WASP

Software Engineering and Cloud Computing 2023

Software Engineering Applied to: Machine Learning for Data-driven Decisions

Author:
Alan LAHOUD

Subject area:
Computer Science

June 30, 2023

1 Introduction to my research

The topic of my research encompasses the combination of structured and constrained optimization problems with machine learning. More specifically, the aim is to solve data-driven optimization problems that are dependent on unseen observations, where those observations can be modelled using training data. Traditional solutions to these problems involve predicting the unknown parameters of the optimization problem first, and then solve the predicted optimization problem via mathematical programming, i.e., using a suitable mathematical solver to the specific OP. However, this approach can face significant challenges, depending on the problem structure and data availability. One major challenge is that even small prediction errors can lead to significant errors in the output of the optimization problem. This is because the error propagation can be uncertain, resulting in traditional approaches having poor performance.

Additionally, in some scenarios, there may be little or no labeled data available for the optimization problem parameters, making traditional approaches impossible to implement. To address these challenges, an alternative approach involves directly learning the optimization problem's result from historical data (if it is available). However, a critical problem with this approach is that the model's output is typically not constrained, making it difficult to satisfy the constraints of the structured and given optimization problem.

Our proposed solution is to improve upon state-of-the-art methods that focus on learning the embedded space between a learnable model and a mathematical programming layer. By doing so, we aim to constrain the model's output to satisfy the optimization problem's constraints while ensuring improved performance in scenarios with limited or uncertain data. This approach involves training a learnable model that takes in historical data as input and outputs a solution that satisfies the optimization problem's constraints. The model is then incorporated into a mathematical programming layer to ensure that the solution satisfies the constraints while minimizing the optimization objective. In summary, our research aims to improve upon traditional methods for solving data-driven optimization problems by incorporating machine learning techniques that constrain the model's output and improve performance in scenarios with limited or uncertain data.

First, we can split the applications of Software Engineering into two parts. The first is the applications directly to the research methods I use in my daily research routine. The second is the applications of the Software Engineering techniques into a product that is a consequence of my research, i.e., as if an ML product was designed based on his research. While Software Engineering techniques are essential for the maintenance of ML products, we will focus this report on the first application, i.e., the research itself.

2 Topics from the lecture

2.1 Code Versioning

The first point to incorporate in the research is code versioning methodology. Code versioning can offer numerous advantages. One of the main benefits is reproducibility. As research often involves iterative and complex processes, tracking changes and reproducing in my research becomes essential. By versioning the code, I can also go back to earlier versions or branches of the code I have done, making it easier to understand how your project has evolved and also to access the original submitted version of the experimental part of the research paper. This is especially beneficial when working with data-driven optimization problems, where outcomes are contingent on a specific combination of parameters and data. When I am dealing with synthetic data, I also generate different size of training data, different types of noise input. In my research it is extremely important to deal with this input variations in an intelligent manner with versioning control. Additionally, if I am collaborating with supervisors or others, versioning allows everyone to work on different aspects of the project simultaneously without overwriting each other's contributions. This can significantly increase the efficiency of the research project.

2.2 Code Testing

The second software engineering concept that is extremely important to my research is code testing. In the context of research on structured and constrained optimization problems with machine learning, code testing plays significant role due to the creation and adjustment of complex models. In this case, even a small bug can lead to erroneous results or inefficient performance (e.g., Combinatorial Optimization Problems can be very sensible to input differences/noise). Incorporating code testing into my daily research routine allows us to catch such errors early on. This way, I can identify and rectify issues before they compromise results or lead the algorithm to output incorrect results. One common approach is unit testing, where individual units of the code (e.g., functions or methods) are tested by itself isolated. This can be particularly useful when developing new algorithms or implementing specific aspects of your machine learning models. By ensuring that each unit performs correctly, it is possible to proceed and build on them knowing that they work as expected.

Specifically to Data-driven Optimization Problems, the techniques of Boundary Value Analysis and Checking Inverse Relationships can be extremely relevant and useful to guarantee the results are according to the expectations. In my research, for example, it would be essential to use two different solvers. Approximated solvers are differentiable and optimal solvers are normally non/differentiable. Assure that the difference between them is always less than a threshold. Moreover, we could have an oracle, at least for simple inputs of the Optimization Problems, that researchers (me, in this case) know the optimal result and then compare with both to guarantee that both the optimal and the approximated solvers are aligned with the oracle at least in the simple scenarios.

From the Dobslaw lecture, a specific technique for testing would be extremely useful in my research is the "Diversity Foundation". The relation between inputs and outputs are of extremely value in the area of Optimization Problem and Mathematical Programming. Among the distances propoed by Dobslaw in his lecture, the idea also reminds a theoretical mathematical condition extremely used in Machine Learning that is called Lipschitz Continuity condition. This condition says that when there is a variation in the input of a function, the output difference should not be more than L times the input difference. I assume this condition or a modified version of this condition can also be used as one of the checking conditions of the program as a whole.

2.3 Sentiment Analysis

Furthermore, the sentiment analysis presented by Calefato could also be used in the optimization problems itself. In a problem where I am optimizing for customer satisfaction, for example, sentiment analysis could provide a more accurate measure of satisfaction than traditional metrics, thus helping optimizing more effectively. The application of sentiment analysis can also assist in understanding the potential implications and real-world impact in my research. By understanding the sentiment and subjective experiences of people involved in or affected by the specific and chosen optimization problem, it is better to argue on the significance and potential applications of the proposed work.

