5 Papers

Title	Publication source	Year
A Deep Multimodal Model for Bug Localization	DMKD	2021
Enhancing Supervised Bug Localization with Metadata and Stack-Trace	KIS	2020
Learning Unified Features from Natural and Programming Languages for Locating Buggy Source Code	IJCAI	2016
Locating Faulty Methods with a Mixed RNN and Attention Model	ICPC	2021
Multi-Dimension Convolutional Neural Network for Bug Localization	TSC	2020

Multi-Dimension Convolutional Neural Network for Bug Localization

汇报人：王昭丹

CONTENTS

BACKGROUND

Static Bug Localization

Mismatch of text similarity

- Structured Information Retrieval
- Semantic Information
- Bug-fixing History

1

Dynamic Bug Localization

Gather information from execution traces of the system

- Spectrum-based
- Model-based

2

Possibility of combining multiple features

How to effectively **combine multiple dimensions of features** for bug localization ?

BACKGROUND

IR-based Bug Localization

- Vector Space Model (VSM)
- Latent Semantic Indexing (LSI)
- Latent Dirichlet Allocation (LDA)
- Unigram Model (UM)
- Cluster Based Document Model (CBDM)

1

ML-based Bug Localization

- Trained BP Neural Network
- BugScout —— an extended LDA
- A Two-phase Recommendation Model
- Learning to Rank
- Combine the LSTM and CNN Model
- HyLoc Combine IR with Six DNNs
- Enhanced CNN

2

More on extracting semantic information in bug reports but ignored many useful IR-base features

Combine the best of both worlds

SOLUTION APPROACH: AN OVERVIEW

Assumption

For a repository of software projects, there are **three repositories** available:

- a repository of historical bug reports,
- a repository of source code files,
- a repository of bug fixing history.

Let **SF** and **BR** denote the set of N_s source files and the set of N_b bug reports.

Source file s ∈ SF	Bug report b ∈ BR	Bug fixing history record
• Source File Identifier • the Class • the Method • the Variable • the Comment • API Documents	• Bug Identifier • the Summary • the Description	• Bug Identifier • Time Stamp tb when the bug report is fixed • Set of source file identities associated with this bug b

SOLUTION APPROACH: AN OVERVIEW

MD-CNN Design Overview

The development of MDCNN bug localization model consists of **two phases**:

- Model Training
- Model Deployment

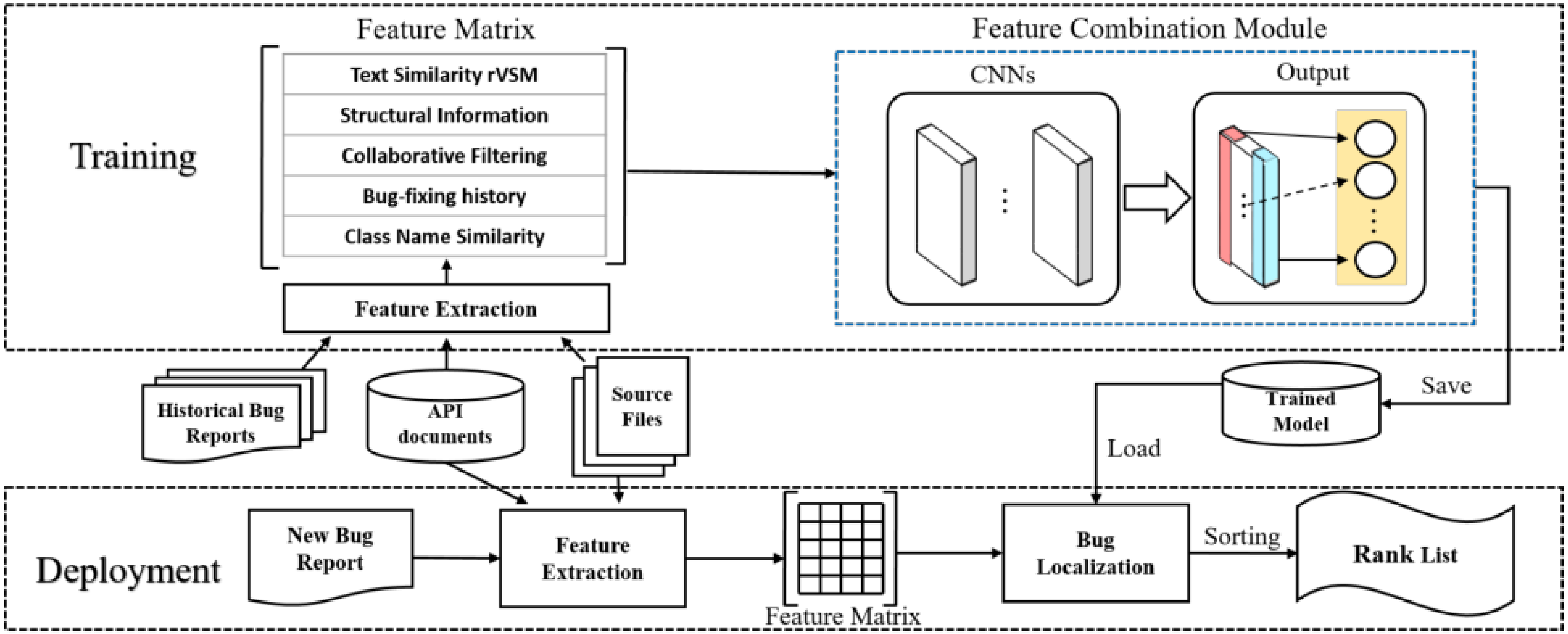


Fig. 1. The Overall Framework of MD-CNN

APPROACH OVERVIEW —— MD-CNN Model Training

Statistical Feature Extraction

The goal of the first task:

- prepare the training dataset by preprocessing the raw input data from a set of historical bug reports stored in the bug tracking system,
- generate a set of statistical features that capture the varying types of relationships between bug reports and source code files.

