Report from Dagstuhl Seminar 20091

SE4ML – Software Engineering for AI-ML-based Systems

Edited by

Kristian Kersting[1], Miryung Kim[2], Guy Van den Broeck[3], and Thomas Zimmermann[4]

1 **TU Darmstadt, DE,** kersting@cs.tu-darmstadt.de
2 **UCLA, US,** miryung@cs.ucla.edu
3 **UCLA, US,** guyvdb@cs.ucla.edu
4 **Microsoft Corporation – Redmond, US,** tzimmer@microsoft.com

Abstract

Multiple research disciplines, from cognitive sciences to biology, finance, physics, and the social sciences, as well as many companies, believe that data-driven and intelligent solutions are necessary. Unfortunately, current artificial intelligence (AI) and machine learning (ML) technologies are not sufficiently democratized – building complex AI and ML systems requires deep expertise in computer science and extensive programming skills to work with various machine reasoning and learning techniques at a rather low level of abstraction. It also requires extensive trial and error exploration for model selection, data cleaning, feature selection, and parameter tuning. Moreover, there is a lack of theoretical understanding that could be used to abstract away these subtleties. Conventional programming languages and software engineering paradigms have also not been designed to address challenges faced by AI and ML practitioners. In 2016, companies invested $26–39 billion in AI and McKinsey predicts that investments will be growing over the next few years. Any AI/ML-based systems will need to be built, tested, and maintained, yet there is a lack of established engineering practices in industry for such systems because they are fundamentally different from traditional software systems.

This Dagstuhl Seminar brought together two rather disjoint communities together, software engineering and programming languages (PL/SE) and artificial intelligence and machine learning (AI-ML) to discuss open problems on how to improve the productivity of data scientists, software engineers, and AI-ML practitioners in industry.

Seminar February 23–28, 2020 – http://www.dagstuhl.de/20091
2012 ACM Subject Classification Software and its engineering, Computing methodologies → Artificial intelligence, Computing methodologies → Machine learning
Keywords and phrases correctness / explainability / traceability / fairness for ml, data scientist productivity, debugging/ testing / verification for ml systems
Digital Object Identifier 10.4230/DagRep.10.2.76

1 Executive Summary

Kristian Kersting (TU Darmstadt, DE)
Miryung Kim (UCLA, US)
Guy Van den Broeck (UCLA, US)
Thomas Zimmermann (Microsoft Corporation – Redmond, US)

Any AI- and ML-based systems will need to be built, tested, and maintained, yet there is a lack of established engineering practices in industry for such systems because they are fundamentally different from traditional software systems. Building such systems requires extensive trial and error exploration for model selection, data cleaning, feature selection, and parameter tuning. Moreover, there is a lack of theoretical understanding that could be used to abstract away these subtleties. Conventional programming languages and software engineering paradigms have also not been designed to address challenges faced by AI and ML practitioners. This seminar brainstormed ideas for developing a new suite of ML-relevant software development tools such as debuggers, testers and verification tools that increase developer productivity in building complex AI systems. It also discussed new innovative AI and ML abstractions that improve programmability in designing intelligent systems.

The seminar brought together a diverse set of attendees, primarily coming from two distinct communities: software engineering and programming languages vs. AI and machine learning. Even within each community, we had attendees with various backgrounds and a different emphasis in their research. For example, within software engineering the profile of our attendees ranged from pure programming languages, development methodologies, to automated testing. Within, AI, this seminar brought together people on the side of classical AI, as well as leading experts on applied machine learning, machine learning systems, and many more. We also had several attendees coming from adjacent fields, for example attendees whose concerns are closer to human-computer interaction, as well as representatives from industry. For these reasons, the first two days of the seminar were devoted to bringing all attendees up to speed with the perspective that each other field takes on the problem of developing, maintaining, and testing AI/ML systems.

On the first day of the seminar, Ahmed Hassan and Tim Menzies represented the field of software engineering. Their talks laid the foundation for a lot of subsequent discussion by presenting some key definitions in software engineering for machine learning (SE4ML), identifying areas where there is a synergy between the fields, informing the seminar about their experiences dealing with industry partners, and listing some important open problems. Sameer Singh and Christopher Ré took care of the first day's introduction to machine learning. Christopher Ré described recent efforts in building machine learning systems to help maintain AI/ML systems, specifically for managing training data, and monitoring a deployed system to ensure it keeps performing adequately. Sameer Singh's talk focused on bug finding, and debugging machine learning systems, either by inspecting black-box explanations, generating realistic adversarial examples in natural language processing (NLP), and doing behavioral testing of NLP models to make them more robust.

