Support Vector Machines for Anti-pattern Detection

Abdou Maiga
Université de Montréal, QC, Canada
abdou.maiga@polymtl.ca

Nasir Ali
École Polytechnique de Montréal, QC, Canada
nasir.ali@polymtl.ca

Neelesh Bhattacharya
École Polytechnique de Montréal, QC, Canada
n.bhattacharya@polymtl.ca

Aminata Sabané
École Polytechnique de Montréal, QC, Canada
a.sabane@polymtl.ca

Yann-Gaël Guéhéneuc
École Polytechnique de Montréal, QC, Canada
guehene@iro.umontreal.ca

Giuliano Antoniol
École Polytechnique de Montréal, QC, Canada
antoniol@ieee.org

Esma Aïmeur
Université de Montréal, QC, Canada
aimeur@iro.umontreal.ca

ABSTRACT

Developers may introduce anti-patterns in their software systems because of time pressure, lack of understanding, communication, and–or skills. Anti-patterns impede development and maintenance activities by making the source code more difficult to understand. Detecting anti-patterns in a is important to ease the maintenance of software. Detecting anti-patterns could reduce costs, effort, and resources. Researchers have proposed approaches to detect occurrences of anti-patterns but these approaches have currently some limitations: they require extensive knowledge of anti-patterns, they have limited precision and recall, and they cannot be applied on subsets of systems. To overcome these limitations, we introduce SVMDetect, a novel approach to detect anti-patterns, based on a machine learning technique—support vector machines. Indeed, through an empirical study involving three subject systems and four anti-patterns, we showed that the accuracy of SVMDetect is greater than of DETEX when detecting anti-patterns occurrences on a set of classes. Concerning, the whole system, SVMDetect is able to find more anti-patterns occurrences than DETEX.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

Design, Verification

Keywords

Anti-pattern, program comprehension, program maintenance, empirical software engineering

1. INTRODUCTION

Anti-patterns are "poor" solutions to recurring design and implementation problems. Researchers have performed empirical studies to show that anti-patterns create hurdles during program comprehension, software evolution and maintenance activities [8]. It is important to detect anti-patterns at an early stage of software development, to reduce the maintenance costs.

Current anti-pattern detection approaches have several limitations: they require extensive knowledge of anti-patterns, they have limited precision and recall and cannot be applied on subsets of systems.

We can apply Support vector machines (SVM) on subsets of systems because it considers system classes one at a time, not collectively as previous rule-based approaches do. To the best of our knowledge, researchers have not yet studied the potential benefits of using SVM to detect anti-patterns.

The contribution of this paper is two-fold. First, we propose our approach, SVMDetect, to detect anti-patterns using SVM. We use both the measures of precision and recall to compare SVMDetect to DETEX [13], the best state-of-the-art approach, on a set of three programs and the four most studied anti-patterns. We showed that the accuracy of SVMDetect is greater than of DETEX when detecting anti-patterns occurrences on a set of classes. Concerning the whole system, SVMDetect is able to find more anti-patterns occurrences than DETEX. We thus conclude that: a SVM-based approach can overcome the limitations of previous approaches.

The paper is organised as follows. Section 2 provides a brief description of the state-of-the-art of anti-patterns detection approaches and SVM. Section 3 describes our approach. Section 4 introduces our empirical study while Section 5 reports and discusses its results. Finally, Section 6 presents the threats to validity whereas Section 7 concludes the paper and outlines future work.

2. RELATED WORK

We now recall the major related work.

Smell/Anti-pattern Detection: Many researchers studied anti-patterns detection. Alikacem et al. [1] detected smells/anti-patterns using a meta-model for representing the source code and fuzzy thresholds. Langelier et al. [10] proposed a visual approach to detect anti-patterns. Marinescu [11] presented detection strategies based on quantifiable expression of rules using metrics to detect anti-patterns in systems. Sahraoui et al. [7] used search-based techniques to detect anti-patterns conjecturing that the more the code deviates from good practices, the more it is likely to be vulnerable to anti-patterns. Moha et al. [13] proposed an approach based on a set of rules (metrics, relations between classes) that describes the characters of each antipattern to identify them.

The works carried out so far suffered from some limitations: they have limited precision and recall (if reported at all), had not been adopted by practitioners yet, cannot be applied on subsets of systems, and required sufficient knowledge of anti-patterns. These limitations can be overcome using Support Vector Machines (SVM).

SVM: SVM has been used in several domains in the past for various applications, *e.g.*, bioinformatics [2], object recognition [4]. Further, SVM is a recent alternative to the classification problems. For example, Guihong et al. [3] used C-SVM, a variant of SVM, for terms classification. SVM was also used in image retrieval systems when Sethia et al. [12] used invariant feature histograms to compare the efficiency of different SVMs claiming that a significant performance gain was obtained with their approach. Kim et al. [9] proposed the change classification approach for predicting latent software bugs based on SVM.

