WASP Software Engineering

Tianyi Zhou

August 2023

1 Abstract of My Research

I am interested in theoretical computer science, data mining, and machine learning. My research focuses on graph algorithms and online media data mining and I am interested in designing efficient algorithms for various data mining tasks with potential real-world applications. To be more concrete, I am focusing on design an algorithmic framework for reducing bias and polarization in online media, like Twitter, Reddit, and other media outlets. Online media is an important part of modern information society, offering a place for public discussion and expressing opinions for billions of individuals. Online media are often credited for breaking information barriers and promoting diversity and democracy. However, the opposite effect is often observed in practice: users tend to favor content that agrees with their existing opinions and avoid conflicting viewpoints, and eventually create increased polarization. Without mediation, current online media will gravitate towards a state in which people are constantly reinforcing their existing opinions. My research focuses on developing a theoretically solid and efficient set of algorithmic techniques to address deficiencies in today's online media. My research is structured along with knowledge discovery, exploration, and content recommendation. In order to accomplish this aim, my research will formulate novel problem representations that provide a deep understanding of the undesirable phenomena observed in online media. In our research project, we will address issues of reducing bias and polarization and creating awareness among users to explore alternative viewpoints. We will also study the effect of different design features on the willingness of the users to explore viewpoints that conflict with their opinions. My research is founded by the ERC project REBOUND.

2 Topics of Lecture

The first interesting topic I would like to talk about it quality assurance and testing in software engineering. I found the concepts in this topic are also important and widely used in machine learning. Verification is about checking of whether we are building the product right while validation is about checking of whether we are building the right product. These two concepts are similar to the concept in PAC learning theory. We aim to find good models for the task, and we need to make sure that we are going to the right way during training. In software engineering, the reason of why we need quality assurance and testing is that the software itself is complex. And people care about the cost of failure for economic and safety reasons. Similar to the research in theoretical computer science, my research cares about the complexity of the problem and the algorithms. Many real-world problems could be abstracted to optimization problems and solved by algorithms. The complexity of the problem may be NP-hard and approximation algorithms are a common family of techniques for solving such hard problems with theoretical quality assurance, namely the lower bound and upper bound of the worst case. I found it interesting since the paradigms used in software quality assurance and test and algorithms analysis share lots of similarities in nature. It raises to an open question, is there some techniques from one field that we could transfer it to another such that difficult problems could be solved? For example, the algorithm analysis techniques may be used to determine the minimum required effort for software testing if we abstract the software structure into graphs. The second interesting topic I would like to talk about is software engineering for AI and ML. It is widely observed that AI systems nowadays are not only about selecting the best models, but most of the efforts also fall into the engineering level. As pointed out form the lecture, the most frequent quality goals of AI-based software systems care for safety, functional robustness and reliability. My research focus on algorithmic and behavioral aspects of the social systems. I believe that algorithmic outcomes should not only care about complexity and efficiency, it should also consider the effect in the system level. That is, if we employ it in real-world systems, is it safe enough to do so? Is it robust enough in real-world systems? In one of my works, we designed an approximation algorithm to connect users who share similar interests on Twitter. A result we aim to scale the algorithm to large-scale of networks with millions of nodes. We care about the position of the algorithms (AI model) in the social systems and their robustness and safety. I believe that we as a researcher who does algorithmic research also need to think in a systematic way. The concepts

and principles of software engineering are very important to system and algorithm design.

3 Topics of Guest Lecture

In the first guest lecture, the boundary values test reminds me the online judge system in the algorithm courses that I take. That is, students are supposed to write code to solve an algorithm problem. The submitted code should pass all test cases. Students should be careful during programming and think about the boundary cases. If one mistakenly forgets some boundary cases, the whole code may be wrong and she at the risk of rewriting everything. On the other hand, if one could identify all boundary cases, it would also guide us to the right solution since the boundary is a variant of the constraint of the solution. In the field of algorithm analysis and design, it is also very important to take care of all boundary cases. But the algorithms themselves may be very complicated and hard to analyze. It is very risky to implement an algorithm or system without test. Thinking about boundary values is usually simpler than coming up with theoretical proof. In practice, we as algorithm designers also test the desired outcomes of the system and set up boundary cases corresponding to constraints to control the risk. The tools and techniques used in traditional software engineering would inspire the topic in algorithm verification and test.

I would also talk about sentiment analysis in the third guest lecture since my own research is highly related to this task. Sentiment analysis, stance detection, and opinion dynamics are broadly studied in machine learning. Empowered by machine learning models, the rich textual datasets of online media could help us to understand the underlying patterns. For example, we could consider sentiment detection as a classification task with text as input and sentiment as output. Other similar tasks include entity extraction, stance detection etc. We refer this kind of task as data annotation and the results can be used for downstream tasks. In this lecture, general-purpose sentiment analysis tools from NLP community were mentioned. But domain-specific tools are needed in different scenarios. As shown in the lecture, software engineering domain-specific tools show its powerful performance in the corresponding domain. Recently, the pre-trained large language models have outperformed other counterparts in various domains and show a potential application of zero-shot learning. It would be also interesting to fine-tune the language model for tasks like sentiment analysis and stance detection.

