三篇论文

SCIENCE AND TECHNOLOGY

《Learning and Evaluating Contextual Embedding of Source Code》	2020	ICML
《Cross-Project Defect Prediction Using a Connectivity-Based Unsupervised Classifier》	2016	ICSE
《An empirical comparison of model validation techniques for defect prediction models》	2016	TSE

GRADUATION DEFENSE

Learning and Evaluating Contextual Embedding of Source Code

2020 ICML

Bert VS CuBert

Bert

one or two sentences, where a sentence is the concatenation of contiguous lines from the source corpus

CuBert

CuBERT is similarly formulated, but a CuBERT line is a logical code line, as defined by the Python standard

Learning and Evaluating Contextual Embedding of Source Code

2020 ICML

5 downsteam claasification tasks

- *Variable-misuse Classification*
- *Wrong Binary Operator*
- *Swapped Operand*
- *Function Docstring Mismatch*
- *Exception Type*

Learning and Evaluating Contextual Embedding of Source Code

2020 ICML

- Do contextual embeddings help with source-code analysis tasks, when pre-trained on an unlabeled code corpus?
- How does the performance of CuBERT on the classification tasks scale with the amount of labeled training data?
- How does context size affect CuBERT?

Learning and Evaluating Contextual Embedding of Source Code

2020 ICML

Setting			Misuse	Operator	Operand	Docstring	Exception
BiLSTM (100 epochs)	From scratch		76.29 %	83.65 %	88.07 %	76.01 %	52.79 %
	CBOW	ns	80.33 %	**86.82 %**	89.80 %	**89.08 %**	**67.01 %**
		hs	78.00 %	85.85 %	**90.14 %**	87.69 %	60.31 %
	Skipgram	ns	77.06 %	85.14 %	89.31 %	83.81 %	60.07 %
		hs	**80.53 %**	86.34 %	89.75 %	88.80 %	65.06 %
CuBERT	2 epochs		94.04 %	89.90 %	92.20 %	97.21 %	61.04 %
	10 epochs		95.14 %	92.15 %	93.62 %	98.08 %	77.97 %
	20 epochs		95.21 %	92.46 %	93.36 %	98.09 %	79.12 %
Transformer	100 epochs		78.28 %	76.55 %	87.83 %	91.02 %	49.56 %

Table 2. Test accuracies of fine-tuned CuBERT against BiLSTM (with and without Word2Vec embeddings) and Transformer trained from scratch on the classification tasks. “ns” and “hs” respectively refer to negative sampling and hierarchical softmax settings used for training CBOW and Skipgram models. “From scratch” refers to training with freshly initialized token embeddings, without pre-training.

Learning and Evaluating Contextual Embedding of Source Code

2020 ICML

Best of # Epochs	Train Fraction	Misuse	Operator	Operand	Docstring	Exception
2	100 %	94.04 %	89.90 %	92.20 %	97.21 %	61.04 %
	66 %	93.11 %	88.76 %	91.61 %	97.04 %	19.49 %
	33 %	91.40 %	86.42 %	90.52 %	96.38 %	20.09 %
10	100 %	95.14 %	92.15 %	93.62 %	98.08 %	77.97 %
	66 %	94.78 %	91.51 %	93.37 %	97.93 %	75.24 %
	33 %	94.28 %	90.66 %	92.58 %	97.36 %	67.34 %
20	100 %	95.21 %	92.46 %	93.36 %	98.09 %	79.12 %
	66 %	94.90 %	91.79 %	93.39 %	97.99 %	77.31 %
	33 %	94.45 %	91.09 %	92.82 %	97.63 %	74.98 %

Table 3. Effects of reducing training-split size on fine-tuning performance on the classification tasks.

