Retrieving Similar Software from Large-scale Open-source Repository by Constructing Representation of Project Description

Chuanyi Li[1,3][a], Jidong Ge[1][b], Victor Chang[2][c] and Bin Luo[1]

[1] *State Key Laboratory for Novel Software Technology, Software Institute, Nanjing University, Nanjing, China*

[2] *School of Computing and Digital Technologies, Teesside University, Middlesbrough, U.K.*

[3] *State key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China*

Keywords: Information Retrieval, Project Similarity, Big Text Data, Learn to Rank.

Abstract: The rise of open source community has greatly promoted the development of software resource reuse in all phases of software process, such as requirements engineering, designing, coding, and testing. However, how to efficiently and accurately locate reusable resources on large-scale open source website remains to be solved. Presently, most open source websites provide text-matching-based searching mechanism while ignoring the semantic of project description. For enabling requirements engineers to find software that are similar to the one to be developed quickly at the very beginning of the project, we propose a searching framework based on constructing semantic embedding for software project with machine learning technique. In the proposed approach, both Type Distribution and Document Vector learnt through different neural network language models are used as project representations. Besides, we integrate searching results of different representations with a Ranking model. For evaluating our approach, we compare search results of different searching strategies manually using an evaluating system. Experimental results on a data set consisting of 24,896 projects show that the proposed searching framework, i.e., combining results derived from Inverted Index, Type Distribution and Document Vector, significantly superior to the text-matching-based one.

1 INTRODUCTION

The large number of open source software products brings great opportunities and challenges to software reuse. No matter at which level of reuse, e.g., requirements reuse, design reuse, code reuse and module reuse and product reuse, the most important thing is to locate reusable artifacts in the massive open source software repository. For reusing, the earlier the stage is, the more benefits there are. So, how to find appropriate open source projects of similar requirements to the one to be developed at the very beginning of software process has always been an important topic to be studied these days.

However, at the very beginning of a project, only the simple description of the product is available. Many existing software similarity measurement approaches relying on other software artifacts cannot be applied. Existing open source software repositories support keywords based similar project searching approaches but ignore the deep semantic of text as well as the concept of software domain knowledge. Upon deriving query results in this way, users still need to review many of the retrieved project description with their idea one by one to check if they are what they want. Besides, for users without software engineering background, it is hard for them to pick up exact keywords for searching.

In this paper, we propose a similar software searching framework according to software description. Figure 1 shows the main steps of our approach. First, we construct inverted index, train Doc2Vec model and train software type predictor for the repository. Then, apply clustering algorithms on document vectors and type distribution vectors respectively to all products to generate clusters for fastening the searching efficiency. While a query is made, three different result sets will be given. Next, we train a ranking model for re-rank the fetched results in the three sets. For training the type predictor,

[a] https://orcid.org/0000-0003-4956-8142
[b] https://orcid.org/0000-0003-1773-0942
[c] https://orcid.org/0000-0002-8012-5852

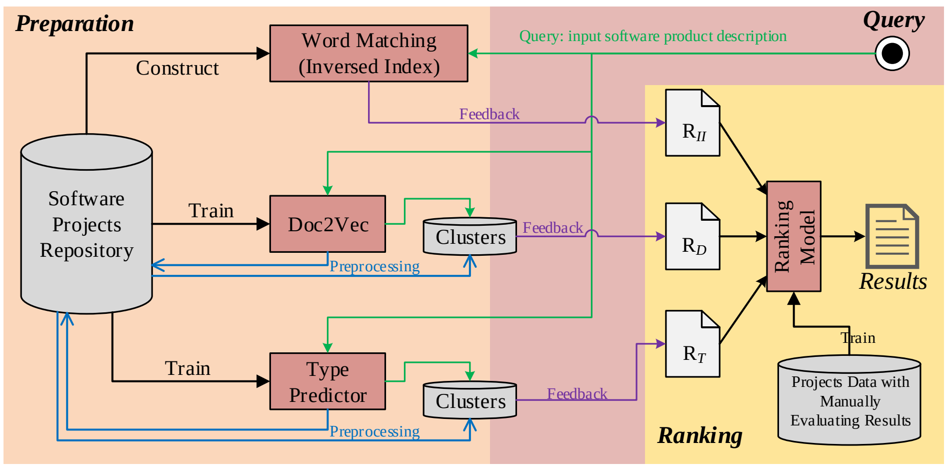

Figure 1: Framework of the proposed Software Project Searching Approach.

we try different machine learning algorithms including Support Vector Machine (i.e., SVM)(Cortes and Vapnik, 1995), full connected Neural Network (i.e., FNN)(Epelbaum, 2017), Convolutional Neural Network (i.e., CNN)(Kim, 2014) and Long Short-Term Memory neural network (i.e., LSTM)(Hochreiter and Schmidhuber, 1997). The experimental results show that our approach improves the similar software product searching results significantly.

