Adversarial Machine Learning: On the Safeness of Recommender Systems in Software Engineering

Phuong Nguyen
Università degli studi dell'Aquila, Italy
Email: phuong.nguyen@univaq.it

A research proposal submitted to VIASM on December 25th, 2020

This research proposal presents an initial investigation on the topic of adversarial machine learning and possible implications on recommender systems for software engineering. It also draws some prospective work for a 2-month research stay at VIASM. First, Section 1 introduces the main goals of the research, while an overview of the addressed topic is given in Section 2. Afterwards, a tentative plan and the corresponding research activities are going to be presented in Section 3. Finally, Section 4 sketches the expected results.

1 Goals

In recent years, there is a dramatic increase in the application of Machine Learning (ML) algorithms in several domains, including the development of recommender systems for software engineering (RSSE). While research has been conducted on the underpinning ML techniques to improve recommendation accuracy, little attention has been paid to make such systems robust and resilient to malicious data. By manipulating the algorithms' training set, i.e., large open source software (OSS) repositories, one could possibly make recommender systems vulnerable to adversarial attacks. This section gives an introduction to the topic being addressed during my research stay.

1.1 Introduction

When developers join a new software project, typically they have to master a large number of information sources. In such a context, the problem is not the lack but instead an overload of information coming from heterogeneous and rapidly evolving sources. Thus, recommender systems in software engineering [1] (RSSE) aim to provide developers with useful recommendations consisting of different items, such as code examples [2], possible third-party components [3], documentation, to name a few.

The development of RSSE encompasses several phases including the design of the underpinning algorithms or the reuse of existing ones. Machine Learning (ML) techniques are amongst the natural choices that developers take when new recommender systems have to be conceived. This means, for example, that the recommendation of code elements, e.g., snippets or APIs, is learned from existing code bases or informal documentation. As a result, the quality of the provided recommendation depends on the quality of such data, whose noisiness has been previously reported [4].

Even worse, online data used to train recommenders can be exploited for malicious purposes. Adversarial Machine Learning [5] (AML) is a field of study that focuses on security issues in ML systems and, specifically, in (general-purpose) recommender systems as well [6]. Research has been done to identify probable threats and seek out adequate countermeasures [7]. For example, Anelli *et al.* [7] use shilling attack to manipulate a collaborative-filtering recommender system operated with Linked Data. Also, Wang and Han [8] propose an improvement of the Bayesian personalized ranking (BPR) technique by exploiting the adversarial training-based mean strategy (AT-MBPR) in the domain of collaborative filtering-based recommender systems.

To defend a system against threats, in the first place, it is necessary to be knowledgeable of various types of adversary activities [9]. Such activities generate perturbations to deceive and disrupt systems by causing a malfunction, compromising their recommendation capabilities. The ultimate aim of these attacks is to manipulate target items, thus creating either a negative or positive influence on the final recommendations [10], depending on the attacker's intention. For instance, in image classification, an attacker crafts an input image by adding non-random noise in a way that it will be imperceptible to ML models, whilst still being properly

recognized by humans [11]. As an example, a panda was recognized as a ribbon by cutting-edge deep neural networks once the input image had been padded with noise, meanwhile humans still correctly perceived the panda from the forged image [12].

AML has been studied in a wide range of domains, e.g., online shopping systems [9] or image classification [11], and addresses both risks and countermeasures. However, to the best of my knowledge, AML has not been investigated in the context of SE applications of ML yet.

1.2 Research objectives

Based on a thorough observation, I realized that most of the efforts related to RSSE have been made to enhance their prediction accuracy, and no work has been dedicated to investigate the problem of using intentionally falsified data to compromise recommender systems' capabilities, as well as conceptualizing the corresponding countermeasures. In this respect, my work is the first one that brings in the issue of AML in RSSE.

During the research stay at VIASM, I am going to address the related issues by answering the following research questions.

- **RQ**$_1$: "*What are the main threats to third-party library and APIs recommender systems?*" I plan to study the threats that cause harm or danger to RSSE suggesting third-party libraries and API function calls. These two types of recommendations have been selected since they are representative of scenarios in which *(i)* the recommendation is learned from OSS repositories; and *(ii)* the outcome of a malicious recommendation, e.g., the usage of a library or an API, can result in severe security holes.
- **RQ**$_2$: "*How can we defend RSSE against adversarial attacks?*" Given the existing threats, it is necessary to conceive effective countermeasures that can ward off attacks tailored to RSSE. This is done by studying learning algorithms being aware of adversarial attacks and seize them. Moreover, adversarial attempts should be turned into features for the training process, i.e., RSSE should not only learn from useful patterns, but also be able to learn how to avoid hostile patterns.