Length	Misuse	Operator	Operand	Docstring	Exception	Misuse on BiLSTM
128	83.97 %	79.29 %	78.02 %	98.19 %	62.03 %	74.32 %
256	92.02 %	88.19 %	88.03 %	98.14 %	72.80 %	78.47 %
512	95.21 %	92.46 %	93.36 %	98.09 %	79.12 %	80.33 %
1024	95.83 %	93.38 %	95.62 %	97.90 %	81.27 %	81.92 %

Table 4. Best out of 20 epochs of fine-tuning, for four example lengths, on the classification tasks. For contrast, we also include results for Variable Misuse using the BiLSTM Word2Vec (CBOW + ns) classifier as length varies.

Cross-Project Defect Prediction Using a Connectivity-Based Unsupervised Classifier

2016 ICSE

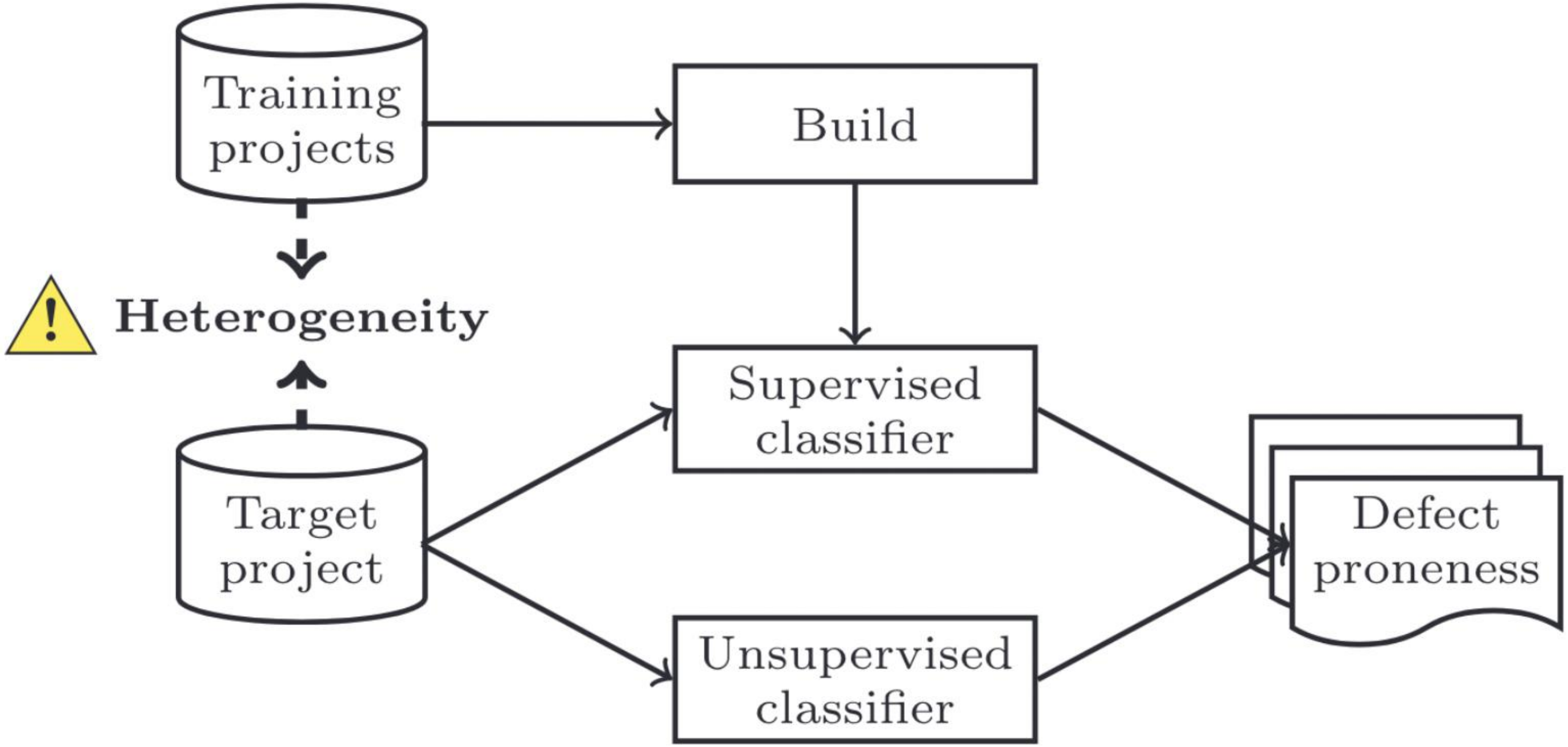

Figure 1: Illustration of the heterogeneity challenge.

