ICIC Express Letters
Volume **15**, Number **5**, May **2021**

 ISSN 1881-803X
pp. **493–500**

FE2VEC: PREDICTION OF SOFTWARE CHANGE PROPAGATION USING A DEEP LEARNING MODEL

Ahmed Hamdi Abdurhman and Jihwan Lee*

Department of Industrial and Data Engineering
Pukyong National University
45, Yongso-ro, Nam-gu, Busan 48513, Korea
*Corresponding author: jihwan@pknu.ac.kr

Received October 2020; accepted December 2020

Abstract. *One of the difficulties that developers might face while collaborating on a project is overloading numbers of files and its dependability on each other specially in a big project. A team that works on the project also might not work in the same environment. Furthermore, programmers who specialized in one part of the project might have their gigs to contribute to the project. Hence, most of the change's records contain information on who has made a change to the file, and the number of involved people who work on the project might get bigger and change as time goes on. This article proposes a recommendation system that assists the developer in recommending based on similar filename embedding, Fe2vec (Filename extension to vector). A detailed technique for learning how a similar text has similar encoding for learning Fe2vec from software changelog data explained. A case study on Vuze-Azureus: an open-source Java-based software development changelog dataset extracted from a version control repository was conducted. We validate our approach by analyzing the software development changelog historical data of over ten years. The result shows that the model predicts the similarity score (0.97) of the filename with an extension similar to the requested filename format.*
Keywords: Deep learning, Fe2vec, Software change propagation, Changelog, Word2vec

1. **Introduction.** Change propagation is a practice to explore how changes made to one of the application versions are migrated to other living versions of the application. [1] examined that the importance of understanding how and why changes propagate during engineering design is critical. The authors stated that most products and systems developed from predecessors and not through clean-sheet design.

Design made at predecessors is at least connected to one other part. For example, if some part of 'A' changed, then part 'B' might need changing. A change occurring at part 'B' might lead to the next part, which might be part 'C' or any list of a chain in the interconnected parts of the given fullset of files. [2] stated predicting and management of change to an existing product resulting from faults or new requirements. The authors advocated that identifying the need for change early in the design process alleviates any additional cost incurred [3], quality problems, and schedule delay.

Identifying and tracking the designed coupling between the system parts is crucial for predicting change propagation [4]. The authors emphasized that to identify the relevant part of a given system element, an analyzer may use a pre-defined parametric relationship, but it is virtually impossible to establish a full set of relationships.

The past few decades have witnessed the marvelous success of deep learning in many application domains such as sequence and time-series [5,6], computer vision [7,8], metal object classification [9], detection [10-12] and many more [13]. Industry practitioners and academic researchers have been in a race to apply this modern neural network in a broader range of applications due to their ability to solve many complicated tasks [14].

DOI: 10.24507/icicel.15.05.493

"Deep learning becomes a general-purpose solution for nearly all learning problems [15]". The field of deep learning in the recommender system is booming. With the growing volume of information, the recommender system is an effective strategy to overcome information overloading. Recommendation lists are generated based on item features, user preferences, user/item past interactions. Here for item, it could be anything.

The recommendation system could be divided into two broad categories: content-based or collaborative. In a collaborative filter algorithm, mainly depend on historical data. For example, if developer X was working on files a.java, b.java, c.java, and developer Y was working on files b.java, c.java, d.java, both developers have similar working knowledge interest in contributing, so X should work on file d.java and Y work on a.java.

Data may gather from different sources to further expand existing analytic; one promising approach is to leverage historical change records, which are available from the modern management system such as Concurrent Version System (CVS) or Product Data Management System (PDMS). As shown in Figure 1, the CVS repository's change record includes the set of system components that frequently changed together in the past. So, extract these co-change patterns from the CVS to predict future changes.