Five Features

- **Text similarity** between bug report and source file
- Bug-source relationship by **similarity of bug reports**
- Bug-source relationship by **recently-fixed source files**
- Bug-source relationship by **class name similarity**
- Bug-source relationship by **structural similarity**

Construct Feature Matrix

The second task is to construct a feature matrix of size **5** $\times N_s$ for each bug report in the training set, say N_b.

- provide N_b training **inputs**, each is represented as a feature matrix of N_s columns and five rows, and corresponds to a bug report in the training set.
  - A column corresponds to a source file in the training set
  - The five statistical features (ranking scores in the range of [0,1]) as its row values
- Configuring a CNN model with varying number of kernels in convolutional layers
- The row values are the **output** generated by the feature extraction task for each pair of source file and bug report

APPROACH OVERVIEW —— Model Deployment

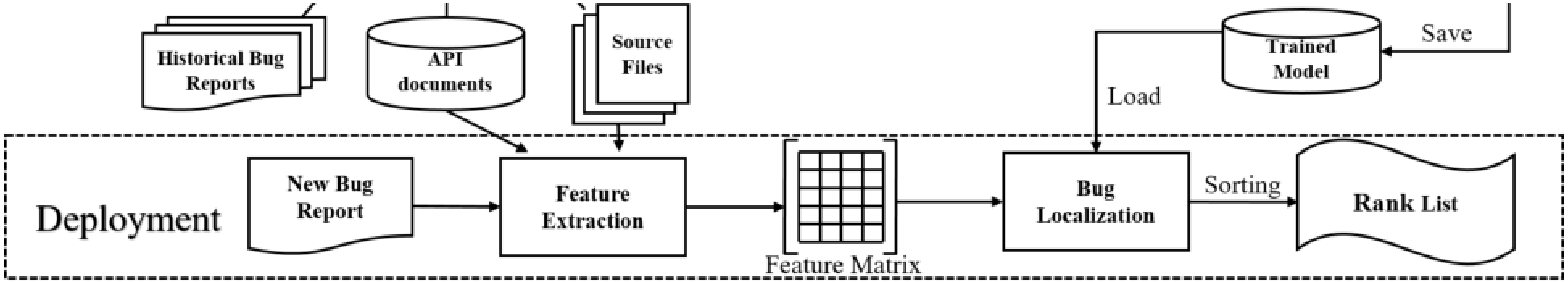
Model Deployment

The pretrained MD-CNN model for automated bug localization prediction will be performed upon request.

Let $\mathbf{f}_{\mathbf{MD-CNN}(\boldsymbol{\theta},\mathbf{r})}$ denote the trained MD-CNN model:

- **r** as the query with the new bug report
- **θ** as the model parameters
  - the number of hidden layers used by the MD-CNN model
  - the number of kernel filters Wi(i > 1)

The **output** of $\mathbf{f}_{\mathbf{MD-CNN}(\boldsymbol{\theta},\mathbf{r})}$ is a probability vector of size N_s for the query **r** with the top **k** highest scores as the top k best source files that match the new bug report.



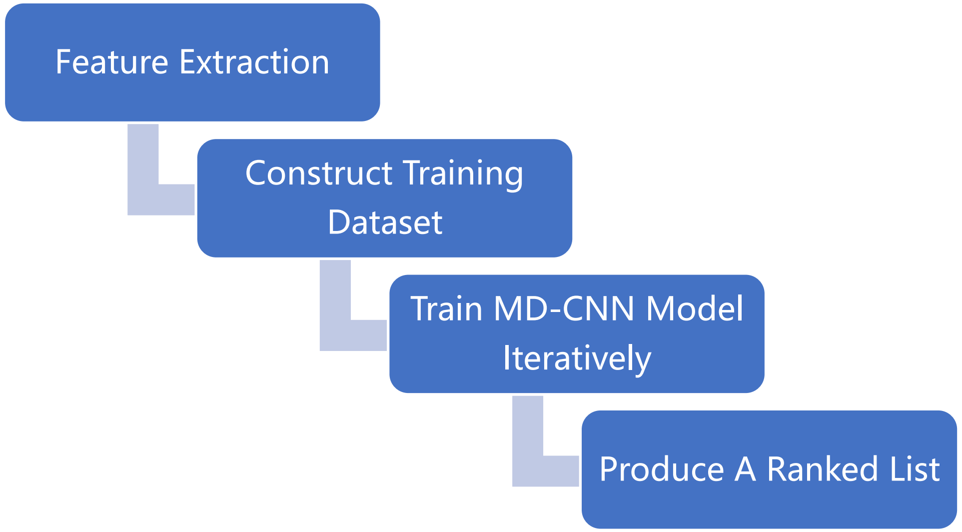
STATISTICAL FEATURE EXTRACTION

Extract Important Dimensions Of Features

- Features between each pair of a bug report and a source file, denoted by **(b, s)**, ∀b ∈ BR, ∀s ∈ SF.
- For each pair (b, s), we extract **k** features from them and build the feature vectors $\boldsymbol{Score(b,s) = [Score_i(b,s)]_{1 \le i \le k}}$.
- Each feature extraction algorithm will take the data **input** from the three repositories and **output** a similarity score for each pair (b, s).

