FunkR-pDAE: Personalized Project Recommendation Using Deep Learning

Lolla Bhavani
PG scholar, Department of MCA, CDNR collage, Bhimavaram, Andhra Pradesh.
K.Venkatesh
(Assistant Professor), Master of Computer Applications, DNR collage, Bhimavaram, Andhra Pradesh.

Abstract

Open source communities are the largest coding platforms that offer software developers an opportunity to engage in social activities relating to the software development. The developers can store and share their codes or projects with wider community of people using the repositories. Enabling its users to find relevant projects is the prime feature of the open source communities. Finding a relevant project among vast open source projects is a difficult task. Recommending a suitable project for developers can save their time and energy. Pengcheng Zhang et.al. Used FunkR-pDAE, a personalized project recommendation approach based on Deep Auto-Encoders, deep learning model. This approach has clear limitations, such as ignoring the dataset description and source code comment documents.

So, proposing a method for recommending a project in open source communities that takes into account the dataset description and source code comment documents, allowing for the discovery of more relationships between developers and projects this might in turn lead to increased precision rate. And also in the proposed work it is planned to generalize the model to recommend other open source communities i.e., github, kaggle and source forge along with integrating the dataset description and source code comment documents in order to mine more relations among the developers and projects.

Keywords - Open Source; project recommendation; deep auto-encoder; GitHub.

Introduction

Open Source Software (OSS) development paradigm is defined as collaborative work between multiple developers. Through online development, geographically dispersed developers can work together to maintain or improve OSS projects. Through this development paradigm, the same

software project includes participants from different areas of the world.

OOS also allows different developers to modify and improve source codes according to their own needs. In addition, unlike traditional organizations, OOS has a composite structure of social-technical processes and does not need to have a specific organization [9].

It usually has a lively software process, constantly changing requirements and fast development phases. For traditional software development, if the software project team is large, the project progress is often very slow. OOS, on the other hand, is developed in an environment where developers can communicate and share code via the Internet rather than in the same geographical place.

Developers gather on social-coding sites in the same virtual environment, such as GitHub and Source forge, for mutual assistance and sharing of experiences. Developers can manipulate the codes through watch, fork, pull-request and so on.

Meanwhile, they can also establish their own social network by following other developers to achieve socialization and transparent programming [2]. According to a report released by GitHub in 2017, GitHub has more than 24 million users, hosts more than 67 million software and combined over one billion code requests.

Due to the rich resources on GitHub, developers and projects are not distributed evenly. It is very hard for developers to find the suitable project for themselves from 67 million code repositories while most of these projects are hosted on GitHub with only a handful of developers involved.

Consequently, personalized project recommendation for GitHub is urgently needed. Project recommendation is a kind of Recommendation Systems in Software Engineering (RSSE) [19]. It lies primarily in the models used, and often relies on the data mining and the predictive nature of its functionality.

In addition to these obvious features, RSSE often differs from other areas. The traditional recommendation system is user-oriented. Users usually create data items directly, for example, in the form of ratings. An important challenge with the traditional recommendation system is to infer and simulate user preferences and needs.

Instead, the main challenge in designing RSSE is to automatically interpret the highly technical data stored in the repository. Recently, researchers have already proposed several novel recommendation approaches for GitHub application [15], [27], [30]. However, existing approaches have some limitations.

Literature Survey

[1] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in Unsupervised and Transfer Learning - Workshop held at ICML 2011, Bellevue, WA, USA, 2012, pp. 37–50.

Auto encoders play a fundamental role in unsupervised learning and in deep architectures for transfer learning and other tasks. In spite of their fundamental role, only linear autoencoders over the real numbers have been solved analytically. Here we present a general mathematical framework for the study of both linear and non-linear autoencoders.

The framework allows one to derive an analytical treatment for the most non-linear autoencoder, the Boolean autoencoder. Learning in the Boolean autoencoder is equivalent to a clustering problem that can be solved in polynomial time when the number of clusters is small and becomes NP complete when the number of clusters is large.

The framework sheds light on the different kinds of autoencoders, their learning complexity, their horizontal and vertical compos ability in deep architectures, their critical points, and their fundamental connections to clustering, Hebbian learning, and information theory. Autoencoders are simple learning circuits which aim to transform inputs into outputs with the least possible amount of distortion.