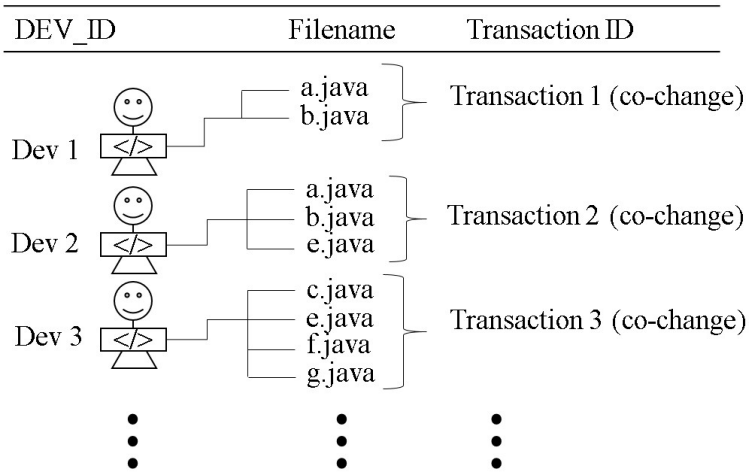

FIGURE 1. An example of historical change records

So far, many scholars have implemented change propagation using different techniques – for example, a probabilistic directed acyclic graphical model, mostly known as Bayesian Networks (BN). BN-based models predict change propagation with probabilistic measures. One of BN's advantages is to provide a quantitative measure to reflect the probability that an element will change. As a limitation of BN, it is computationally expensive. [16] used Bayesian belief network as a probabilistic tool to predict the possible affected system models, given a change in the system. A systematic framework for modeling change propagation using Dependency Network (DN) [4] claims that DN provides a straightforward and computationally efficient algorithm that can overcome BN's limitation in modeling dependency between system elements.

[17] used a heuristics approach to predict a change in one source code entity propagate to other entities. Develop a stochastic model known as K3B, which predicts how far a set of change propagates throughout the system [18]. Software stability has become one of the most critical attributes that affect maintenance costs. [3] proposed a Class Coupling Network (CCN) to model software structure at the class level to evaluate software stability. The authors validated software stability metrics theoretically using Weyuker's properties and empirically using a set of open-source Java software systems.

Deep learning in the recommendation system is significantly improving over the conventional system for recommendation systems. Some notable recent research application areas such as learning item embedding, deep collaborative filtering, feature extraction

from content, and session-based recommendation with RNN [19] give us hope to further dive into this field.

A two-layer neural network that processes text by 'vectorizing' known as word2vec [20] gains much traction and provides state-of-the-art word embedding. While word2vec is not a deep neural network, it turns text into a numerical form that deep neural networks can understand.

Understanding the value of software development tools in assisting developers by recommending an entity where a particular change will likely propagate during software development and maintenance is profoundly required. So, considering this, we propose a systematic approach for modeling changing propagation using deep learning tools as a recommendation system. As a case study, we used the Vuze-Azureus [21] software development changelog dataset, an open-source BitTorrent client-and-tracker, developed in Java.

This paper is organized as follows. Section 2 describes the steps of the proposed approach in detail. The experimental result and validation method are discussed in Section 3. The conclusion and future work is to be found in Section 4.

2. Proposed Approach.

This section will discuss some essential steps of the proposed approach to two main subsections: first, we will extract information and then make our data ready for the proposed approach model.

2.1. Extracting changelogs.

In this sub-section, we first start extracting the required information out of software repository changelogs. To get meaning out of software history changelog and use it for the proposed system approach, we extract them in the form of event batch or co-changes simultaneously. We use this data as a primary source of input. If the file was committed together in the past, we consider them; they will be most likely to change together again.

The data we have at hand might not explain what is going on in the actual file(s), but we have known that the numbers of committed files simultaneously may indicate the total numbers of files affected. We consider all committed changes as our valuable information to train the proposed model.

As illustrated in Figure 2, in our case, the first step toward the proposed approach started with cleaning data. They were then extracting filename from all file paths. Furthermore, we then assigned a unique number to represent the extracted filename. Almost all the data is categorical, so we added extra columns to map the categorical data to numerical as depicted in Table 1.