TABLE 1
Features Used in the MD-CNN Model

Dimension	Formula
Text Similarity	$Score_{t-sim}(b,s) = g(n_t) \times cos(b,s) = \frac{1}{1+e^{\gamma mm(n_t)}} \times \frac{\vec{b}\cdot\vec{s}}{\|\vec{b}\|\,\|\vec{s}\|}$
Similar Bug History	$Score_{cf-sim}(b,s) = \sum_{i=1}^{k} \frac{1}{i} sim - rank(b, B(s))$
Bug-fixing History	$Score_{h-sim}(b,s) = \sum_{s \in H_m} \frac{1}{1+e^{-\frac{12t_{elapse}(s,b)}{m}+w(s)}}$
Class Name Similarity	$Socre_{c-sim}(b,s) = \begin{cases} max\_len(cn) & if\ cn \in s.class \cap b.class \\ 0 & otherwise \end{cases}$
Structural Similarity	$Score_{s-sim}(b,s) = \sum_{b_p \in b} \sum_{s_p \in s} sim(b_p, s_p)$



STATISTICAL FEATURE EXTRACTION

Text Similarity

- rVSM

Term frequency $\boldsymbol{tf(t,d)}$

Inverse document frequency $\boldsymbol{idf(t,d)}$

Term weight $\boldsymbol{w_t}$ in each document vector of size n

number of occurrences of a term **t** in a document **d**

number of documents that contain the term **t**

D = BR∪SF

$$tf(t,d) = log(\boxed{f_{td}} + 1)$$
$$idf(t,\boxed{D}) = log(\frac{|D|}{\boxed{d_t}}) \quad (1)$$

$$w_{t\in d} = tf(t,d) \times idf_{t,D} = log\,(f_{td}+1) \times log(\frac{|D|}{d_t}) \quad (2)$$

- cosine similarity → Equation (4)

total number of distinct terms in the source file s

$$Score_{t-sim}(b,s) = g(\boxed{n_t}) \times cos(b,s)$$
$$= \frac{1}{1+e^{\boxed{\gamma_{mm}(n_t)}}} \times \frac{\vec{b}\cdot\vec{s}}{\|\vec{b}\|\|\vec{s}\|} \quad (4)$$

length of document s

takes into account of larger documents during the ranking

Min-Max normalization method to normalize nt

- **Bug report**, extract the text of
  - ➢ summary, description and comments
- **Source file**, extract the
  - ➢ string-literal in addition to comments and identifiers
- **API documentation** in the source file
  - ➢ text description of the classes and interfaces through term extraction from API speciation

Bug Report

Project: Eclipse_Platform_UI
Bug_ID: 407505
Summary: Maximise-Restore causes hidden editor area to be shown
Description: In our Eclipse-based RCP we don't always need to have an editor area, so hide it using **WorkbenchPage**.setEditorAreaVisible(false).
......
Even though it is not visible it is getting added to elements-ToMinimize which means is gets tagged with **MINIMIZED** & MINIMIZED_BY_ZOOM and therefore set to visible when restore is called.

Bug_Files:
bundles/org.eclipse.e4.ui.workbench.addons.swt/src/org/eclipse/e4/ui/workbench/addons/minmax/**MinMaxAddon.java**

Source File

File_Name: MinMaxAddon.java
Content:
import org.eclipse.swt.widgets.**Shell**;
...
final **Shell** winShell = (**Shell**) window.getWidget();
...
partService.requestActivation();

API Document

API_Name: Shell
Content:
...
Instances that do have a parent are described as secondary or dialog shells. Instances are always displayed in one of the maximized, **minimized** or normal states: When an instance is marked as maximized, the window manager will typically resize it to fill the entire visible area of the display, and the instance is usually put in a state where it can not be resized (even if it has style RESIZE) until it is no longer maximized. When an instance is in the normal state (neither maximized or **minimized**), its appearance is controlled by the style constants which were specified when it was created and the restrictions of the window manager (see below). When an instance has been marked as **minimized**
...

API_Name: partService
Content:
A part service tracks the creation and activation of parts within a **workbench page**. This service can be acquired from your service locator: IPartService service = (IPartService) getSite().getService(IPartService.class); This service is not available globally, only from the **workbench** window level down.
See Also: I**WorkbenchPage**, IServiceLocator.getService(Class)
Restriction: This interface is not intended to be implemented by clients.

STATISTICAL FEATURE EXTRACTION

Similarity to Historical Bug Reports

Examine the bug fixing history to extract those previously fixed bug reports that are textually similar to the current bug report.

Let **br(b, s)** denote the set of historical bug reports associated with a source file **s** and are fixed before the current bug report **b**.