While conceptually simple, they play an important role in machine learning. Autoencoders were first introduced in the 1980s by Hinton and the PDP group (Rumelhart et al., 1986) to address the problem of "back propagation without a teacher", by using the input data as the teacher.

[2] A. Begel, J. Bosch, and M. D. Storey, "Social networking meets software development: Perspectives from github, msdn, stack exchange, and top coder," IEEE Software, vol. 30, no. 1, pp. 52–66, 2013.

One form of crowdsourcing is the competition, which poses an open call for competing solutions. Commercial systems such as Top Coder have begun to explore the application of competitions to software development, but have important limitations diminishing the potential benefits drawn from the crowd. In particular, they employ a model of independent work that ignores the opportunity for designs to arise from the ideas of multiple designers.

In this paper, we examine the potential for software design competitions to incorporate recombination, in which competing designers are given the designs of others and encouraged to use them to revise their own designs. To explore this, we conducted two software design competitions in which participants were asked to produce both an initial and a revised design, drawing on lessons learned from the crowd.

[3] T. Chang, W. Hsiao, and W. Chang, "An ordinal regression model with SVD hebbian learning for collaborative recommendation," J. Inf. Sci. Eng., vol. 30, no. 2, pp. 387–401, 2014.

The Internet is disseminating more and more information as it continues to grow. This large amount of information, however, can cause an information overload problem for users. Recommender systems to help predict user preferences for new information can ease users' mental loads. The model-based collaborative filtering (CF) approach and its variants for recommender systems have recently received considerable attention.

Nonetheless, two issues should be carefully considered in practical applications. First, the data reliability of the rating matrix can affect the prediction performance. Second, most current models view the measurement scale of output classes as nominal instead of ordinal ratings. This study proposes a model-based CF approach that deals with both issues.

[4] X. Chen and J. C. Grundy, "Improving automated documentation to code traceability by combining retrieval techniques," in 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, 2011, pp. 223–232.

Documentation written in natural language and source code are two of the major artifacts of a software system. Tracking a variety of traceability links between software documentation and source code assists software developers in comprehension, efficient development, and effective management of a system.

Automated traceability systems to date have been faced with a major open research challenge: how to extract these links with both high precision and high recall. In this paper we introduce an approach that combines three supporting techniques, Regular Expression, Key Phrases, and Clustering, with a Vector Space Model (VSM) to improve the performance of automated traceability between documents and source code.

This combination approach takes advantage of strengths of the three techniques to ameliorate limitations of VSM. Four case studies have been used to evaluate our combined technique approach. Experimental results indicate that our approach improves the performance of VSM, increases the precision of retrieved links, and recovers more true links than VSM alone.

Proposed method

To implement this project, we have designed following modules

1) **Upload Open-Source Dataset:** using this module we will upload dataset to application

2) **Pre-process Code & Comments:** using this module we will read all comments and values and then build a training vector and, in this module, we will replace all missing values with 0. Using this module, we will split dataset into train and test where application used 80% dataset size for training and 20% for testing

3) **Train KNN Algorithm:** processed dataset will be input to KNN algorithm to train a model and then model will be applied on NEW test data to predict SOURCE CODE REPOSITORY name. 20% test data will be applied on trained model and then calculate accuracy of correct prediction

4) **Comparison Graph:** using this module we will plot accuracy, precision and recall graph

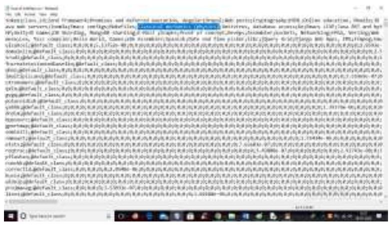
5) **Predict Open-Source Project:** using this module user can enter any source code related query and then KNN will predict GITHUB REPOSITORY name of that query.

Result

Now-a-days lots of developers are making many open source software project development and this developed projects modules can be useful in some other projects. So we can reuse such modules in new project without development from scratch. Many existing open source project communities exists but they won't consider SOURCE CODE or dataset comments and this may not give accurate search result which may decrease prediction accuracy.

To overcome from this problem we are using source code and dataset comments to train Machine Learning algorithm called KNN and after training this model can be used to predict repository for given search query.

