Recommending Third-party Library Updates with LSTM Neural Networks

Phuong T. Nguyen[1], Juri Di Rocco[1], Riccardo Rubei[1], Claudio Di Sipio[1] and Davide Di Ruscio[1]

[1]*Università degli studi dell'Aquila, 67100 L'Aquila, Italy*

Abstract

During the lifecycle of a software project, oftentimes developers have the need to update third-party libraries (TPLs) from an old version to a newer one. This aims to keep their code up-to-date with the latest functionalities offered by the libraries. In practice, choosing the next version for a library is a daunting task since it is crucial to maintain a harmonious relationship with other libraries. We propose DeepLib, a novel approach to the recommendation of an upgrade plan for software projects with respect to library usage. We mine migration history to build matrices and train deep neural networks, which are eventually used to forecast the subsequent versions of the related libraries. We evaluate the framework on a dataset from the Maven Central Repository. The results show promising outcomes: DeepLib can recommend the next version for the library of interest, earning a high prediction accuracy.

Keywords

Mining software repositories, Third-party library updates, Deep learning, LSTM

1. Introduction

When working with coding tasks, developers usually make use of third-party libraries (TPLs) that offer desired functionalities [1, 2, 3], e.g., database administration, log management, and file utility to name a few. Reusing existing TPLs allows developers to leverage well-founded programming utilities without re-implementing software functionalities from scratch. This indeed helps them save time as well as increase productivity. Nevertheless, libraries evolve, many API functions are added, and many others are removed or deprecated. In this way, it is necessary to migrate an old library to a new one to enforce the new functionalities of the project. However, choosing the wrong version of a library may break the mutual dependencies among different libraries, resulting in unavoidable disruptions [4]. To upgrade a library, a developer needs to be knowledgeable of both versions' documentation and choose the right matching between methods. In fact, due to the fear of incompatibility and breaking changes [5], developers are highly reluctant to upgrade TPLs [4, 6]. In this respect, it is essential to have the proper machinery to assist them in choosing suitable updates.

IIR 2021 – 11th Italian Information Retrieval Workshop, September 13–15, 2021, Bari, Italy
phuong.nguyen@univaq.it (P. T. Nguyen); juri.dirocco@univaq.it (J. Di Rocco); riccardo.rubei@graduate.univaq.it (R. Rubei); claudio.disipio@graduate.univaq.it (C. Di Sipio); davide.diruscio@univaq.it (D. Di Ruscio)
0000-0002-3666-4162 (P. T. Nguyen); 0000-0002-7909-3902 (J. Di Rocco); 0000-0002-5077-6793 (D. Di Ruscio)

We propose DeepLib, a novel approach to recommendation of library updates, exploiting cutting-edge deep learning (DL) techniques. By analyzing the migration history of mined projects, we build matrices containing libraries and their versions in chronological order, which are fed to the recommendation engine. In addition, a long short-term memory recurrent network [7] (LSTM) was built to predict a set of versions. As output, DeepLib delivers the next version for a single library *lib* that the developer wants to upgrade. To the best of our knowledge, there exist no comparable tools providing this type of recommendation. Thus, we cannot compare our system with any reusable baselines, but evaluate it employing extensive experiments on a considerably large dataset from the Maven Central Repository. The experimental results show that DeepLib can effectively suggest the next version for a single library, demonstrating its feasibility in the field.

Structure. Section 2 presents a motivating example, and background to LSTMs. We introduce the proposed approach in Sect. 3, and present the evaluation materials and methods in Sect. 4. Afterwards, Sect. 5 reports and analyzes the experimental results, as well as the probable threats to validity. The related work is reviewed in Sect. 6 and the paper is concluded in Sect. 7.

2. Motivations and Background

Section 2.1 describes a motivating example, afterwards, Sect. 2.2 briefly recalls background related to long short-term memory recurrent neural networks.