2 Project overview

Most RSSE heavily rely on open data sources, such as GitHub, the Maven Central Repository, or Stack Overflow, which can be easily steered by adversaries [13]. In other words, these systems are vulnerable to attacks equipped with forged input data. Therefore, there is the need for comprehending the likely threats, with the ultimate aim of conceiving counteractions to increase the safeness of RSSE.

2.1 Classification of attacks

In a supervising learning task, given a training dataset D with n pairs of input sample and the corresponding label $(x, y) \in \mathsf{X} \times \mathsf{Y}$, classification is defined as seeking a candidate function that can predict the class label y around the input sample x. This boils down to solving an empirical risk minimization (ERM) problem by means of the following equation [6]:

$$\min_{a} \sum_{x,y \in D} l(f(x_i, \theta), y_i) \tag{1}$$

where $l(.)$ is the empirical risk function, or the loss function, θ is the model parameter. Adversarial attempts try to generate perturbed examples in the form of: $x_p = x + \delta$, by means of a non-random perturbation δ, which leads to an erroneous prediction, e.g., misclassification.

In a similar fashion, attackers to RSSE may try to craft the training data with their malicious examples, which once being recommended, can cause harm the target system. Attacks to recommender systems are classified into two main categories as follows [6]:

- *Poisoning attacks* spoil an ML model by falsifying the input data;
- *Evasion attacks* attempt to avoid being detected by hiding malicious contents, which then will be classified as legitimate by ML models.

With poisoning attacks, there are two possible interventions:

– *Push attacks* favor the targeted items, thus increasing the possibility of being recommended;

– In contrast, *nuke attacks* try to downgrade/defame the targeted items [7, 9], compelling them to disappear from the recommendation list.

In the research stay at VIASM, I am going to focus on poisoning attacks as they are easy to conduct, yet effective. The remaining attacks are left to my future work.

2.2 Generative Adversarial Networks

Generative Adversarial Networks [14] (GANs) have been widely used in image processing to produce crafted images, which resemble real ones. According to my investigation, GANs can also be exploited to generate fake training data to feed RSSE. This section briefly recalls the main technical details of GANs.[1]

Figure 1 illustrates a GAN which consists of two main components, namely Generator and Discriminator. Generator ② is a deep neural network and accepts as input both real training data ④ and crafted data ①, i.e., noise. Discriminator ⑤ is also a deep neural network and it learns to distinguish the real training data from the crated data, to provide the final prediction ⑥. In other words, Generator is trained with real and forged data to trick Discriminator, which in turns attempts to avoid being tricked by learning from real training data.

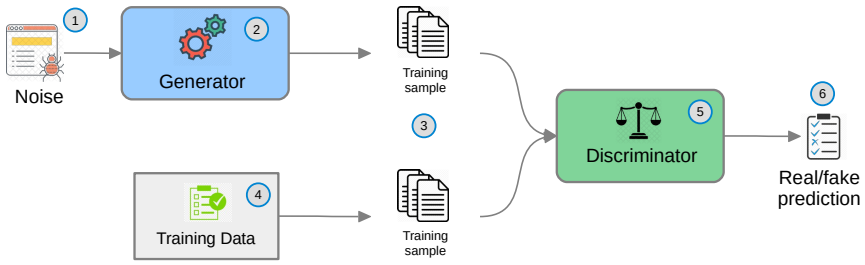

Figure 1: Architecture of a Generative Adversarial Network.

In this context, Discriminator plays the role of a loss/error function to train Generator. Using Discriminator to train Generator boils down to minimizing the JS-divergence between the distributions of the generated and real data. The final aim is to minimize the loss, which is defined using the following formula [14]:

$$E_{x\sim p_d(x)}\left[logD^*(x)\right] + E_{z\sim p_z(z)}\left[log(1-D^*(G(z)))\right] \tag{2}$$

$D^*(x)$ is defined as follows:

$$D^*(x) = \frac{p_d(x)}{p_d(x)+p_g(x)} \tag{3}$$

where $p_d(x)$ is the distribution of the training data, and $p_z(x)$ is the distribution of the noise.

The first component, i.e., $E_{x\sim p_d(x)}\left[logD^*(x)\right]$ corresponds to attempts to recognize real data better, while the second component, i.e., $E_{z\sim p_z(z)}\left[log(1-D^*(G(z)))\right]$ means that the engine can spot fake data better.

It is my belief that GANs can be used to produce crated training OSS projects to spoil RSSE. In this respect, it is necessary to perform a detailed investigation on this topic, as well as to find effective countermeasures that can deal with this type of adversarial attempts.

2.3 Potential risks to RSSE for mining libraries and API calls

This section reviews notable RSSE that support the development of software projects by delivering third-party libraries and API function calls. Table 1 lists the considered systems according to their functionality in chronological order. The possible vulnerabilities are discussed according to the previously-given categorization of attacks (cf. Section 2.1), based on the systems' internal design.