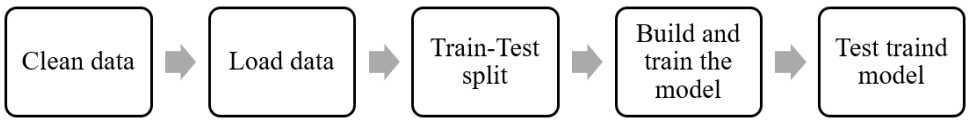

FIGURE 2. Overall implementation process

Load data: at this step, turn each sequence of the filename that co-changed simultaneously as a sentence into a list of integer filename indexes and return to index mapping to test the model.

2.2. Applying Fe2vec model on non-textual data.

During the development stage or maintenance, change occurs in a single file that might propagate into other related elements. If we can present each filename by a vector, then we can easily find similar filenames. If a developer commits a file, we can easily extract the filename associated with the file, as shown in Figure 3, and recommend similar files using the vector similarity score between the filenames. To get a vector representation of files, we applied the Fe2vec

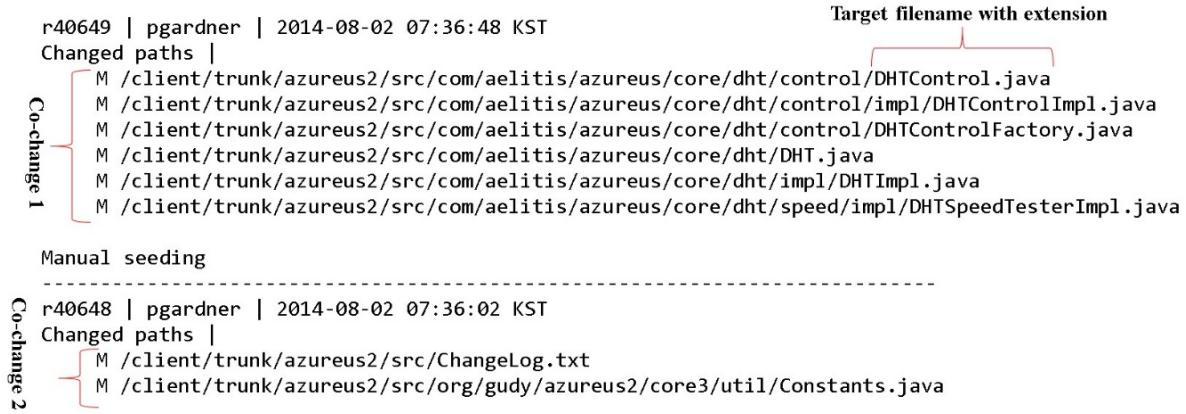

FIGURE 3. Vuze-Azureus raw historical development changelog

(Filename extension to vector) model, software changelog of a developer as a sentence, and the filenames as its words.

Fe2vec (Filename extension to vector), originally known as word2vec [20], is adopted because it allows grouping the vectors of similar words together in vector space. Without human intervention, Fe2vec creates vectors that are distributed, numerical representations of the filename with extension features. Word2vec can make highly accurate guesses about a word's meaning based on past appearances. Those guesses can be used to establish a filename's association with other filenames or cluster documents and classify them accordingly.

As depicted in Figure 3, each filename in a vector trains alongside other that neighbor them in the input corpus. It can utilize either of two models: Skip-gram model (predict surrounding context words given a center word) or continuous word of bag (using context to predict a target word).