The second day of the seminar continued to introduce the attendees to some prominent approaches for tackling the SE4ML problem. Elena Glassman presented her work at the intersection of human-computer interaction and software engineering, while Jie Zhang gave an overview of software testing for ML, based on her recent survey of the field. Significant attention during the seminar was spent on the problem of deploying machine learning models in environments that change over time, where the behavior of the AI/ML system diverges from the intended behavior when the model was first developed. For example, such issues were discussed by Barbara Hammer in her talk on machine learning in non-stationary environments. Isabel Valera introduced the seminar to another important consideration when developing AI/ML-based systems: interpretability and algorithmic fairness. Andrea Passerini's talk was aimed at explaining some of the basic principles of machine learning for a non-machine learning audience; for example generalization, regularization, and overfitting, as well as some recent trands in combining learning with symbolic reasoning.

The remainder of the seminar was centered around various breakout sessions and working groups, including sessions on (1) Specifications and Requirements, (2) Debugging and Testing, (3) Model Evolution and Management, and (4) Knowledge Transfer and Education. There were extended discussions on the question “what is a bug?” in an AI/ML setting, what is a taxonomy of such bugs, and can we list real-world examples of such bugs happening in practice. Interleaved with these working groups, there were several demand-driven talks, designed to answer questions that came up during the discussions. For example, Steven Holtzen and Parisa Kordjamshidi introduced the seminar to efforts in the AI community to build higher-level languages for machine learning, in particular probabilistic programming and declaritive learning-based programming. Christian Kästner shared his insights from teaching software engineering for AI/ML-based systems using realistic case studies. Molham Aref gave his unique view on developing such systems from industry, which was a tremendously valuable perspective to include in these discussions.

Overall, this seminar produced numerous new insights into how complex AI-ML systems are designed, debugged, and tested. It was able to build important scientific bridges between otherwise disparate fields, and has spurred collaborations and follow-up work.

2 Table of Contents

3 Overview of Talks

3.1 Machine Learning in non-stationary environments

Barbara Hammer (Universität Bielefeld, DE)

Joint work of Michael Biehl, Viktor Losing, Benjamin Paassen, Alexander Schulz, Heiko Wersing

One of the main assumptions of classical machine learning is that data are generated by a stationary concept. This, however, is violated in practical applications e.g. in the context of life long learning, for the task of system personalisation, or whenever sensor degradation or non-stationary environments cause a fundamental change of the observed signals. Within the talk, we will give an overview about recent developments in the field of learning with concept drift, and we will address two particular challenges in more detail: (1) How to cope with a fundamental change of the data representation which is caused e.g. by a misplacement or exchange of sensors? (2) How to deal with heterogeneous concept drift, i.e. mixed rapid or smooth, virtual or real drift, e.g. caused by a real-life non-stationary environment? We will present novel intuitive distance-based classification approaches which can tackle such settings by means of suitable metric learning and brain-inspired adaptive memory concepts, respectively, and we will demonstrate their performance in different application domains ranging from computer vision to the control of protheses.

References

1 Viktor Losing, Taizo Yoshikawa, Martina Hasenjäger, Barbara Hammer, Heiko Wersing: Personalized Online Learning of Whole-Body Motion Classes using Multiple Inertial Measurement Units. ICRA 2019: 9530-9536
2 Michiel Straat, Fthi Abadi, Christina Göpfert, Barbara Hammer, Michael Biehl: Statistical Mechanics of On-Line Learning Under Concept Drift. Entropy 20(10): 775 (2018)
3 Viktor Losing, Barbara Hammer, Heiko Wersing: Incremental on-line learning: A review and comparison of state of the art algorithms. Neurocomputing 275: 1261-1274 (2018)
4 Benjamin Paaßen, Alexander Schulz, Janne Hahne, Barbara Hammer: Expectation maximization transfer learning and its application for bionic hand prostheses. Neurocomputing 298: 122-133 (2018)
5 Viktor Losing, Barbara Hammer, Heiko Wersing: Tackling heterogeneous concept drift with the Self-Adjusting Memory (SAM). Knowl. Inf. Syst. 54(1): 171-201 (2018)
6 Viktor Losing, Barbara Hammer, Heiko Wersing: Self-Adjusting Memory: How to Deal with Diverse Drift Types. IJCAI 2017: 4899-4903
7 Viktor Losing, Barbara Hammer, Heiko Wersing: Personalized maneuver prediction at intersections. ITSC 2017: 1-6

