Low-shot classification for handling class imbalancelement in software defects prediction

Ioana-Gabriela Chelaru
Babeș-Bolyai University

WeADL 2025 Workshop

The workshop is organized under the umbrella of WinDMiL, project funded by CCCDI-UEFISCDI, project number PN-IV-P7-7.1-PED-2024-0121, within PNCDI IV

Contents

What is Software Defect Prediction (SDP)?

- SDP: Detecting software modules likely to contain faults in new releases.
- Essential for all software lifecycle stages:
  - Development
  - Testing
  - Maintenance
  - Evolution
- Guides testing and code review by highlighting risky code areas.

Why is SDP Important?

- Early defect detection $\rightarrow$ higher software quality.
- Efficient resource allocation for testing and maintenance.
- Reduces costs and prevents failures in production.

Challenges in SDP

- **Severe class imbalance:** Defective modules are rare [1].
- **Data scarcity:** Few labeled defects, especially in new projects.
- **Feature selection:** determining the relevant features correlated with the error-proneness of a piece of code [2, 3]

Literature review

- Conventional ML, evolutionary, fuzzy, and deep learning methods [4].
- Few-shot and one-shot learning (FSL/OSL) with Siamese Networks: promising for low-data, imbalanced scenarios.
- Previous SNN-SDP models: shallow networks [5] trained on small and outdated datasets.

Few-shot Learning & Siamese neural networks

The Few-shot Learning Paradigm focuses on obtaining **accurate predictions** when training **data is scarce**.

Siamese neural networks: two neural networks that receive different inputs and are joined at the top by function that computes the distance between the feature-based representations of the inputs, thus deciding if the inputs belong to the same class or not

Research Questions

- **RQ1:** How to design an FSL-based approach using SNNs to predict the defect-proneness of a software entity?
- **RQ2:** To what extent does the feature-based representation used to characterize the software entities influence the fault detection performance?
- **RQ3:** How does the predictive performance of our proposed approach compare to similar related work?
- **RQ4:** What are the insights that can be obtained from applying an explainability method to our FSL-based classifier?

Case study: Apache Calcite

- 16 versions of Apache Calcite.
- Each class: 3278 software metrics, binary defect label.
- Highly imbalanced: Defect rate drops to 3.3% in later versions.

Version	# Classes	# Defects	Defective rate	Version	# Classes	# Defects	Defective rate
1.0.0	1075	178	0.166	1.8.0	1301	101	0.078
1.1.0	1103	113	0.102	1.9.0	1310	90	0.069
1.2.0	1108	126	0.114	1.10.0	1310	84	0.064
1.3.0	1115	112	0.100	1.11.0	1331	80	0.060
1.4.0	1127	123	0.109	1.12.0	1415	81	0.057
1.5.0	1176	103	0.088	1.13.0	1275	53	0.042
1.6.0	1193	107	0.090	1.14.0	1308	53	0.041
1.7.0	1252	128	0.102	1.15.0	1352	45	0.033

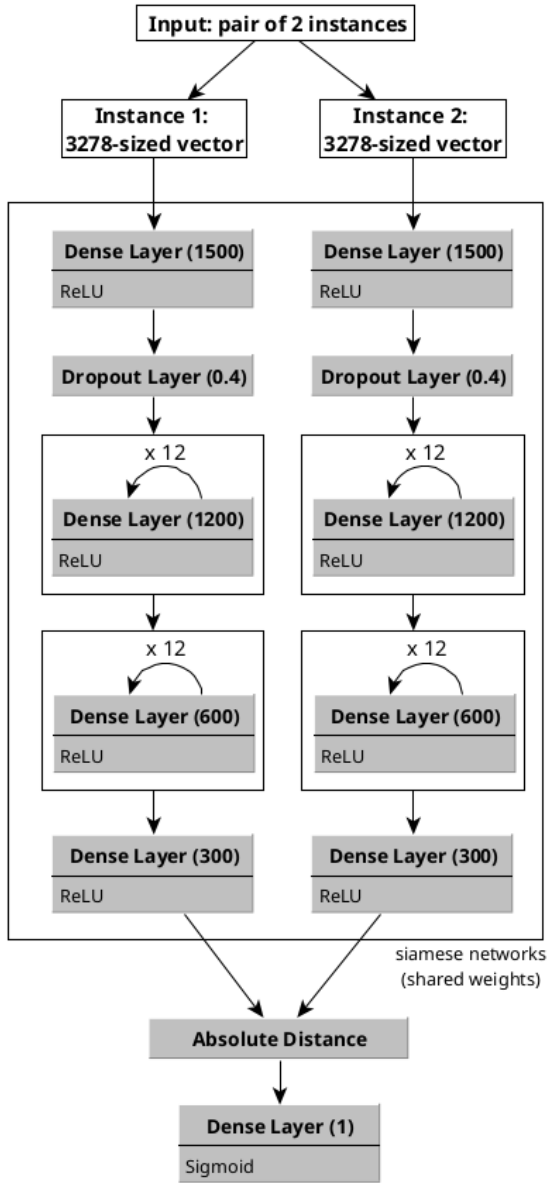
FS-iSDP: the architecture - RQ1

Two identical neural networks composed of 26 Dense hidden layers and one dropout (40%) layer:

- 1st layer of size 1500
- 12 layers of size 1200
- 12 layers of size 600
- last layer of size 600

Input: 2 instances represented by 3278-sized vectors

Output: the predicted probability that the two instances are in the same class or in different classes



Model training

- **Objective:** to predict defects for version $x \in [1, 15]$, where the training dataset is $\bigcup_{i=0}^{x-1} Calcite_i$
- **Data preprocessing:** randomly extract a subset of non-defects to rebalance the number of defects to non-defects and reduce the model's bias
- The training dataset is split into *validation* (30%) and *training* (70%)
- The model is fed pairs of instances containing at least one positive instance to further emphasize the recognition of defects

Classification

Let c denote an instance from the testing dataset, $score_+(c)$ be the average similarity between c and the defective *training* instances and $score_-(c)$ is the average similarity between c and the non-defective *training* instances.

The classification of c is then the class with the highest score, i.e.,

$$class(c) = \begin{cases} + & \text{if } score_+(c) > score_-(c) \\ - & otherwise \end{cases}$$

Performance evaluation

$\forall k, 1 \leq k \leq 15$, the model is trained on releases $0..k-1$ and tested on release k

Based on the values from the confusion matrix we compute:

- *Precision* for the positive (defect) class
- *Probability of detection* (Recall or Sensitivity)
- *Critical success index* (CSI)
- *Specificity* or *true negative rate*
- *Area under the Receiver Operator Characteristic curve* (AUC)
- *Area under the Precision-Recall curve* (AUPRC)
- *Matthews Correlation Coefficient* (MCC)
- *F1-score* for the defect (positive)

Results and discussion I

Results obtained for the 9 experiments performed based on the best 2500 features (selected using *Univariate feature selection*)

Version for testing (k)	**Versions for training (0..k-1)**	**TP**	**FP**	**TN**	**FN**	*Prec*(↑)	*POD* (↑)	*Spec* (↑)	*CSI* (↑)	*AUC* (↑)	*AUPRC* (↑)	*MCC*(↑)	*F1* (↑)
1.7.0	1.0.0-1.6.0	123	404	720	5	0.233	0.961	0.641	0.231	0.801	0.597	0.369	0.376
1.8.0	1.0.0-1.7.0	95	287	913	6	**0.249**	0.941	0.761	0.245	0.851	0.595	0.412	0.393
1.9.0	1.0.0-1.8.0	77	254	966	13	0.233	0.856	0.792	0.224	0.824	0.544	0.377	0.366
1.10.0	1.0.0-1.9.0	83	252	974	1	0.248	0.988	0.794	**0.247**	0.891	**0.618**	**0.439**	**0.396**
1.11.0	1.0.0-1.10.0	69	218	1033	11	0.240	0.863	**0.826**	0.232	0.844	0.551	0.398	0.376
1.12.0	1.0.0-1.11.0	69	234	1100	12	0.228	0.852	0.825	0.219	0.838	0.540	0.383	0.359
1.13.0	1.0.0-1.12.0	49	231	991	4	0.175	0.925	0.811	0.173	0.868	0.550	0.355	0.294
1.14.0	1.0.0-1.13.0	47	235	1020	6	0.167	0.887	0.813	0.163	0.850	0.527	0.335	0.281
1.15.0	1.0.0-1.14.0	45	239	1068	0	0.158	**1.000**	0.817	0.158	**0.909**	0.579	0.360	0.274