In this paper, we used Skip-gram model architecture, as depicted in Figure 4. Instead of a word, we used *names* associated with *files*, for example, [*filename.extension*] and sequence of commit co-change simultaneously as a sentence. Once we have a developer software co-changelog history as a sequence, we can easily apply a Fe2vec algorithm to

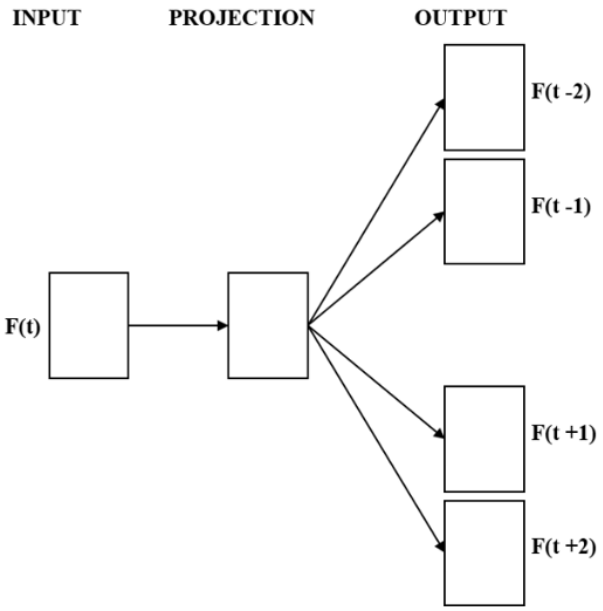

FIGURE 4. Skip-gram model architecture [22]

creating vectors for these change commits files and then compare them to calculate a similarity score. More formally, this model's main objective is to maximize the average log probability given sequence of the training filename $f_1, f_2, f_3, \ldots, f_N$.

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{-c\leq i\leq c,\, i\neq 0}\log(f_{n+i}|f_t) \tag{1}$$

where c is the size of the training context, the Skip-gram model, for example, if input target filename $F(t)$ is given to the model, there will be four targets context filenames, i.e., our window size is ($\max$ *context location* $C = 2$) (*context window size* $K = 4$), four activation function, four predictions, and four errors summed to get total error. There, the network is going to tell us the probability for every filename in our vocabulary V of being the nearby filename that we chose window size. The vocabulary we were mentioning here is built from filename. For example, if we have 2,000 unique filenames and use "DHC.java" as a one-hot encoding vector, the vector will have 2,000 components (one for every filename in our vocabulary) and place "1" in position corresponding to the filename "DHC.java" which is unique spot and "0s" in all of the other position.

The dot product at the hidden layer (no activation function at this point) passed to the output layer. The output layer computes the dot product between the hidden layer's output vectors and weighs the output layer matrix. Then, the Softmax function computes the probability of filename appearing to be in the context of $F(t)$ at a given context location, which helps us distribute the probability throughout each output node, i.e., means that because we want to convert out the last layer output in terms of probability and the softmax function can easily convert values into probability values.

Let us represent two sets of vectors that give the probability of context filename U_f when f is the context filename and V_f when f center word.

$$p(f_o|f_c) = \frac{\exp\left(u_o^T v_c\right)}{\sum_{f=1}^{F}\exp\left(u_f^T v_c\right)} \tag{2}$$

where o = center filename and c = context filename. The softmax shown in Equation (2), the dot product in the numerator between o and c, allows us to capture the similarity between the two vectors, as mentioned earlier. The higher the similarity, the higher the probability will be. Simultaneously, the denominator provides us a way to normalize the value of probability over the entire vocabulary so that it all adds up to 1.

Skip-gram along with "Negative sampling" yielded better results and reduced the compute burden of the training process. For more details about Skip-gram, please refer to [22].

3. **Experiments.**

This section describes the dataset, system environment used for conducting our experimental, evaluation of the proposed approach, and presents results.

3.1. **System environment and data knowledge.**

The experiment was conducted using a desktop computer with an Intel®Core™i7-8700T CPU, 16.0 GB installed memory (RAM). Python Version (V. 3.7.6), Pandas V. 1.0.1, Plot V. 0.6.5, Umap V. 0.1.1, Numpy V. 1.18.5, Gensim V. 3.8.3, Keras V. 2.4.3, and Tensorflow 2.3.0.