3.2 Data Driven Decision Making for the Development of Trustworthy Software

Ahmed E. Hassan (Queen's University – Kingston, CA)

Software systems produce an enormous amount of rich data while being used (e.g., crashes, logs, telemetry data, and user reviews) and while being developed (e.g., historical code changes, test results, and feature requests). Leveraging such rich data through machine learning (ML), we can deliver better software in a cost-effective manner.

In this talk, I share my team's experience working closely with industrial partners over the past decade to address software development and operation (e.g., AIOps) challenges using ML. Then I discuss essential technical and non-technical goals to ensure the long-term successful integration of such ML solutions into daily practice.

3.3 Probabilistic Programming

Steven Holtzen (UCLA, US)

This talk provides a gentle introduction to probabilistic modeling and probabilistic programs. First, we ask what is a probabilistic program and how can they be used to solve problems? After some motivating examples, we discuss challenges in automating probabilistic inference. We highlight several example probabilistic programming languages and their diverse approaches to probabilistic inference, including (1) Stan [1], (2) Problog [2], (3) Dice [3], and (4) Figaro [4]. We close with a discussion of existing systems and prospects for integrating probabilistic programs and software engineering.

References

1 Bob Carpenter, Andrew Gelman, Matt Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Michael A Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2016. *Stan: A probabilistic programming language.* Journal of Statistical Software (2016)

2 Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. 2013. *Inference and learning in probabilistic logic programs using weighted Boolean formulas.* J. Theory and Practice of Logic Programming 15(3) (2013), 358 – 401.

3 Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2020. *Dice: Compiling Discrete Probabilistic Programs for Scalable Inference.* arXiv preprint arXiv:2005.09089.

4 Avi Pfeffer. 2009. *Figaro: An object-oriented probabilistic programming language.* Charles River Analytics Technical Report 137 (2009).

3.4 Declarative Learning-Based Programming as an Interface to AI Systems

Parisa Kordjamshidi (Michigan State University – East Lansing, US)

Joint work of Parisa Kordjamshidi, Kristian Kersting, Dan Roth
Main reference Parisa Kordjamshidi, Dan Roth, Kristian Kersting: "Declarative Learning-Based Programming as an Interface to AI Systems", CoRR, Vol. abs/1906.07809, 2019.
URL https://arxiv.org/abs/1906.07809

Data-driven approaches are becoming more common as problem-solving techniques in many areas of research and industry. In most cases, machine learning models are the key component of these solutions, but a solution involves multiple such models, along with significant levels of reasoning with the models' output and input. Current technologies do not make such techniques easy to use for application experts who are not fluent in machine learning nor for machine learning experts who aim at testing ideas and models on real-world data in the context of the overall AI system. We review key efforts made by various AI communities to provide languages for high-level abstractions over learning and reasoning techniques needed for designing complex AI systems. We classify the existing frameworks based on the type of techniques and the data and knowledge representations they use, provide a comparative study of the way they address the challenges of programming real-world applications, and highlight some shortcomings and future directions.

3.5 Teaching Software Engineering for AI-enabled Systems

Christian Kästner (Carnegie Mellon University – Pittsburgh, US)

Main reference Christian Kästner, Eunsuk Kang: "Teaching Software Engineering for AI-Enabled Systems", 2020.
URL https://arxiv.org/abs/2001.06691.9

Software engineers have significant experience to offer when building intelligence systems, drawing on decades of methods for building systems that scale and are robust, even when built on unreliable components. Systems with AI/ML components raise new challenges and require careful engineering, for which we designed a new course. We specifically go beyond traditional ML courses that teach modeling techniques under artificial conditions and focus on realism with large and changing datasets, robust and evolvable infrastructures and requirements engineering that considers also ethics and fairness. We share all course material

- Software Engineering for AI-Enabled Systems (SE4AI)
  https://ckaestne.github.io/seai/
- Software Engineering for AI/ML – An Annotated Bibliography
  https://github.com/ckaestne/seaibib

3.6 SE for (AI+SE)

Tim Menzies (North Carolina State University – Raleigh, US)

What should this community tell the world abut SE and AI? What are our "seven deadly sins" and our "dozen" best practices?

