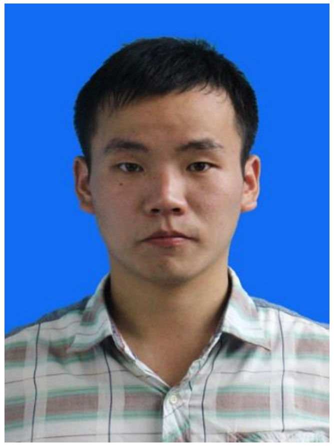
Cross-Entropy: A New Metric for Software Defect Prediction

Xian Zhang[*], Kerong Ben, Jie Zeng
Department of Computer and Data Engineering
Naval University of Engineering, Wuhan, China

Authors

Xian Zhan, born in 1990. PhD candidate. Student member of China Computer Federation. His research interests include software analysis, software defect mining, and machine learning.

Kerong Ben, born in 1963, PhD, professor, PhD supervisor. Senior member of China Computer Federation. His main research interests include software quality assurance and artificial intelligence.



Jie Zeng, born in 1993. PhD candidate. Student member of China Computer Federation. His research interests include software analysis, software defect mining, and machine learning.

Outline

Outline

Introduction

The accurate prediction of where defects are likely to occur in code is important since it can help direct test effort, reduce costs and improve the quality of software [29].

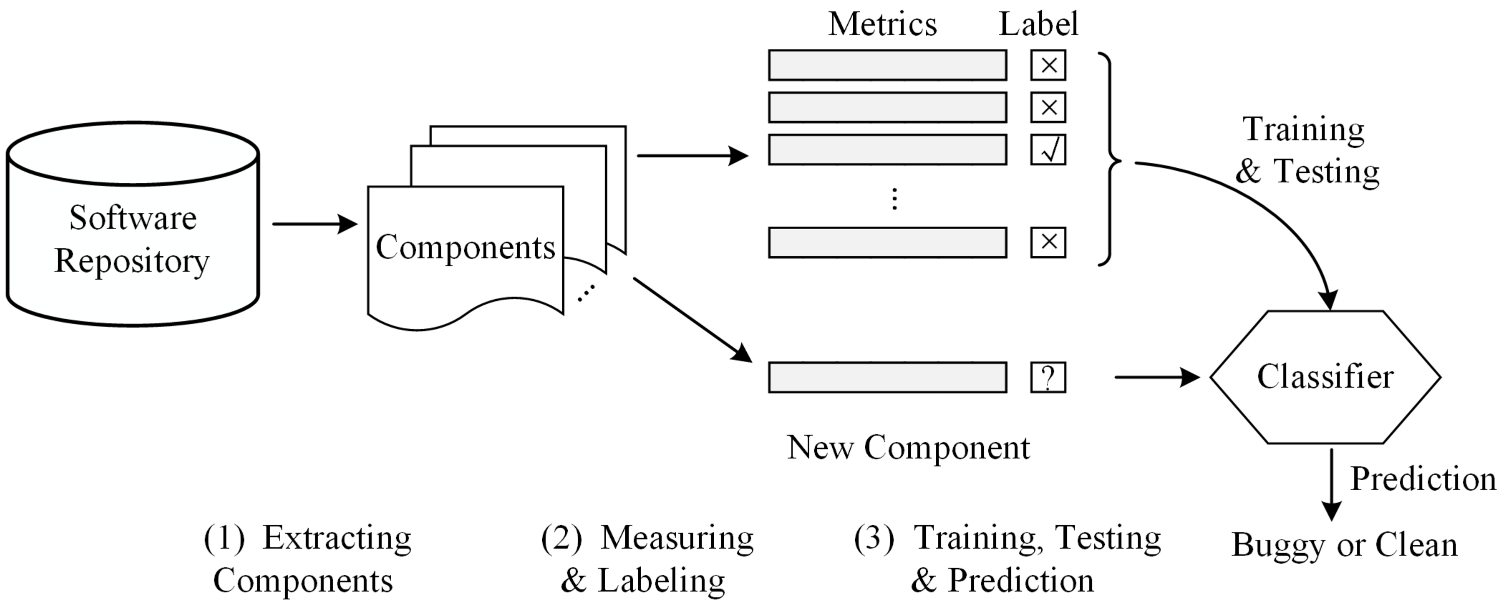


Fig. 1. A typical process of software defect prediction

(seen as a typical binary classification problem)

Introduction

To improve defect prediction performance, many software metrics (also called features) are proposed [8], e.g., McCabe metrics [31], CK metrics [25], QMOOD metrics [13], code change metrics [30], other process metrics [16], and automatically learned features [14][27].

However, there is no one metric set can perform well in any case yet. It indicates that relative to the diversity of defect patterns, the aforementioned metrics are the inadequate representation of software defects.

For this reason, this paper attempts to characterize defects from different perspective for better defect prediction.

Introduction

In recent years, the large and growing software repositories have gathered tremendous amount of source code and other data. This abundance of artifacts motivates new, data-driven approaches to analyze software [17]. From the perspective of **natural language processing** (NLP), a series of research has studied the statistical properties of code.

- Hindle, et al. "On the naturalness of software," (ICSE'12)
- Z. Tu, et al. "On the localness of software," (FSE'14)
- Ray, et al. "On the naturalness of buggy code," (ICSE'16)
- S. Wang, et al. "Bugram: Bug detection with n-gram language models," (ASE'16).
- M. Allamanis, et al. "A survey of machine learning for big code and naturalness," (arXiv: 1709.06182, 2017).

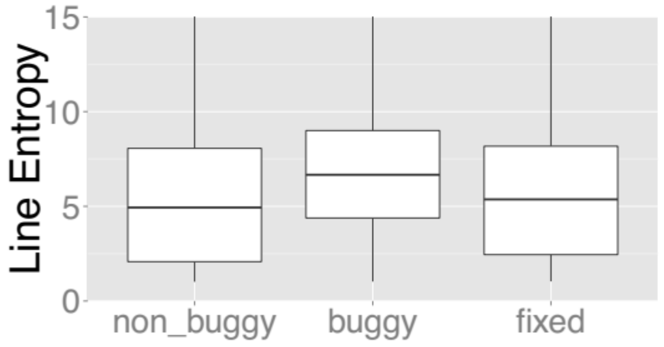
Introduction

Hindle et al. [2] first presented the **naturalness hypothesis**: "programs that real people actually write are mostly simple and repetitive, and thus they have usefully predictable statistical properties that can be captured in language models and leveraged for software engineering tasks".

Learning from a corpus, a language model will assign low **cross-entropy** (or high probability) to code that appears often in practice, i.e., is natural. In [2], the authors used the classic *n*-gram language model for code naturalness analysis.

Introduction

Based on this work, Ray et al. [4] suggested another hypothesis: unnatural code is more likely to be buggy. They first presented evidence that **code with bugs tends to be more entropic**, becoming less so as bugs are fixed. Wang et al. [26] leveraged n-gram language models to detect bugs and got better results than rule-based approaches.



(a) Entropy difference between non-buggy, buggy, and fixed lines at bug-fix threshold 7.

(Ray et al. [4])

Introduction

Inspired by these previous works, we plan to absorb cross-entropy as a new code metric into typical defect prediction at file level (*Contribution* ①).

We propose a recurrent neural network (RNN) based defect prediction framework, called *DefectLearner*, to automatically generate the cross-entropy metric of software components, and combine other metrics together for the identification of buggy code (*Contribution* ②).

We give a comprehensive evaluation of the effectiveness of cross-entropy metric. The experimental results suggest that **cross-entropy is complementary to traditional metrics** in defect prediction tasks (*Contribution* ③).

Outline

Background

A. Language model

Language model is a kind of basic model in NLP [7], aiming to learn the joint probability function of sequences of words in a language. Given a token sequence $S = w_1 w_2 \cdots w_l$, the probability of this sequence occurring can be estimated as a product of a series of conditional probabilities:

$$\begin{aligned} P(S) &= P(w_1) P(w_2 \mid w_1) \cdots P(w_l \mid w_1, w_2, \cdots, w_{l-1}) \\ &= \prod_{i=1}^{l} P(w_i \mid w_1, \cdots, w_{i-1}) \end{aligned} \quad (1)$$

where l denotes sequence length. In practice, given a corpus C of programs, we can use a language model to estimate $P(S)$ in the maximum-likelihood way.