In conclusion, we make the following contributions in this paper:

(1) Organizing a dataset consisting of 24,896 software projects for conducting similar software product searching research. 2000 projects are used as testing data and the others are used as the searching database.

(2) Designing and implementing an integrated similar software product/project searching approach by adopting learning to rank techniques.

(3) Developing a reusable manual annotating system for evaluating different searching methods.

The rest of this article is organized as follows. Section 2 introduces some related work. Section 3 describes the dataset. Details of the proposed approach are illustrated in Section 4. Experiments and evaluations are presented in Section 5. Section 6 concludes the paper.

2 RELATED WORK

Most existing related work mainly measure project similarity based on software code or other artifacts. Lannan Luo et al. (Luo et al., 2017) proposed a binary code similarity comparison method using the longest common subsequence to judge the similarity degree of items according to the code. Naohiro Kawamitsu et al. (Kawamitsu et al., 2014) proposed a technique to automatically identify the source code reuse relationship between two code bases. Shuai Wang et al. (Wang and Wu, 2017) proposed a new method for binary code similarity analysis using memory fuzziness. The main difference between our work and these is that we measure the natural language similarity but not the code similarity. For non-software engineering domain text similarity measuring, there are many existing works based on machine learning techniques. Lin Yao et al. (Yao et al., 2018) propose a new LSTM encoder for the existing similarity measurement algorithm of short text. Chenghao Liu et al. (Liu et al., 2017) proposed an online bayesian inference algorithm for collaborative subject regression (CTR) model to act on the recommendation system. Naresh Kumar Nagwani et al. (Nagwani, 2015) proposed a similarity measure of text processing (SMTP) for knowledge discovery of text collection. Yuhua Li et al. (Li et al., 2006) proposed an algorithm that takes into account the semantic information and word order information implied in sentences. Different from these works, we aim at solving the software project description similarity problem by considering the software engineering domain knowledge while adopting the machine learning techniques.

3 CONSTRUCTED DATA SET

We prepare the data set according to the large open-source and business software platform SourceForge (i.e., https://sourceforge.net). We downloaded the Top 24,896 open-source software projects from Source-

Forge by setting the searching filter of Operating System to Windows in January of 2018. All these projects are belonging to 17 categories. The number of projects of each category is as shown in Table 1. For the projects in the prepared data set, there are their *names*, *categories*, and *descriptions*.

For approach evaluation, we randomly choose 2000 projects from each category at a same ratio for comparing different approaches. The left projects are used in constructing the searching database. In the remain parts of this paper, we use *Evaluating Set* referring to 2000 selected projects and *Database* for the other projects. For training and testing the ranking strategy, the evaluating set is utilized.

4 APPROACH

4.1 Database Preparation

There are three concrete searching methods in our approach, namely, words matching, text semantic matching and software category matching. For each method, we need to prepare searching index based on database.

Words Matching. For implementing efficient words matching searching, we generate Inverted Index for descriptions of software projects in our searching data base. Inverted indexing technology for text character matching is widely used in all kinds of search engines. It represents the connection among texts to some extents, especially the Top-K searching results performs well in most application scenario. We utilize inverted indexing for modeling the text similarity from the words matching perspective and provide similar software projects as part of the integrated searching approach. We only retrieve keywords in the description for generating the inverted index. The keywords are those have top-25 TF-IDF (Term Frequency-Inverse Document Frequency) values in each project description. We do not take stopwords into account in building the inverted index and the stopwords are those in Default English stopwords list[1]. We adopt the open-source search library Lucene[2] for implementing words matching.

