A study on the relevance of semantic features extracted using BERT-based language models for enhancing the performance of software defect classifiers

Anamaria Briciu, Gabriela Czibula, Mihaiela Lupea

Department of Computer Science, Babeș-Bolyai University Cluj-Napoca, Romania

Outline

Outline

Software defect prediction

Identifying the software entities (classes, modules, methods, functions, etc.) that are defective in a new version of a software system

Relevance:

- improvement of software quality
- ease software maintenance and evolution

Proposed study

Investigate the relevance of source code representations automatically learned using pre-trained language models.

Contributions

1. examining the discriminating power of extracted BERT-based embeddings in the context of software defect prediction
2. analysis of the capacity of natural language patterns to discriminate between defective and non-defective instances
3. novel evaluation strategy

Research questions

RQ1 What is the relevance of the programming language-specific information learned by source code-based pre-trained models compared to the features encoded by natural-language based pre-trained models in a task of SDP?

RQ2 To what extent does the use of deep semantic and contextual features of the source codes extracted using pre-trained BERT-based language models improve the performance of software defect predictors compared to the semantic features learned by natural language-based models such as `doc2vec` and LSI?

Outline

Semantic features used for SDP

- Abstract Syntax Trees-based representations [WLT16, WLNT18, MHX+20, DPN+19]
- graph-based representations [PLNB17]
- token embeddings [HYLZ18, ULA+22] and document embeddings [MTC22]

Classification models

- CNN [MHX+20, PLNB17]
- LSTM, BiLSTM [DPN+19, MVAK+20, ULA+22, LL21]
- DBN [WLT16, WLNT18]
- BERT fine-tuning [PLX21]

Outline

Case study

The Apache Calcite data set is used [BCRH+18].

- 16 versions (1.0.0 - 1.15.0)
- ground truth labels for all included application classes:
  - $+$ = defective
  - $-$ = non-defective
- Data imbalance: defective rate varies from *0.033* (version 1.15.0) to *0.166* (version 1.0.0)

Formalization of the SDP problem

Formalization as a supervised binary classification problem.

- an object-oriented software system $\mathcal{S} = \{c_1, c_2, \ldots, c_n\}$ consisting of software entities (i.e. application classes)
- software entities characterized by m features: $f_1, \ldots f_m \rightarrow$ each application class c_i, $1 \leq i \leq n$ is represented by a high dimensional numerical vector $c_i = (c_{i1}, \ldots, c_{im})$, where c_{ij} $(\forall 1 \leq j \leq m)$ is the value of the feature f_j computed/learned for the application class c_i.
- **GOAL**: predict if a certain software entity is likely to belong to the $+$ or $-$ target class

Data representation

Source code embeddings obtained using 2 BERT-based models: RoBERTa-base (a natural language model) & CodeBERT-base-MLM (a programming language model).

Feature extraction approach

1. Remove comments, documentation
2. Tokenize source code files (maximum length = 512, padding and truncation applied)
3. Extract last hidden layer representations from BERT model (512x768)
4. Apply mean pooling to obtain a representation for the entire input sequence

The SDP classifier

The SDP classifier used is based on an artificial neural network (ANN).

- architecture: 768 (input) - 128 - 32 - 16 - 1 (output)
- hidden layers: ReLU activation function
- output layer: sigmoid activation function
- to account for data imbalance: class weights
- early stopping criterion based on AUPRC value on the validation set

Training & classification

Follow the historical evolution of the Calcite software and assess the real-life defect prediction capabilities of the classifier.

- **train** ANN on the application classes from versions $0..k$
  - 80% will be used for *training*
  - 20% will be used for *validation*
- **test** on version $k+1$ $(\forall k, 0 \leq k \leq 14)$.

Testing & evaluation: Evaluation

After classification, the *TP (true positive)*, *TN (true negative)*, *FP (false positive)* and *FN (false negative)* are reported in a confusion matrix.