Source code file: **AjBuildManager.java**	
Bug ID: 272591	Summary: couldn't find **aspectjrt.jar** on **classpath**
	Description: I am using the **aspectj** runtime **jar** that is in the spring source bundle repository. The have renamed their jar to match their naming conventions and it is causing the warning to occur. Their bundle is named com.springsource.org.**aspectj**.runtime-1.6.3.RELEASE.jar. It would be nice if this **warning** was not printed out in this case.
Bug ID:34951	Summary: NPE **compiling** without **aspectjrt.jar**
	Description: **Compiling** spacewar without specifying **aspectjrt.jar** on the **classpath** causes a NPE. Expected an error message "**aspectjrt.jar** required". Steps to reproduce: 1) install latest 2) cd doc/examples3) java -jar ../../lib/aspectjtools.jar -verbose @spacewar/debug.lst Result :NPE in attached log
Bug ID: 112830	**Warning** "couldn't find **aspectjrt.jar** on **classpath**"
	The **compiler** makes this warning if "**aspectrt.jar**" file has a different name like "**aspectrt-1.3.jar**", which is the case when **compiling** with maven.

Fig. 3. Bug reports that are similar with a single source file

Computes the textual similarity between the current bug report **b** and the summaries of all the bug reports in br(b, s).

$$Score_{hbs-sim}(b,s) = cosine(b, br(b,s)) \qquad (5)$$

Normalize the CF score of the similar historical bugs for each source file.

the similarity ranked list
in descending order

$$Score_{cf-sim}(b,s) = \sum_{i=1}^{k} \frac{1}{i} \boxed{sim - rank(b, \boxed{B(s)})} \qquad (6)$$

set of bug reports for which the
source file s was fixed before the
current bug report b was received

STATISTICAL FEATURE EXTRACTION

Similarity to Recent Buggy Source Files

- The change history data of source code in the version control systems.
- A source file is more likely to contain faults if it has recently been changed by fixing bugs.

Bug_ID : 272354
Report_Time : 2009-04-15 14:21
Modify_Time : 2009-04-16 03:00
Buggy_Files :
org.eclipse.jdt.junit.core/src/org/eclipse/jdt/internal/junit/buildpath/**P2Utils.java**
org.eclipse.jdt.junit/src/org/eclipse/jdt/internal/junit/buildpath/**P2Utils.java**

Bug_ID : 272418
Report_Time : 2009-04-15 19:57
Buggy_Files :
org.eclipse.jdt.junit.core/src/org/eclipse/jdt/internal/junit/buildpath/**P2Utils.java**

Bug_ID : 274041
Report_Time : 2009-04-28 10:07
Buggy_Files :
org.eclipse.jdt.junit.core/src/org/eclipse/jdt/internal/junit/buildpath/**P2Utils.java**
org.eclipse.jdt.junit/src/org/eclipse/jdt/internal/junit/buildpath/**P2Utils.java**

Fig. 5. Three bug reports corresponding to the same source code file

- Time based decaying method
- Equation (7) defines the similarity to the recent buggy files
- **w(s)** denotes the shortest time between a bug-fixing commit for the source file **s** and the current bug report **b**

set of buggy source files that are found in **m** days before receiving the bug report **b** at **tb**

$$Score_{h-sim}(b,s) = \sum_{s\in \boxed{H_m}} \frac{1}{1+e^{-\frac{12t_{elapse}(s,b)}{m}+w(s)}} \qquad (7)$$

$$w(s) = min_{s\in H_m, t_{elapse}(s,b)\leq m} \boxed{t_{elapse}(s,b)} \qquad (8)$$

number of days that have elapsed between a bug-fixing commits and a newly submitted bug report **b**

STATISTICAL FEATURE EXTRACTION

Class Name Similarity

- Checking whether the name of each class in the source code file is also included in the bug report.
- For all the class names present in the bug report, use the maximum length of the class name as the similarity value for the $\boldsymbol{Score_{cn}(b,s)}$.