Background

Neural language models are designed to model language sequences by using the semantic similarity between words. Commonly, the **distributed representation** of words [41] (also called **word-embeddings** or word vector) and neural network technique are used together to estimate the joint probability distribution of sequences (see (1)).

The neural language model was formally proposed by Bengio et al. in 2001 [40], and has made great progress in many NLP tasks [41], such as machine translation, information retrieval, and dialogue systems.

Background

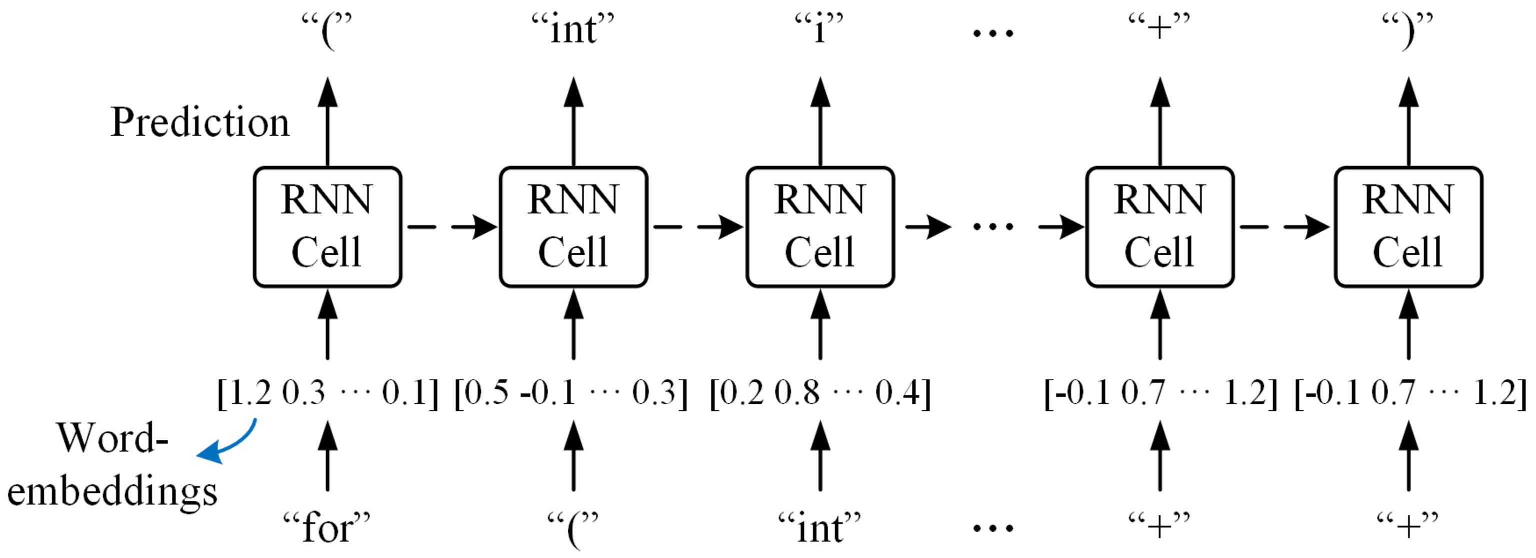


Fig. 2. Basic RNN language model with word-embeddings as inputs

The model aims at predicting a probability distribution over the "next" token, given some preceding tokens. Each token can be mapped into a fixed-length continuous vector. For example, **token "for"—> [1.2 0.3 … 0.1]**.

Background

B. Cross-Entropy

In information theory, cross-entropy is commonly used to quantify the difference between two probability distributions [7]. In this paper, it is used as a code metric which denotes the naturalness (i.e., the likelihood of occurrence) of software components [2][4].

As Ray et al. [4] present, unnatural code is more likely to be buggy, which means that *code with bugs tends to be more entropic*. Hence, we attempt to use the cross-entropy metric of software components for better defect prediction.

Background

Formally, given a corpus (or a software component, etc.) $C=\{S_1 S_2 \cdots S_N\}$ and a language model M, the cross-entropy of the model on the corpus can be calculated as follows:

$$\begin{aligned} CE_M(C) &= -\frac{1}{l_C}\log_2 P_M(C) \\ &= -\frac{1}{l_C}\sum_{j=1}^{N}\log_2 P_M(S_j) \qquad (2) \\ &= -\frac{1}{l_C}\sum_{j=1}^{N}\sum_{i=1}^{l_j}\log_2 P_M\left(w_{j,i} | w_{j,1},\cdots,w_{j,i\text{-}1}\right) \end{aligned}$$

where $S_j \in C$ is the j-th sequence in the corpus, N is the total number of sequences, l_j is the length of S_j, $l_C = \sum_{j=1}^{N} l_j$

Background

is the total length of the sequences in the corpus, $P_M\left(S_j\right)$ is the probability of occurrence of S_j estimated by M, $P_M\left(C\right)=\prod_{j=1}^{N}P\left(S_j\right)$ is the probability of occurrence of the corpus.

Perplexity is the exponential form of cross-entropy, which is defined as follows:

$$PP_M\left(C\right)=2^{CE_M\left(C\right)} \quad (3)$$

where $CE_M\left(C\right)\in\left[0,+\infty\right)$ and $PP_M\left(C\right)\in\left[1,+\infty\right)$. This paper uses perplexity as the loss function for language model training.

Outline

- Introduction
- Background
- **Approach**
- Evaluation
- Conclusion

Approach

A. Overall Framework

Fig. 3 illustrates the overview of our proposed ***DefectLearner***, a framework designed to automatically generate cross-entropy scores of software components with arbitrary granularity, and combine traditional metrics together for accurate software defect prediction.

Approach

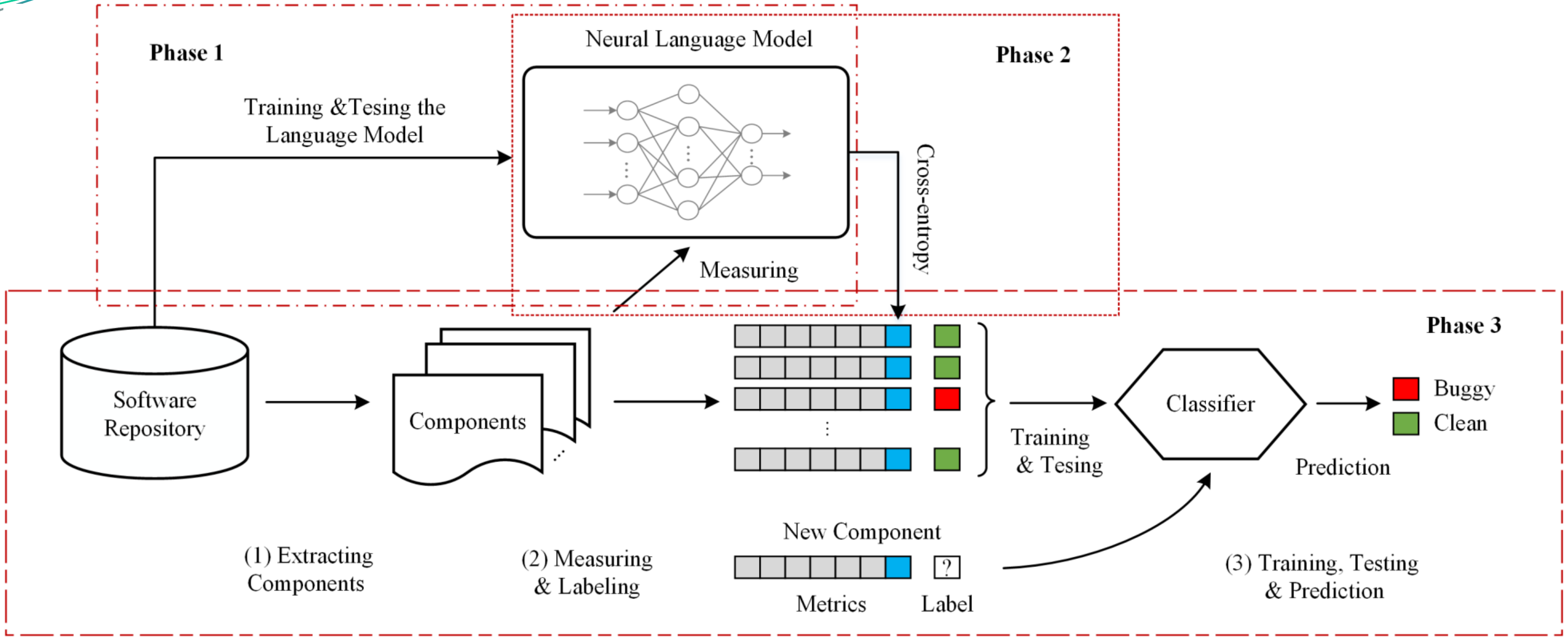


Fig. 3. The overall workflow of *DefectLearner*

The framework mainly contains three phases: **a learning phase** (Phase 1), **a measuring phase** (Phase 2), and **a prediction phase** (Phase 3).