We conducted our experiment on Vuze-Azureus [21] dataset that includes a range index 5883 cases for software development changelogs and 40 uniquely identified developer ID. Almost all of the data are categorical. During our cleaning phase, some of the data was converted into numerical data types. Table 1 provides a small portion of the data with both numerical and categorical inline in the same row. The dataset contains software changelogs over the last 10 years that spans between 2003-2014.

The raw data look like Figure 3, and after extracting and conversion look like Table 1. For a specific step, we used MS excel to structure the dataset and extract the filename from the changelog path. The authors develop the rest of the required parts.

TABLE 1. Processed first 10 rows of Vuze-Azureus software changelogs

	EVENT_ID	EVENT_ID_	DEV_ID	DEV_ID_	OPERATION	OPERATION_ID_	FILE_NAME	FILE_ID_	TIME_STAMP
0	r40649	12538	pgardner	27	M	2	DHTControl.java	838	2014-08-02 07:36:48 KST
1	r40649	12538	pgardner	27	M	2	DHTControlImpl.java	844	2014-08-02 07:36:48 KST
2	r40649	12538	pgardner	27	M	2	DHTControlFactory.java	843	2014-08-02 07:36:48 KST
3	r40649	12538	pgardner	27	M	2	DHT.java	837	2014-08-02 07:36:48 KST
4	r40649	12538	pgardner	27	M	2	DHTImpl.java	859	2014-08-02 07:36:48 KST
5	r40649	12538	pgardner	27	M	2	DHTSpeedTesterImpl.java	909	2014-08-02 07:36:48 KST
6	r40648	12537	pgardner	27	M	2	ChangeLog.txt	520	2014-08-02 07:36:02 KST
7	r40648	12537	pgardner	27	M	2	Constants.java	718	2014-08-02 07:36:02 KST
8	r40645	12536	pgardner	27	M	2	Constants.java	718	2014-07-31 03:03:03 KST
9	r40644	12535	pgardner	27	M	2	ChangeLog.txt	520	2014-07-31 03:01:28 KST

In this experiment, we have chosen to work on an end filename from a given changelog path. Filenames could be uniquely identified to address in the given directory system and exclusively recommended to co-change occurring in the past.

To keep the model from biased toward any characteristic of the data, we shuffled and randomly split the data into training (90%), and testing (10%) sets. Then, the model trains using training data. In contrast, the test set was used to validate the result. Finally, the model performance is evaluated by employing a similarity score.

3.2. **Evaluation method.** The feature vector considers being "good" in order to get acceptable classification accuracy. To answer how good filename embedding algorithms are, we can use either analogies or similarity. This article used a similarity score to measure how the model predicts filename with an extension similar to the requested filename and format. Furthermore, the similarity of the filename also might have a very close dependency on each other. These evaluation methods are just a representation of how well they are working in our particular problem approach with our particular dataset.

As depicted in Figure 5, we used UMAP (Uniform Manifold Approximation and Projection) [23] as a dimensional reduction algorithm that allows us to visualize higher dimensions into lower-dimensional space.