To answer these questions, I offer (tiny) summaries of SE and AI practice. The focus here will be "what are the surprises?", i.e., what are the *new* things we know *now* that we didn't know before. For example

- "Programmers" do much more than programming. And in fact, social factors between programming can be just as predictive for bugs as any programming language feature.
- Some (not all) SE data is inherently low dimensional and we can exploit that great benefit.

For more on this talk, see http://tiny.cc/se4ml

3.7 Some Machine Learning Basics + Random Stuff

Andrea Passerini (University of Trento, IT)

I will give a quick overview of the basic concepts used in machine learning from generalisation to regularized loss minimizations to model selection. I will quickly present how to use the scikit learn framework to train multiple classifiers and give a bird's eye view of deep learning. I will end up presenting some work of mine focused on the combination of learning and constraints.

3.8 Experiences Building & Maintaining Software 2.0 Systems.

Christopher Ré (Stanford University, US)

This talk describes our group's recent working building and maintaining a new breed of ML software systems. The talk focusses on how engineer time is spent in building and maintaining these systems. Two main example systems were discussed.

1. Snorkel. A system to make creating and maintaining training sets a 1st class problem in both software and statistical theory.
2. Overton A system built at Apple that focused engineer time on maintaining supervision and monitoring its output quality – not more building.

3.9 Testing and Finding Bugs in NLP Models

Sameer Singh (University of California – Irvine, US)

Current evaluation of NLP systems, and much of ML, consists of measuring accuracy on held-out instances. Since held-out instances are gathered using similar annotation process as the training data, they include the same biases, providing "shortcuts" to NLP models. Further, single aggregate metric hides the actual strengths and weaknesses of the model, making it difficult to focus engineering and research efforts.

In this talk, I presented a few approaches we are exploring to perform a more thorough evaluation of NLP systems.

1. I will introduce our work on *generating black-box explanations* for ML models (LIME and Anchors) and their use in finding bugs.
2. I will describe automatic techniques for perturbing instances to identify shortcuts via *semantic adversarial examples*.
3. I will propose novel ML paradigms that introduce "testing for ML", in particular *Checklist* for creating behavioral tests for NLP.

The talk will be grounded in latest NLP benchmarks such as QA, sentiment analysis, and textual entailment, on SOTA models like BERT.

3.10 ML for Consequential Decision Making

Isabel Valera (MPI für Intelligente Systeme – Tübingen, DE)

This talk provided a brief overview of fairness and interpretability in ML, painting out the main challenges in the topic. Then, I introduce an example of how formal verification approaches can help explainable ML by providing with a model and similarity agnostic, as well as modular framework to generate (nearest) counterfactual explanations for the outcomes of algorithmic decision making systems. This example was later extended with some existing work on software engineering for adversarial robustness in ML. We close the presentation opening up questions on how software engineering may be helpful to define, test, and verify specification on the ethics of ML

- Some references on Counterfactual explanations:
  https://arxiv.org/pdf/1905.11190.pdf
  https://arxiv.org/abs/2002.06278
- Some work on fairness:
  http://jmlr.org/papers/v20/18-262.html
  https://arxiv.org/abs/1902.02979

3.11 Software Testing for Machine Learning

Jie Zhang (University College London, GB)

Main reference J. M. Zhang, M. Harman, L. Ma, Y. Liu: "Machine Learning Testing: Survey, Landscapes and Horizons", IEEE Transactions on Software Engineering, pp. 1–1, 2020.
URL https://doi.org/10.1109/TSE.2019.2962027

Machine learning systems are a type of software. This talk builds the connection between software testing and machine learning.

I first gave a brief introduction on software testing. Software testing aims to evaluate a software to check whether its behaviors meet the requirements. I introduced the properties of interest, the testing component, the testing workflow, and some key techniques in automated software testing.

Based on traditional software testing, I introduced machine learning testing (MLT). MLT detects the imperfections in machine learning systems that violate the expectation. The properties of interest may include correctness, fairness, privacy, security, interpretability. Different from traditional software testing, MLT bugs may exist in the data, learning programs, or frameworks. Many traditional testing techniques can be adopted in MLT.