Approach

B.Neural language model

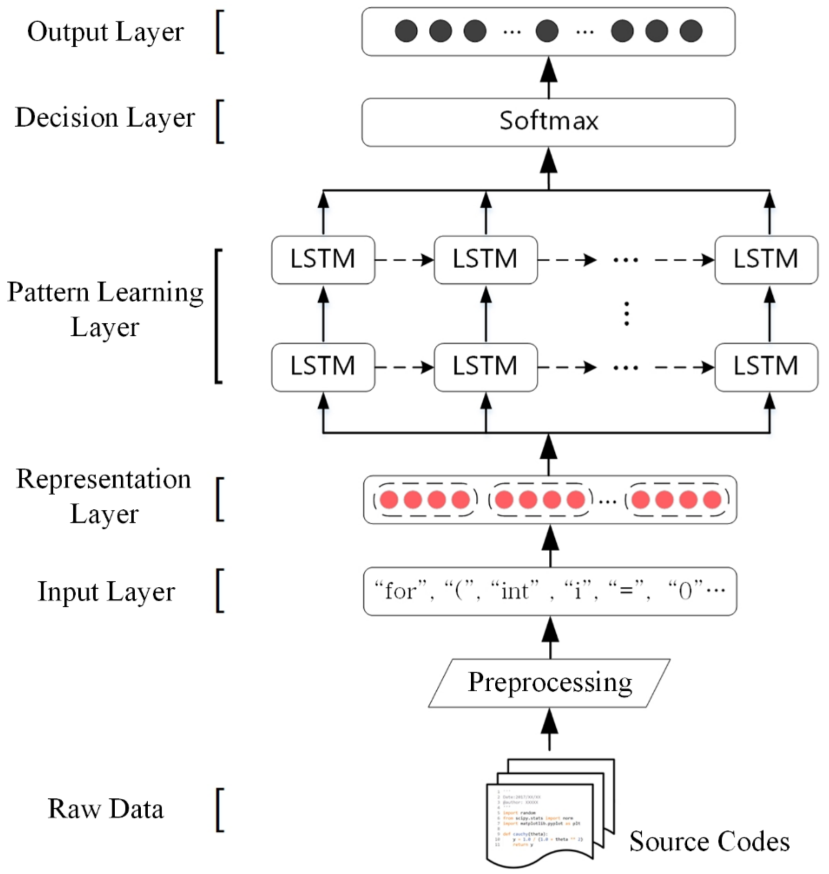


Fig. 4. The neural language model designed in *DefectLearner*

Approach

Fig. 4 presents our long short-term memory (LSTM) cell based RNN language model for cross-entropy calculation.

First, we need to train the model, whose goal is to learn common patterns or regularities in code token sequences by mining software repository.

After training, the language model will be utilized to measure the cross-entropy values of software components of interest (see (2)).

Approach

The core architecture of our network is a multi-layer **long short-term memory** (LSTM) cell based RNN. LSTM cell [24] is one of the most popular and efficient methods for understanding sequential dependencies [11], which uses memory gates to control its inputs and outputs. The LSTM cell we selected [35] are shown in Fig. 5.

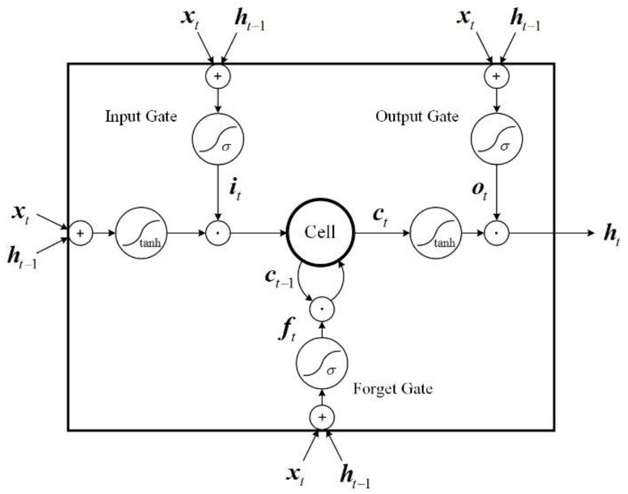


Fig. 5. LSTM cell

Approach

Deep network architectures often encounter many troubles during the training process, such as overfitting, gradient explosion, or gradient vanishing. Hence, this paper adopts several optimization strategies:

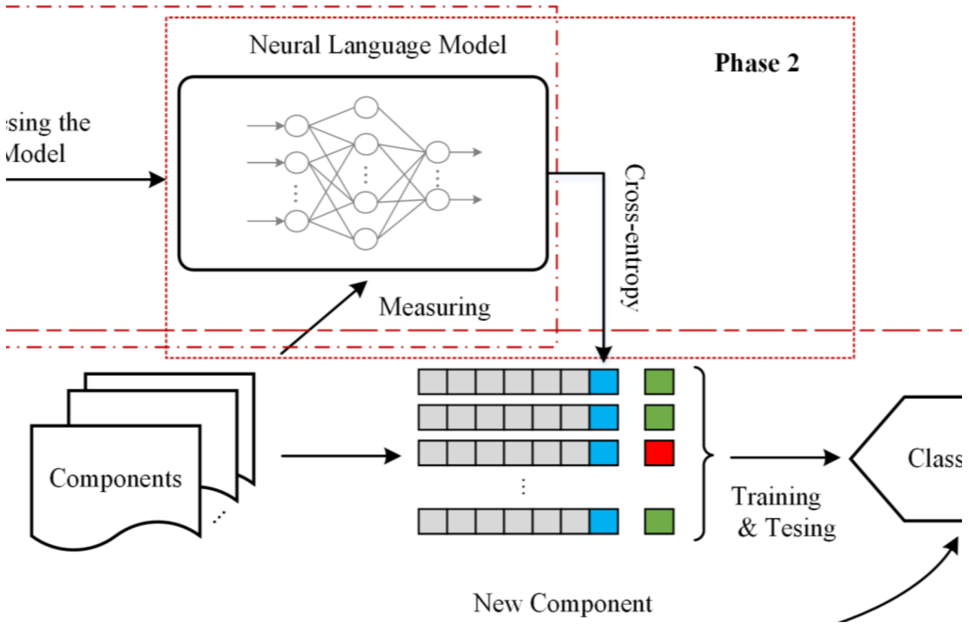
- Dropout [35]
- Gradient clipping [23]
- Adaptive learning rate.

For model testing, we choose 10-fold cross validation with perplexity measure (see (3)) as loss function.

Approach

C.Measurement of Software Component

After the learning phase (Phase 2), we use the trained language model to measure the naturalness of software components by (2), i.e., to generate the cross-entropy values for defect prediction.



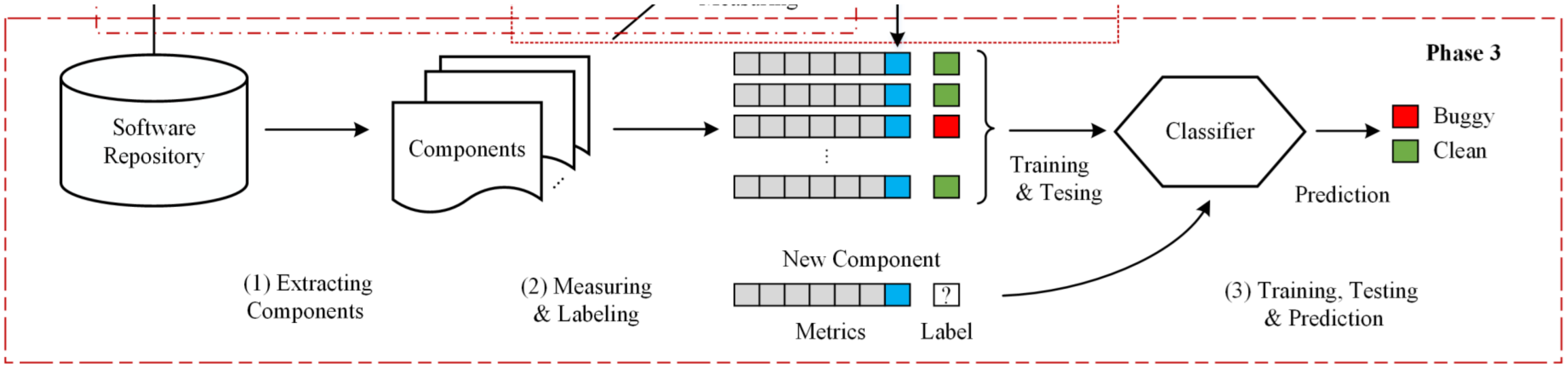
Approach

Specifically, we first match the evaluated components with their source code areas in the repository. Before feeding the code to language model, we should do the same preprocessing, in which each component will be cut into fixed-length token sequences. Then, we utilize the trained model to measure these inputs separately. As a result, the average cross-entropy of these sequences is namely the (estimated) naturalness score of the software component.