▷ **Library recommendation.** LibRec [15] recommends libraries using a combination of rule mining and a collaborative-filtering technique to mine libraries from projects similar to the one being developed. LibCUP [16] suggests libraries that have strong ties by using a clustering approach to identify and recommend co-usage patterns. LibD [17] provides libraries to Android apps using a clustering technique. First, it decompiles applications

[1]The background of GANs as well as the corresponding notions originate from `https://bit.ly/3hizVtf`

Table 1: Notable RSSE for mining libraries and APIs.

	System	Venue	Year	Data source
Library rec.	LibRec [15]	Working Conference on Reverse Engineering (now SANER)	2013	GitHub
	LibCUP [16]	Journal of Systems and Software	2017	GitHub
	LibD [17]	International Conference on Software Engineering	2017	Android markets
	LibFinder [18]	Information and Software Technology Journal	2018	GitHub
	CrossRec [3]	Journal of Systems and Software	2020	GitHub
	LibSeek [19]	IEEE Transactions on Software Engineering	2020	Google Play, GitHub, MVN
API rec.	MAPO [20]	ECOOP	2009	SourceForge
	UP-Miner [21]	International Conference on Mining Software Repositories	2013	Microsoft Codebase
	DeepAPI [22]	ESEC/FSE	2016	GitHub
	PAM [23]	ESEC/FSE	2016	GitHub
	FINE-GRAPE [24]	Empirical Software Engineering Journal	2017	GitHub
	FOCUS [2]	International Conference on Software Engineering	2019	GitHub, MVN

to build a control flow graph composed of packages, classes, and methods belonging to the projects. Then, the graph is used to extract features, and grouped by a similarity function. Ouni *et al.* proposed LibFinder [18], providing libraries based on a multi-objective search-based algorithm. Being built with a collaborative-filtering technique, CrossRec extracts libraries from similar projects [3]. LibSeek [19] relies on a matrix factorization technique to deliver relevant libraries for mobile apps, obtained by collecting neighborhood information, i.e., characteristics of similar libraries.

As it can be seen, all the considered systems leverage open sources, e.g., GitHub or Android markets, for training. Moreover, they mine libraries using similarity-based measures, either a similarity function, or a clustering technique. Thus, they are exposed to perturbations with malicious content hidden in OSS projects. I therefore conjecture that, by fabricating projects with bogus data, attackers can favor (push attack) or defame (nuke attack) a library [6, 7]. In other words, they can make a good/useful library out of being recommended, or even worse, promote a malicious library to a higher rank in the recommendations, so that users of the recommender system would unintendedly adopt it.

▷ **API recommendation.** MAPO [20] recommends API patterns by extracting API related information from the developer's context. The resulting data is clustered and ranked according to their similarity with the client code. In this respect, the system can be fooled with malicious code intentionally inserted into similar projects. Wang *et al.* proposed UP-Miner [21] to mine from source code. Since UP-Miner relies on a similarity measure, it may recommend to developers malicious code embedded in projects disguised as similar. DeepAPI [22] generates relevant API sequences starting from a natural language query. It employs an RNN Encoder-Decoder to encode words in context vectors used to train the model. As the corpus is collected from GitHub projects, a hostile user can easily inject perturbations during the data gathering phase, i.e., feeding the system with interfered projects. PAM [23] has been proposed to extract relevant API patterns from client code by using the structural Expectation-Maximization (EM) algorithm to infer the most probable items. The mined API patterns are then ranked according to their probability. Push and nuke attacks could easily modify the final ranking obtained by the tool, i.e., operating on terms' occurrences to favor or defame a certain pattern. FINE-GRAPE [24] delivers relevant APIs by relying on the history of the related files. It parses GitHub projects, discovers, and ranks the relevant API calls according to their history, i.e., methods, annotations, and classes from every API version. FINE-GRAPE is prone to manipulations which forge an artificial history of API calls in GitHub projects. FOCUS [2] suggests APIs by encoding projects in a tensor and using a collaborative-filtering technique. Since it works on data mined from similar projects, FOCUS is not immune from attacks, i.e., an adversary can create fake projects with toxic APIs and pose them as legitimate to trick FOCUS into recommending these calls.

3 Project plan and main activities

- July 1st 2022: Arrive at VIASM to start the research stay.
- From July 1st 2022 to July 31st 2022: Conduct an evaluation on two third-party library recommender systems to perturb their recommendations.
- Organize the first seminar on the topic of: "*Machine Learning for Software Engineering.*"
- From August 1st 2022 to August 31st 2022: Investigate countermeasures to combat adversarial attempts.
- Organize the second seminar on the topic of: "*Adversarial Machine Learning.*"
- End of August 2022: Present the final report on the research stay.