2.1. Motivating example

While working with a software project, developers often need to upgrade the constituent third-party libraries from an old version to a newer one, aiming to approach the latest functionalities offered by the libraries. Hereafter, we consider the following terms: *(i) library* or *dependency*: A software module which is developed by a third party, and provides tailored functionalities. A library evolves over the course of time by offering new functionalities or bug fixes [4]; *(ii) repository* or *client*: A software project that is hosted in OSS platforms, e.g., GitHub, Maven and that makes use of some TPLs.

We consider in Table 1 a running example with maintainers working on the repository named *org.apache.hadoop:hadoop-auth*[1] which depends on a set of four libraries as follows: lib_1: *log4j:log4j*; lib_2: *org.slf4j:slf4j-log4j12*; lib_3: *org.apache.httpcomponents:httpclient*; and lib_4: *commons-codec:commons-codec*.

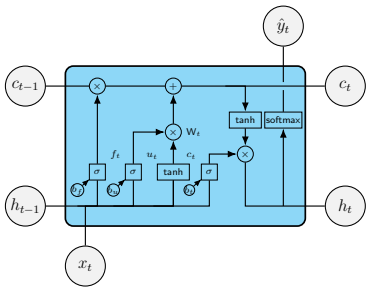
The latest version of the *hadoop-auth* repository is *3.0.0-alpha3* (the green row), and we assume that the maintainers want to upgrade the libraries. However, they do not know for sure which version should be used for the constituent libraries, i.e., all the cells are filled with a question mark (**?**). One may think of a simple heuristic that migrates a library to the next version or the latest one. However, we see that such a heuristic does not work in every case. In particular, there are two additional possible changes that developers can perform on library dependencies: *(i)* removal of a library; and *(ii)* downgrade migration, as we explain as follows.
▷ **Removal of a library.** In the table, the repository versions are listed in chronological order, i.e., using their timestamp. A cell with 0 implies that the library in the column is not included

[1] https://bit.ly/2WP3ysS

Table 1
Migration path of the *org.apache.hadoop:hadoop-auth* repository.

Client version	lib_1	lib_2	lib_3	lib_4	Timestamp
2.0.2-alpha	1.2.17	1.6.1	0	1.4	2012-10-02T00:44:04
2.3.0	1.2.17	1.7.5	4.2.5	1.4	2014-02-11T13:55:58
2.4.1	0	1.7.5	4.2.5	1.4	2014-06-21T06:08:34
2.5.1	0	0	4.2.5	0	2014-09-05T23:05:15
2.6.0	1.2.17	0	4.3.1	0	2014-11-13T22:35:37
2.7.2	1.2.17	1.7.10	4.2.5	1.4	2016-01-14T21:32:14
3.0.0-alpha3	1.2.17	1.7.10	4.5.2	1.4	2017-05-26T20:39:35
*	?	?	?	?	

Table 2
Background for LSTMs.

An LSTM cell [9, 10]	Formula
	• $\sigma(x) = (1 + exp(-x))^{-1}$ • $\tanh(x) = 2 \cdot \sigma(2x) - 1$ • $f_t = \sigma(W_f \cdot i_t + b_f)$ • $u_t = \sigma(W_u \cdot i_t + b_u)$ • $c_t = tanh(W_c \cdot i_t + b_c)$ • $W_t = c_{t-1} \cdot f_t + N_t \cdot u_t$

by the repository version represented in the row. The presence of a library is subject to change from version to version. For instance, lib_1 has been used by version *2.0.0-alpha*, *2.0.2-alpha*, and *2.3.0*. When the repository is upgraded from *2.3.0* to *2.4.1*, lib_1 is removed. However, the library is then re-introduced when moving from *2.5.1* to *2.6.0*. Thus, we see that the ability to recommend a 0 is also useful.

▷ **Downgrade migrations.** We can see that the upgrading is not always done upward, i.e., moving the library to a higher version since there are also backward migrations. For instance, when the client moves from *2.6.0* to *2.7.2*, lib_3 is downgraded from version *4.3.1* to *4.2.5*. However, the library is then updated to version *4.5.2* by client *3.0.0-alpha3*.