Approach

D.Performing Defect prediction

The Phase 3 in Fig. 3 depicts a typical process of defect-proneness prediction, which is commonly adopted in previous studies [1][12][14][27][37].



This paper selects four common classifiers for prediction [5], including support vector machine (SVM), logistic regression (LR), naive Bayes (NB), and random forest (RF).

Outline

Evaluation

In this section, we evaluate the effectiveness of our *DefectLearner* framework. The core task of the evaluation is to verify the effectiveness of the cross-entropy metric.

Our experiments are designed to address the following research questions (RQ):

- RQ1: *Is cross-entropy more class-discriminative than traditional code metrics in defect prediction?*
- RQ2: *Can cross-entropy improve the performance of prediction models?*
- RQ3: *How is the performance of cross-entropy on different datasets?*

Evaluation

A. Datasets

We select 12 Java open-source projects from the tera-PROMISE Repository[1]. These datasets have been widely used in previous studies [1][12][14][27].

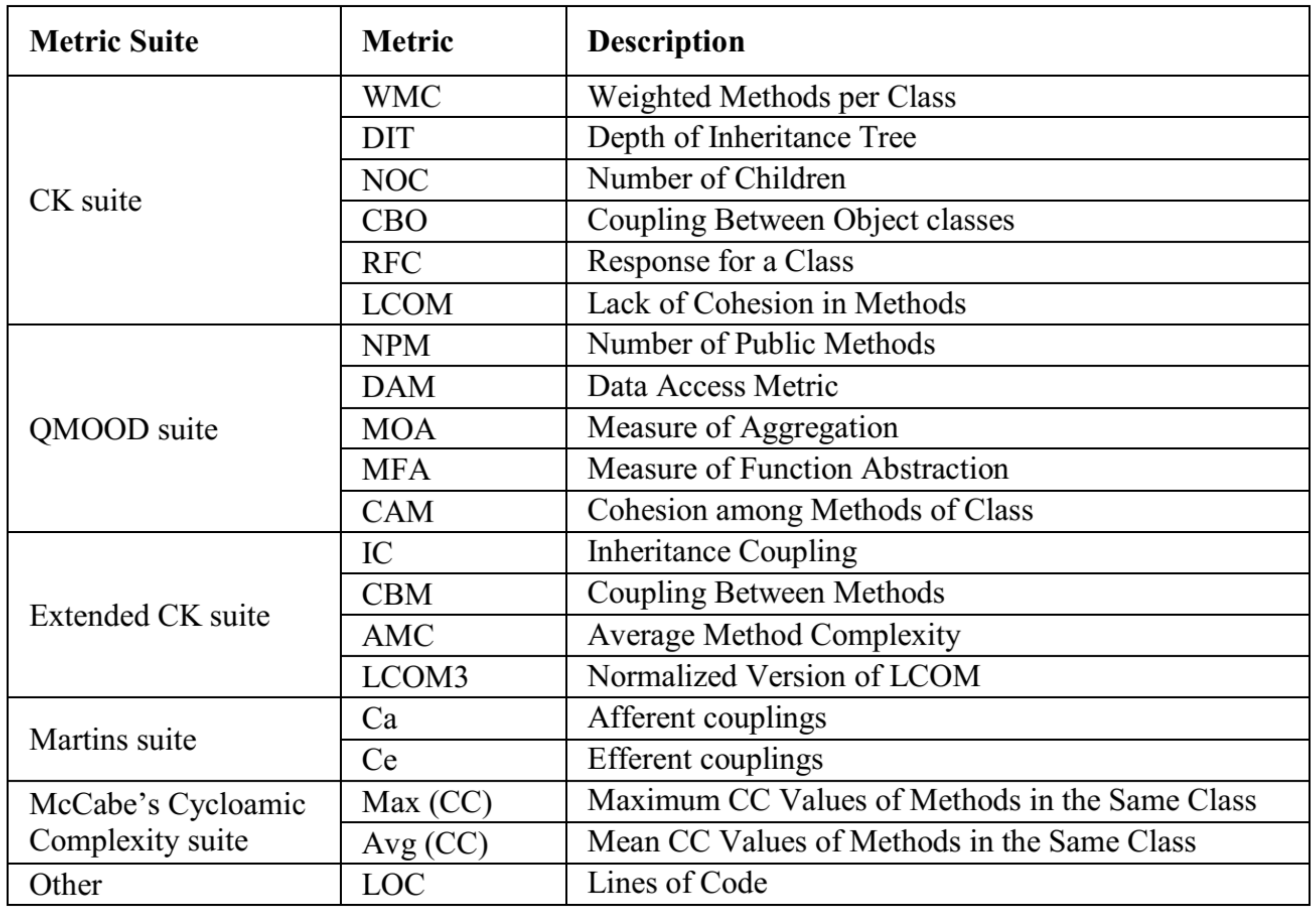
TABLE I. THE DATESETS USED IN OUR EXPERIMENTS

Project	**Version**	**Evaluated Components**	**Buggy Components**	**Buggy Rate**
Ant	1.7.0	745	166	22.28%
Camel	1.6.0	965	188	19.48%
Ivy	2.0.0	352	40	11.36%
jEdit	4.2	367	48	13.08%
Log4j	1.1	109	37	33.95%
Lucene	2.4.0	340	203	59.71%
PBeans	2.0	51	10	19.61%
POI	3.0	442	281	63.58%
Synapse	1.2	256	86	33.59%
Velocity	1.6.1	229	78	34.06%
Xalan-J	2.6.0	885	411	46.44%
Xerces	1.3.0	453	69	15.23%
Total	—	5,194	1,617	31.13%

[1] http://openscience.us/repo/defect/

Evaluation

TABLE II. THE 20 TRADITIONAL METRICS IN THE DATASETS USED AS THE BASELINE



Metric Suite	Metric	Description
CK suite	WMC	Weighted Methods per Class
	DIT	Depth of Inheritance Tree
	NOC	Number of Children
	CBO	Coupling Between Object classes
	RFC	Response for a Class
	LCOM	Lack of Cohesion in Methods
QMOOD suite	NPM	Number of Public Methods
	DAM	Data Access Metric
	MOA	Measure of Aggregation
	MFA	Measure of Function Abstraction
	CAM	Cohesion among Methods of Class
Extended CK suite	IC	Inheritance Coupling
	CBM	Coupling Between Methods
	AMC	Average Method Complexity
	LCOM3	Normalized Version of LCOM
Martins suite	Ca	Afferent couplings
	Ce	Efferent couplings
McCabe's Cycloamic Complexity suite	Max (CC)	Maximum CC Values of Methods in the Same Class
	Avg (CC)	Mean CC Values of Methods in the Same Class
Other	LOC	Lines of Code

Evaluation

B. Model Parameters

The parameter settings of *DefectLearner* are given as follows: the vocabulary size $V = 5{,}000$; the dimension of word vectors $dim_{\mathrm{w}} = 500$; the number of LSTM layers $N_{\mathrm{layer}} = 2$; the numbers of hidden nodes in each LSTM layer $\boldsymbol{dim}_{\mathrm{h}} = [800 \quad 1200]$; the keep probability of dropout $p_{\mathrm{keep}} = 0.55$; the length of input sequences $N_{\mathrm{step}} = 35$; the batch size $N_{\mathrm{batch}} = 30$; the number of epochs $N_{\mathrm{epoch}} = 15$; the maximum gradient norm $grad_{\mathrm{max}} = 5$; the parameters of learning rate $\lambda = 11$, $\eta = 0.5$.

