Support Vector Machines for Anti-pattern Detection

Abdou Maiga, Nasir Ali, Neelesh Bhattacharya, Aminata Sabané, Yann-Gaël Guéhéneuc, Giuliano Antoniol, Esma Aïmeur

DGIGL et DIRO, École Polytechnique et Université de Montréal, Québec, Canada

E-mails: {abdou.maiga, nasir.ali, n.bhattacharya, a.sabane, giuliano.antoniol}@polymtl.ca, {guehene, aimeur}@iro.umontreal.ca

ASE 2012
August 15, 2013
Outline

Introduction

Related Work

Our Approach: SVMDetect

Study Results

Discussions

Conclusion and Future Work

Questions
Introduction

Motivation

- Anti-patterns: “poor” solutions to recurring design and implementation problems.
- **Impact** program comprehension, software evolution and maintenance activities [8].
- Important to detect them early in software development process, 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**
- they cannot be **applied on subsets** of systems.

We propose

- **Apply SVM on subsets** 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.
Introduction

Contributions

- **SVMDetect** to detect anti-patterns using SVM
- Use of precision and recall to **compare SVMDetect to DETEX** [13], the best state-of-the-art approach, on 3 programs and 4 anti-patterns.
- The **accuracy** of SVMDetect is **greater** than of DETEX on subsets.
- For whole system, SVMDetect find **more anti-patterns occurrences** than DETEX.

We thus conclude that: a **SVM-based approach** can **overcome** the **limitations** of previous approaches.
Related Work

Smell/Anti-pattern Detection

Many researchers studied anti-patterns detection.

- Alikacem et al. [1] used meta-model for representing the source code and fuzzy thresholds.
- Langelier et al. [10] used a visual approach.
- Sahraoui et al. [7] used search-based techniques.
- Moha et al. [13] proposed an approach based on a set of rules that describes each anti-pattern.

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
- required sufficient knowledge of anti-patterns.
Related Work

SVM Applications

- SVM in several domains for various applications, e.g., bioinformatics [2], object recognition [4].
- SVM is a recent alternative to the classification problems.
- Guihong et al. [3] used SVM, for terms classification.
- SVM used in image retrieval systems by Sethia et al. [12]
- 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.
SVMDetect is based on Support Vector Machines (SVM) using a polynomial kernel to detect occurrences of anti-patterns.

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.
Our Approach: SVMDetect

SVMDetect - Steps

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 $TDS$ with object-oriented metrics for classes.

- **Step 2 (Train the SVM Classifier)**
  Train SVMDetect with $TDS$ defined in Step 1.

- **Step 3 (Construction of the dataset $DDS$ and detection of the occurrences of an anti-pattern)**
  Build detection dataset $DDS$ and apply SVMDetect trained in step 2 to $DDS$.

We use Weka to implement SVMDetect using its SVM classifier.
Empirical Study

- **goal**: validate that SVMDetect can **overcome** previous approaches’ limitations

- **quality focus**: **accuracy** of SVMDetect, in terms of **precision** and **recall**.

- **perspective**: researchers and practitioners interested in **verifying** if SVMDetect can be **effective** in detecting various kinds of anti-patterns, and in **overcoming** the previous limitations.
Empirical Study

Research Questions

- 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?
Empirical Study

<table>
<thead>
<tr>
<th>Objects</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Names</strong></td>
<td><strong>Versions</strong></td>
<td><strong># Lines of Code</strong></td>
<td><strong># Classes</strong></td>
<td><strong># Interfaces</strong></td>
</tr>
<tr>
<td>ArgoUML</td>
<td>0.19.8</td>
<td>113,017</td>
<td>1,230</td>
<td>67</td>
</tr>
<tr>
<td>Azureus</td>
<td>2.3.0.6</td>
<td>191,963</td>
<td>1,449</td>
<td>546</td>
</tr>
<tr>
<td>Xerces</td>
<td>2.7.0</td>
<td>71,217</td>
<td>513</td>
<td>162</td>
</tr>
</tbody>
</table>

**Table:** Description of the objects of the study
Empirical Study

Subjects

The subjects of our study are the following four anti-patterns:

- Blob
- Functional Decomposition (FD)
- Spaghetti Code (SC)
- Swiss Army Knife (SAK)

These four anti-patterns because known anti-patterns, commonly studied in previous work for comparison.
Study Results