Cross-Project Defect Prediction Using a Connectivity-Based Unsupervised Classifier

2016 ICSE

Unsupervised Defect Prediction

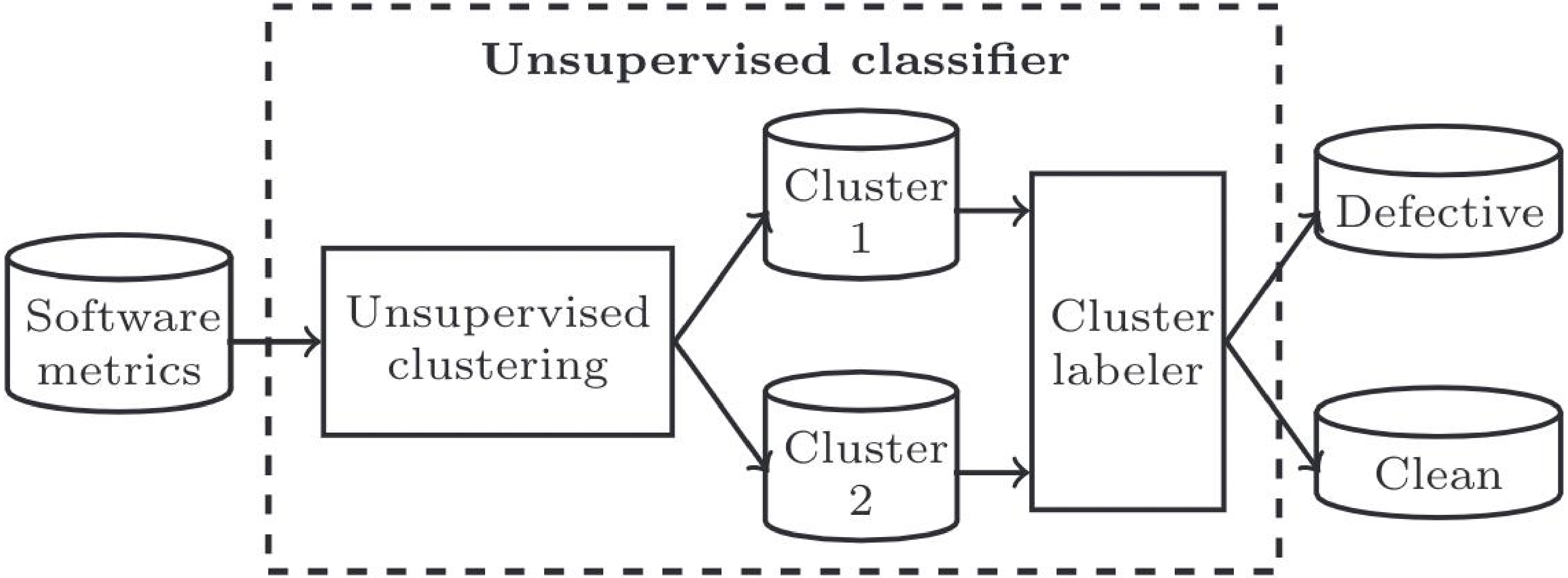

Figure 2: A typical process to do defect prediction using an unsupervised classifier.

Cross-Project Defect Prediction Using a Connectivity-Based Unsupervised Classifier

2016 ICSE

Spectral Clustering

Distance-based：K-means Clustering

Algorithm 1: Spectral clustering based classifier for defect prediction

Input: A matrix with rows as software entities and columns as metrics.

Output: A vector of defect proneness of all software entities.