Evaluation

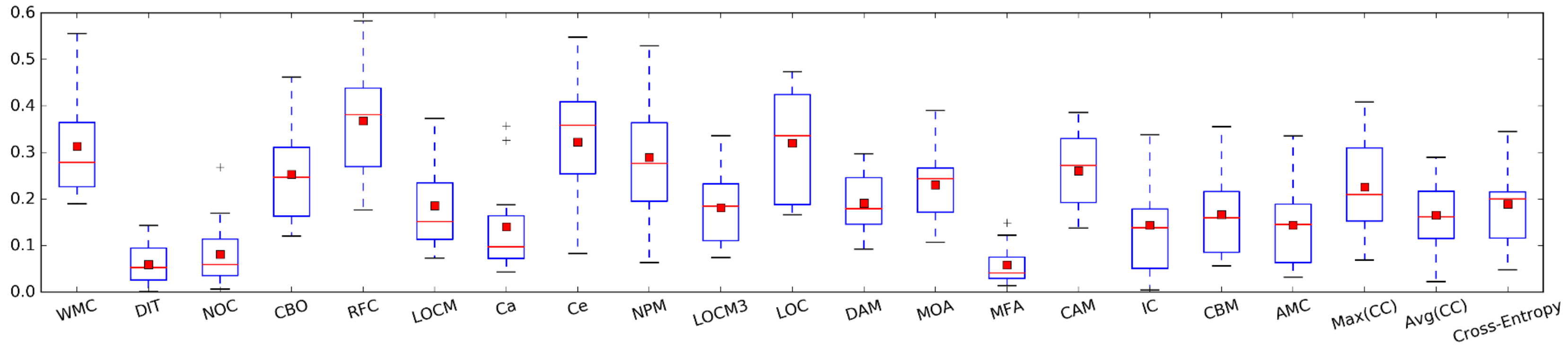
C. Results and Analysis

1) Discrimination of cross-entropy **(RQ1).**

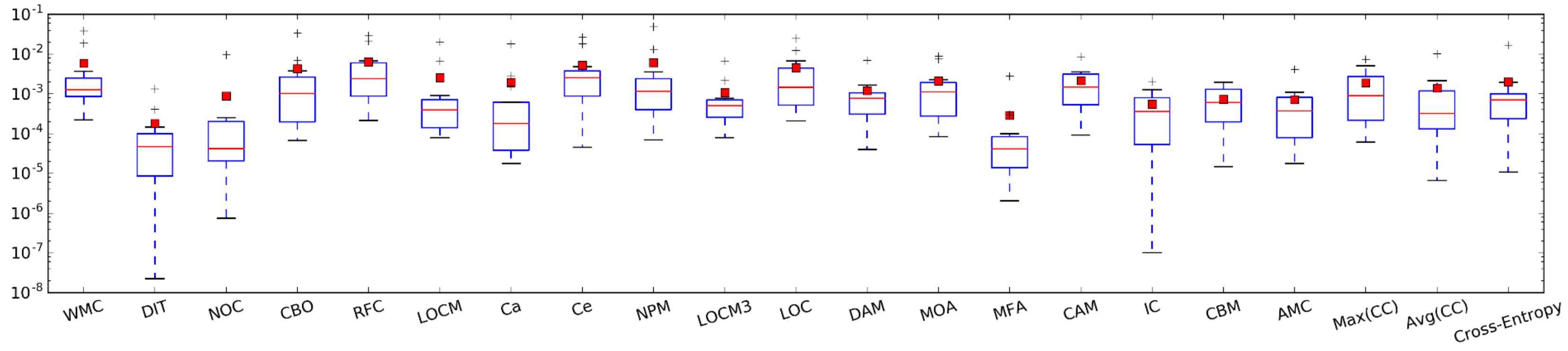
Generally, the feature more effective in classification tasks tends to own higher ability to distinguish samples. Hence, to assess the significance of cross-entropy, we first measure its discrimination for buggy proneness.

We select three common discrimination measures, viz., *Pearson's correlation*, *Fisher's criterion*, and *Information gain*.

Evaluation

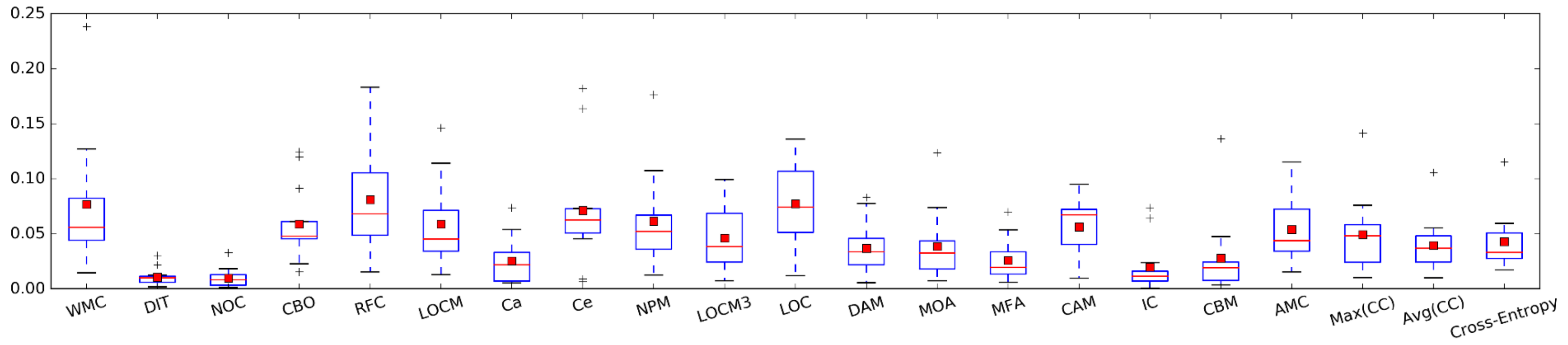


(a) Pearson's correlation



(b) Information gain

Evaluation



(c) Information gain

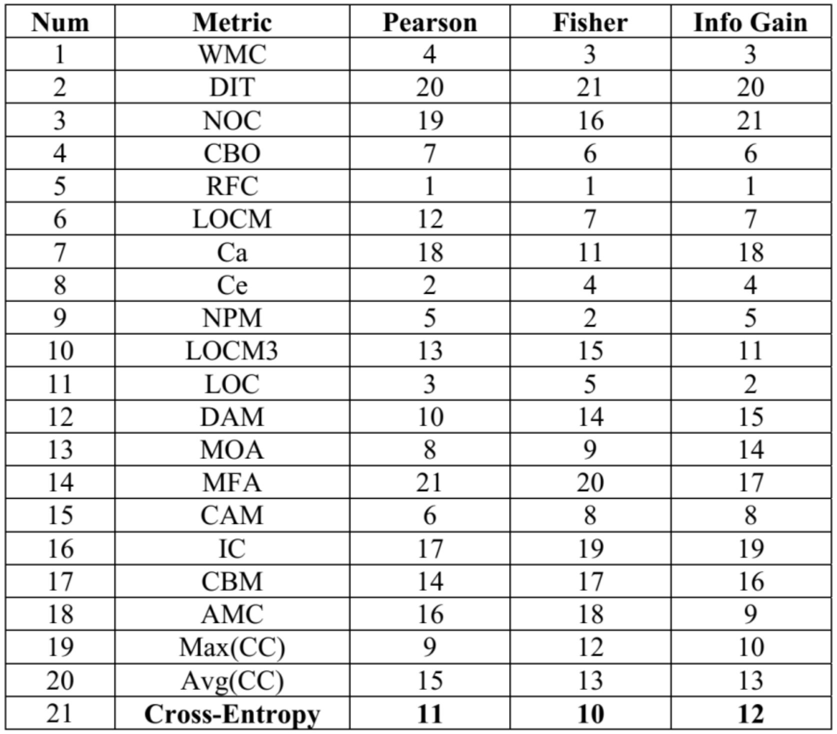
Fig. 6. Discrimination scores of the evaluated metrics

Compared to other metrics, cross-entropy is at the medium level in terms of both dispersion and mean, which indicates that cross-entropy is more relevant to defect proneness than half of the baseline metrics.

More explicitly, to give a comparison of discrimination, we **sort** the 21 metrics by **the mean values** of measures.