Metrics used:

- **PPV** *(precision for the positive class)* $= \frac{TP}{TP+FP}$
- **POD** *(probability of detection)* $= \frac{TP}{TP+FN}$
- *Spec (specificity* or *true negative rate)* $= \frac{TN}{TN+FP}$
- *FAR (false alarm ratio)* $= \frac{FP}{TP+FP}$
- *CSI (critical success index)* $= \frac{TP}{TP+FN+FP}$
- *AUC (Area under the ROC curve)* $= \frac{POD+Spec}{2}$
- *MCC (Matthews Correlation Coefficient)* $= \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP) \cdot (TP+FN) \cdot (TN+FP) \cdot (TN+FN)}}$
- *F1 (F-score for the positive class)* $= \frac{2 \cdot PPV \cdot POD}{PPV+POD}$.

Outline

Results (I)

Version k	Model used for code embedding	TP	FP	TN	FN	*PPV* (↑)	*POD* (↑)	*Spec* (↑)	*FAR* (↓)	*CSI* (↑)	*AUC* (↑)	*MCC* (↑)	*F1* (↑)
1.0.0	**RoBERTa**	50	136	854	63	0.269	0.442	0.863	0.731	0.201	0.653	0.247	0.334
	CodeBERT-MLM	69	97	893	44	0.416	0.611	0.902	0.584	0.329	0.756	0.435	0.495
1.1.0	**RoBERTa**	109	60	922	17	0.645	0.865	0.939	0.355	0.586	0.902	0.775	0.739
	CodeBERT-MLM	100	24	958	26	0.806	0.794	0.976	0.194	0.667	0.885	0.753	0.800
1.2.0	**RoBERTa**	98	43	960	14	0.695	0.875	0.957	0.305	0.632	0.916	0.681	0.775
	CodeBERT-MLM	106	88	915	6	0.546	0.946	0.912	0.454	0.530	0.929	0.643	0.693
1.3.0	**RoBERTa**	100	72	932	23	0.581	0.813	0.928	0.419	0.513	0.871	0.643	0.678
	CodeBERT-MLM	105	136	868	18	0.436	0.854	0.865	0.654	0.405	0.859	0.546	0.577
1.4.0	**RoBERTa**	87	112	961	16	0.437	0.845	0.896	0.563	0.405	0.870	0.558	0.576
	CodeBERT-MLM	101	312	761	2	0.245	0.981	0.709	0.755	0.243	0.845	0.409	0.391
1.5.0	**RoBERTa**	99	113	973	8	0.467	0.925	0.896	0.533	0.450	0.911	0.614	0.621
	CodeBERT-MLM	90	68	1018	17	0.570	0.841	0.937	0.430	0.514	0.889	0.656	0.679
1.6.0	**RoBERTa**	109	90	1034	19	0.548	0.852	0.920	0.452	0.500	0.886	0.639	0.667
	CodeBERT-MLM	93	45	1079	35	0.674	0.727	0.960	0.326	0.538	0.843	0.664	0.699
1.7.0	**RoBERTa**	71	125	1075	30	0.362	0.703	0.896	0.638	0.314	0.799	0.448	0.478
	CodeBERT-MLM	95	113	1087	6	0.457	0.941	0.906	0.543	0.444	0.923	0.618	0.615
1.8.0	**RoBERTa**	81	155	1065	9	0.343	0.900	0.873	0.657	0.331	0.886	0.509	0.497
	CodeBERT-MLM	82	127	1093	8	0.392	0.911	0.896	0.608	0.378	0.904	0.557	0.548
1.9.0	**RoBERTa**	81	166	1060	3	0.328	0.964	0.865	0.672	0.324	0.914	0.519	0.489
	CodeBERT-MLM	80	114	1112	4	0.412	0.952	0.907	0.588	0.404	0.930	0.593	0.576
1.10.0	**RoBERTa**	61	91	1160	19	0.401	0.763	0.927	0.599	0.357	0.845	0.515	0.526
	CodeBERT-MLM	61	73	1178	19	0.455	0.763	0.942	0.545	0.399	0.852	0.556	0.570
1.11.0	**RoBERTa**	60	95	1239	21	0.387	0.741	0.929	0.613	0.341	0.835	0.498	0.508
	CodeBERT-MLM	69	182	1152	12	0.275	0.852	0.864	0.725	0.262	0.858	0.435	0.416
1.12.0	**RoBERTa**	47	156	1066	6	0.232	0.887	0.872	0.768	0.225	0.880	0.414	0.367
	CodeBERT-MLM	48	142	1080	5	0.253	0.906	0.884	0.747	0.246	0.895	0.442	0.395
1.13.0	**RoBERTa**	42	107	1148	11	0.282	0.792	0.915	0.718	0.263	0.854	0.439	0.416
	CodeBERT-MLM	42	108	1147	11	0.280	0.792	0.914	0.720	0.261	0.853	0.437	0.414
1.14.0	**RoBERTa**	42	157	1150	3	0.211	0.933	0.880	0.789	0.208	0.907	0.412	0.344
	CodeBERT-MLM	40	152	1155	5	0.208	0.889	0.884	0.792	0.203	0.886	0.397	0.338