3 Topics of Software Engineering and Literature

3.1 Automated Software Testing

Automated Software Testing is when the software testing such as comparisons of expected and provided outcomes from functions or programs are performed automatically. Automated software testing is normally done by using specific tools or designed programs only for test comparisons. Regarding automated testing applied to my research, I can see a good potential fo it specifically in the area of Mathematical Programming. Automated testing can be used to validate the mathematical models. By comparing the output of a model with expected results, testing frameworks can provide insights into the accuracy and reliability of the model. They can also capture the input region (i.e., optimization problem parameters) such that lead to more problematic outputs such as infinite values or infeasible solutions due to approximation problems. Specifically o my area (data-driven optimization problems), it is common to deal with some elements of randomness or elements with stochastic nature. Automated testing can be designed to handle multiple runs and verify the consistency in a Monte Carlo fashion.

It seems that the research paths of Automated Software Testing and Mathematical Programming or Artificial Intelligence in general are more towards the applications of those mathematical tools into the Software Testing instead of researching techniques of software testing into the development of Machine Learning or Mathematical Programming tools. As an example, [3] conducted a literature review of various AI techniques and algorithms used in software testing, including decision trees, support vector regression, hybrid genetic algorithms, k-means clustering, and natural language processing. These techniques have been applied to various aspects of software testing, such as refining black-box test specifications, predicting coverage in automated testing, enhancing regression testing, identifying infeasible GUI test cases, and detecting duplicate defect reports. The mentioned paper highlights the advantages of AI in software testing, such as reducing the software development life cycle, increasing efficiency, and improving the quality of software products.

A similar research direction is demonstrated by [1]. The authors identified more than hundred studies published between 1991 and 2016, which they classified based on the software testing activity they support, the ML algorithm used, the type of software under test, and the quality of the empirical evidence provided. This paper shows that most of the studies presented focus on automating test case and test data generation, with decision trees being the most commonly used ML algorithm. The authors also found that the majority of the studies provide preliminary empirical evidence to support their claims, but there is a clear need for more robust empirical studies. In sum, it is possible to see from an overview of the literature that it is more common to apply ML and AI into software testing than the other way around, leaving space to see how Automated Software Testing can be used to help on assuring robustness into the Mathematical Programming and ML field.

3.2 Quality Assurance

Quality assurance is an essential step for any research or product development. This include the area of Machine Learning and Mathematical Programming for Optimization Problems. Quality assurance involves the systematic monitoring and evaluation of the project pieces to make sure that the quality of the product as a whole satisfies the planned requirements. In mahine learning, for example, the quality of the data used for training models is important and crucial. Quality Assurance can be also used to make sure that the data is reliable and free from bias or errors.

Quality assurance can also validate the performance of machine learning models by checking if it performs as expected and that it's generalizing well with noise or slightly different data.

From a perspective of Mathematical Programming, Quality Assurance can be used to verify the solution of the mathematical solver. This can involve checking that the solution meets all the constraints, that it's optimal or near-optimal, and that it's feasible in the real world.

The authors in [2] propose a framework for evaluating the quality of AI-based systems. The proposed framework evaluate the quality of the system based on data integrity, model robustness, system quality, costumer expectation and process agility. The data integrity checks if the data is accurate, consistent and representative in the domain. The model robustness involves the evaluation of the model performance through different input data and checking if it is resilient to adversarial attacks. System Quality focuses on the reliability and security of the system as a whole. Costumer Expectation checks if the system satisfy the expectation of the costumers by taking human feedback. Finally, the process agility assess the system's flexibility and how the system respond to changes in the requirements.

The proposed framework can be used as a guideline for ML researchers when designing a new idea. Checking some of these aspects might be extremely relevant to ensure that the final product derived from the research would be indeed a high-quality product.

4 Future Directions

Over the next 5-10 years, the trend of big ML and AI products such as autonomous vehicles and large language models is likely to continue. This leads to the need of professionals who guarantees the quality and robustness of these models. In other words, I can see an important demand for software engineering staff who are not only good in technical aspects but also knowledgeable in AI in general to be part of the AI change and advancements.

Another trend is the challenge to apply AI to the health domain. This is due to the fact that, although Neural Networks has a big hype due to their advances in Machine Learning, Deep Learning and AI in general, there is a lack of explainability in thee models. This leads to a growing demand for techniques and software tools that can help humans understand AI systems and rely on them to apply in health systems in the daily life.

References

[1] V. H. Durelli, R. S. Durelli, S. S. Borges, A. T. Endo, M. M. Eler, D. R. Dias, and M. P. Guimarães. Machine learning applied to software testing: A systematic mapping study. *IEEE Transactions on Reliability*, 68(3):1189–1212, 2019.

[2] G. Fujii, K. Hamada, F. Ishikawa, S. Masuda, M. Matsuya, T. Myojin, Y. Nishi, H. Ogawa, T. Toku, S. Tokumoto, et al. Guidelines for quality assurance of machine learning-based artificial intelligence. *International journal of software engineering and knowledge engineering*, 30(11n12):1589–1606, 2020.

[3] H. Hourani, A. Hammad, and M. Lafi. The impact of artificial intelligence on software testing. In *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, pages 565–570. IEEE, 2019.