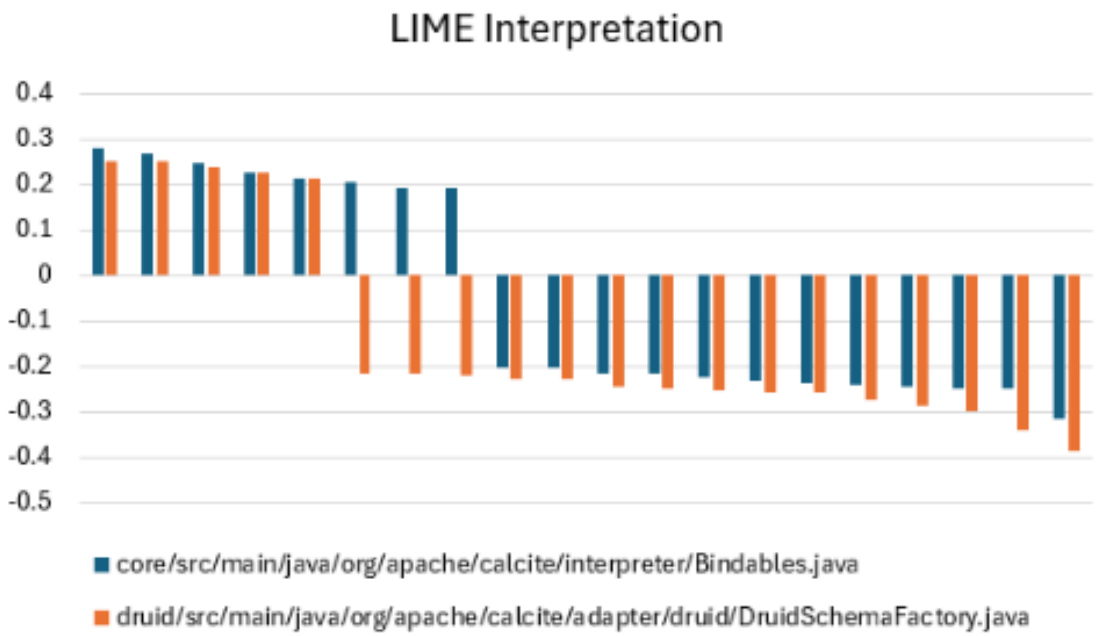
Results and discussions II - RQ2

By applying *Univariate feature selection* we achieved:

- precision was improved by 16%
- the model's ability to distinguish between classes benefited: AUC increased by 2.5%, while the AUPRC improved by 6.6%
- the False Positive Rate decreased by 24%
- the False Negative Rate rose slightly from 0.068 to 0.081, reflecting a trade-off in favor of detecting more defects
- the training time per fold was reduced by approximately 25–30%

Interpretability - RQ4

- Applied LIME to 2 false-positives from version 1.10.0 to observe the 20 most important features that contributed to the misclassification
- We note that even if more than 10 software metrics contribute to the classification of the instances as belonging to the non-defective class, their contribution is relatively low, as the instances are still misclassified



Comparison to Related Work - RQ3

Model	**Dataset**	*Prec*(↑)	*POD* (↑)	*Spec* (↑)	*CSI* (↑)	*AUC* (↑)	*AUPRC* (↑)	*MCC*(↑)	*F*1 (↑)
$OCSVM_+$[1]	Calcite	0.102	0.602	0.563	0.095	0.582	0.352	0.087	0.172
$OCSVM_-$ [1]	Calcite	0.103	0.598	0.565	0.095	0.582	0.351	0.085	0.172
SVC [1]	Calcite	**0.447**	0.642	**0.921**	**0.356**	0.782	0.545	**0.479**	**0.521**
Our `FS-iSDP`	Calcite	0.215	**0.919**	0.787	0.210	**0.853**	**0.567**	0.381	0.346

- Our model outperforms all other models in recall, AUC, and AUPRC, thus making it better suited in imbalanced classification tasks
- The SVC model shows strong performance in precision, specificity, and MCC, suggesting that it is highly conservative and less likely to produce false positives

Overall, our `FS-iSDP` model achieves balanced performance across all metrics and demonstrates resilience to data imbalance.

Conclusion

- FS-iSDP: Robust, interpretable, and effective for realistic SDP scenarios.
- Handles severe imbalance and data scarcity.
- Outperforms prior SNN and traditional models on Calcite.
- Provides actionable insights via LIME.

Thank you!

Q&A

Bibliography I


G. Ciubotariu, G. Czibula, I. G. Czibula, and I.-G. Chelaru, "Uncovering behavioural patterns of one- and binary-class SVM-based software defect predictors," in *Proceedings of ICSOFT 2023*, pp. 249–257, SciTePress, 2023.


A. Briciu, G. Czibula, and M. Lupea, "A study on the relevance of semantic features extracted using bert-based language models for enhancing the performance of software defect classifiers," *Procedia Computer Science*, vol. 225, pp. 1601–1610, 2023.


G. Czibula, I.-G. Chelaru, I. G. Czibula, and A.-J. Molnar, "An unsupervised learning-based methodology for uncovering behavioural patterns for specific types of software defects," *Procedia Computer Science*, vol. 225, pp. 2644–2653, 2023.


I. Batool and T. A. Khan, "Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review," *Computers and Electrical Engineering*, vol. 100, p. 107886, 2022.


L. Zhao, Z. Shang, L. Zhao, A. Qin, and Y. Y. Tang, "Siamese dense neural network for software defect prediction with small data," *IEEE Access*, vol. 7, pp. 7663–7677, 2019.