4 Expected results

The results of the research stay would be an article submitted (and accepted for publication) to one of the following journals:

- Elsevier Journal of Systems and Software (ISI/SCIE, Q1).
- Elsevier Information and Software Technology Journal (ISI/SCIE, Q1).
- IEEE Transactions on Software Engineering (ISI/SCIE, Q1).

References

[1] Martin P. Robillard, Walid Maalej, Robert J. Walker, and Thomas Zimmermann. *Recommendation Systems in Software Engineering*. Springer, 2014.

[2] Phuong T. Nguyen, Juri Di Rocco, Davide Di Ruscio, Lina Ochoa, Thomas Degueule, and Massimiliano Di Penta. FOCUS: A Recommender System for Mining API Function Calls and Usage Patterns. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, pages 1050–1060, Piscataway, NJ, USA, 2019. IEEE Press.

[3] Phuong T. Nguyen, Juri Di Rocco, Davide Di Ruscio, and Massimiliano Di Penta. CrossRec: Supporting Software Developers by Recommending Third-party Libraries. *Journal of Systems and Software*, page 110460, 2020.

[4] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. Germán, and Daniela E. Damian. The promises and perils of mining github. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, pages 92–101, 2014.

[5] J. D. Tygar. Adversarial machine learning. *IEEE Internet Computing*, 15(5):4–6, 2011.

[6] Yashar Deldjoo, Tommaso Di Noia, and Felice Antonio Merra. Adversarial machine learning in recommender systems: State of the art and challenges. *ArXiv e-prints*, May 2020. Under Review.

[7] Vito Walter Anelli, Yashar Deldjoo, Tommaso Di Noia, Eugenio Di Sciascio, and Felice Antonio Merra. Sasha: Semantic-aware shilling attacks on recommender systems exploiting knowledge graphs. In *The Semantic Web*, pages 307–323, Cham, 2020. Springer International Publishing.

[8] Jianfang Wang and Pengfei Han. Adversarial Training-Based Mean Bayesian Personalized Ranking for Recommender System. *IEEE Access*, 8:7958–7968, 2020. Conference Name: IEEE Access.

[9] B. Mobasher, R. Burke, R. Bhaumik, and J. J. Sandvig. Attacks and remedies in collaborative recommendation. *IEEE Intelligent Systems*, 22(3):56–63, 2007.

[10] Ihsan Gunes, Cihan Kaleli, Alper Bilge, and Huseyin Polat. Shilling attacks against recommender systems: A comprehensive survey. *Artif. Intell. Rev.*, 42(4):767–799, December 2014.

[11] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*, pages 427–436. IEEE Computer Society, 2015.

[12] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[13] Yiming Zhang, Yujie Fan, Shifu Hou, Yanfang Ye, Xusheng Xiao, Pan Li, Chuan Shi, Liang Zhao, and Shouhuai Xu. Cyber-guided Deep Neural Network for Malicious Repository Detection in GitHub. In *2020 IEEE International Conference on Knowledge Graph (ICKG)*, pages 458–465, August 2020.

[14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[15] Ferdian Thung, David Lo, and Julia Lawall. Automated library recommendation. In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pages 182–191, Oct 2013.

[16] Mohamed Aymen Saied, Ali Ouni, Houari Sahraoui, Raula Gaikovina Kula, Katsuro Inoue, and David Lo. Improving reusability of software libraries through usage pattern mining. *Journal of Systems and Software*, 145:164 – 179, 2018.

[17] M. Li, W. Wang, P. Wang, S. Wang, D. Wu, J. Liu, R. Xue, and W. Huo. Libd: Scalable and precise third-party library detection in android markets. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 335–346, 2017.

[18] Ali Ouni, Raula Gaikovina Kula, Marouane Kessentini, Takashi Ishio, Daniel M. German, and Katsuro Inoue. Search-based software library recommendation using multi-objective optimization. *Inf. Softw. Technol.*, 83(C):55–75, March 2017.

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

[20] Hao Zhong, Tao Xie, Lu Zhang, Jian Pei, and Hong Mei. MAPO: Mining and Recommending API Usage Patterns. In *23rd European Conference on Object-Oriented Programming*, pages 318–343, Berlin, Heidelberg, 2009. Springer.

[21] J. Wang, Y. Dang, H. Zhang, K. Chen, T. Xie, and D. Zhang. Mining Succinct and High-coverage API Usage Patterns from Source Code. In *10th MSR*, pages 319–328, Piscataway, 2013. IEEE.

[22] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. Deep API Learning. In *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 631–642, New York, 2016. ACM.

[23] Jaroslav Fowkes and Charles Sutton. Parameter-free Probabilistic API Mining Across GitHub. In *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 254–265, New York, 2016. ACM.

[24] Anand Ashok Sawant and Alberto Bacchelli. fine-GRAPE: fine-grained APi usage extractor – an approach and dataset to investigate API usage. *Empirical Software Engineering*, 22(3):1348–1371, June 2017.