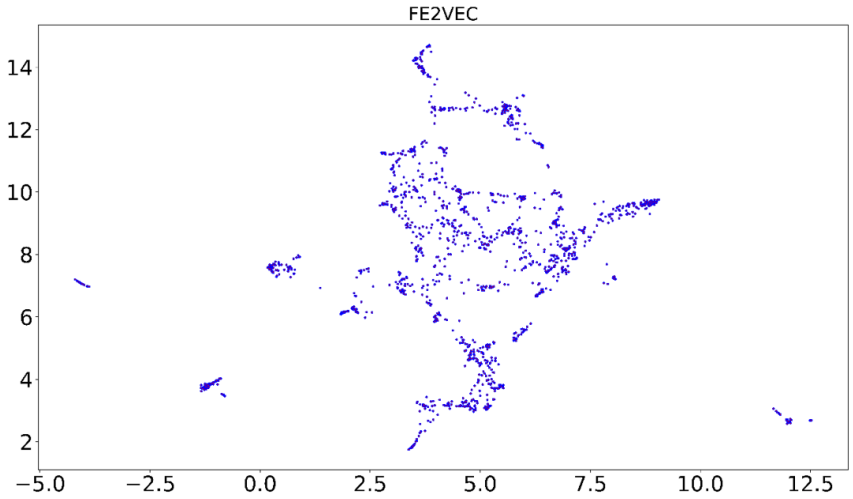

FIGURE 5. Each dot in this scattered plot is a filename vocabulary. The more dense cluster shows that the closer to each other in the vector space.

Figure 6 obtained the result by optimizing the following hyperparameters: *Window size* = 6, and *Negative sampling* = 6. The hyper parameter might differ according to a different dataset, but we found it well-suited in our use case. The more frequent the file got committed, the probability of getting opted for a negative sample will increase.

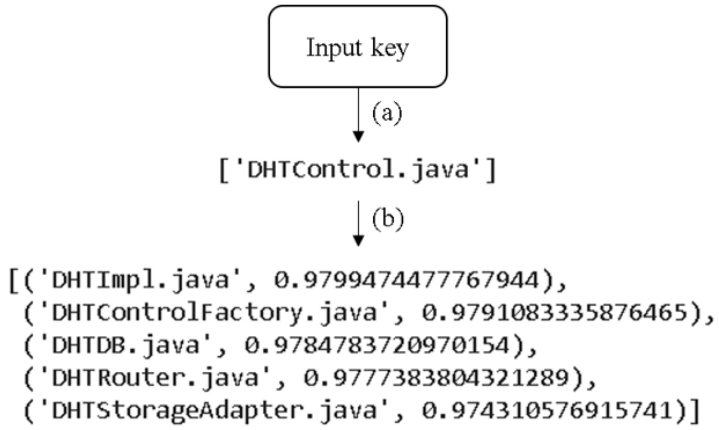

FIGURE 6. Input filename (a) and stop 5 output (b) based on similarity score

4. **Conclusion and Future Work.** This paper proposed a recommendation system using a software development changelog dataset and recommended a similar file that might need changing the change propagation probabilities at the file level. We have provided a systematic approach to recommending files that are most likely similar to each other based on their score that may need to consider retouching it. The proposed system may greatly help developers join the development team after the initial launch of the project. Furthermore, participating in the project helps them gain insight into similar files based on the current working filename. The file score also considers the batch commit in the past to gain a better score in the current calculation. To validate our approach, a case study of the Vuze-Azureus software development changelog, an open-source Java-based project, was used. In this study, the filename vocabulary and context were obtained from the changelogs accumulated between 2003-2014.

In our future research, we will add more open-source software changelog and other deep learning models to investigate the most suitable models for the proposed approach.

Acknowledgment. This research was financially supported by the Ministry of Trade, Industry and Energy (MOTIE) and Korea Institute for Advancement of Technology (KIAT) through the National Innovation Cluster R&D program (Project No. P0006910, Title: Development of Recommendation Business System Based Supplies with Blockchain).

REFERENCES

[1] M. Giffin, O. de Weck, G. Bounova, R. Keller, C. Eckert and P. J. Clarkson, Change propagation analysis in complex technical systems, *Journal of Mechanical Design*, vol.131, no.8, 2009.

[2] P. J. Clarkson, C. Simons and C. Eckert, Predicting change propagation in complex design, *Journal of Mechanical Design*, pp.788-797, 2004.

[3] W. Pan, H. Jiang, H. Ming, C. Chai, B. Chen and H. Li, Characterizing software stability via change propagation simulation, *Complexity*, pp.1-17, 2019.

[4] J. Lee and Y. S. Hong, Data-driven prediction of change propagation using dependency network, *Engineering Applications of Artificial Intelligence*, pp.149-158, 2018.