To the best of our knowledge, no previous approach used SVM for anti-pattern Detection.

3. OUR APPROACH: SVMDETECT

SVMDetect is based on Support Vector Machines (SVM) using a polynomial kernel to detect occurrences of anti-patterns.

SVM is a set of techniques based on statistical theory of supervised learning introduced by Vapnik [5]. It relies on the existence of a linear classifier in an appropriate space and uses a set of training data to train the parameters of the classifier. It is based on the use of functions called kernel, which allows an optimal separation of data into two categories by a hyperplane.

As other machine learning techniques, applying SVMDetect requires preprocessing. Indeed, we must first train SVMDetect on some sets of known occurrences of the anti-patterns, one anti-pattern at a time, before applying it on some set of classes.

We use SVMDetect to detect the well-known anti-patterns: Blob, Functional Decomposition, Spaghetti code, and Swiss Army Knife. For each anti-pattern detection, the detection process is identical.

We illustrate the detection process with the Blob anti-pattern for the sake of clarity. We define:

- $TDS = \{C_i, i = 1, \ldots, p\}$, a set of classes C_i derived from an object-oriented system that constitutes the training dataset;
- $\forall i, C_i$ is labelled as Blob (B) or not (N);
- DDS is the set of the classes of a system in which we want to detect the Blob classes.

To detect the Blob classes in the set DDS, we apply SVMDetect through the following steps:

Step 1 (Object Oriented Metric Specification): SVMDetect takes as input the training dataset TDS. For each class from TDS, we calculate object-oriented metrics that will be used as the attributes x_i for each class in TDS. We use POM[1] to compute metrics for all the studied systems. POM is an extensible framework, based on the PADL meta-model, which provides more than 60 metrics [6], including the well-known metrics by Chidamber and Kemerer.

Step 2 (Train the SVM Classifier): We train the SVM classifier using the dataset TDS and the set of metrics computed in Step 1. We define the training dataset as: $TDS = \{(x_i, y_i) | x_i \in \mathbb{R}^p, y_i \in \{-1, 1\}, \forall i \in (1, \ldots, n)\}$ where y_i is either 1 or -1, indicating respectively if a class x_i is a Blob or not. Each x_i is a p-dimensional real vector with p the number of metrics.

The objective of the training step is to find the optimal hyperplane that divides the classes into the two different groups, Blob or Not-Blob.

Step 3 (Construction of the dataset DDS and detection of the occurrences of an anti-pattern): We build the dataset of the system on which we want to detect an anti-pattern as follows: for each class of the system, we compute the same set of metrics as in Step 1. We use the SVM classifier trained in Step 2 to detect the new occurrences of the anti-pattern in the dataset DDS.

We use Weka[2] to implement SVMDetect using its SVM classifier.

4. EMPIRICAL STUDY

The *goal* of our empirical study is to validate that SVMDetect can overcome the limitations of previous approaches, by comparing our approach, SVMDetect, with DETEX [13]. The *quality focus* of our study is the accuracy of SVMDetect, in terms of precision and recall. The *perspective* is that of researchers and practitioners interested in verifying if SVMDetect can be effective in detecting various kinds of anti-patterns, and in overcoming the previous limitations.

4.1 Research Questions

To see whether SVMDetect overcomes the previous limitations, we ask the following research question:

- RQ1: How does the accuracy of SVMDetect compare with that of DETEX, in terms of precision and recall? We decompose RQ1 as follows:
  - RQ1$_1$: How does the accuracy of SVMDetect compare with that of DETEX, in terms of precision and recall, when applied on a same subset of a system?
  - RQ1$_2$: How many occurrences of Blob SVMDetect can detect when comparing with that of DETEX on a same entire system?

[1]`http://wiki.ptidej.net/doku.php?id=pom`
[2]`http://www.cs.waikato.ac.nz/ml/weka/`

4.2 Objects

The objects[3] of our study are ArgoUML v0.19.8 (1,230 classes and 113,017 lines of code), Azureus v2.3.0.6 (1,449 classes and 191,963 lines of code), and Xerces v2.7.0(513 classes and 71,217 lines of code), three open-source Java systems. We chose these systems on several factors. First, we selected open-source systems that are freely available so that other researchers can replicate our study. Second, we selected systems that have been used by other researchers to allow comparisons [13].

4.3 Subjects

The subjects of our study are the following four anti-patterns: Blob, Functional Decomposition (FD), Spaghetti Code (SC), and Swiss Army Knife (SAK). We chose these four anti-patterns because these are known anti-patterns and commonly studied in previous work, particularly by Moha et al. [13] for comparison. More details on the description of these anti-patterns can be found in [13].