Semantic Matching. In order to modeling deep semantics in measuring software product similarity, we adopt Doc2Vec (Le and Mikolov, 2014) for generating representation of software description. For calculating similarities among a large number of texts, a general method for transforming each text into a fixed-length feature vector is needed. Doc2Vec is an unsupervised algorithm that learns fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents. In our approach, we apply the Doc2Vec tools provided by gensim[3] to transform each software project description into a dense vector. All projects in the data set are used in training the Doc2Vec model. Values of core parameters for training are $vector\_size = 150$, $window = 5$, $min\_count = 3$. These values are most frequently used in applying Doc2Vec tools. Another consideration is that the smaller the vector size is, the higher the querying response efficiency is.

Clustering. In order to fasten the searching efficiency, we apply k-Means(MacQueen et al., 1967) clustering algorithm on derived document vectors of projects in the database. Since clustering can cluster the sample data with similar semantics, it can effectively reduce the range of candidate sets and reduce the time cost and noise impact caused by excessive data. The vector similarity algorithm used in k-Means is cosine similarity(Gomaa and Fahmy, 2013). The number of clusters is set to 34, i.e., twice of the category number. While the searching is requested, the input project description will be firstly turn to a document vector and then find the nearest cluster center. The Top-K nearest projects in the same cluster will be retrieved as similar projects. Details of querying will be described in next subsection.

Category Matching. The first two parts of matching have little relation of software engineering domain knowledge. Software project taxonomy is one kind of software engineering domain specific knowledge. The category names are usually software property related words, such as Windows, Desktop Application, Mobile Application, and Security. We could use software category information of a project to represent the project for searching similar ones. Inspired by the neural network language model, we utilize machine learning methods to learn vectors of software project's category distribution from project description. We model this as a multiclass classification problem. We try different machine learning algorithms and different input feature representing methods. Combinations of learning algorithms and features are as following:

(1) **SVM + TF-IDF.** We adopt the multiclass SVM method which casts the multiclass classification problem into a single optimization problem. The tool we use is libsvm[4] developed by Chih-Chung Chang and

[1]Defined here: https://www.ranks.nl/stopwords/

[2]Found here: http://lucene.apache.org/

[3]Found here: https://radimrehurek.com/gensim/models/doc2vec.html

[4]Found here: https://www.csie.ntu.edu.tw/~cjlin/libsvm/

Table 1: Statistics on software projects in the prepared data set.

Category	# of projects	Examples	Category	# of projects	Examples
Development	4321	Hibernate, TortoiseSVN	Security & Utilities	690	Tor Browser, ophcrack
System Administration	2554	ScpToolkit, Rufus	Home & Education	596	Logisim, FMSLogo
Science & Engineering	2020	OpenCV, Octave Forge	Desktop	574	3D Desktop, Adobe Flash Updater
Games	1626	Amidst, Antimicro	Terminals	133	Alarm, BBSSH
Internet	1480	4chan Downloader, 51Degrees-PHP	Multimedia	91	BlackBox ISO Burner, Joystick To Mouse
Business & Enterprise	1238	KeePass, GnuWin	Mobile-apps	47	KeePass for J2ME, LineageOS on LeEco
Audio & Video	1105	Equalizer APO, ffdshow	Formats and protocols	43	MoodleRest Java Library, Podcast Generator
Communications	1244	Moodle, Scrollout F1	Nonlisted Topics	6244	Pokemon Online, Pokemon–Dark Rift
Graphics	890	Sweet Home 3D, gnuplot	***Total Number of projects***	24,896	

Chih-Jen Lin. We use TF-IDF vectors to represent each project's description. The size of vocabulary is 50,000 because we choose to keep words with top-50000 document frequency for representing each document. For tuning the parameter of SVM algorithm, i.e., Cost, we apply five-fold cross-validation on the data set consisting of all projects in the database. We choose the value for parameter *–c* that derives the highest average predicting performance in cross-validation. Details of evaluation metrics are described in the experiment section.

(2) **FNN + TF-IDF.** Different from SVM's constructing separating gap, neural network is to build a mapping function between input data instance and its labels. We firstly try simple full connected neural network to learn a project category predictor. Each description of project is represented by the TF-IDF vector which is the same as used in SVM. We adopt simple settings for the fully connected neural network. There are three hidden layers and they have 1,500, 750 and 17 neurons separately. The activation function is *Tanh*. The input and output layers have 50,000 and 17 neurons respectively. The output layer is also a $softmax$ layer applied on the third hidden layer.