1: Normalize software metrics using z-score.
2: Construct a weighted adjacency matrix W.
3: Calculate the Laplacian matrix L_{sym}.
4: Perform the eigendecomposition on L_{sym}.
5: Select the second smallest eigenvector $\mathbf{v_1}$.
6: Perform the bipartition on v_1 using zero.
7: Label each cluster as defective or clean.

Cross-Project Defect Prediction Using a Connectivity-Based Unsupervised Classifier

2016 ICSE

RF、NB、LR、DT、LMT

k-means clustering (KM)
partition around medoids (PAM)
fuzzy Cmeans (FCM)
neural-gas (NG)

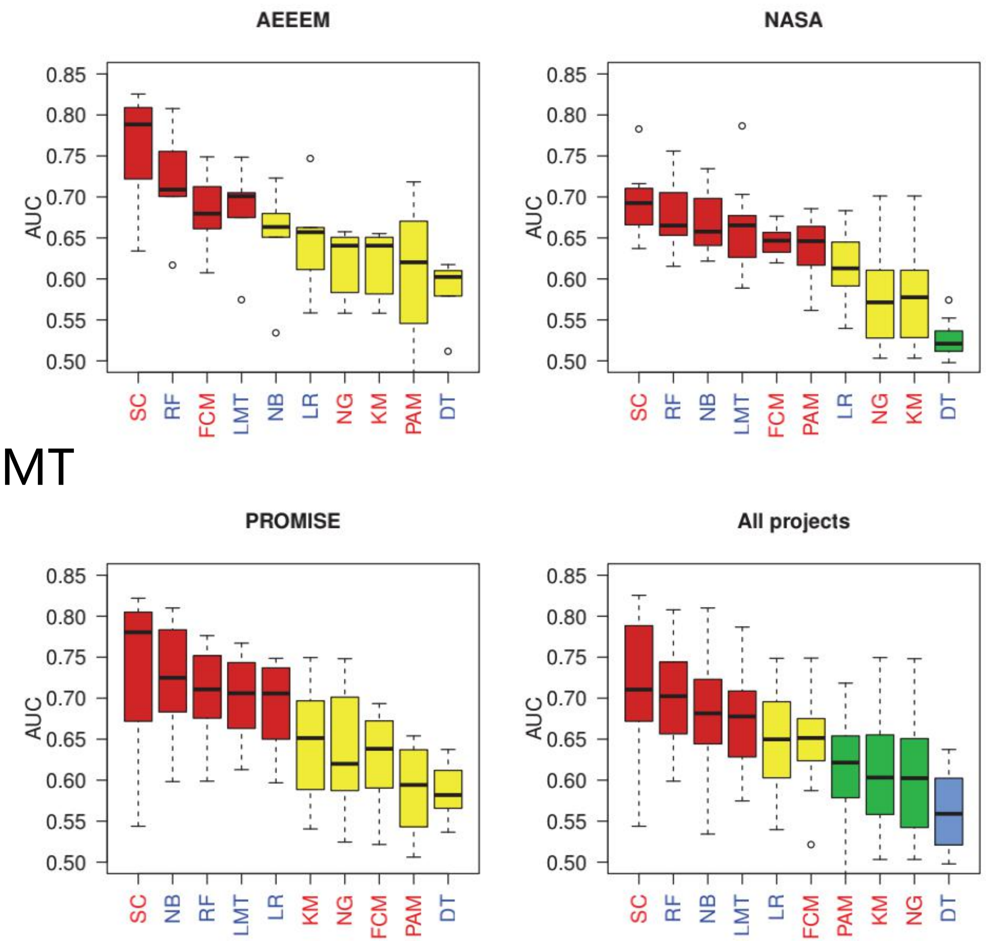

Figure 3: The boxplots of AUC values of all studied supervised (blue labels) and unsupervised (red labels) classifiers (for the abbreviations, see Section 4.3). Different colors represents different ranks (red > yellow > green > blue).

Table 2: The AUC values of the top four classifiers in cross-project defect prediction (Bold font highlights the best performance).