In the following subsection, we review an LSTM neural network dealing with time series data as a base for further presentations.

2.2. Long short-term memory neural networks

Recurrent neural networks (RNNs) [8] are a family of neural networks specialized in dealing with sequence data. An RNN stores information about past events to predict future occurrences. However, a main drawback of RNNs is that they cannot learn long-term dependencies well. Thus, long short-term memory recurrent neural networks (LSTMs) have been proposed to transcend the limitation [7]. LSTMs learn better long-term dependencies by memorizing the input sequence of data. Moreover, they have the mechanisms to remove or add information to remember worthy information and discard useless ones.

We refer to Table 2 to illustrate how an LSTM works. Given that $i_t = [h_{t-1}, x_t]$ is the concatenation of h_{t-1} the hidden state vector from the previous time step, and x_t is the current input vector, then two states are propagated to the next cell, i.e., cell state c_t and hidden state h_t. The output of the previous unit, together with the current input, is fed as the input data for a cell. The sigmoid function is used to discard useless information and retain useful information. $W_\times$ and $b_\times$ are the weight and bias matrices for different network entry, hidden state matrix. *Softmax* is used as the activation function, converting a set of real numbers to probabilities which sum to 1.0 [11]. Given C classes, and y_k is the output of the kth neuron, the final prediction is the class that gets the maximum probability, i.e., $\hat{y} = argmax\ p_k$, $k \in \{1, 2, .., C\}$, where p_k is computed as follows: $p_k = exp(y_k) / \sum_{k=1}^{C} exp(y_k)$.

3. DeepLib: Forecasting the next versions for third-party libraries

DeepLib is built on top of an LSTM to accept as input a set of versions and returns the future version for each library. Given a set of projects, we populate a matrix by reading each client and filling the correct version for all libraries. We obtain a matrix where each row represents a client with its versions. From the matrix, we insert one more column on the right side. For each client, the last cell is filled with the version of the library by the next client.

Figure 1(b) depicts the migration matrix for lib_1 for the motivating example in Table 1. The left side depicts the original migration matrix, and the right side is the resulting migration matrix for lib_1. For instance, the first row contains the versions of the four libraries, i.e., *(1.2.15, 1.6.1, 0, 1.4)* while the last column is the future version of lib_1, i.e., *1.2.17*, which is actually the version of lib_1 by the next client (Version *2.0.2*). This can be interpreted as follows: *"Given that in the current client we use* 1.2.15 *for* lib_1, 1.6.1 *for* lib_2, *no* lib_3, *and version* 1.4 *for* lib_4, *then in the next version of the client we should adopt* 1.2.17 *for* lib_1*."*

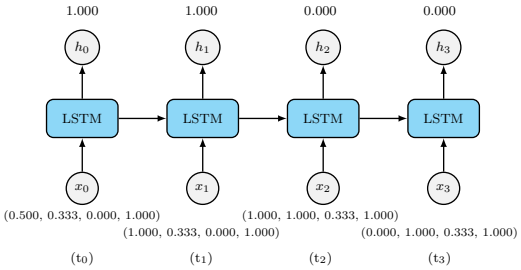
Since LSTMs only work with numbers, we need to encode each library version using a unique number. Moreover, the σ and $\tanh$ functions (cf. Section 2.2) accept values in the [0..1] range, we also need to normalize all the numbers to meet this requirement. The right most part of Fig. 1(b) depicts the migration matrix after the encoding and normalizing phases.[2]

Figure 1(a) explains how DeepLib works: The data to feed the system is a tuple of the form x_t=<lib_1^{v1}, lib_2^{v1}, lib_3^{v1}, lib_4^{v1}> and y_t=<lib_1^{v2}>, which captures the migration path of a client. DeepLib uses input features from recent events, i.e., X = $\{x_t\}$, $t \in T^p$ to forecast the future version of each libraries, Y = $\{y_t\}$, $t \in T^f$, where T^p and T^f are time in the past and the future, respectively. By each time step t, only one vector x_t is fed to the LSTM cell.