To implement this project we have use Open Source repository dataset from GITHUB which contains more than 16000 repositories with comments and below screen showing dataset details



In above dataset screen first 5 lines contains WORDS from source acode and dataset COMMENTS and in all rows first column contains names of GITHUB user repository and other column contains values as 0 and greater than 0. If repository contains comment words then value will be >0 else 0.

You can see dataset details from below link

https://data.world/vmarkovtsev/github-source-code-names/workspace/file?filename=artm.csv

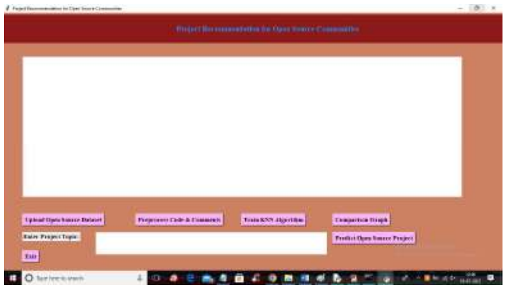
By using above dataset we will train KNN algorithm and then user can search such repository by entering any query.

To implement this project we have designed following modules

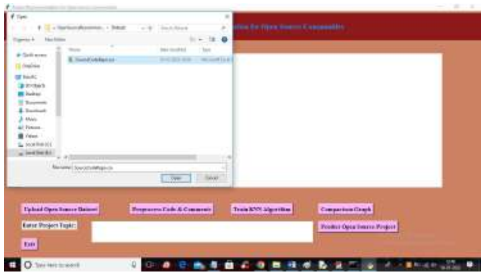
1) Upload Open Source Dataset: using this module we will upload dataset to application
2) Pre-process Code & Comments: using this module we will read all comments and values and then build a training vector and in this module we will replace all missing values with 0. Using this module we will split dataset into train and test where application used 80% dataset size for training and 20% for testing
3) Train KNN Algorithm: processed dataset will be input to KNN algorithm to train a model and then model will be applied on NEW test data to predict SOURCE CODE REPOSITORY name. 20% test data will be applied on trained model and then calculate accuracy of correct prediction
4) Comparison Graph: using this module we will plot accuracy, precision and recall graph
5) Predict Open Source Project: using this module user can enter any source code related query and then KNN will predict GITHUB REPOSITORY name of that query.

SCREEN SHOTS

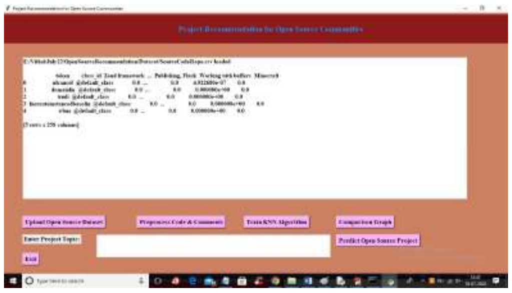
To run project double click on 'run.bat' file to get below screen


Upload Open Source Dataset
Preprocess Code & Comments
Train KNN Algorithm
Comparison Graph
Enter Project Topic:
Predict Open Source Project
Exit

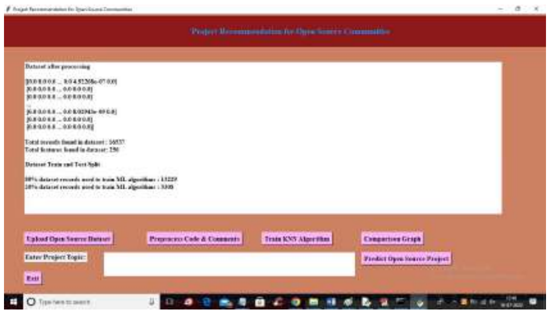
In above screen click on 'Upload Open Source Dataset' button to upload dataset and get below screen



In above screen selecting and uploading 'source code repository' file and then click on 'Open' button to load dataset and get below screen

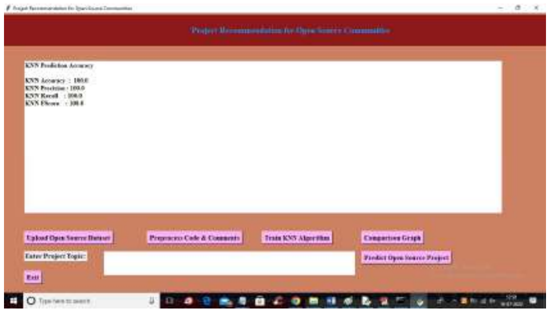


In above screen dataset loaded and we can see dataset contains both numeric and non-numeric data but machine learning algorithms only take integer values so click on 'Pre-process Code & Comments' button to replace missing values and then convert dataset into numeric vector and get below output