Dataset	Project	SC	RF	NB	LMT
AEEEM	Eclipse JDT Core	**0.83**	0.81	0.68	0.75
	Equinox	**0.81**	0.70	0.66	0.71
	Apache Lucene	**0.79**	0.76	0.72	0.70
	Mylyn	**0.63**	0.62	0.53	0.57
	Eclipse PDE UI	**0.72**	0.71	0.65	0.67
NASA	CM1	**0.67**	0.66	0.66	0.62
	JM1	**0.66**	0.62	0.64	0.60
	KC3	0.64	**0.65**	0.62	0.63
	MC1	0.69	**0.71**	0.66	0.67
	MC2	**0.68**	0.62	0.64	0.59
	MW1	**0.70**	0.67	**0.70**	0.67
	PC1	0.71	**0.73**	0.70	0.70
	PC2	0.78	0.76	0.73	**0.79**
	PC3	**0.72**	0.70	0.70	0.68
	PC4	0.65	**0.67**	0.63	**0.67**
	PC5	**0.71**	0.66	0.66	0.63
PROMISE	Ant v1.7	**0.79**	0.75	0.77	0.75
	Camel v1.6	**0.62**	0.60	0.60	0.61
	Ivy v1.4	0.70	**0.71**	0.68	0.70
	Jedit v4.0	**0.79**	0.74	0.75	0.73
	Log4j v1.0	**0.82**	0.76	0.81	0.74
	Lucene v2.4	0.67	0.68	**0.69**	0.66
	POI v3.0	**0.82**	0.71	0.78	0.69
	Tomcat v6.0	**0.80**	0.78	**0.80**	0.77
	Xalan v2.6	0.54	**0.66**	0.60	0.62
	Xerces v1.3	**0.77**	0.69	0.70	0.71
Median		**0.71**	0.70	0.68	0.68

An empirical comparison of model validation techniques for defect prediction models

2016 TSE

2 Concepts

Bias

How much do the performance estimates differ from the model performance on unseen data?

Variance

How much do performance estimates vary when the experiment is repeated on the same data?

An empirical comparison of model validation techniques for defect prediction models

2016 TSE

TABLE 1: Summary of model validation techniques.

Family	Technique	Training sample	Testing sample	Estimated performance	Iteration(s)
Holdout	Holdout 0.5	50% of original	Independent: 50% of original	A single estimate	1
	Holdout 0.7	70% of original	Independent: 30% of original	A single estimate	1
	Repeated Holdout 0.5	50% of original	Independent: 50% of original	Average of performance of several samples	100
	Repeated Holdout 0.7	70% of original	Independent: 30% of original	Average of performance of several samples	100
Cross-validation	Leave-one-out	N-1 of original	Independent: Subject that is not included in the training sample	Average of performance of several samples	N
	2-Fold	50% of original	Independent: 50% of original	Average of performance of several samples	2
	10-Fold	90% of original	Independent: 10% of original	Average of performance of several samples	10
	10 $\times$ 10-Fold	90% of original	Independent: 10% of original	Average of performance of several samples	100
Bootstrapping	Ordinary	Bootstrap	Original	Average of performance of several samples	100
	Optimism-reduced	Bootstrap	Original	Apparent† - optimism	100
	Out-of-sample	Bootstrap	Independent: the training subjects that are not sampled in bootstrap	Average of performance of several samples	100
	.632 Bootstrap	Bootstrap	Independent: the training subjects that are not sampled in bootstrap	0.368 $\times$ Apparent† + 0.632 $\times$ average(out-of-sample)	100

†*Apparent performance* is computed from a model that is trained and tested on the original sample.

An empirical comparison of model validation techniques for defect prediction models

2016 TSE

- RQ1: Which model validation techniques are the least biased for defect prediction models?

the out-of-sample bootstrap tends to provide the least biased performance

- RQ2: Which model validation techniques are the most stable for defect prediction models?

the ordinary bootstrap is the most stable model validation technique