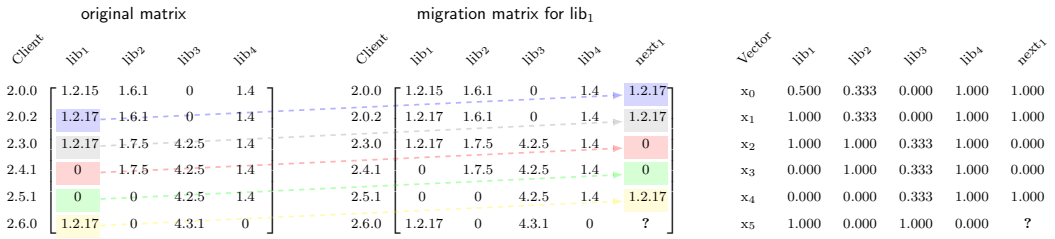
The conceived architecture is depicted in Fig. 2. DeepLib has been implemented on top of the Keras framework[3] and trained using Google Colab. Data is fetched from OSS platforms ①, e.g., GitHub and Maven with Crawler ②. The collected data is then aligned, sorted, and transformed into a suitable format to store in CSV files by Converter ③. It is necessary

[2] Matrix encoding and normalizing is conveniently done with the *LabelEncoder()* and *MinMaxScaler()* utilities embedded in Python.

[3] https://keras.io/



(a) Architecture of DeepLib



(b) Migration matrices for lib_1

Figure 1: Network configuration and data representation.

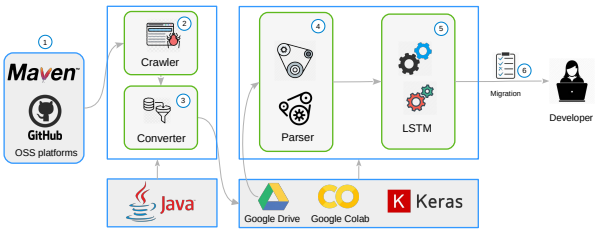


Figure 2: System architecture.

to upload the data to Google Drive for further processing. The PARSER component ④ builds migration matrices for LSTM ⑤ which eventually provides library updates ⑥.

4. Evaluation

We evaluate DeepLib to study its capability to provide a developer with accurate recommendations featuring suitable upgrades. In Sect. 4.1 we introduce the datasets, while in Sect. 4.2 we describe the settings and evaluation metrics.

4.1. Data extraction

We rely on a dataset collected from more than 1,000 public Maven repositories. The dataset consists of migration history for the *top ten* popular libraries. Given a set of libraries, we crawled all of their versions together with the list of clients and their corresponding release date. Moreover, we mined dependency links from a client to the used libraries with Maven Dependency Graph, a graph-based representation of the collected artifacts in Maven and their

Table 3
Summary of the datasets.

Library	Alias	η_V	η_C	η_M
junit:junit	L_{01}	29	101,541	2,073
org.slf4j:slf4j-api	L_{02}	74	44,233	16,187
org.scala-lang:scala-library	L_{03}	228	25,417	19,508
com.google.guava:guava	L_{04}	90	24,532	8,921
org.mockito:mockito-core	L_{05}	259	20,762	855
com.android.support:appcompat-v7	L_{06}	59	19,772	1,194
commons-io:commons-io	L_{07}	25	19,198	3,332
ch.qos.logback:logback-classic	L_{08}	75	18,655	3,100
org.commons:commons-lang3	L_{09}	18	17,224	3,915
org.clojure:clojure	L_{10}	67	15,954	234

relationships. Then, additional steps were performed to remove unuseful clients, using the following constraints: A client should *(i)* have more than one version; *(ii)* migrate at least one library among the considered libraries; and *(iii)* use at least four of the given libraries. This allows us to keep the resulting matrices not too sparse. Table 3 reports the main characteristics of the datasets: each row features an input library with its name, the number of versions (η_V), the number of clients that use at least one version of the library (η_C), the number of clients that migrate from one version to another (η_M).