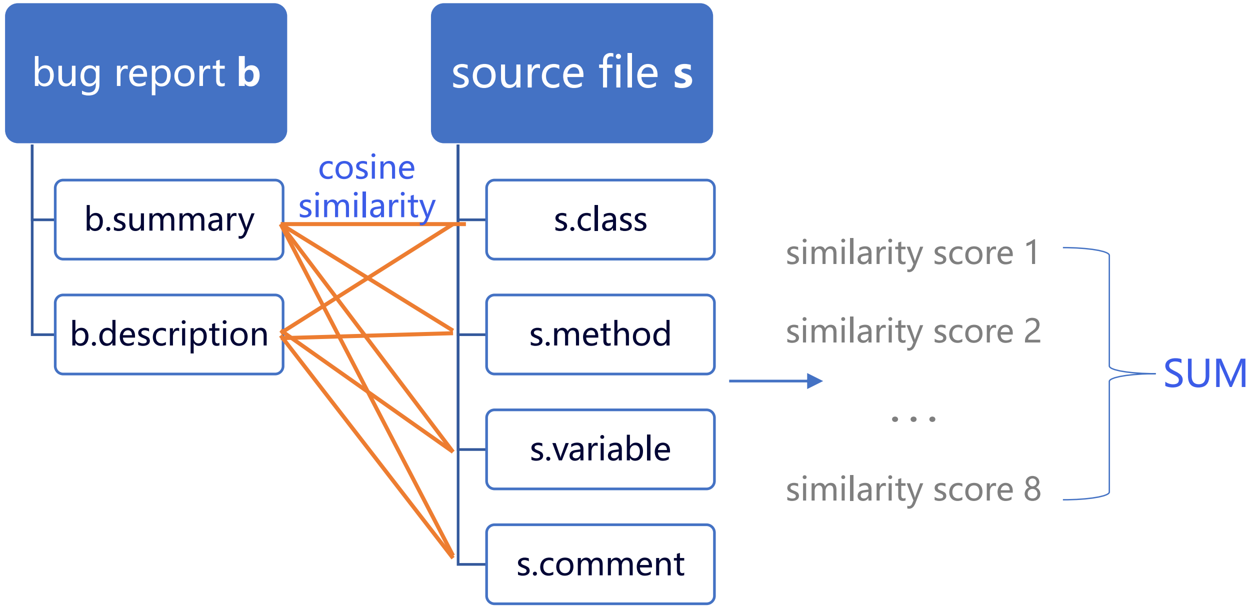
set of words in the bug report b

$$Socre_{c-sim}(b,s) = \begin{cases} \boxed{max\_len(b,s)} & if\ \ cn \in \boxed{s.class} \cap \boxed{b.class} \\ 0 & otherwise \end{cases} \tag{9}$$

longest class name of the source file s which appeared in the bug report b

set of class names in a source file s

Structural Similarity



one of the four text segments in a source file s

$$Score_{s-sim}(b,s) = \sum_{b_p \in b} \sum_{s_p \in s} sim(\boxed{b_p}, \boxed{s_p}) \tag{10}$$

one of the two text segments in a bug report b

MD-CNN MODELING

Feature Scaling with Min-Max Normalization

For a given feature **ξ**, set **ξmin** and **ξmax** as the minimum and the maximum observed values in the training dataset.

- If $\xi > \xi_{max}$, set ξ to be ξ_{max};
- if $\xi < \xi_{min}$, then set $\xi = 0$.
- For $\xi_{min} \leq \xi \leq \xi_{max}$, if $\xi > 1$, then need to employ the min-max normalization to scale ξ to the value range of [0,1].

$$\frac{\xi - \xi_{min}}{\xi_{max} - \xi_{min}},$$

MD-CNN MODELING

Feature Combination

- Linear models fail to capture the hidden and non-linear relationship among the different types of features across bug reports and source files.
- Replace the linear model for combining the bug-source file similarity features for bug localization by using deep neural networks such as convolutional neural network.
- Take a **two phase approach** to develop a CNN-based non-linear feature combinator

Project: AspectJ
Bug_ID: **263837**
Summary: Error during Delete AJ Markers
Description: Error sent through the AJDT mailing list. I believe this is an LTW weaving error, so not raising it against AJDT.
Bug_Files: weaver/src/org/aspectj/weaver/bcel/**BcelClassWeaver.java**
weaver/src/org/aspectj/weaver/bcel/**BcelTypeMunger.java**
weaver/src/org/aspectj/weaver/bcel/**BcelWeaver.java**

scores after normalization	*t_sim*	*cf_sim*	*h_sim*	*c_sim*	*s_sim*
BcelClassWeaver.java	0	0.87	0.03	0.43	0
BcelTypeMunger.java	0.01	0.59	0.8	0	0
BcelWeaver.java	0.86	0.71	0.81	0.43	0.08

Fig. 6. An example of the non-linear relationship between the features

MD-CNN MODELING

Convolutional Neural Network Structure

- two epochs
- the learning rate = 1e3 and batch size = 32
- brute force method

$$Lost(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{T} W * t_{ij}log(y_{ij}) + (1 - t_{ij})log(1 - y_{ij})) \qquad (11)$$