(3) **CNN + Word2Vec.** Word2Vec (Mikolov et al., 2013) is a novel model architecture for computing continuous vector representations of words from very large data sets. In our approach, we adopt Word2Vec to train embedding for words. In our experiments, we length of vectors for words is set to 150 which is the same as that of vectors for the entire document.

Convolutional neural network uses convolution kernel to extract local features, and integrates and analyzes features in the way of multi-layer convolution, which shows great advantages in image processing (LeCun et al., 1998). We also try CNN with simple settings in training project category predictor. The number of input words of each project description is equal to the average length of all descriptions in the database, i.e., 46. For documents whose length is more than 46, we use the middle 46 words to represent the document. Each word is represented by a 150-dimension vector derived from Word2Vec model. We use three kernels of different sizes, namely 3, 4 and 5. There is only one convolution-pooling layer for each kernel and eventually they are concatenated together to predict the project categories. Like that in fully connected neural network, the output layer is a *softmax* layer.

(4) **LSTM + Word2Vec.** LSTM is an artificial recurrent neural network (RNN) architecture(Hochreiter and Schmidhuber, 1997). Both CNN and LSTM can be utilized as feature extracting approaches in processing natural language texts. CNN extract effective features from the entire text while LSTM extract effective semantic features specifically. In our approach, we only want to explore which one is better in encoding project category distribution for searching similar software projects among SVM, NN, CNN and LSTM. So, we also adopt simple LSTM in training project category predictor.

There are some common settings for NN, CNN and LSTM: (1) using *tanh* activation function, (2) using *Mean Squared Error* loss function, (3) use Stochastic Gradient Descent (SGD) loss optimizer, and (4) *softmax* layer as output layer.

Upon the category distribution vectors of projects in the database are derived, the clustering algorithm is also applied for indexing projects.

4.2 Query

The querying process is intuitively described in Figure 1. The details of each modules' work are illustrated as following:

Words Matching. Only non-stopwords are retrieved in searching similar projects using inverted index. The remained words in the description are transferred to the Lucene API to searching similar projects. Eventually, top-10 projects returned by Lucene would be used as the words matching similar projects.

Semantic Matching. The description of the input project is firstly transformed into a 150-dimensional document vector by the pre-trained Doc2Vec model. Then the cosine similarity between the document vector and each cluster center are calculated. Upon the nearest cluster is detected, top-10 nearest projects in the cluster measured by cosine similarity algorithm would be retrieved as ultimate similar projects of semantic matching.

Category Matching. This process is the same as that of the semantic matching searching. Firstly, the description of the input project is given to the pre-trained category predictor to generate a category distribution vector for the project. Then this vector is used to detect the nearest cluster and retrieve the top-10 nearest projects in that cluster with cosine similarity algorithm.

As shown in Figure 1, for each input project, the querying process would return three different lists of similar projects retrieved by different approaches. We would also manually evaluate the quality of different lists of projects. Since different approaches has different similarity aspects consideration and each gives some good feedbacks, we want to integrate their results by re-ranking the three lists of projects. Details are illustrated in next subsection.

4.3 Ranking

Ranking approach is to conduct a secondary sort on the three different tok-10 project lists returned by different searching strategies.

For each project in the evaluating set, its corresponding thirty similar projects are retrieved through three different matching approaches. Then, for each project, each of their similar projects is manually annotated a score. Eventually, the similar projects can be re-ranked according to the scores. The ranker to be trained should predict the ultimate order of projects as similar to manually ordering as possible. The input project description is called a *query* and each result is called a *document*. The task is to rank *documents* belonging to a same *query*.

In this paper, we adopt pair-wise ranking algorithm for re-ranking similar projects. It formulates ranking task as a two-class classification problem(Clariana and Wallace, 2009) and focuses on relative preference between two items of a same query. Let (x_i, y_i) and (x_j, y_j) be two result documents of a same query, where x is the feature vector of the result and y is the score of the result, then a new data instance (x', y')=($x_i - x_j$, $y_i - y_j$) would be generated for training the pair-wise ranker. The tool we use is SVM Rank[5]. The following commonly used features are adopted:

- TF-IDF vector of the *document*, i.e., descriptions of result projects.
- Latent Dirichlet Allocation (i.e., LDA) topic model vector of the *document*.
- Length of the *document*.
- The number of words that appear in both the *query* and the *document*.
- The one hot vector of unigram pairs in the (*query*, *document*) pairs.
- The string edit distance between the *query* and the *document*.
- The cosine similarity of doc2vecs and category vectors between the *query* and the *document*.
- The searching approach, 0, 1 and 2 for Words, Semantic and Category Matching respectively.