4 Research Opportunities

I would like to elaborate on **quality assurances** since I find it is super important in software engineering. The definition of quality assurance is any systematic process of determining whether a product or service meets specified requirements. As shown in the lecture, we need validation and verification to make sure that we are building the right product and building the product right. This opens many research opportunities in machine learning systems in software engineering. Abdullah and Gias et al.[1] studied the quality assurance challenges for machine learning software applications. They propose a taxonomy of machine learning software application quality assurance issues by mapping the various ML adoption challenges in different phases. The software development life cycle phases include requirement collection, design, testing, deployment, and maintenance. Various challenges are listed in corresponding phases. Combined with my research interests, a potentially interesting research topic in quality assurance-related design and implementation would be graph mining and neutral language processing. During the implementation phase, different software modules and functions would have dependencies on other packages. The complexity would grow when the software project gets larger. It is hard to manually control the dependency risk and bugs raised by the complex nature. One potential solution is to minimize the unnecessary dependencies and remove the redundancy. We could naturally represent the function dependencies as a graph and introduce graph theory and graph data mining techniques to find potential risks in the current structure. One could also formalize the objective function to minimize or maximize the desired output by design constraints. At the application level, tools like auto-completion and bug detection are widely used to accelerate the productivity in development phase. A potential application of quality assurance in the implementation phase would be to integrate the graph data mining models to help developers identify potential redundancy and risky dependencies. On the other hand, neutral language processing techniques can also be used on source code. It could help to detect potentially redundant functions in the source code by comparing embedding the dependency information and its functionalities. The results could help developers to simplify the software structure and reduce the risk. Ideally, this kind of machine learning tool for quality assurance could be part of the modern AI-powered IDE. Similar products like the GPT-based copilots on GitHub. Machine learning tools can also be used in the testing phase. Hooda and Chhillar et al[2] study the test case optimization and redundancy reduction using neural networks. This line of research shows

the potential of applying machine learning techniques to improve quality assurance.

Maintenance and Evolution is another topic that I would like to elaborate on in detail. According to the definition, maintenance refers to enhancements and modifications required to keep the software continually responsive while evolution refers to a change from a basic to a more advanced state of the software. As shown in [3], the proportion of software maintenance costs is over 70% percent. A concrete example would be open-source software. The community-based large-scale social coding pattern drives the evolution of the software to advanced states and a lot of effort from the community contributes to maintenance. In practice, people may overlook the cost of maintenance and only focus on the cost of design, implementation, and test. Software maintenance I wildly studied, and it is particularly important in safety-critical and mission-critical applications where failure may incur huge cost [4]. Code analysis tools used for maintenance prediction would be an interesting potential research direction. Predictive maintenance would be possible empowered by machine learning and AI models. It could help to determine the strange condition and estimate when the maintenance should be done. Ideally, we should solve the abnormal problem before the failure happens and reduce the cost of maintenance. Maintenance anomaly detection methods could be a potential direction to explore. In the evolution of software, AI-based tools like dev-bot could be powerful tools to increase productivity. For example, in the open-source community, dev-bots are widely used to merge requests and answer simple questions. With the success of large language models like ChatGPT, more intelligence-intensive works can be done by dev-bots. It has the potential to summarize the state of the software and answer questions when software is large and hard to maintain. In open-source communities, it is very important to engage more developers. AI-based tools could help newcomers to start to get familiar with the project in smoothly by providing tutorials and answering questions immediately. Such tools have great business value.

At the end, I would like to talk about my thoughts about the future trends of ML/AI engineering and software engineering. I think the two show an upward spiral. ML and AI are not only about accuracy but also safety and robustness in real-world applications. Building AI systems borrows the idea and experience from software engineering. The advance of AI tools helps to overcome difficulties in traditional software engineering. My research focuses on the algorithmic aspect of machine learning and AI, but I realized that engineering thinking is also important. In my future career, engineering thinking should be an essential skill for algorithm and ML designers.

References

[1] Alamin, Al., Abdullah, Md., and Uddin, Gias. "Quality assurance challenges for machine learning software applications during software development life cycle phases." *2021 IEEE International Conference on Autonomous Systems (ICAS).* IEEE, 2021.

[2] Hooda, Itti, and Chhillar, R. S. "Test case optimization and redundancy reduction using GA and neural networks." *International Journal of Electrical and Computer Engineering*, 8(6), 5449, 2018.

[3] Koskinen, Jussi. "Software maintenance costs." *Information Technology Research Institute, ELTIS-Project University of Jyväskylä*, 2003.

[4] Lenarduzzi, Valentina, Sillitti, Alberto, and Taibi, Davide. "A survey on code analysis tools for software maintenance prediction." *Proceedings of 6th International Conference in Software Engineering for Defence Applications: SEDA 2018*, 6, Springer International Publishing, 2020.