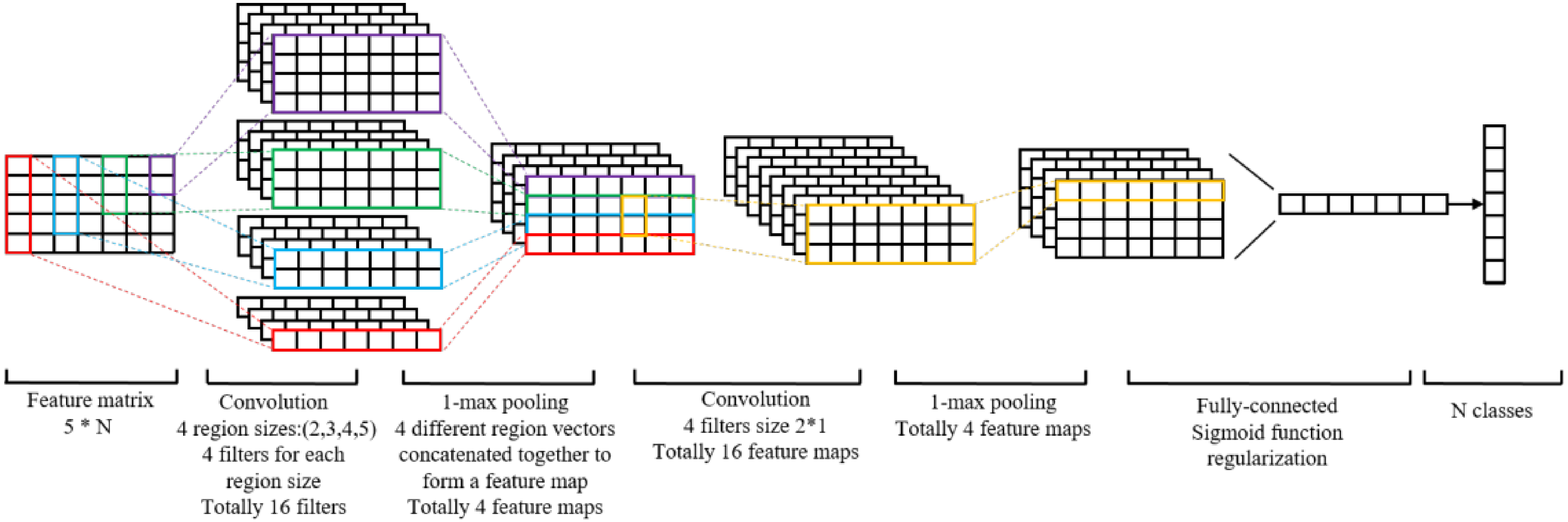


Fig. 7. The architecture of convolutional neural network

EXPERIMENTAL EVALUATION

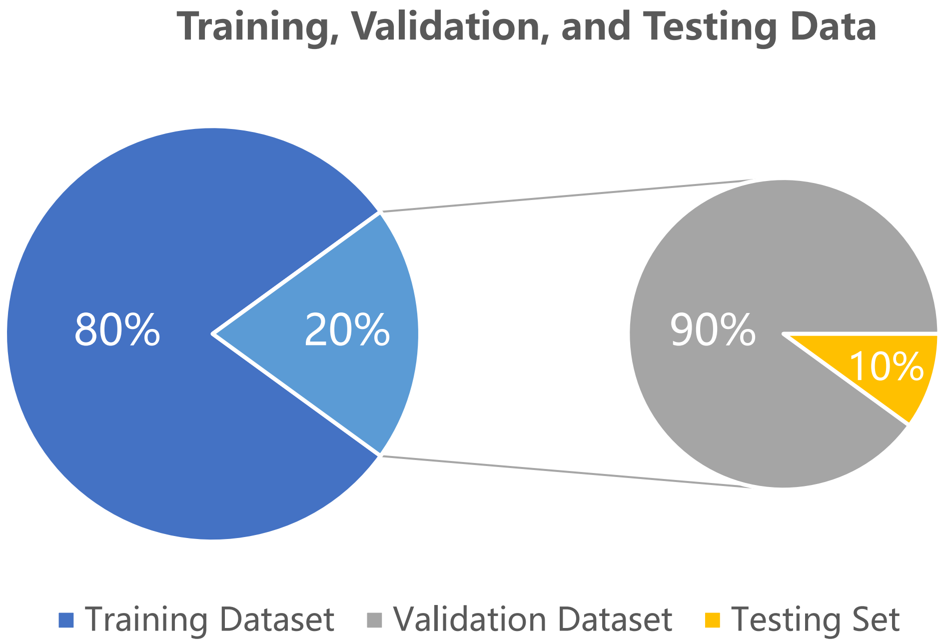
Dataset

- For comparison, use the same collection of datasets provided in LR.
- It contains a total of 22,747 bug reports from six popular open-source projects: Eclipse Platform UI, JDT, Bir747t, SWT, Tomcat, and AspectJ.

1

TABLE 2
Benchmark Datasets

Project	Time Range	#bug reports	#source files	#API entries
Eclipse	10/01–01/14	6495	3454	1314
JDT	10/01–01/14	6274	8184	1329
Birt	06/05–12/13	4178	6841	957
SWT	02/02–01/14	4151	2056	161
Tomcat	07/02–01/14	1056	1552	389
AspectJ	03/02–01/04	593	4439	54



EXPERIMENTAL EVALUATION

Evaluation Metrics

- **Accuracy@k**: the percentage of the bug reports that have found at least one buggy source files in the top k (k= 1,5,10,20) ranked files returned.
- **MAP (Mean Average Precision)**: the mean of the Average Precision (AvgP) scores across all bug report queries.
- **MRR (Mean inverse Rank)**: the mean of the Reciprocal Rank for all queries.

Evaluation Plan

Four evaluation objectives:

- The effectiveness of our MD-CNN by comparing it with existing representative bug localization systems.
- The importance of different features on the overall performance of our MD-CNN.
- The impact of training data on the performance of our MD-CNN.
- The impact of multiple system parameters on the performance of MD-CNN.

EXPERIMENTAL EVALUATION

Performance and Effectiveness of MD-CNN

Tour representative baseline approaches:

- Learning to Rank (LR)
- BugLocator (BL)
- The standard VSM method VSM
- Deep Neural Networks (DNN)

1

Performance Impact of Each Feature on MD-CNN

- Compute the MAP of each feature on all six datasets
- Greedy algorithm to sort the five features on each dataset
- Performance of combining features

2

EXPERIMENTAL EVALUATION

- TensorFlow to construct the CNN model
- A server with Intel Xeon CPU E5-2650 and NVIDIA GPU TITAN V

TABLE 3
Different cliff's delta and effectiveness level [6]

Cliff's Delta ($\|\delta\|$)	Effectiveness Level
$0.000 \leq \|\delta\| < 0.147$	Negligible
$0.147 \leq \|\delta\| < 0.330$	Small
$0.330 \leq \|\delta\| < 0.474$	Medium
$0.474 \leq \|\delta\| \leq 1.000$	Large