[5]Found here: http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html

All features are calculated based on the keywords extracted from each project's description. We conduct five-fold cross-validation for evaluating using the evaluating set.

5 EXPERIMENTS

In this section, we introduce the manual annotating process and evaluation metrics.

5.1 Annotation

The annotators for annotating similarity between software projects are second-year graduate students in software engineering. Each group of projects are annotated by five students. The average scores are adopted as the ultimate scores for each project. The annotators are asked to assign scores to the result projects according to the input project. Six different similarity aspects are considered in manually measuring the similarity. If two projects have no common properties among the six aspects, then the score is 0. If they match one more aspect, more scores they should be assigned. Descriptions of these aspects and an example is shown in Table 2. We set different scores for different aspects, i.e., Platform and Architecture are 1 score while the others are 2 scores. Through manual annotating, projects in different searching result lists of all projects in the evaluating set would get manual scores representing their similarities with the input projects.

5.2 Evaluation Metrics

For evaluating the classification performances of category predictors, we use the *precision*, *recall*, and *accuracy* values. Since the numbers of projects of different categories are imbalanced, the macro *precision*, *recall* and *F1* are also used in evaluation.

For comparing the performances of different approaches in searching similar software projects, we compare the average manually annotated scores of each project in the result list, i.e., *Average Score*. Besides, we merge result lists of different approaches

Table 2: Descriptions of scoring aspects and the annotating example.

Scoring Aspects	Description	Example		√/×
Platform	Products are used in the same platform, such as mobile phones, laptops, servers, clusters, cloud, etc.	This is a map internet web service based on a huge raster maps or satellite images for tracking and monitoring the mobile objects (cars etc) using GPS.	QLandkarte GT is the ultimate outdoor aficionado's tool. It supports GPS maps in GeoTiff format as well as Garmin's img vector map format. Additional it is the PC side front end to QLandkarte M, a moving map application for mobile devices.	√
Architecture	Client, Server, Client/Server, Browser/Server, etc.			×
Users	No specific user group, developers, financiers, designers, etc.			×
Domain	Financial, Words, Painting, Game, Education, etc.			√
Tools	Products are related the same third party tools, libraries or other type of resources.			√
Function	The products are to satisfy the same functional requirements.			×

and re-rank projects according to their manually annotated scores. Then the *Average Hit* of Top-10 projects in the ranked list of different approaches are calculated. The higher average score and average hit are, the better the searching approach is.

For evaluating the ranking approach, we adopt the commonly used evaluation metric Mean Average Precision (i.e., MAP) for algorithms that do information retrieval.

5.3 Answering Research Questions

RQ1. Which machine learning method works best for software type encoding in the searching scenario?

Results and Discussion: According to classification performances showing in Table 3, the full connected neural network using TF-IDF as input works best in transforming project description into category distribution vector. We also check the accuracy of top-k categories predicted by classifiers in Table 3. It proves that FNN with TF-IDF has better predicting capability than others taking top-k categories into account. The average manual annotated score shows the overall performance of each method. For further describing that the recommending performance of FNN is better than others, we plot the average score curves of Top-K recommended projects for each method. In Figure 2(a), we calculate the average score of k-th recommended project in the initial order. It shows that FNN has the highest scores for all k-th projects. In Figure 2(b), we firstly rank all recommended list of each method for each evaluating project according to the manual annotated scores, then calculate the average scores of the ranked k-th projects on all the evaluating projects and plot the score curves. Although CNN achieves higher average scores for top-5 projects, the last 5 projects derives much lower scores than FNN. The average scores of ranked list derived by FNN is also higher than the other two methods. All these details prove that FNN with TF-IDF as text features is the best simple method for encoding software project category distribution in the similar project retrieving scenario. In the following parts, FNN would be used as the representative of Category Matching approach.

RQ2. What are the performances of different searching methods based on words matching, pure semantic similarity and type semantic respectively?