Evaluation

TABLE III. RANKS OF METRICS' CLASS-DISCRIMINATIVE PERFORMANCE



Num	Metric	Pearson	Fisher	Info Gain
1	WMC	4	3	3
2	DIT	20	21	20
3	NOC	19	16	21
4	CBO	7	6	6
5	RFC	1	1	1
6	LOCM	12	7	7
7	Ca	18	11	18
8	Ce	2	4	4
9	NPM	5	2	5
10	LOCM3	13	15	11
11	LOC	3	5	2
12	DAM	10	14	15
13	MOA	8	9	14
14	MFA	21	20	17
15	CAM	6	8	8
16	IC	17	19	19
17	CBM	14	17	16
18	AMC	16	18	9
19	Max(CC)	9	12	10
20	Avg(CC)	15	13	13
21	**Cross-Entropy**	**11**	**10**	**12**

Cross-entropy is more class-discriminative than 50% of the traditional code metrics and owns discrimination in defect-proneness prediction.

Evaluation

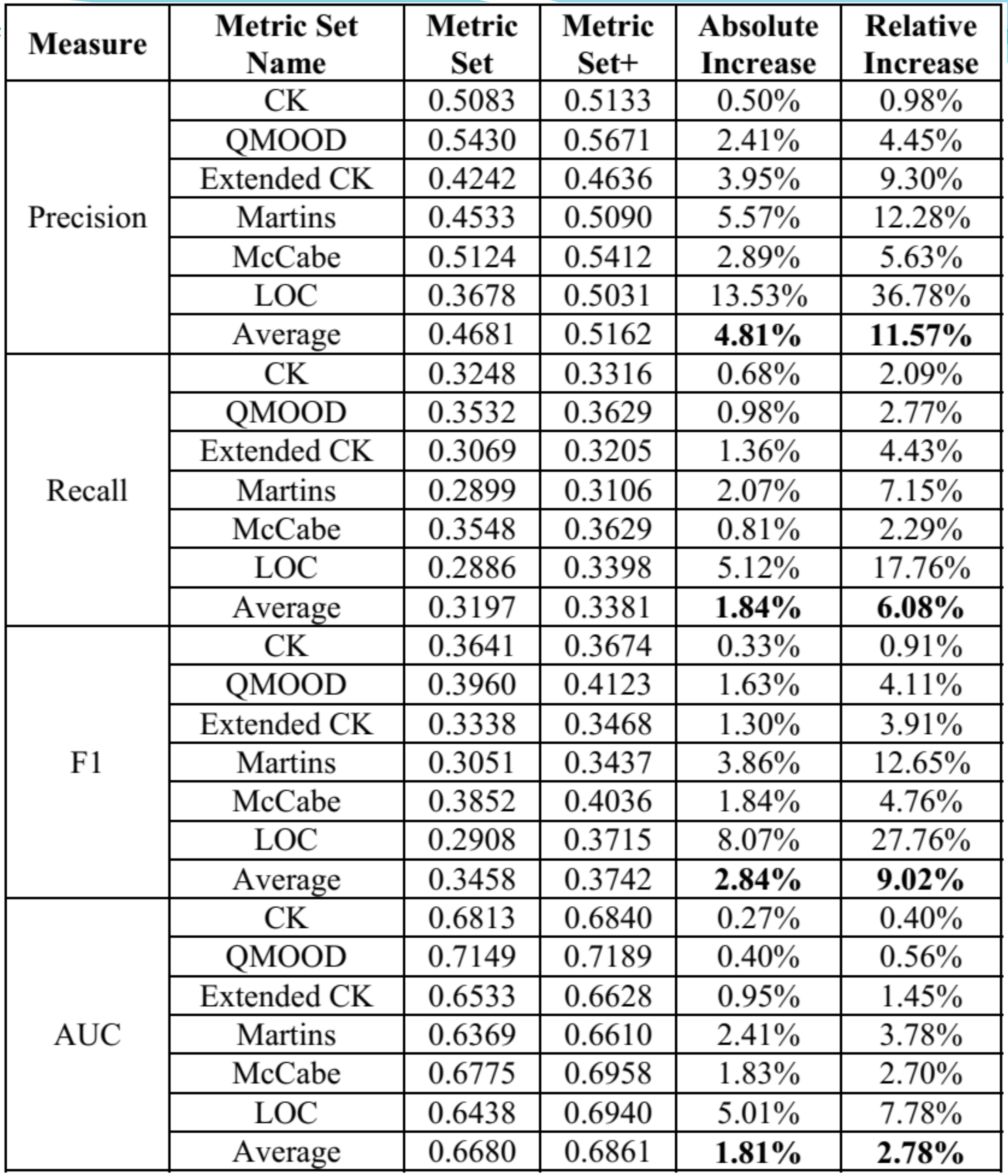
2) *Performance of prediction with cross-entropy* ***(RQ2)***

To evaluate a metric, we also want to find out how much contribution it can bring. Therefore, we design experiments to compare the performance improvement when cross-entropy is combined with different traditional metric suites.

As listed in TABLE II, there are six widely used metric suites, including CK suite, QMOOD suite, Extended CK suite, Martins suite, McCabe's CC suite, and LOC, which characterize different aspects of software. To give a comprehensive evaluation, we **combine cross-entropy with the six metric suites** together to feed prediction models respectively to compare the performance changes.

Evaluation

TABLE IV. The Performance Changes on Different Traditional Metric Sets with Cross-Entropy Added



Measure	Metric Set Name	Metric Set	Metric Set+	Absolute Increase	Relative Increase
Precision	CK	0.5083	0.5133	0.50%	0.98%
	QMOOD	0.5430	0.5671	2.41%	4.45%
	Extended CK	0.4242	0.4636	3.95%	9.30%
	Martins	0.4533	0.5090	5.57%	12.28%
	McCabe	0.5124	0.5412	2.89%	5.63%
	LOC	0.3678	0.5031	13.53%	36.78%
	Average	0.4681	0.5162	**4.81%**	**11.57%**
Recall	CK	0.3248	0.3316	0.68%	2.09%
	QMOOD	0.3532	0.3629	0.98%	2.77%
	Extended CK	0.3069	0.3205	1.36%	4.43%
	Martins	0.2899	0.3106	2.07%	7.15%
	McCabe	0.3548	0.3629	0.81%	2.29%
	LOC	0.2886	0.3398	5.12%	17.76%
	Average	0.3197	0.3381	**1.84%**	**6.08%**
F1	CK	0.3641	0.3674	0.33%	0.91%
	QMOOD	0.3960	0.4123	1.63%	4.11%
	Extended CK	0.3338	0.3468	1.30%	3.91%
	Martins	0.3051	0.3437	3.86%	12.65%
	McCabe	0.3852	0.4036	1.84%	4.76%
	LOC	0.2908	0.3715	8.07%	27.76%
	Average	0.3458	0.3742	**2.84%**	**9.02%**
AUC	CK	0.6813	0.6840	0.27%	0.40%
	QMOOD	0.7149	0.7189	0.40%	0.56%
	Extended CK	0.6533	0.6628	0.95%	1.45%
	Martins	0.6369	0.6610	2.41%	3.78%
	McCabe	0.6775	0.6958	1.83%	2.70%
	LOC	0.6438	0.6940	5.01%	7.78%
	Average	0.6680	0.6861	**1.81%**	**2.78%**