Project Recommendation for Open Source Communities
Upload Open Source Dataset
Preprocess Code & Comments
Train KNN Algorithm
Comparison Graph
Enter Project Topic:
Predict Open Source Project
Exit

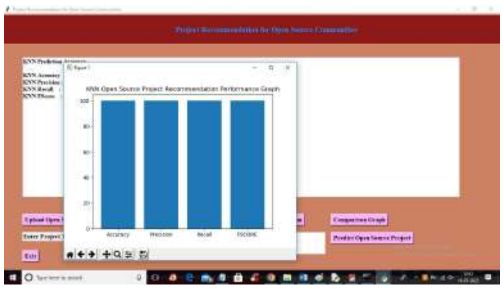
```
Dataset after processing

[[0.0 0.0 0.0 ... 0.0 4.51268e-07 0.0]
 [0.0 0.0 0.0 ... 0.0 0.0 0.0]
 [0.0 0.0 0.0 ... 0.0 0.0 0.0]
 ...
 [0.0 0.0 0.0 ... 0.0 8.02943e-09 0.0]
 [0.0 0.0 0.0 ... 0.0 0.0 0.0]
 [0.0 0.0 0.0 ... 0.0 0.0 0.0]]

Total records found in dataset : 16537
Total features found in dataset: 256

Dataset Train and Test Split

80% dataset records used to train ML algorithms : 13229
20% dataset records used to train ML algorithms : 3308
```

In above screen we can see entire dataset converted to numeric vector and we can see dataset contains 16537 repository and comments size is 256 and then application split dataset into train and test where 13229 records used for training and 3308 records used for testing and then will get prediction accuracy of test data.

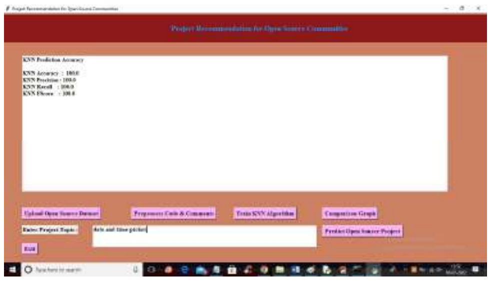

Project Recommendation for Open Source Communities
Upload Open Source Dataset
Preprocess Code & Comments
Train KNN Algorithm
Comparison Graph
Enter Project Topic:
Predict Open Source Project
Exit

```
KNN Prediction Accuracy

KNN Accuracy  : 100.0
KNN Precision : 100.0
KNN Recall    : 100.0
KNN FScore    : 100.0
```

In above screen with KNN we got all metrics values as 100% and now model is ready and now click on 'Comparison Graph' button to get below graph

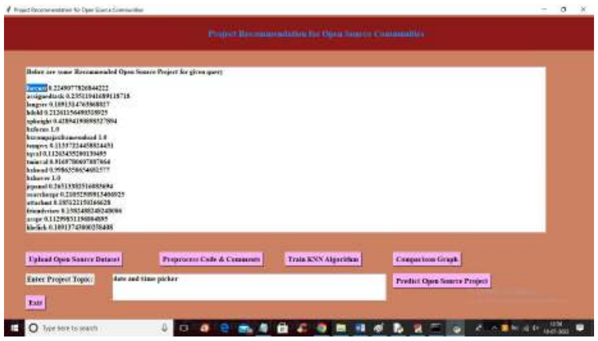


In above graph x-axis represents metrics as accuracy, precision etc and y-axis represents metric values and now close above graph and then enter any query and then press 'Predict Open Source Project' button to get below output


Project Recommendation for Open Source Communities
Upload Open Source Dataset
Preprocess Code & Comments
Train KNN Algorithm
Comparison Graph
Enter Project Topic:
date and time picker
Predict Open Source Project
Exit