Results and Discussion: According to average manual scores shown in Table 4, the category matching method achieves the best performance among all the three approaches. However, there are no big differences between the words matching and category matching. The average hits at position 1 to 3 of words matching is even much higher than twice of that of the category matching, implying the most similar projects could be derived by matching words. But matching category distribution does better in retrieving other similar projects. The average hits at other top-k ranges of category matching are higher than those of the other two methods. Note that there are many projects derived by different methods tied for tenth. It could be seen that category matching also derives the most number of tied tenth. The MAP values represent the consistencies between manual ordering and searching ordering. So, category matching can best represent people's point of view in sorting projects. We plot average scores and average hits at each top-k position of different approaches in Figure 3. It shows that words matching does the best in searching the most similar projects while category matching does best on average. Since each approach has some high quality feedbacks, using any one singly as ultimate searching strategy is not a good strategy. So we design the Ranking step.

RQ3. To what extent would the ranking strategy improves searching performance while combining the results of different searching methods?

Results and Discussion: Before ranking, there are six different strategies for combining result lists of the three searching approaches. All MAP values of these strategies are shown in the first six columns of Table

Table 3: Evaluating results of different Category Predictors.

Method	Macro			Accuracy					Average Score
	precision	recall	F1	Top-1	Top-2	Top-3	Top-4	Top-5	
SVM	0.475	0.230	0.445	0.475	0.593	0.642	0.685	0.716	4.246
FNN	**0.641**	**0.507**	**0.566**	**0.581**	**0.717**	**0.786**	**0.837**	**0.863**	**5.258**
CNN	0.553	0.317	0.403	0.428	0.602	0.687	0.743	0.797	4.996
LSTM	0.612	0.371	0.462	0.489	0.648	0.733	0.793	0.837	4.644

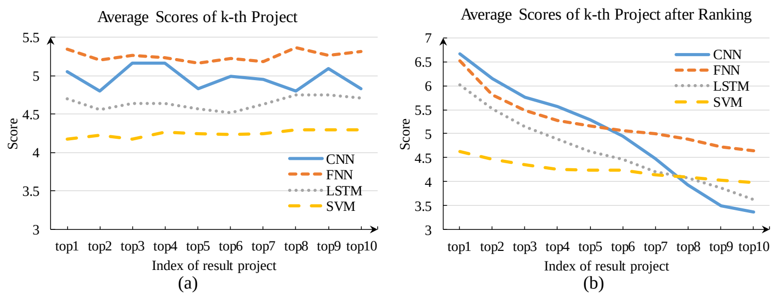

Figure 2: Average scores of k-th projects in the result list of different category predictors.

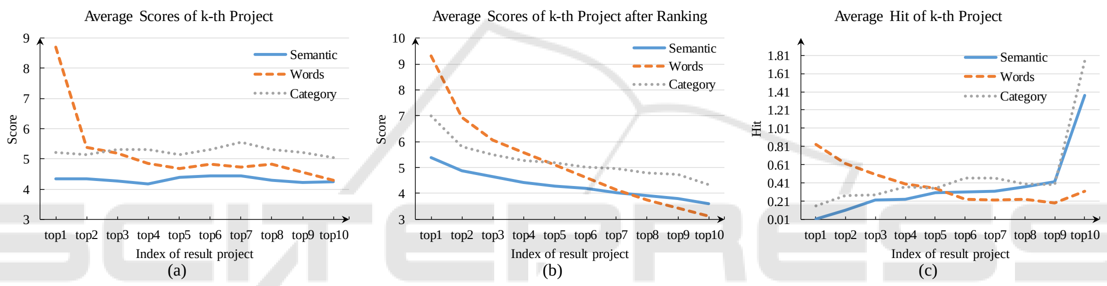

Figure 3: Average scores and average hits of different approaches at each Top-K position.

5. The last column of Table 5 is the MAP value of the result lists automatically ranked by the trained ranker. The other columns of After Ranking are MAP results derived by projecting projects of the auto-ranked list to the different approaches and then using different strategies to combine them. The automatically ranked project list of the searching result has slightly higher MAP value for top-10 projects than the lists derived from concatenating results of different searching approaches with different orders, for both before and after ranking. But for the top-20 and top-30 MAP values, the automatically ranked results does not as good as combining in the order of Category Matching, Words Matching and Semantic Matching. Comparing the first three columns of Before and After Ranking, we find that the ranker improves the project orders for both category and semantic matching, but not the words matching. Besides, Table 4 and Figure 3 tell that the quality of the first several results in words matching are much higher than that of the semantic matching. Combining these clues, the ranker must give some projects in the semantic matching result higher priorities than some in the word matching result which should be put forward. Two *CSW* columns and two *CWS* columns in Table 5 also tell that if semantic matching results put in front of the words matching results, the entire MAP value would be declined. So, for improving the ranker, some features that distinguish words matching results from semantic matching ones should be designed and added. The ultimat conclusion is that the best way to utilize the ranker is to sort results of different approaches at first and then concatenate them in the order of Category Matching, Words Matching, and Semantic Matching.