[5] M. Längkvist, L. Karlsson and A. Loutfi, A review of unsupervised feature learning and deep learning for time-series modeling, *Pattern Recognition Letters*, vol.42, pp.11-24, 2014.

[6] Z. Shen, Y. Zhang, J. Lu, J. Xu and G. Xiao, A novel time series forecasting model with deep learning, *Neurocomputing*, vol.396, pp.302-313, 2020.

[7] N. Akhtar and A. Mian, Threat of adversarial attacks on deep learning in computer vision: A survey, *IEEE Access*, vol.6, pp.14410-14430, 2018.
[8] N. Doulamis, A. Doulamis and E. Protopapadakis, Deep learning for computer vision: A brief review, *Computational Intelligence and Neuroscience*, 2018.
[9] T. L. Dang, T. Cao and Y. Hoshino, Classification of metal objects using deep neural networks in waste processing line, *International Journal of Innovative Computing, Information and Control*, vol.15, no.5, pp.1901-1912, 2019.
[10] Y. Shin, M. Kim, K.-W. Pak and D. Kim, Practical methods of image data preprocessing for enhancing the performance of deep learning based road crack detection, *ICIC Express Letters, Part B: Applications*, vol.11, no.4, pp.373-379, 2020.
[11] Y. Omae, M. Kobayashi, K. Sakai, T. Akiduki, A. Shionoya and H. Takahashi, Detection of swimming stroke start timing by deep learning from an inertial sensor, *ICIC Express Letters, Part B: Applications*, vol.11, no.3, pp.245-251, 2020.
[12] M. Nakano, T. Kuwazawa and Y. Oyama, Detection of objects jumping in front of car using deep learning, *ICIC Express Letters, Part B: Applications*, vol.10, no.5, pp.395-403, 2019.
[13] A. Yadav and D. K. Vishwakarma, Sentiment analysis using deep learning architectures: A review, *Artificial Intelligence Review*, vol.53, no.6, pp.4335-4385, 2020.
[14] C. Paul, A. Jay and S. Emre, Deep neural networks for YouTube recommendations, *ACM Conference on Recommender Systems*, Boston, MA, USA, 2016.
[15] P. Covington, J. Adams and E. Sargin, Deep neural networks for YouTube recommendations, *The 10th ACM Conference on Recommender Systems (RecSys'16)*, New York, NY, USA, 2016.
[16] M. Siavash, H. Alaa and T. Ladan, Using Bayesian belief networks to predict change propagation in software systems, *Proc. of the 15th IEEE International Conference on Program Comprehension*, Banff, Alberta, BC, 2007.
[17] A. Hassan and R. Holt, Predicting change propagation in software systems, *Proc. of the 20th IEEE International Conference on Software Maintenance*, Chicago, IL, USA, 2004.
[18] K. A. M. Ferreira, M. A. S. Bigonha, R. S. Bigonha, B. N. de Lima, B. M. Gomes and L. F. O. Mendes, A model for estimating change propagation in software, *Software Quality Control*, 2017.
[19] A. Karatzoglou and B. Hidasi, Deep learning for recommender systems, *The 11th ACM Conference on Recommender Systems (RecSys'17)*, New York, NY, USA, 2017.
[20] T. Mikolov, K. Chen, G. Corrado and J. Dean, Efficient estimation of word representations in vector space, *Proc. of Workshop at International Conference on Learning Representations*, 2013.
[21] *Vuze-Azureus*, Sourceforge, https://sourceforge.net/projects/azureus/, Accessed on 11-9-2020.
[22] T. Mikolov, I. Sutskever, K. Chen, G. Corrado and J. Dean, Distributed representations of words and phrases and their compositionality, *CoRR*, 2013.
[23] L. H. J. McInnes, UMAP: Uniform manifold approximation and projection for dimension reduction, *arXiv.org*, arXiv: 1802.03426, 2018.