4.2. Settings and metrics

▷ **Experimental settings.** We opted for the ten-fold cross-validation technique [12], each dataset (cf. Section 4.1) is split into k=10 equal parts, so-called *folds*. For each validation round, one fold is used as testing data, and the remaining k-1 folds are combined to form the training data. The evaluation simulates a real development scheme: *the system needs to provide the active projects with recommendations using the data from a set of existing projects.*
▷ **Metrics.** We evaluate how well DeepLib recommends versions that eventually match with those stored in the ground-truth data. We compute accuracy according to each library (Acc_{lib}): the metric measures the ratio of clients with correct predictions (δ) to the total number of clients (n), i.e., $Acc_{lib} = \frac{\delta}{n}$. Moreover, we compute correlation efficients using the Spearman ρ and the Kendall τ, and measure the effect size with Cliff's delta [13].

5. Results

Section 5.1 reports an example recommended by DeepLib, and Sect. 5.2 analyzes the results.

5.1. Explanatory example

Fig. 3 shows the recommendation for the *com.hubspot:SingularityService* repository.[4] The left side depicts the list of versions for the libraries of the client numbered *0.4.2*, which invokes four libraries, i.e., L_{02}, L_{04}, L_{08}, and L_{09} (cf. Table 3). The remaining cells are filled with 0, indicating that the corresponding libraries are not present. The top row on the right depicts the real versions of all libraries for the next client *0.6.1*. The scenario is challenging as it requires

[4] https://bit.ly/2MnvnXn

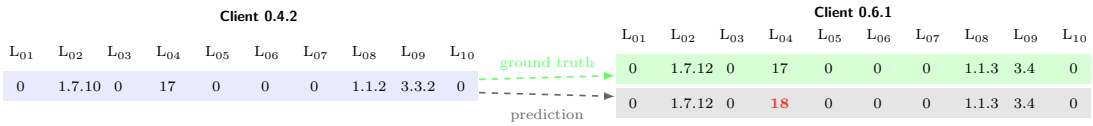


Figure 3: Recommendation for the *com.hubspot:SingularityService* repository.

Table 4
Acc$_{lib}$ obtained by DeepLib.

	Cross validation										
L_{01}	0.975	0.956	0.988	0.991	0.954	0.954	0.962	0.965	0.986	0.970	0.970
L_{02}	0.558	0.728	0.792	0.667	0.347	0.635	0.656	0.742	0.611	0.515	0.625
L_{03}	0.748	0.824	0.741	0.856	0.908	0.944	0.902	0.670	0.552	0.669	0.781
L_{04}	0.767	0.735	0.805	0.776	0.678	0.608	0.766	0.776	0.954	0.857	0.772
L_{05}	0.961	0.989	0.988	0.972	0.976	0.623	0.952	0.994	0.966	0.974	0.939
L_{06}	0.997	0.994	1.000	0.999	1.000	1.000	1.000	1.000	1.000	1.000	0.999
L_{07}	0.952	0.981	0.970	0.969	0.945	0.947	0.952	0.960	0.990	0.961	0.963
L_{08}	0.889	0.950	0.958	0.922	0.967	0.971	0.921	0.924	0.992	0.967	0.946
L_{09}	0.937	0.952	0.967	0.966	0.960	0.852	0.966	0.900	0.984	0.952	0.944
L_{10}	0.994	1.000	0.998	1.000	1.000	0.993	0.987	1.000	1.000	1.000	0.997

a big migration step, i.e., upgrading almost all the constituent libraries at once. We expect DeepLib to provide proper recommendations to assist developers in migrating their clients, as big migrations may make the prediction more challenging.