4.4 Analysis Methods

For the purpose of our empirical study, we build two datasets for each system and each anti-pattern, composed of anti-patterns and non-anti-patterns classes in equal numbers. To answer $RQ1_1$ (respectively $RQ1_2$) , we train SVMDetect on a dataset DDS_1 and detect occurrences of an anti-pattern on DDS_2 (respectively on the rest of the whole system). We compute the precision and recall of SVMDetect on DDS_2 (respectively we compute the number recovered occurrences of BLOB on the rest of the whole system). We then run DETEX on the same dataset, DDS_2 (respectively on the rest of the whole system), and compute its precision and recall.

5. RESULTS AND DISCUSSION

This section reports and discusses the results of our empirical study. The data, for replication purpose, is available online[4].

Subsets of System: $RQ1_1$: Table 1 reports the precision and recall values when applying DETEX and SVMDetect on the DDS_2 datasets. When applied on subsets of systems, we observed that DETEX could not detect occurrences of some anti-patterns and, when it did, the precision and recall values were quite low, mostly 0. We explain this observation by the use by DETEX of boxplots and thresholds. When DETEX analyses a few classes, its use of boxplots and thresholds yields most of the classes to fall under (respectively above) the thresholds and, hence, it is not to be reported. This problem does not arise when analysing an entire system because then the boxplots quartiles are different and more classes falls within the threshold values set in the rules. SVMDetect can work on the whole system and part of a system.

Complete System: $RQ1_2$: Table 2 shows the total number of anti-patterns' occurrences of blob detected by DETEX and SVMDetect. When applied on the whole system for detecting Blob occurrences, SVMDetect, on an average, performed better than DETEX. SVMDetect could detect

[3]`argouml.tigris.org/`, `azureus.sourceforge.net/`, and `xerces.apache.org/xerces-c/`

[4]`http://www.ptidej.net/download/experiments/ase12/`

Table 2: Total recovered occurrences of BLOB by DETEX and SVMDetect

	DETEX	**SVMDetect**
ArgoUML	25	40
Azureus	38	48
Xerces	39	55
Total	102	**143**

143 Blob occurrences; whereas DETEX could detect 102 Blob occurrences, in spite of DETEX being able to perform at its best when detecting Blob. Further, we applied SVMDetect using a trained dataset of one system A, for detecting an anti-pattern of another system B. For example, trained dataset of ArgoUML was used to detect anti-patterns of the whole system Xerces. The results obtained were significantly better than DETEX and the results can be generalised for any system. We cannot perform experiment on other anti-patterns detection due to the lack of manually validated oracle. But looking at the nature of Blob detection, we believe that SVMDetect would even perform better when detecting other anti-patterns.

Thus, we answer RQ1: "How does the accuracy of SVMDetect compare with that of DETEX, in terms of precision and recall?" as follows: on subsets of systems, SVMDetect dramatically outperforms DETEX, while on entire systems, SVMDetect detects more occurrences of Blob than DETEX.

6. THREATS TO VALIDITY

We now discuss threats to the validity of our results.

Construct validity: Threats to construct validity concern the relation between theory and observation. In our study, they are mainly related to the identification of classes suspected to be anti-patterns. To reduce the effect of this threat, the occurrences of anti-patterns have been manually validated by independent engineers.

Internal Validity: Threats to internal validity concern the dependence of the obtained results that depend on the chosen anti-patterns and systems. These threats do not affect our study because we used four well-known and representative anti-patterns. These anti-patterns also have been used in previous works. We also used three open-source systems with different sizes, which have been used by previous researchers.

Reliability Validity: Reliability validity threats concern the possibility of replicating the study concerned. To mitigate this threat, we used three open-source systems that can be freely downloaded from the Internet. We attempted to provide all the necessary details to replicate our study. Moreover, the results of the validation and the datasets are available on-line.

External Validity: Threats to external validity concern the possibility to generalise our results. We studied three systems with different sizes and different domains. Further, we also used a representative subset of anti-patterns. However, we will apply SVMDetect on other systems and anti-patterns in future work.

