Assignment 2

WASP Software Engineering Course

Amirhossein Ahmadian

Sep 2022

About my research

Statistical machine learning is the main area of my PhD studies, and my research is currently focused on handling Out-of-Distribution (OOD) data, that is the data that do not come from the same distribution as training data. More specifically, OOD detection (anomaly/novelty detection) has been one of the main problems that I have explored. This problem is concerned with automatic identification of data that deviate in an unusual or significant way from a reference set of "normal data". The data can be in principle of any type such as image, voice, or text, though I have worked with image data in my project so far. The OOD detection problem is addressed via deep learning methods, and particularly neural network based models, in my research. Identifying OOD inputs can be directly the problem of interest (e.g., detecting accidents and suspicious activities) as well as a way to enhance the robustness of other AI and machine learning models. As a simple example, assume that we have a face recognition system for door access control that receives input from a camera in order to detect human faces. In this case, images of human faces are basically considered as normal data, whereas image of an "animal face" is an OOD for instance that needs to be detected before propagation to later stages of the recognition system.

Topics in Software Engineering and their potential connection to my research

Automated Software Testing: Automated testing lies in the broad area of software quality assurance and involves leveraging dynamic techniques to verify software implementation. The goal in automated software testing is basically to check (in a systematic and scalable way) whether the outputs of a computer program given a set of inputs match the outputs that are expected and/or satisfy the expected requirements. Testing is usually run using a software separated from the one under the test. The inputs whose corresponding outputs are verified to this end are called the "test cases". The primary question is how to choose the test cases to be representative, and how to specify the conditions for identifying the correct outputs. The simplest strategy is to randomly take many cases from the space of possible inputs. More sophisticated ideas are around as well, such as focusing on boundary value cases (e.g., placing very large numbers in the input) and generating random inputs based on a template of user actions sequence. In the most basic form, the correct output(s) for each test case is specified by an oracle (human). A more powerful approach is property-based testing (PBT), where a set of general rules, that are required on all cases, are written by a human expert. For example, we know that a function that reserves a list of numbers should not change the length of the list. Such properties can be automatically checked on the input-output pairs by the testing tool, and a test case passes the test only if it satisfies all of them.

Machine learning software projects pose some unique challenges in software testing, in addition to the routine software testing that is done on the classic parts e.g., the user interface and I/O. In case of a software that implements ML and data science algorithms, part of the validation and verification procedures should be designed based on the literature of ML, deep learning, or

other data science area. This type of testing is often done through measuring the performance of the system on a 'test' set of data which have never been used to train or tune the system before. However, there are several potential pitfalls in choosing the test set and performance metric. For example, considering human related data, a biased sampling for the test set can lead to ignoring minority groups in measuring performance. Similarly, many performance measures (such as 'average' accuracy) do not reveal the biases or malfunction of the system with respect to the minority groups. Another common problem is 'data leakage' between training and test stages, meaning the data used for testing is not completely independent from the data used for training, which in turn can lead to over-optimistic performance results. The testing becomes even more complicated in the particular area of anomaly/out-of-distribution detection. While we usually have examples of normal (in-distribution) data, the space of abnormal (OOD) inputs can be very huge. Anomalies can have different types, such as irrelevant (noise) data, rare events, and intentional adversarial inputs [1]. It can be interesting but challenging to design software testing tools that assess the robustness of AI systems to a wide range of abnormal data.

Security and Privacy: In general, security is the methods and precautions taken to keep data safe from attackers and unauthorized users. Passwords and encryption methods are among the well-known means to software security, for example. Privacy has the security as a prerequisite, but it also aims at preventing the use of personal (human related) data for any purpose that is not consented by the user (e.g. being accessed by third-parties), or in any way that is against the state/regional regulations (e.g. does not comply with GDPR rules). In providing privacy, first it is important to understand the concept of "personal data". Personal data is any kind of data which contains "identifiers", that is, information which can be exploited to identify a person (data subject) directly or indirectly. Therefore, obvious examples of name, image, address are personal data, as well as things like IP address, browser configuration, and similar. According to the EU regulations on protecting personal data (GDPR), a data controller is a person or organization that processes some personal data, and of course it needs to have valid reasons for processing and keeping data. An agent (e.g., a tech company) that processes the personal data in order to serve the needs of data controller is called a data processor. GDPR requires that data controllers and their corresponding data processors establish agreements which guarantee protection of privacy during transfer and process of data. For instance, if we use a cloud service in our website that can access any form of users data (even ip-addresss), we need to communicate with the cloud service provider to ensure GDPR is not violated in any step.