The second row of Fig. 3 presents the versions suggested by DeepLib for client *0.6.1*. The tool recommends correct migration for L_{02}, L_{08}, and L_{09}. It only mispredicts for L_{04}, by providing *18* instead of *17*, the correct one. Moreover, DeepLib accurately predicts all the zeros, i.e., the libraries that are not invoked. This seems to be trivial at first sight, however as we pointed in Section 2.1, recommending a zero makes sense. In summary, we see that our tool can provide relevant recommendations to the repository, even when a big migration step is required.

5.2. Result analysis

We performed experiments on the considered datasets using the ten-fold cross-validation technique. The prediction results are shown in Table 4. For each library, besides the accuracy for each fold from **F01** to **F10**, we also averaged out the scores to get the final accuracy, which is shown in the last column of the tables. Moreover, the cells with an accuracy smaller than 0.700 are marked using the light red color, signaling an inferior performance.

Overall, the table demonstrates that DeepLib can provide accurate predictions for almost all the libraries. For instance, with L_{01}, by all the testing rounds DeepLib always gets an accuracy larger than 0.90, and the average accuracy is 0.970. This also applies to other libraries, such as L_{05} or L_{07}. Especially, by L_{06} and L_{10} we see a maximum accuracy for most of the folds: DeepLib gets an average accuracy of 0.999 with L_{06}.

Finding 1. Being fed with proper data, DeepLib recommends the next version for a single library, obtaining a high accuracy for the majority of the libraries.

We see that DeepLib gets an encouraging result for most of the libraries. Nevertheless, it *fails* in some certain cases. It is necessary to find out the rationale behind such a setback, as this

helps reveal the pitfalls that one can avoid when deploying DeepLib. According to Table 3, there are three variables: number of versions (η_V), number of clients (η_C), and number of migrations (η_M). We perform quantitative analyses to study the relationships between these variables and the average accuracy, using the Spearman ρ and the Kendall τ, and measure the effect size with Cliff's delta [13].

There is a low correlation between accuracy and η_V, and this is enforced by both coefficients, i.e., $\rho = -4.84 \times 10^{-1}$ and $\tau = -3.03 \times 10^{-1}$. Moreover, the difference is statistically significant, i.e., p-value$= 7.78 \times 10^{-3}$ and 5.98×10^{-9}. The table also shows that the effect is large by the considered relationships, i.e., Cliff's delta is 1.0. This essentially means that the more versions a library has, the lower accuracy DeepLib obtains. A similar trend is seen with the relationship between accuracy and η_M. In particular, $\rho = -8.48 \times 10^{-1}$ and $\tau = -6.71 \times 10^{-1}$, which means accuracy is disproportionate to the number of migrations. The difference is statistically significant and the effect size is large. Altogether, this suggests that it is more difficult for DeepLib to provide good recommendations for a library associated with a large number of migrations.

We suppose that this happens due to the structure of the networks, i.e., if there are more versions or migrations, the network fails to absorb all the patterns. Such a limitation can be overcome with deeper networks, i.e., by padding additional hidden units to DeepLib. To validate the hypothesis, we increased the number of network units from 40 to 100 and reran the experiments on the libraries with which DeepLib gets a low accuracy by most of the folds, i.e., L_{02}, L_{03}. As expected, we see a gain in accuracy by these libraries. For the sake of clarity, we report the change in accuracy with respect to Table 4 as follows: $\text{Acc}_{lib}(L_{02})$: $0.625 \rightarrow 0.632$, $\text{Acc}_{lib}(L_{03})$: $0.772 \rightarrow 0.781$.

Finding 2. DeepLib suffers a deficiency in performance on libraries with a large number of versions and/or migrations. However, depending on the input data, the system's performance can be enhanced with deeper networks.

5.3. Threats to validity

Threats to *internal validity* are related to the factors in the approach and evaluation that could have affected the final results. A possible threat is that the datasets might not fully reflect real-world development scenarios as we could consider only popular libraries. To mitigate the threats, we crawled a wide range of clients across several repositories. Still, we believe that considering data from other sources, e.g., GitHub, can help eliminate the threat.