Table: Experimental results. For each testing case (the SDP classifier trained on all Calcite releases from 0 to k, then tested on version $k+1$) and two code embeddings (generated using RoBERTa and CodeBERT-MLM), the obtained confusion matrix and the metrics values are provided.

Discussion (I)

- good performance of the SDP classifier for the **defect** class for most test configurations
- classifier trained on CodeBERT-MLM representations outperforms the one trained on RoBERTa representations
- RoBERTa-based model achieved comparable results

Results (II)

Difficulty	Model used to learn features	k (Training data from Calcite release 1.k.0. testing data from Calcite release 1.k+1.0)														
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Defect class	doc2vec+LSI [MTC22]	0.684	0.681	0.692	0.686	0.690	0.687	0.688	0.694	0.688	0.687	0.690	0.691	0.696	0.691	0.692
	CodeBERT-MLM	0.436	0.372	0.378	0.371	0.363	0.340	0.318	0.336	0.350	0.341	0.338	0.351	0.352	0.350	0.343
Non-Defect class	doc2vec+LSI [MTC22]	0.095	0.089	0.088	0.085	0.079	0.076	0.076	0.072	0.068	0.064	0.062	0.060	0.057	0.055	0.052
	CodeBERT-MLM	0.093	0.087	0.081	0.081	0.072	0.062	0.058	0.055	0.052	0.052	0.051	0.047	0.047	0.047	0.044
Overall	doc2vec+LSI [MTC22]	0.174	0.164	0.160	0.156	0.148	0.143	0.142	0.137	0.130	0.124	0.120	0.117	0.112	0.108	0.103
	CodeBERT-MLM	0.139	0.123	0.117	0.115	0.105	0.092	0.086	0.084	0.082	0.079	0.078	0.075	0.073	0.072	0.068

Table: Difficulty values for all fifteen testing configurations, for two representations: based on doc2vec+LSI, and learned by CodeBERT-MLM.