The particular topic of anomaly detection has direct applications in computer security, to the best of my knowledge. Generally, cyber attacks and breaches can be seen as abnormal (rare/novel) events, which can have complicated patterns. Machine learning algorithms can help to detect such events (e.g., via monitoring network traffic, activity logs, etc.) automatically and with minimum delay [2]. Of course, since we relatively have few examples of positive cases (attacks do not happen regularly, fortunately), small training data can be a challenge in practice. Thinking about privacy, it is clear that protection of personal data should be a vital aspect of any machine learning application or research that deals with human subjects, particularly since today's deep models might need big training datasets collected from various resources (and possibly with different original purposes).

Human-Computer Interaction: The broad area of human-computer interaction (HCI) is concerned with studying and enhancement of interfaces that connect the human users to computer software. The word "interface" refers to any method, program, or hardware that plays the intermediate role of transmitting the user commands to the core software, and/or delivering the outputs from that to the user. Examples of an interface range from the obvious classical hardware like monitors and keyboards to the modern AI based personal voice assistants, such as Google Assistant and Microsoft Cortana. It seems generally that the trend in modern HCI favors interfaces that ask

users about their aims (i.e., "what would you like to do?") instead of the more classical approach of getting a sequence of commands explicitly (e.g., checking some check boxes and clicking some buttons). Moreover, in the modality aspect, interfaces are evolving towards more natural forms of communication which is also used in human-human interactions, such as speech, and natural language. AI and machine learning have contributed significantly in all of the mentioned directions. More specifically, natural language processing (NLP), speech recognition, and speech synthesis technologies have had (and will have) a pivotal role in changing how we interact with computers and machines [3], e.g. by simply asking questions instead of exploring documents manually. NLP's influence is not only on the input side but it has also led to more sophisticated forms of presenting outputs, like the automatic summaries that the Google search engine provides. The role of computer vision and graphics in HCI is remarkable as well [4]. Basic examples for this include the eye tracking (e.g., to help people with disabilities) and handwriting recognition systems.

Future Trends in Software Engineering

Considering my experience in machine learning field, I would like to make comments on two significant arenas in which software engineering and AI/ML are interacting with each other, and will do more in future. These two domains that I identify are (I) "SE for ML", that is adapting and extending software engineering methods to develop and deploy ML and data science models and applications, and (II) "ML for SE" which means leveraging the ML models and algorithms to improve or replace the classical software engineering methods. In the latter case, the end product is not necessarily related to ML/AI, but ML is used to speed up or optimize the software production process. Whereas the course contents were more focused on the topic (I), I will begin the discussion and elaborate mostly in the opposite direction, that is "ML for SE".