The main threat to *external validity* concerns the generalizability of our findings. DeepLib has been evaluated on projects collected from Maven, since we have suitable software to fetch the data. We anticipate that our tool is also applicable to other platforms, as long as they support versioning. We plan to generalize DeepLib to data from GitHub in our future work.

6. Related Work

We review notable recommender systems by focusing on those related to the adoption of TPLs and API migrations.

Ouni *et al.* [14] develops LibFinder that uses a multi-objective algorithm to detect semantic similarity in source code. CrossRec [2, 15] assists developers in selecting suitable TPLs. The system exploits a collaborative filtering technique to recommend libraries by relying on the set of dependencies, which have been included in the project being developed. LibSeek [16] employs the matrix factorization (MF) technique to predict relevant TPLs for mobile apps. It adopts an adaptive weighting scheme to reduce the skewness caused by popular libraries. Furthermore, the MF-based algorithm is used to integrate neighborhood information by computing the similarity of libraries contained in the matrix.

Req2Lib [17] has been recently proposed to recommend TPLs given textual description of project requirements. The tool employs a seq2seq LSTM which is trained with description and libraries belonging to configuration file. Additionally, a domain-specific embedding model obtained from Stack Overflow is used to encode words in high-dimensional vectors. Xu *et al.* propose Meditor [18] to analyze GitHub commits to extract migration-related (MR) changes by mining *pom.xml* files. Once MR updates have been found, the tool employs the WALA framework to check their consistency by analyzing the developer's context and apply them directly. Apiwave [19] infers and retrieves relevant information related to TPLs, i.e., popularity and migration data. The tool uses two different modules to discover the popularity by analyzing import statements of projects, i.e., the removal of certain API decreases its popularity. Additionally, the system can infer migration data from each API replacement.

Differently from the aforementioned approaches, DeepLib can learn from what other projects have done to recommend the next upgrades that maintainers should operate on one or more libraries already in their project. DeepLib recommends also removals of dependencies according to existing migrations. Migrating the source code that might get affected by the recommended upgrades is not in the scope of this paper, and we plan it as future work.

7. Conclusion and Future Work

To reduce the burden related to the identification of the upgrades that need to be operated on the current system we proposed DeepLib, a novel approach to recommendation of the next version for the used TPLs by considering migration histories of several OSS projects. Our proposed tool is able to extract relevant migration data and encode it in matrices. Then, deep learning techniques are employed to provide recommendations that are relevant for the current configuration. As future work, we plan to evaluate DeepLib on specific ecosystems including that of Android apps. Moreover, we also intend to investigate the possibility of applying the technique to support the migration of source code, which can be affected by the proposed upgrade plans.

8. Acknowledgments

The research described in this paper has been carried out as part of the CROSSMINER Project, which has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant 732223.

References

[1] S. Raemaekers, A. van Deursen, J. Visser, Semantic versioning and impact of breaking changes in the maven repository, Journal of Systems and Software 129 (2017) 140 – 158. URL: http://www.sciencedirect.com/science/article/pii/S0164121216300243. doi:https://doi.org/10.1016/j.jss.2016.04.008.

[2] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, M. Di Penta, CrossRec: Supporting Software Developers by Recommending Third-party Libraries, Journal of Systems and Software (2019) 110460. URL: http://www.sciencedirect.com/science/article/pii/S0164121219302341. doi:https://doi.org/10.1016/j.jss.2019.110460.

[3] J. Di Rocco, D. Di Ruscio, C. Di Sipio, P. T. Nguyen, R. Rubei, Development of recommendation systems for software engineering: the CROSSMINER experience, Empirical Software Engineering 26 (2021) 69. URL: https://doi.org/10.1007/s10664-021-09963-7. doi:10.1007/s10664-021-09963-7.