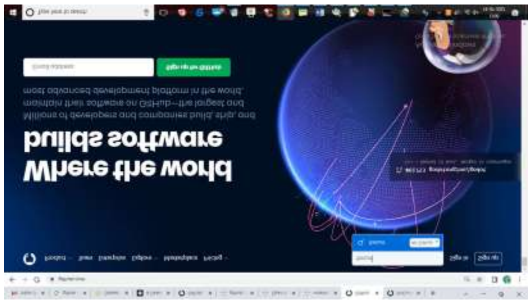
```
KNN Prediction Accuracy

KNN Accuracy  : 100.0
KNN Precision : 100.0
KNN Recall    : 100.0
KNN FScore    : 100.0
```

In above screen in small TEXT FIELD I entered query 'date and time picker' and then press button to get all those repository which implements date and time picker module

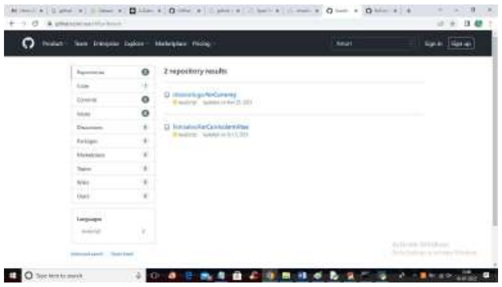

Project Recommendation for Open Source Communities
Upload Open Source Dataset
Preprocess Code & Comments
Train KNN Algorithm
Comparison Graph
Enter Project Topic:
date and time picker
Predict Open Source Project
Exit

```
Below are some Recommended Open Source Project for given query

forcurr 0.2249977826844222
assignedtask 0.23511941689118718
longrev 0.1891314765868817
hdold 0.21261156490518925
sphheight 0.42894190898527894
hxfocus 1.0
hxcompareprojectnameasked 1.0
tempvcs 0.11537224458824451
spval 0.11263435200139485
tmincal 0.9169780697887864
hxhead 0.9986358654681577
hxhover 1.0
jqpanel 0.2651398251685694
searchurge 0.21052509813408925
attachmt 0.195122159169628
friendsview 0.1592489248249086
scope 0.11299851190884895
klnclick 0.18913743000258488
```

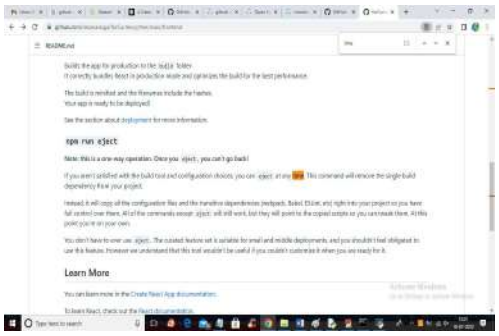
In above screen display list of repository user and the matching similarity for example 'forcurr' is the repository name and 0.22 is the matching score and now in GITHUB we search for that repository and then check weather its description contains any 'time or date details'. In below screen I am searching for repository in GITHUB



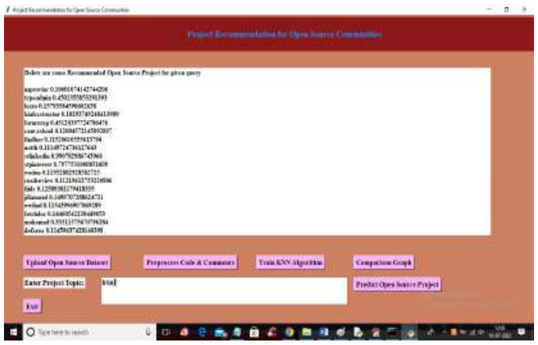
In above screen in TOP left small text field I entered repository name as 'forcurr' and below is the search output



In above screen click on first link called ‘for Currency’ to get below screen



In above screen in comments or description we can see word ‘time’ found. Similarly you can search any comment given in dataset first row to repository names which contains implementation code


Project Recommendation for Open Source Communities
Below are some Recommended Open Source Project for given query
Upload Open Source Dataset
Preprocess Code & Comments
Train KNN Algorithm
Comparison Graph
Enter Project Topic:
html
Predict Open Source Project
Exit

In above screen I entered query as ‘html’ and get all repositories which done HTML projects.

Similarly user can enter any query and then get repository code name as recommendation.