Subsets of System: RQ1

<table>
<thead>
<tr>
<th>Subsets of System</th>
<th>ArgouML</th>
<th>Azureus</th>
<th>Xerces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blob</td>
<td>DETEX</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>SVMDetect</td>
<td>97.09</td>
<td>97.32</td>
</tr>
<tr>
<td>FD</td>
<td>DETEX</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>SVMDetect</td>
<td>70.68</td>
<td>72.01</td>
</tr>
<tr>
<td>SC</td>
<td>DETEX</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>SVMDetect</td>
<td>85.00</td>
<td>88.00</td>
</tr>
<tr>
<td>SAK</td>
<td>DETEX</td>
<td>10.00</td>
<td>10.00</td>
</tr>
<tr>
<td></td>
<td>SVMDetect</td>
<td>75.46</td>
<td>84.54</td>
</tr>
</tbody>
</table>
### Study Results

#### Subsets of System: RQ1

**Table:** Recall of SVMDetect vs. DETEX in subsets (%)

<table>
<thead>
<tr>
<th>Subset</th>
<th>DETEX</th>
<th>SVMDetect</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgouML</td>
<td>0.00</td>
<td>84.09</td>
</tr>
<tr>
<td>Azureus</td>
<td>0.00</td>
<td>91.33</td>
</tr>
<tr>
<td>Xerces</td>
<td>0.00</td>
<td>95.29</td>
</tr>
<tr>
<td>FD</td>
<td>0.00</td>
<td>57.50</td>
</tr>
<tr>
<td>DETEX</td>
<td>0.00</td>
<td>84.28</td>
</tr>
<tr>
<td>SVMDetect</td>
<td>0.00</td>
<td>70.00</td>
</tr>
<tr>
<td>SC</td>
<td>0.00</td>
<td>71.00</td>
</tr>
<tr>
<td>DETEX</td>
<td>0.00</td>
<td>89.00</td>
</tr>
<tr>
<td>SVMDetect</td>
<td>0.00</td>
<td>86.00</td>
</tr>
<tr>
<td>SAK</td>
<td>0.00</td>
<td>77.14</td>
</tr>
<tr>
<td>DETEX</td>
<td>0.00</td>
<td>85.71</td>
</tr>
<tr>
<td>SVMDetect</td>
<td>0.00</td>
<td>75.50</td>
</tr>
</tbody>
</table>
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.
- on entire systems, SVMDetect detects more occurrences of Blob than DETEX.
Discussions

Threats to Validity

Threats to the validity of our results.

- **Construct validity** (Measurement errors, subjectivity): occurrences of anti-patterns **manually validated**.
- **Internal Validity** (dependence of the obtained results on chosen anti-patterns and systems.): used four well-known and representative anti-patterns. used in previous works. used 3 **open-source systems with different sizes**, used in previous works.
- **Reliability Validity** (possibility of replication): used 3 **open-source systems available on-line**.
- **External Validity** (Generalisability): 3 systems with different sizes and different domains. Representative subset of anti-patterns.
Conclusion and Future Work

Conclusion

- introduced a novel approach to detect anti-patterns, SVMDetect, based on SVM.
- SVMDetect performs on 3 systems (ArgoUML v0.19.8, Azureus v2.3.0.6, and Xerces v2.7.0) and 4 anti-patterns (Blob, Functional Decomposition, Spaghetti Code, and Swiss Army Knife)
- the accuracy of SVMDetect is greater than that of DETEX on a subset of classes.
- on whole system, SVMDetect is able to find more anti-patterns occurrences than DETEX
- SVM-based approach can overcome the limitations of the previous approaches and could be more readily adopted by practitioners.
Future Work

Future work includes:

- use SVMDetect in real-world environments.
- reproduce the study with other systems and anti-patterns to increase our confidence in the generalisability of our conclusions.
- take into account the user feedback.
- evaluate the impact of the quality of training dataset and feedback set on SVMDetect results.
References

E. H. Alikacem and H. A. Sahraoui. Détection d’anomalies utilisant un langage de règle de qualité.

J. Bedo, C. Sanderson, and A. Kowalczyk. An efficient alternative to svm based recursive feature elimination with applications in natural language processing and bioinformatics.
References

G. Cao, J.-Y. Nie, J. Gao, and S. Robertson. Selecting good expansion terms for pseudo-relevance feedback.


References


References


References

R. Marinescu.
Detection strategies: Metrics-based rules for detecting design flaws.

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

DECOR: A method for the specification and detection of code and design smells.
Transactions on Software Engineering (TSE), 2009.