Table 1: **Precision (left) and Recall (right) of SVMDetect vs. DETEX in subsets (%)**

Precision:

		ArgoUML	Azureus	Xerces
Blob	DETEX	0.00	0.00	0.00
	SVMDetect	97.09	97.32	95.51
FD	DETEX	0.00	0.00	0.00
	SVMDetect	70.68	72.01	66.93
SC	DETEX	0.00	0.00	0.00
	SVMDetect	85.00	88.00	86.00
SAK	DETEX	10.00	10.00	0.00
	SVMDetect	75.46	84.54	80.76

Recall:

		ArgoUML	Azureus	Xerces
Blob	DETEX	0.00	0.00	0.00
	SVMDetect	84.09	91.33	95.29
FD	DETEX	0.00	0.00	0.00
	SVMDetect	57.50	84.28	70.00
SC	DETEX	0.00	0.00	0.00
	SVMDetect	71.00	89.00	86.00
SAK	DETEX	0.00	0.00	0.00
	SVMDetect	77.14	85.71	75.50

7. CONCLUSION AND FUTURE WORK

Anti-patterns are a fact of developers' life when developing software systems, under the conditions prevailing nowadays: distribution in time and space, time pressure and complexity. Anti-patterns in particular impede program comprehension and thus have negative impact on both development and maintenance activities. We observed, as other authors [13], that current anti-patterns detection approaches have some limitations: they require extensive knowledge of anti-patterns, they have limited precision and recall, and they cannot be applied on subsets of systems. To overcome these limitations, we introduced a novel approach to detect anti-patterns, SVMDetect, based on support vector machines (SVM). We designed an empirical study that allowed us to compare the results of DETEX, the state-of-the-art approach by Moha et al. [13] based on rules, with that of SVMDetect, our approach based on a SVM. We overcame the difficulty of finding a training set for SVMDetect and of applying DETEX and SVMDetect on the same input. We performed experiments to show how SVMDetect performs on a set of three systems (ArgoUML v0.19.8, Azureus v2.3.0.6, and Xerces v2.7.0) and four anti-patterns (Blob, Functional Decomposition, Spaghetti Code, and Swiss Army Knife).

We showed that the accuracy of SVMDetect is greater than that of DETEX when detecting anti-patterns occurrences on a set of classes. Concerning the whole system, SVMDetect is able to find more anti-patterns occurrences than DETEX.

We thus conclude that our conjecture is correct: a SVM-based approach can overcome the limitations of the previous approaches and could be more readily adopted by practitioners.

Future work includes performing an empirical study taking into account the user feedback. It also includes the use of SVMDetect in real-world environments. We would ask our industrial partners help in realising such a study. Further, we would also reproduce the study with other systems and anti-patterns to increase our confidence in the generalisability of our conclusions. Another interesting study could be the evaluation of the impact of the quality of feedback on SVMDetect results.

8. REFERENCES

[1] E. H. Alikacem and H. A. Sahraoui. Détection d'anomalies utilisant un langage de règle de qualité. In *LMO*. Hermes Science Publications, 2006.

[2] J. Bedo, C. Sanderson, and A. Kowalczyk. An efficient alternative to svm based recursive feature elimination with applications in natural language processing and bioinformatics. In A. Sattar and B.-h. Kang, editors, *AI 2006: Advances in Artificial Intelligence*, volume 4304 of *Lecture Notes in Computer Science*, pages 170–180. Springer Berlin Heidelberg, 2006.

[3] G. Cao, J.-Y. Nie, J. Gao, and S. Robertson. Selecting good expansion terms for pseudo-relevance feedback. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008*, pages 243–250. ACM, 2008.

[4] M. J. Choi, A. Torralba, and A. S. Willsky. A tree-based context model for object recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34:240–252, Feb. 2012.

[5] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. 10.1007/BF00994018.

[6] Y.-G. Guéhéneuc, H. Sahraoui, and Farouk Zaidi. Fingerprinting design patterns. In E. Stroulia and A. de Lucia, editors, *Proceedings of the* 11^{th} *Working Conference on Reverse Engineering (WCRE)*. IEEE Computer Society Press, November 2004.

[7] M. Kessentini, S. Vaucher, and H. Sahraoui. Deviance from perfection is a better criterion than closeness to evil when identifying risky code. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ASE '10, pages 113–122, New York, NY, USA, 2010. ACM.

[8] F. Khomh, M. D. Penta, and Y.-G. Guéhéneuc. An exploratory study of the impact of antipatterns on class change-and fault-proneness. *Journal of Empirical Software Engineering (EMSE)*, 2011.

[9] S. Kim, E. J. W. Jr., and Y. Zhang. Classifying software changes: Clean or buggy? *IEEE Trans. Software Eng.*, 34(2):181–196, 2008.

[10] G. Langelier, H. Sahraoui, and P. Poulin. Visualization-based analysis of quality for large-scale software systems. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, pages 214–223, New York, NY, USA, 2005. ACM.

[11] R. Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *In Proceedings of the IEEE 20th International Conference on Software Maintenance*. IEEE Computer Society Press, 2004.

[12] L. Setia, J. Ick, and H. Burkhardt. Svm-based relevance feedback in image retrieval using invariant feature histograms.

[13] Naouel Moha, Y.-G. Guéhéneuc, L. Duchien, and A.-F. L. Meur. DECOR: A method for the specification and detection of code and design smells. *Transactions on Software Engineering (TSE)*, 2009.