TABLE 5
Performance comparison (MAP and MRR) with four representative baseline methods (VSM, BL, LR, DNN)

Dataset	MAP					MRR				
	VSM [21]	BL [49]	LR [44]	DNN	MD-CNN	VSM [21]	BL [49]	LR [44]	DNN	MD-CNN
AspectJ	0.12	0.22	0.38	0.40	**0.41**	0.16	0.32	0.44	0.46	**0.46**
Birt	0.05	0.14	0.17	0.21	**0.22**	0.07	0.18	0.21	0.23	**0.25**
Eclipse	0.20	0.31	0.44	0.47	**0.48**	0.25	0.37	0.51	0.54	**0.54**
JDT	0.12	0.23	0.40	0.45	**0.45**	0.15	0.30	0.47	0.53	**0.53**
SWT	0.08	0.38	0.40	0.51	**0.53**	0.09	0.44	0.46	0.56	**0.57**
Tomcat	0.33	0.43	0.52	0.54	**0.55**	0.36	0.48	0.55	0.59	**0.60**
Average	0.15	0.285	0.385	0.43	0.44	0.18	0.348	0.44	0.485	0.492
Improved%	+193.3	+54.4	+14.3	+2.3	-	+173.3	+41.4	+11.8	+1.4	-
p-Value	<0.01	<0.05	>0.05	-	-	<0.01	>0.05	>0.05	-	-
δ	0.944	0.667	0.5	-	-	0.889	0.667	0.388	-	-

TABLE 6
Training and Test Time of MD-CNN (in minutes)

Project	Training time on the dataset (Average)		Test time for one report (Average)	
	Feature Extraction	Feature Combination	Feature Extraction	Feature Combination
AspectJ	89	177	0.46	0.07
Birt	138	181	0.68	0.08
Eclipse	196	193	0.39	0.07
JDT	210	199	0.82	0.08
SWT	129	186	0.25	0.06
Tomcat	68	170	0.15	0.05

TABLE 4
Performance comparison (Accuracy@k, k=1,5,10,20) of four representative baseline methods (VSM, BL, LR, DNN)

Dataset	Accuracy@1					Accuracy@5					Accuracy@10					Accuracy@20				
	VSM [21]	BL [49]	LR [44]	DNN	MD-CNN	VSM [21]	BL [49]	LR [44]	DNN	MD-CNN	VSM [21]	BL [49]	LR [44]	DNN	MD-CNN	VSM [21]	BL [49]	LR [44]	DNN	MF-CNN
AspectJ	0.116	0.251	0.374	0.402	**0.453**	0.209	0.404	0.523	0.567	**0.59**	0.285	0.48	0.637	0.738	**0.752**	0.40	0.505	0.738	0.785	**0.801**
Birt	0.043	0.111	0.124	0.173	**0.197**	0.096	0.259	0.289	0.371	**0.403**	0.117	0.321	0.381	0.513	**0.530**	0.178	0.399	0.489	0.568	**0.609**
Eclipse	0.185	0.271	0.397	0.402	**0.448**	0.337	0.538	0.654	0.713	**0.735**	0.422	0.616	0.743	0.805	**0.822**	0.523	0.710	0.821	0.833	**0.844**
JDT	0.097	0.181	0.334	0.427	**0.439**	0.201	0.39	0.635	0.714	**0.720**	0.287	0.502	0.729	0.800	**0.811**	0.396	0.604	**0.832**	0.827	0.831
SWT	0.044	0.198	0.313	0.418	**0.464**	0.118	0.381	0.624	0.739	**0.752**	0.199	0.496	0.75	0.852	**0.867**	0.433	0.612	0.835	0.878	**0.893**
Tomcat	0.208	0.351	0.419	0.434	**0.467**	0.487	0.651	0.715	0.743	**0.776**	0.599	0.716	0.802	0.827	**0.859**	0.680	0.815	**0.898**	0.856	0.871
Average	0.116	0.227	0.327	0.376	**0.412**	0.241	0.437	0.573	0.641	**0.663**	0.318	0.522	0.673	0.756	**0.773**	0.435	0.608	0.769	0.791	**0.813**
Improved%	+255.17	+81.49	+25.99	+9.57	-	+175.1	+51.71	+15.71	+3.43	-	+143.08	+48.08	+14.86	+2.25	-	+86.9	+33.72	+5.72	+2.78	-
p-Value	<0.01	<0.05	<0.05	-	-	<0.01	<0.05	>0.05	-	-	<0.01	<0.05	>0.05	-	-	<0.01	<0.05	>0.05	-	-
δ	0.944	0.778	0.722	-	-	0.944	0.778	0.5	-	-	0.944	0.889	0.667	-	-	0.944	0.778	0.167	-	-

EXPERIMENTAL EVALUATION

Impact of Each Feature on MD-CNN

- These five dimensions features contribute differently to each dataset in terms of the accuracy of bug localization.
- Every feature in these five is useful, effective, and necessary.