6 CONCLUSION

In this paper, we propose an integrated approach for retrieving similar projects from large-scale open-source software repository with project description as single input. For improving the traditional keywords matching performance, we adopt semantic matching and category distribution matching methods. For better integrating the results derived by different matching approaches, we adopt learn to rank algorithm to train a ranker for re-sorting the searching results. The

Table 4: Evaluation results of three different searching approaches.

Method	Average Score	Average Hits					MAP@10	
		@Top10	@1-3	@4-6	@7-9	@10	Initial	Gold
Words	5.205	3.89	**1.95**	0.98	0.64	0.32	0.2929	*0.389*
Semantic	4.312	3.66	0.34	0.84	1.11	1.37	0.291	*0.366*
Category	**5.258**	**4.89**	0.71	**1.18**	**1.25**	**1.75**	**0.404**	*0.489*

Table 5: MAP values of different searching result lists.

MAP @Top	Before Ranking						After Ranking						
	WCS	CWS	SCW	CSW	SWC	WSC	WCS	CWS	SCW	CSW	SWC	WSC	Auto
10	0.2929	***0.404***	0.291	0.404	0.291	0.2929	0.2926↓	0.406↑	0.303↑	0.406	0.303	0.292	**0.407**
20	0.273	***0.303***	0.235	0.273	0.227	0.233	0.272	**0.304**	0.241	0.275	0.233	0.234	0.301
30	0.226	***0.247***	0.213	0.239	0.212	0.216	0.227	**0.248**	0.217	0.240	0.216	0.217	0.247

experimental results show that the ranker derives a slight improvement in MAP value at top-10 projects.

Design effective text features in training rankers , analyze concrete effectiveness of different features, find the way to omit the impact of imbalanced projects distribution and train better category classifiersare, and taking user requirements into consideration for measuring project similarity are the future works of this paper.

ACKNOWLEDGEMENTS

This work was supported by the National Natural Sciences Foundation, China (No. 61802167), Open Foundation of State key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) (SKLNST-2019-2-15), and the Fundamental Research Funds for the Central Universities. Jidong Ge is the corresponding author.

REFERENCES

Clariana, R. B. and Wallace, P. (2009). A comparison of pair-wise, list-wise, and clustering approaches for eliciting structural knowledge. *International Journal of Instructional Media*, 36(3):287–302.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Epelbaum, T. (2017). Deep learning: Technical introduction. *arXiv preprint arXiv:1709.01412*.

Gomaa, W. H. and Fahmy, A. A. (2013). A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Kawamitsu, N., Ishio, T., Kanda, T., Kula, R. G., De Roover, C., and Inoue, K. (2014). Identifying source code reuse across repositories using lcs-based source code similarity. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, pages 305–314. IEEE.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, Y., McLean, D., Bandar, Z. A., O'shea, J. D., and Crockett, K. (2006). Sentence similarity based on semantic nets and corpus statistics. *IEEE transactions on knowledge and data engineering*, 18(8):1138–1150.

Liu, C., Jin, T., Hoi, S. C., Zhao, P., and Sun, J. (2017). Collaborative topic regression for online recommender systems: an online and bayesian approach. *Machine Learning*, 106(5):651–670.

Luo, L., Ming, J., Wu, D., Liu, P., and Zhu, S. (2017). Semantics-based obfuscation-resilient binary code similarity comparison with applications to software and algorithm plagiarism detection. *IEEE Transactions on Software Engineering*, 43(12):1157–1177.

MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Nagwani, N. K. (2015). A comment on "a similarity measure for text classification and clustering". *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2589–2590.

Wang, S. and Wu, D. (2017). In-memory fuzzing for binary code similarity analysis. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pages 319–330. IEEE Press.

Yao, L., Pan, Z., and Ning, H. (2018). Unlabeled short text similarity with lstm encoder. *IEEE Access*, 7:3430–3437.