I believe that one of the most impressive developments in this line are the recent deep learning based automatic code completion and generation tools, that is related to the informal idea of "ML will eat software". The recently introduced GitHub Copilot [5] is a sophisticated AI based code auto-completion tool, which can be used with a number of programming languages like Python and JavaScript. Copilot suggests lines and snippets of code to the programmer based on the comments (in natural language) and some uncompleted code written by the programmer. The deep learning model at the core of Copilot is OpenAI Codex [6], a variant of the GPT-3 language model, trained on billions of lines of source code. I believe that the market of such intelligent coding assistant tools will continue to grow, given the rising interest in programming careers and the diversities in programmers' backgrounds. As more and more people are attracted to software development around the world, we see more beginners and ones coming from backgrounds other than computer science or engineering in the field. Tools like Copilot can be a good help for such groups to make progress in their career, particularly because they help in the steps that need some algorithms or database theory knowledge (e.g. sorting functions). The other group who can benefit much from such tools are the free-lance programmers that would like to focus on some core area (data science, for example) while developing an application for end-users. The load of many routine functionalities such as user interface and I/O can be much reduced by automatic coding, letting the programmer to focus on their main expertise. Of course, programmers can already find tons of code snippets and examples on the web, but using Copilot-fashion tools is much more convenient in terms of integrating the external code and tuning it rapidly for the specific project requirements. Nevertheless, I can see some potential social/ethical downsides of AI based automatic programming tools. Relying on these tools blindly can leave us with programmers who do not know how their own code works exactly, leading to various security risks and harmful bugs.

Machine learning has been employed in a number of other areas of SE as well, including quality assurance and requirement analysis [7]. An example for the former is ML based bug prediction,

formulated as a binary classification problem [8]. A classifier can detect whether a proposed change in a source code (i.e., a "commit" operation) will create a bug in the program, given the history of previous changes. Another example is the approaches that analyze the control flow of programs based on probabilistic models in order to create optimal test cases [9]. I expect this type of ML applications in SE to get more attention in coming years, due to the recent advancements in deep learning (which can basically solve more complicated tasks) and the availability of vast amount of public code repositories (which can be used as training data). However, I believe that the role of software engineers cannot be totally replaced by ML models in such tasks, considering the extreme complexity (high dimensionality) of the space representing computer programs. Deep learning models are still not very good in generalization and dealing with novel cases in such spaces, which means human supervisors cannot be excluded, at least in the near future.

"SE for ML" is the other direction in research and industry, which seems active and interesting as well. My impression is that in this context, both the SE and ML community will need to move towards more "standardization" in adopted procedures, frameworks, and benchmarks. For instance, looking at the deep learning research implementations, we observe that a large part of the code is about the non-core aspects of the software such as data preprocessing and managing configurations, which are implemented differently by different developers. This can be addressed by promoting protocols on data processing pipelines and other side components of ML-centric software. Another important and challenging aspect is the quality assurance for the software which rely on ML and deep learning, particularly due to their data-driven nature. Any industrial protocol for testing such software should clearly specify the data collection, labeling, and processing stages as well as appropriate performance measures, and their interpretation, taking high-level goals into account.

References

[1] Chalapathy, Raghavendra, and Sanjay Chawla. "Deep learning for anomaly detection: A survey" arXiv preprint arXiv:1901.03407, 2019.

[2] Ahmed, Mohiuddin, Abdun Naser Mahmood, and Jiankun Hu. "A survey of network anomaly detection techniques" Journal of Network and Computer Applications, 2016.

[3] Panda, Soumya Priyadarsini. "Automated speech recognition system in advancement of human-computer interaction." International Conference on Computing Methodologies and Communication (ICCMC), IEEE, 2017.

[4] Jaimes, Alejandro, and Nicu Sebe. "Multimodal human–computer interaction: A survey." Computer vision and image understanding, 2007.

[5] https://github.com/features/copilot

[6] Chen, Mark, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards et al. "Evaluating large language models trained on code." arXiv preprint arXiv:2107.03374, 2021.

[7] Shafiq, Saad, Atif Mashkoor, Christoph Mayr-Dorn, and Alexander Egyed. "Machine learning for software engineering: A systematic mapping." arXiv preprint arXiv:2005.13299, 2020.

[8] Shivaji, Shivkumar, E. James Whitehead, Ram Akella, and Sunghun Kim. "Reducing features to improve bug prediction." IEEE/ACM International Conference on Automated Software Engineering, 2009.

[9] Baskiotis, Nicolas, Michele Sebag, Marie-Claude Gaudel, and Sandrine-Dominique Gouraud. "A Machine Learning Approach for Statistical Software Testing." International Joint Conference on Artificial Intelligence, 2007.