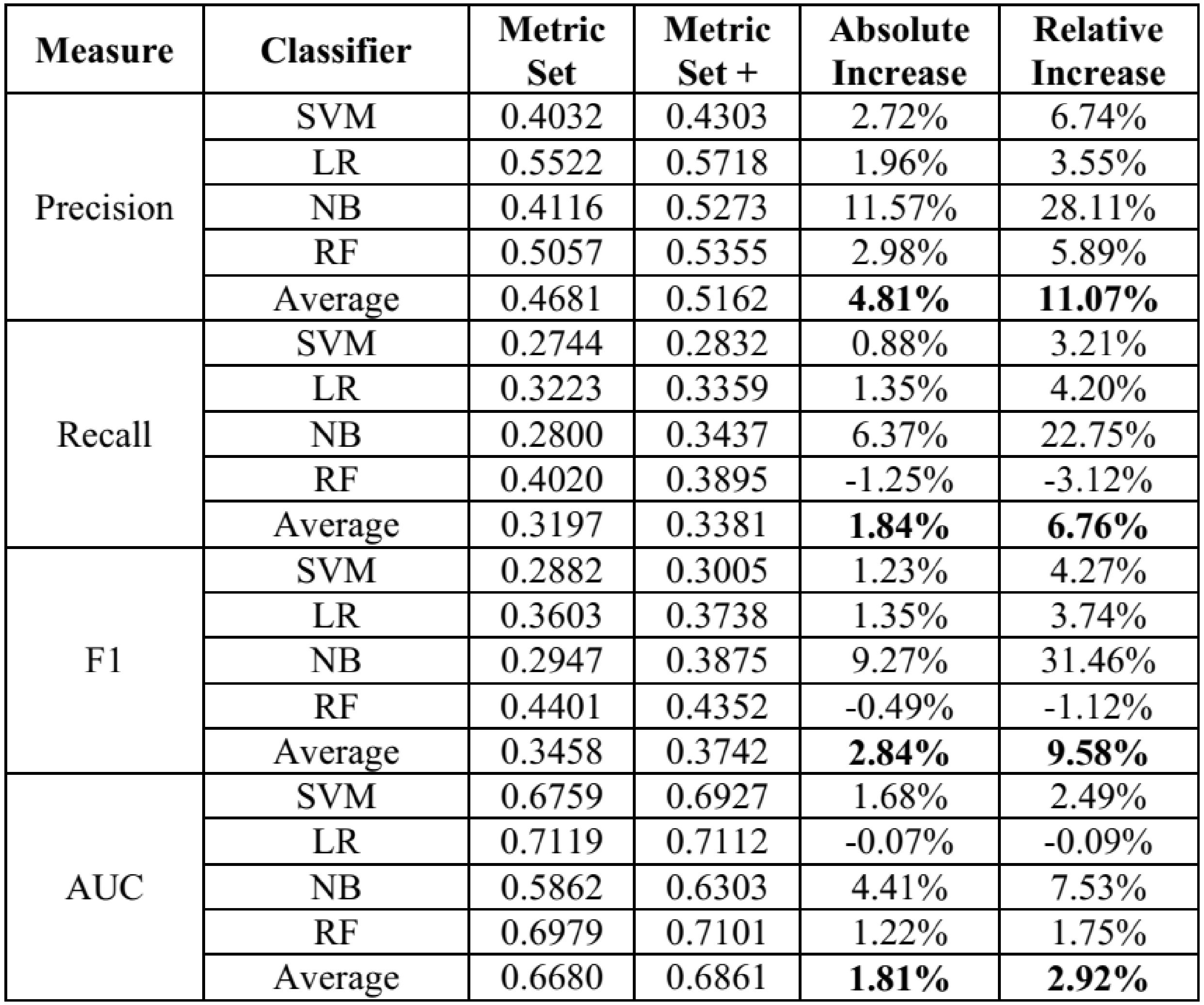
Evaluation

TABLE IV presents the performance changes with cross-entropy added, where the results are the mean values of four classifiers on 12 projects, and the “Metric Set+” denotes the new set with the integration of cross-entropy.

From this table, we can find that on average, cross-entropy bring absolute improvement of 4.81%, 1.84%, 2.84%, and 1.81% respectively in Precision, Recall, F1, and AUC.

Evaluation

TABLE V. THE PERFORMANCE CHANGES OF DIFFERENT CLASSIFIERS WITH CROSS-ENTROPY ADDED



Measure	Classifier	Metric Set	Metric Set +	Absolute Increase	Relative Increase
Precision	SVM	0.4032	0.4303	2.72%	6.74%
	LR	0.5522	0.5718	1.96%	3.55%
	NB	0.4116	0.5273	11.57%	28.11%
	RF	0.5057	0.5355	2.98%	5.89%
	Average	0.4681	0.5162	**4.81%**	**11.07%**
Recall	SVM	0.2744	0.2832	0.88%	3.21%
	LR	0.3223	0.3359	1.35%	4.20%
	NB	0.2800	0.3437	6.37%	22.75%
	RF	0.4020	0.3895	-1.25%	-3.12%
	Average	0.3197	0.3381	**1.84%**	**6.76%**
F1	SVM	0.2882	0.3005	1.23%	4.27%
	LR	0.3603	0.3738	1.35%	3.74%
	NB	0.2947	0.3875	9.27%	31.46%
	RF	0.4401	0.4352	-0.49%	-1.12%
	Average	0.3458	0.3742	**2.84%**	**9.58%**
AUC	SVM	0.6759	0.6927	1.68%	2.49%
	LR	0.7119	0.7112	-0.07%	-0.09%
	NB	0.5862	0.6303	4.41%	7.53%
	RF	0.6979	0.7101	1.22%	1.75%
	Average	0.6680	0.6861	**1.81%**	**2.92%**

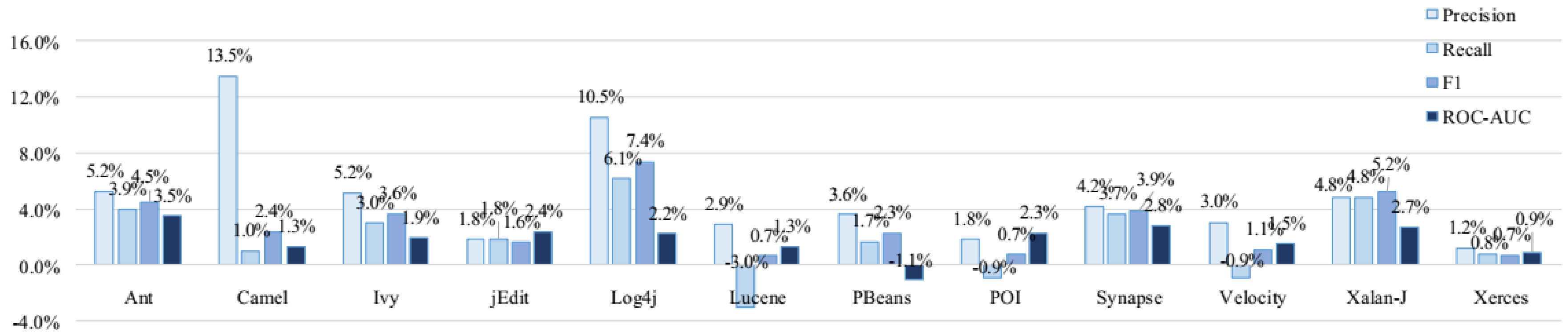
Evaluation

From this table, we can find that four classifiers acquire different performance changes. Roughly, cross-entropy contributes most to NB and least to RF. Even though, cross-entropy does bring improvement in most cases, with an average of absolute increases being 4.81%, 1.84%, 2.84%, and 1.81% respectively in Precision, Recall, F1, and AUC measure.

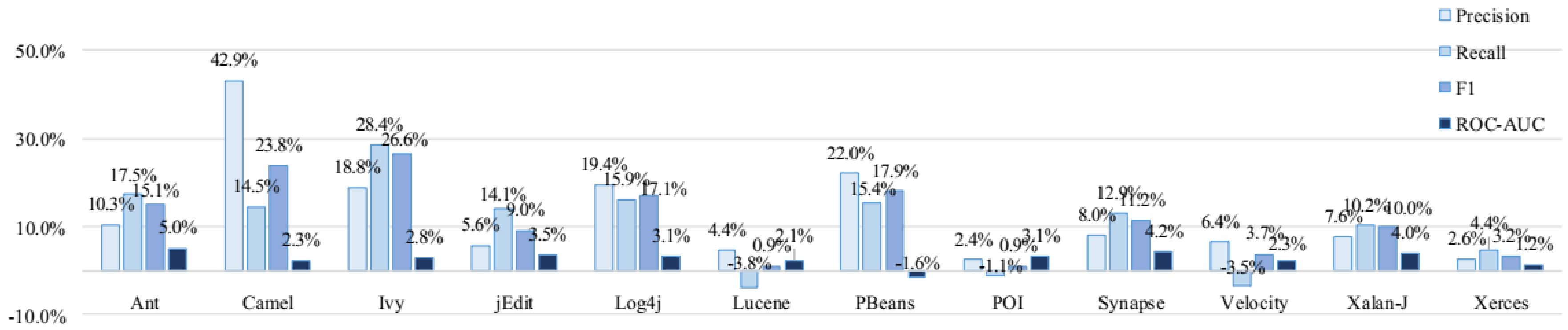
Cross-entropy is complementary to traditional metric sets in defect-proneness prediction.

Evaluation

3) Performance on different projects (RQ3).