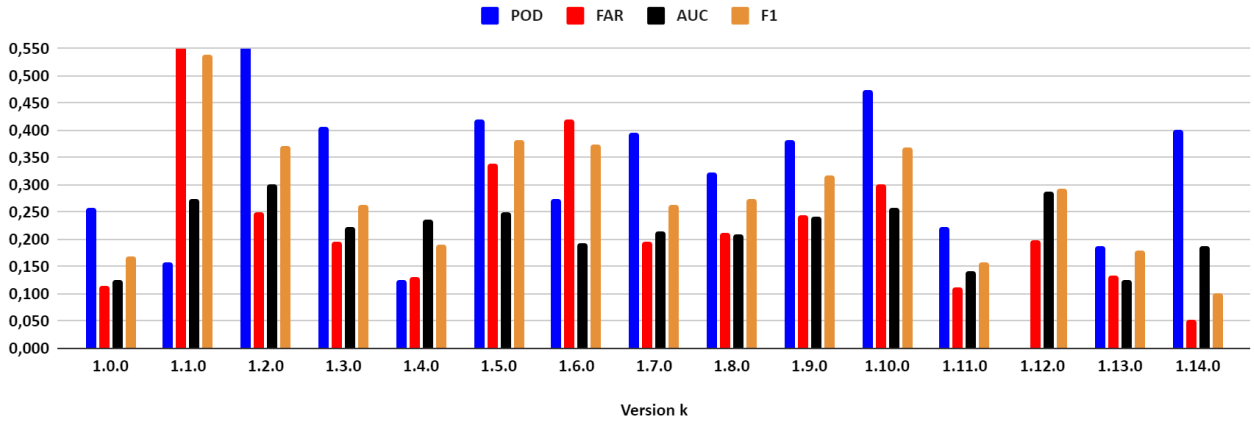

Figure: Classification improvement of CodeBERT-MLM representations over doc2vec+LSI representations in all testing configurations.

Discussion (II)

Same training & testing methodology used for the `doc2vec` and LSI representations introduced in [MTC22].

- significant differences in computed difficulty values for the positive (defect) class
- overall descending trend: highest difficulty obtained for testing configuration with $k = 0$
- considerable improvements observed when using BERT-based features as opposed to doc2vec+LSI features [MTC22]

Outline

Conclusions

- CodeBERT-MLM model, trained on source codes, provided more effective representations for detecting software defects than a model that focuses on natural language, such as RoBERTa
- the semantic and contextual features extracted by the pre-trained CodeBERT-MLM model better at discriminating between defective and non-defective source codes, compared to the features encoded in the source code semantic representations learned by `doc2vec` and LSI (two natural language-based models that learn only from the analyzed input source codes)
- **future work:** add comments from the source code and/or other software artifacts

Bibliography

Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J. Mior, and Daniel Lemire.
Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources.
In *SIGMOD '18: Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 221–230, New York, NY, USA, 2018. Association for Computing Machinery.

Hoa Khanh Dam, Trang Pham, Shien Wee Ng, Truyen Tran, John Grundy, Aditya Ghose, Taeksu Kim, and Chul-Joo Kim.
Lessons learned from using a deep tree-based model for software defect prediction in practice.
In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 46–57. IEEE, 2019.


Xuan Huo, Yang Yang, Ming Li, and De-Chuan Zhan.
Learning semantic features for software defect prediction by code comments embedding.
In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 1049–1054, 2018.


Junhao Lin and Lu Lu.
Semantic feature learning via dual sequences for defect prediction.
IEEE Access, 9:13112–13124, 2021.


Shi Meilong, Peng He, Haitao Xiao, Huixin Li, and Cheng Zeng.
An approach to semantic and structural features learning for software defect prediction.
Mathematical Problems in Engineering, 2020:1–13, 2020.


Diana-Lucia Miholca, Vlad-Ioan Tomescu, and Gabriela Czibula.

An in-depth analysis of the software features' impact on the performance of deep learning-based software defect predictors.
IEEE Access, 10:64801–64818, 2022.

Amirabbas Majd, Mojtaba Vahidi-Asl, Alireza Khalilian, Pooria Poorsarvi-Tehrani, and Hassan Haghighi.
SIdeep: Statement-level software defect prediction using deep-learning model on static code features.
Expert Systems with Applications, 147:113156, 2020.

Anh Viet Phan, Minh Le Nguyen, and Lam Thu Bui.
Convolutional neural networks over control flow graphs for software defect prediction.
In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 45–52. IEEE, 2017.

Cong Pan, Minyan Lu, and Biao Xu.
An empirical study on software defect prediction using CodeBERT model.
Applied Sciences, 11(11):Article No. 4793, 2021.


Md Nasir Uddin, Bixin Li, Zafar Ali, Pavlos Kefalas, Inayat Khan, and Islam Zada.
Software defect prediction employing bilstm and bert-based semantic feature.
Soft Computing, 26(16):7877–7891, 2022.


Song Wang, Taiyue Liu, Jaechang Nam, and Lin Tan.
Deep semantic feature learning for software defect prediction.
IEEE Transactions on Software Engineering, 46(12):1267–1293, 2018.


Song Wang, Taiyue Liu, and Lin Tan.
Automatically learning semantic features for defect prediction.
In *Proceedings of the 38th International Conference on Software Engineering*, pages 297–308, 2016.