[4] E. Derr, S. Bugiel, S. Fahl, Y. Acar, M. Backes, Keep me updated: An empirical study of third-party library updatability on android., in: B. M. Thuraisingham, D. Evans, T. Malkin, D. Xu (Eds.), ACM Conference on Computer and Communications Security, ACM, 2017, pp. 2187–2200. URL: http://dblp.uni-trier.de/db/conf/ccs/ccs2017.html#DerrBFA017.

[5] J. Huang, N. Borges, S. Bugiel, M. Backes, Up-to-crash: Evaluating third-party library updatability on android, in: 2019 IEEE European Symposium on Security and Privacy (EuroS P), 2019, pp. 15–30. doi:10.1109/EuroSP.2019.00012.

[6] R. G. Kula, D. M. German, A. Ouni, T. Ishio, K. Inoue, Do developers update their library dependencies?: An empirical study on the impact of security advisories on library migration, Empirical Software Engineering 23 (2018) 384–417. doi:10.1007/s10664-017-9521-5.

[7] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780. URL: https://doi.org/10.1162/neco.1997.9.8.1735. doi:10.1162/neco.1997.9.8.1735.

[8] S. Alemany, J. Beltran, A. Pérez, S. Ganzfried, Predicting hurricane trajectories using a recurrent neural network, in: The Thirty-Third Conference on Artificial Intelligence, AAAI 2019, The Ninth Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, AAAI Press, 2019, pp. 468–475. URL: https://doi.org/10.1609/aaai.v33i01.3301468. doi:10.1609/aaai.v33i01.3301468.

[9] C. Olah, Understanding LSTM Networks, 2020. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[10] L. Iovino, P. T. Nguyen, A. D. Salle, F. Gallo, M. Flammini, Unavailable transit feed specification: Making it available with recurrent neural networks, IEEE Transactions on Intelligent Transportation Systems 22 (2021) 2111–2122. doi:10.1109/TITS.2021.3053373.

[11] W. Rawat, Z. Wang, Deep convolutional neural networks for image classification: A comprehensive review, Neural Comput. 29 (2017) 2352–2449. URL: https://doi.org/10.1162/neco_a_00990. doi:10.1162/neco_a_00990.

[12] R. Kohavi, A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection, in: 14th International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers Inc., San Francisco, 1995, pp. 1137–1143.

[13] R. J. Grissom, J. J. Kim, Effect sizes for research: A broad practical approach, 2nd edition ed., Lawrence Earlbaum Associates, 2005.

[14] A. Ouni, R. G. Kula, M. Kessentini, T. Ishio, D. M. German, K. Inoue, Search-based software library recommendation using multi-objective optimization, Inf. Softw. Technol. 83 (2017) 55–75. URL: https://doi.org/10.1016/j.infsof.2016.11.007. doi:10.1016/j.infsof.2016.11.007.

[15] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, Mining software repositories to support OSS developers: A recommender systems approach, in: Proceedings of the 9th Italian Information Retrieval Workshop, Rome, Italy, May, 28-30, 2018., 2018. URL: http://ceur-ws.org/Vol-2140/paper9.pdf.

[16] Q. He, B. Li, F. Chen, J. Grundy, X. Xia, Y. Yang, Diversified third-party library prediction for mobile app development, IEEE Transactions on Software Engineering (2020) 1–1.

[17] Z. Sun, Y. Liu, Z. Cheng, C. Yang, P. Che, Req2Lib: A Semantic Neural Model for Software Library Recommendation, in: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2020, pp. 542–546. doi:10.1109/SANER48275.2020.9054865, iSSN: 1534-5351.

[18] S. Xu, Z. Dong, N. Meng, Meditor: Inference and Application of API Migration Edits, in: 2019 IEEE/ACM 27th Int. Conf. on Program Comprehension (ICPC), 2019, pp. 335–346. doi:10.1109/ICPC.2019.00052.

[19] A. Hora, M. T. Valente, Apiwave: Keeping track of API popularity and migration, in: 2015 IEEE Int. Conf. on Software Maintenance and Evolution (ICSME), 2015, pp. 321–323. doi:10.1109/ICSM.2015.7332478.