(a) The absolute increases



(b) The relative increases

Fig. 7. The performance improvement after cross-entropy metric is added

Evaluation

Overall, as we can see from Fig. 7, cross-entropy plays a positive role on the 12 investigated projects, no matter in terms of relative comparison or absolute comparison.

Cross-entropy can improve the prediction performance on different defect datasets.

Outline

- Introduction
- Background
- Approach
- Evaluation
- **Conclusion**

Conclusion

The cross-entropy of a sequence measures its likelihood of appearance in the corpus. Common sequences usually own higher probability of occurrence, that is, lower cross-entropy. In recent years, a series of work have used cross-entropy for code measurement and found that defective codes tend to be more entropic.

Based on this finding, this paper introduces cross-entropy as a new metric for defect prediction. To make use of the advantages of deep architecture in pattern learning, we construct a RNN based framework *DefectLearner* to automatically calculate the cross-entropy value of each software component and perform defect-proneness recognition.

Conclusion

Then, we present substantial empirical evidence of the effectiveness of this new metric. As the experimental results show, the cross-entropy metric owns more discrimination power for defect classification than half of 20 common metrics, and does bring improvement to prediction models on different projects. Besides, cross-entropy is a learnable feature that exploits the implicit knowledge in software repositories, and also of good interpretability.

In conclusion, these findings suggest that **cross-entropy is useful to predict software defects and complementary to existing metric suits.**

Conclusion

Recent & Future Work

(1) Following this work, we have explored an extended study [39] recently, in which a new code metric *CE-AST*, i.e., the cross-entropy value of the sequence of AST nodes, is proposed for better defect prediction.

[39] X. Zhang, et al. "Using Cross-entropy value of code for better defect prediction," in ISSSR'18: the International Symposium on System and Software Reliability, 2018, accepted. https://github.com/TOM-ZXian/

(2) The language model can be regarded as a learner. It should pay more attention to the codes with high quality. Therefore, we try to feed the model the quality information of codes, so that different samples will have different weights. After training, the cross entropy measured by the weighted language model can get better performance.

Reference

[1] A. Agrawal and T. Menzies, "Is 'better data' better than 'better data miners'?: On the benefits of tuning SMOTE for defect prediction," in *ICSE'18*: *40th International Conference on Software Engineering*, 2018. https://doi.org/10.1145/3180155.3180197.
[2] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the naturalness of software," in *ICSE'12: Proc. of the International Conference on Software Engineering*, pp. 837-847, 2012.
[3] B. Ghotra, S. McIntosh, and A. E. Hassan, "A large-scale study of the impact of feature selection techniques on defect classification models," in *MSR'17: IEEE/ACM 14th International Conference on Mining Software Repositories*, pp. 146-157, 2017.
[4] B. Ray, V. Hellendoorn, S. Godhane, Z. Tu, A. Bacchelli, and P. Devanbu, "On the naturalness of buggy code," in *ICSE'16: Proc. of the International Conference on Software Engineering*, pp. 428-439, 2016.
[5] C. M. Bishop, Pattern recognition and machine learning, New York: Springer, 2006.
[6] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction model," in *ICSE'16: Proc. of the International Conference on Software Engineering*, pp. 321-332, 2016.
[7] D. Jurafsky and J. H. Martin, Speech and language processing, 2nd ed., Upper Saddle River: Pearson/Prentice Hall, 2009.
[8] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and Software Technology*, vol. 55, no.8, pp. 1397-1418, 2013.
[9] D. Feng, J. Wang, Q. Li, and S. Zhang, "Defect prediction in Android binary executables using deep neural network," *Wireless Personal Communications*, 2017. https://doi.org/10.1007/s11277-017-5069-3.
[10] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16-28, 2014.
[11] H. Salehinejad, J. Baarbe, S. Sankar, J. Barfett, E. Colak, and S. Valaee, "Recent advances in recurrent neural networks," *arXiv preprint*, *arXiv*: 1801.01078, 2018.
[12] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Information and Software Technology*, vol. 58, pp. 388-402, 2015.
[13] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on software engineering*, vol. 28, no. 1, pp. 4-17, 2002.

[14] J. Li, P. He, J. Zhu, and R. L. Michael, "Software defect prediction via convolutional neural network," in *QRS'17: Proc. of the International Conference on Software Quality, Reliability and Security*, pp. 318-328, 2017.
[15] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering* , 2017. DOI: 10.1109/TSE.2017.2720603.
[16] L. Madeyski and M. Jureczko, "Which process metrics can significantly improve defect prediction models? An empirical study," *Software Quality Journal*, vol.23, no.3, pp. 393-422, 2015.
[17] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *arXiv preprint, arXiv*: 1709.06182, 2017.
[18] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, "DeepCoder: Learning to write programs," in *ICLR'17: Proc. of the International Conference on Learning Representations*, 2017.
[19] M. H. Halstead, "Elements of Software Science (Operating and Programming Systems Series)," New York: Elsevier Science Inc., 1977.
[20] M. Jureczko, and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proc. of the International Conference on Predictive Models in Software Engineering*, 2010.
[21] R. Gupta, S. Pal, A. Kanade, and S. Shevade, "DeepFix: fixing common C language errors by deep learning," in *AAAI'17: Proc. of the AAAI Conference on Artificial Intelligence*, pp. 1345-1351, 2017.
[22] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504-518, 2015.
[23] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *ICML'13*: *International Conference on Machine Learning*. pp. 1310-1318, 2013.
[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[25] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476-493, 1994.
[26] S. Wang, D. Chollak, D. Movshovitz-Attias, and L. Tian, "Bugram: Bug detection with n-gram language models," in *ASE'16: Proc. of the IEEE/ACM International Conference on Automated Software Engineering*, pp. 708-719, 2016.
[27] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *ICSE'16: Proc. of the International Conference on Software Engineering*, pp. 297-308, 2016.
[28] T. Fawcett, "An introduction to ROC analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861-874, 2006.
[29] T. Hall, S. Beecham, D. Bowes, D. Grayc, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering,". *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276-1304, 2012.

[30] T. Jiang, L. Tan, and S. Kim, "Personalized defect prediction," in *ASE'13: Proc. of the IEEE/ACM International Conference on Automated Software Engineering*, pp. 279-289, 2013.
[31] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308-320, 1976.
[32] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S Khudanpur, "Recurrent neural network based language model," in *INTERSPEECH'10*: *Annual Conference of the International Speech Communication Association*, pp. 1045-1048, 2010.
[33] T. Mikolov, W. T. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 746-751, 2013.
[34] V. Murali, S. Chaudhuri, and C. Jermaine. "Bayesian specification learning for finding API usage errors," in *FSE'17: Proc. of the Joint Meeting on Foundations of Software Engineering*, pp. 151-162, 2017.
[35] W. Zaremba, I. Sutskever, and O. Vinyals. "Recurrent neural network regularization," *arXiv preprint, arXiv*: 1409.2329, 2014.
[36] X. Y. Jing, F. Wu, X. Dong, and B. Xu, "An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems," *IEEE Transactions on Software Engineering*, vol. 43, no. 4, pp. 321-339, 2017.
[37] X. Y. Jing, S. Ying, Z. Zhang, S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *ICSE'14: Proc. of the International Conference on Software Engineering*, 2016.
[38] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, "Deep learning for just-in-time defect prediction," in *QRS'15: Proc. of the International Conference on Software Quality, Reliability and Security*, pp. 17-26, 2015.
[39] X. Zhang, K. Ben, and J. Zeng, "Using Cross-entropy value of code for better defect prediction," in *ISSSR'18: the International Symposium on System and Software Reliability*, 2018, accepted. https://github.com/TOM-ZXian/-Cross-entropy-metric-of-code-for-defect-prediction
[40] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *NIPS'01: Proc. of Advances in Neural Information Processing Systems*, pp. 932-938, 2001.
[41] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 512, no. 7553, pp. 436-444, 2015.
[42] Z. Li, X. Y. Jing, X. Zhu, H. Zhang, B. Xu, and S. Ying, "On the multiple sources and privacy preservation issues for heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, DOI: 10.1109/TSE.2017.2780222, 2017.
[43] Z. Tu, Z. Su, and P. Devanbu, "On the localness of software," in *FSE'14: Proc. of the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 269-280, 2014.
[44] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The impact of feature selection on defect prediction performance: An empirical comparison," in *ISSRE'16: IEEE 27th International Symposium on Software Reliability Engineering*, pp. 309-320, 2016.

Thanks