Note: Entered search word can be found in any source file of Repository as comments or description

Conclusion:

The paper presents FunkR-pDAE, a personalized project recommendation approach based on an existing deep learning model: Deep Auto-Encoders. The experimental results show that FunkR-pDAE achieves the desired recommendation effect compare with other approaches. For future work, we plan to study the dataset description and source code comment documents, which can mine more relations among developers and projects. Furthermore, we plan to generalize our model to recommend other open source communities.

REFERENCES

[1] P. Baldi, “Autoencoders, unsupervised learning, and deep architectures,” in Unsupervised and Transfer Learning - Workshop held at ICML 2011, Bellevue, WA, USA, 2012, pp. 37–50.

[2] A. Begel, J. Bosch, and M. D. Storey, “Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder,” IEEE Software, vol. 30, no. 1, pp. 52–66, 2013.

[3] T. Chang, W. Hsiao, and W. Chang, “An ordinal regression model with SVD hebbian learning for collaborative recommendation,” J. Inf. Sci. Eng., vol. 30, no. 2, pp. 387–401, 2014.

[4] X. Chen and J. C. Grundy, “Improving automated documentation to code traceability by combining retrieval techniques,” in 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, 2011, pp. 223–232.

[5] H. Dai, Y. Wang, R. Trivedi, and L. Song, “Recurrent Convolutionary latent feature processes for continuous-time recommendation,” in Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, Boston, MA, USA, 2016, pp. 29–34.

[6] X. Dong, L. Yu, Z. Wu, Y. Sun, L. Yuan, and F. Zhang, “A hybrid collaborative filtering model with deep structure for recommender systems,” in Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, CA, USA., 2017, pp. 1309–1315.

[7] G. Gousios, “The ghtorent dataset and tool suite,” in Proceedings of the 10th Working Conference on Mining Software Repositories, MSR ’13, San Francisco, CA, USA, 2013, pp. 233–236.

[8] G. E. Hinton, “Training products of experts by minimizing contrastive divergence,” Neural Computation, vol. 14, no. 8, pp. 1771– 1800, 2002.

[9] Y. Hu, S. Wang, Y. Ren, and K. R. Choo, “User influence analysis for github developer social networks,” Expert Syst. Appl., vol. 108, pp. 108–118, 2018.

[10] O. Irsoy and E. Alpaydin, “Unsupervised feature extraction with autoencoder trees,” Neurocomputing, vol. 258, pp. 63–73, 2017.

[11] G. Jeong, S. Kim, and T. Zimmermann, “Improving bug triage with bug tossing graphs,” in Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, Amsterdam, The Netherlands, 2009, pp. 111–120.

[12] Y. Kawamoto, H. Takagi, H. Nishiyama, and N. Kato, “Efficient resource allocation utilizing q-learning in multiple ua communications,” IEEE Transactions on Network Science and Engineering, 2018. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TNSE.2018.2842246

[13] O. Kuchaiev and B. Ginsburg, “Training deep autoencoders for collaborative filtering,” CoRR, vol. abs/1708.01715, 2017. [Online]. Available: http://arxiv.org/abs/1708.01715

[14] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, “Routing or computing? the paradigm shift towards intelligent computer network packet transmission based on deep learning,” IEEE Trans. Computers, vol. 66, no. 11, pp. 1946–1960, 2017.

[15] T. Matek and S. T. Zebec, “Github open source project recommendation system,” CoRR, vol. abs/1602.02594, 2016.

[16] A. T. Nguyen, M. Hilton, M. Codoban, H. A. Nguyen, L. Mast, E. Rademacher, T. N. Nguyen, and D. Dig, “API code recommendation using statistical learning from fine-grained changes,” in Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Seattle, WA, USA, 2016, pp. 511–522.

[17] D. Ramachandram and G. W. Taylor, “Deep multimodal learning: A survey on recent advances and trends,” IEEE Signal Process. Mag., vol. 34, no. 6, pp. 96–108, 2017.

[18] P. C. Rigby and M. P. Robillard, “Discovering essential code elements in informal documentation,” in 35th International Conference on Software Engineering, ICSE ’13, San Francisco, CA, USA, 2013, pp. 832–841.

[19] M. P. Robillard, R. J. Walker, and T. Zimmermann, “Recommendation systems for software engineering,” IEEE Software, vol. 27, no. 4, pp. 80–86, 2010.

[20] R. Salakhutdinov, A. Mnih, and G. E. Hinton, “Restricted boltzmann machines for collaborative filtering,” in Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, 2007, pp. 791–798.