.0	80.5	93.3	86.5	71.4	45.5	55.6	60.9	46.7	52.8	57.1	30.8	40.0
Basic+Ext 2	76.4	0.0	0.0	0.0	79.1	94.4	86.0	87.5	63.6	73.7	56.8	41.7	48.1	66.7	20.5	31.4
Basic+Ext1,2	78.9	100.0	12.5	22.2	81.1	95.1	87.5	71.4	45.5	55.6	70.7	48.3	57.4	61.9	33.3	43.3
Basic+Ext1,2,3	80.2	100.0	12.5	22.2	82.8	95.1	88.5	77.8	63.6	70.0	73.3	55.0	62.9	54.5	30.8	39.3

particular ODC category. Moreover, they were asked to identify the synonyms among these relevant words/phrases.

Evaluation metrics. We employ two evaluation metrics. First, we report overall accuracy, which is the percentage of records in the test set correctly classified by AutoODC. Second, we compute the precision and recall for each ODC category. Given an ODC category *c*, its recall is the percentage of defect records belonging to *c* that are correctly classified by AutoODC; and its precision is the percentage of records classified by AutoODC as *c* that indeed belong to *c*.

Evaluation methodology. All performance numbers we report in Section III.B are obtained via 5-fold cross-validation experiments. Specifically, we first randomly partition the 403 defect records into five equal-sized subsets (or *folds*). Then, in each fold experiment, we reserve one of the five folds for testing and use the remaining four folds as the training set. We repeat this five times, each time using a different fold as a test set. Finally, we average the result obtained over the five folds.

B. Experimental Results on the Basic System and the Annotation Relevance Framework

The basic defect classification system. We first report the performance of the basic defect classification system. Each record is represented as a binary vector with the 1089 unique words appearing in the data set as features. We use SVM^{light} to train a soft-margin SVM classifier on the training set. Results of the basic defect classification system are shown in Table I, which reports performance in terms of accuracy, as well as precision (P), recall (R) and F-score (the harmonic mean of precision and recall) for each ODC class. Note that the table contains results for all but the Performance class. We omitted the results for Performance for the obvious reason: The data set has only one instance of Performance, and since this instance can appear in either the training set or the test set for a given fold experiment, the F-score achieves for this ODC class will always be zero. As we can see, this basic system achieves an accuracy of 74.2%, offering the best F-score on Capability due to its frequent occurrence.

Effectiveness of our three extensions to the basic system. We begin by incorporating Extension 1 (generating pseudo-instances for training) into the basic system. Results of this extension are shown in row 2 of Table I. As we can see, this extension gives a 2.7% improvement in accuracy over the basic system. The F-scores on three under-represented classes (Integrity, Usability, and Requirements) improve considerably.

Next, we incorporate Extension 2 (employing the relevant annotations to generate additional features). To better understand its contribution, we first apply Extension 2 to the basic system in isolation (row 3 of Table I), and then apply it in combination with Extension 1 (row 4 of Table I). When Extension 2 is applied in isolation to the basic system, we see that it improves not only the accuracy of the basic system by 2.2%, but also the F-scores of all classes except Reliability. These results demonstrate the effectiveness of Extension 2. When the two extensions are applied in combination to the basic system, we can see that performance rises further: in comparison to the basic system, accuracy improves by 4.7%, and F-scores on all classes improve. Finally, we incorporate Extension 3 into the system (classification with domain knowledge). Specifically, the results in row 5 of Table I are obtained by applying all three extensions to the basic system. Comparing the results in rows 4 and 5, we can see that Extension 3 provides useful knowledge: accuracy improves by 1.3%. In comparison to the basic system, the three extensions together improve accuracy by 6%, which represents a 23% reduction in relative error. Overall, these results demonstrate the effectiveness of each of our extensions.

IV. CONCLUSION AND FUTURE WORK

AutoODC, when used in tandem with traditional manual approaches, can largely reduce the human effort and improve an analyst's confidence in ODC classification. It will have a significant impact on systematic defect analysis for software quality assurance. This paper seeks to improve ODC classification by presenting a novel framework for acquiring a defect classifier from relevant annotations to robustly analyze natural language defect descriptions. Based on this framework, we developed AutoODC, a text classification tool for automated ODC classification which can be adjusted or extended to other text classification problems in the software engineering domain. Our evaluation shows that AutoODC is accurate in classifying defects by the "impact" attribute with an accuracy of 80.2% when using manually classified differences as a basis of evaluation. To envisage our future work, we will extend AutoODC to perform defect classifications on other ODC attributes and experiment it on other industrial and open source defect repositories.

REFERENCES

[1] R. Chillarege, I.S. Bhandari, J.K. Chaar, M.J. Halliday, D.S. Moebus, B.K. Ray, M.-Y.Wong "Orthogonal Defect Classification-A Concept for In-Process Measurements," IEEE TSE. Vol.18, pp. 943-956, Nov. 1992.

[2] R. Lutz and C. Mikulski. "Empirical analysis of safety-critical anomalies during operations", IEEE TSE, vol. 30, no. 3, March, 2004.

[3] J. Zheng, L. Williams, N. Nagappan, J. Hudpohl, "On the value of static analysis tools for fault detection", IEEE transactions on software engineering. 2006, vol. 32, no. 44.

[4] V. Vapnik, The Nature of Statistical Learning. Springer, N.Y., 1995.

[5] M.F. Porter, "An algorithm for suffix stripping", Program, 14(3) pp 130−137, 1980.

[6] C. Fellbaum, WordNet: An electronic lexical database, MIT Press, Cambridge, MA. 1998.