TABLE 7
The MAP of each feature on six projects

Feature	AspectJ	Birt	Eclipse	JDT	SWT	Tomcat
Text Similarity	0.264	0.157	0.352	0.312	0.344	0.457
Similar Bug History	0.090	0.178	0.212	0.372	0.413	0.307
Bug-fixing History	0.247	0.049	0.089	0.042	0.135	0.037
Class Name Similarity	0.133	0.050	0.197	0.146	0.167	0.094
Structural Similarity	0.093	0.076	0.194	0.126	0.105	0.196
MD-CNN (5 combo)	**0.41**	**0.22**	**0.48**	**0.45**	**0.53**	**0.55**

TABLE 8
The importance of features using greedy algorithm (Feature NO.1: Text Similarity; Feature NO.2: Similar Bug History; Feature NO.3: Bug-fixing History; Feature NO.4: Class Name Similarity; Feature NO.5: Structural Similarity;)

Dataset		First	Second	Third	Fourth	Fifth
AspectJ	Feature	1	3	4	2	5
	MAP	0.264	0.359	0.387	0.405	0.412
	Improved	-	+36.0%	+7.8%	+4.7%	+1.7%
Birt	Feature	2	1	4	5	3
	MAP	0.178	0.195	0.209	0.216	0.220
	Improved	-	+9.6%	+7.2%	+3.3%	+1.9%
Eclipse	Feature	1	2	4	5	3
	MAP	0.352	0.415	0.441	0.466	0.479
	Improved	-	+17.9%	+6.3%	+5.7%	+2.8%
JDT	Feature	2	1	4	5	3
	MAP	0.372	0.413	0.431	0.444	0.452
	Improved	-	+11.0%	+4.4%	+3.0%	+1.8%
SWT	Feature	2	1	4	3	5
	MAP	0.413	0.475	0.507	0.523	0.534
	Improved	-	+15.0%	+6.7%	+3.2%	+2.1%
Tomcat	Feature	1	2	5	4	3
	MAP	0.457	0.494	0.524	0.543	0.548
	Improved	-	+8.1%	+6.1%	+3.6%	+0.9%

EXPERIMENTAL EVALUATION

Impact of Model Parameters on MD-CNN

Three parameters:

- cf-sim feature (similarity to the historical fixed bugs)
- h-sim feature (similarity to the recent-fixed source files)
- number of convolutional layers

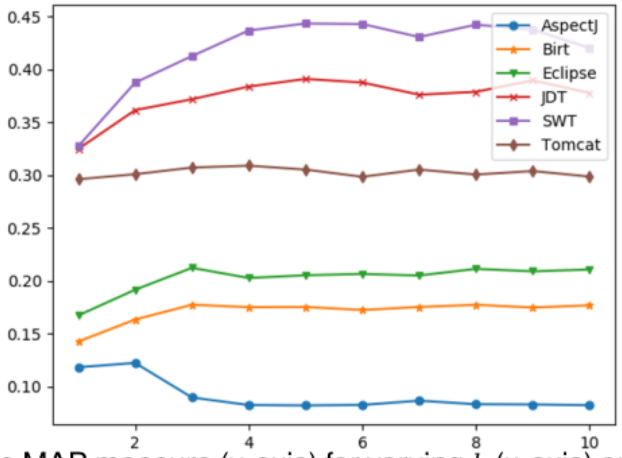


Fig. 8. The MAP measure (y-axis) for varying k (x-axis) on Similar Bug History feature (cf-sim)

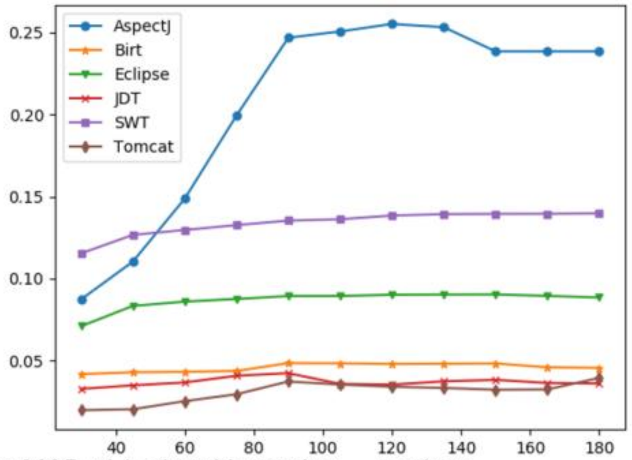


Fig. 9. The MAP of *h-sim* with varying m settings

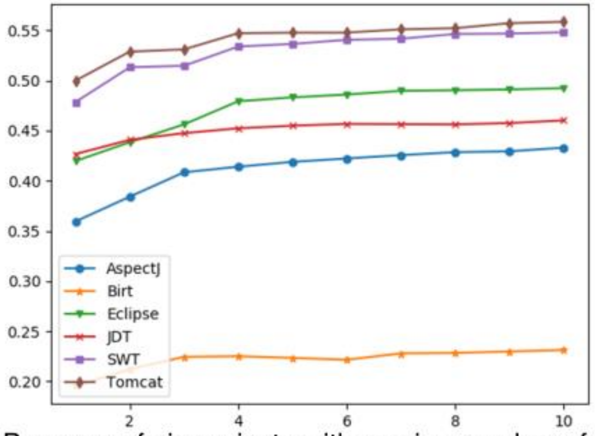


Fig. 10. MAP scores of six projects with varying number of convolution layers in MD-CNN

论　文　汇　报　展　示

感 谢 您 的 聆 听

汇报人：王昭丹