I gave an overview of the related work in MLT so far. The details of the related work can be found in our survey: *Machine Learning Testing: Survey, Landscapes, and Horizons* (TSE 2020)

The last part of my talk is about my two recent practices in improving ML systems.

- *Perturbation Validation (PV)* is a compliment for out-of-sample validation in model validation. It does not use validation or test but checks whether the learner detects a small ratio of incorrect labels in training data.

- *Black-box repair* fixes machine translation problems without model retraining, so it is automatic, fast, light-weight, and can target and repair specific cases without touching other well-formed translations.

4 Working groups

4.1 Agile Development of AI/ML-based Systems

Andreas Metzger (Universität Duisburg – Essen, DE), Christian Kästner (Carnegie Mellon University – Pittsburgh, US), and Daniel Speicher (Universität Bonn, DE)

This breakout working session aimed at identifying how typical management and engineering practices of agile methods may be affected when developing AI/ML-based systems. To this end, typical practices of XP (eXtreme Programming [1]) were used to serve to structure the discussions. The main outcomes of this session were open questions that may provide an opportunity for further investigation, such as empirical studies.

The practice of the *planning game* in XP allows customers and developers to steer the work towards the most useful system the team can deliver. Functionality or quality increments are described, often estimated for the required implementation effort, and finally selected. Customers contribute their knowledge about the business value of increments. Developers contribute their knowledge about technical complexity and risks. Questions regarding the planning game included: (1) Is effort planning for AI/ML components more challenging and less precise (e.g., since creating an AI/ML model may be more explorative and experimental)? (2) How to assess the value contribution of AI/ML components? (3) Can sufficiently small work items (i.e., user stories) be defined?

The practice of *pair programming* (worth its own book [2]) has several goals, such as knowledge diffusion and skill transfer within the team. This keeps the team in the position to evolve every part of the system even when a member of the team leaves. The practice of *collective code ownership* allows all team members to (carefully) change any part of the system. Questions regarding pair programming and collective code ownership were: (1) How can AI/ML experts and software engineers work together? (2) Given joint teams of AI/ML experts and software engineers, what may happen if either may change a machine learning model and program code? (3) Do we need these practices for AI/ML components at all (e.g., concerns such as technical debt may be addressed via AutoML etc.)?

The practice of *simple design* encourages developers to stay with simple solutions. Simpler solutions lead to faster results and allow earlier customer feedback. Unnecessary technological complexity may burden future change and development. “Simplicity” here is not an absolute term, but relative to the team’s knowledge and experience. The central question regarding simple design was: How to make a trade-off between deep learning and “shallow” learning? While deep learning may generally lead to less interpretable and explainable AI/ML models than “shallow” learning, deep learning requires less feature engineering and thus requires less engineering effort.

The practice of *refactoring* has the goal to keep the design simple and to maintain code quality (such as maintainability, changeability, understandability). To guide refactoring, developers have described refactoring opportunities, often called “code smells”. Questions regarding refactoring included: (1) How can AI/ML model quality be defined in the first place? (2) How to refactor towards good AI/ML model quality? (3) As AI/ML models are generated and not hand-crafted, is refactoring needed at all?

The practice of *test-driven development* has the goal to establish a solid base of automated tests and to guide development. Also, automated tests safeguard existing functionality during refactoring and functionality addition. The key question regarding test-driven development was: How to define feasible test cases up-front (and not just non-functional constraints on the output of the AI/ML model)? We realized that the answer to this question was very much tied to the problem of how to specify AI/ML-based systems and whether machine learning may be requirements engineering – a topic that was discussed throughout the seminar [3].

References

1 Kent Beck. *Extreme Programming Explained: Embrace Change*. Second Edition. Addison-Wesley, Reading, MA, 2005

2 Laurie Williams, Robert Kessler. *Pair programming illuminated*. Addison-Wesley Longman Publishing Co., Inc., 2002.

3 Christian Kästner. "Machine Learning is Requirements Engineering – On the Role of Bugs, Verification, and Validation in Machine Learning", Medium post, Accessed April 25, 2020. https://medium.com/analytics-vidhya/machine-learning-is-requirements-engineering-8957aee55ef4

Participants

- Hadil Abukwaik
ABB – Ladenburg, DE
- Molham Aref
relationalAI – Berkeley, US
- Earl T. Barr
University College London, GB
- Houssem Ben Braiek
Polytechnique Montréal, CA
- Pavol Bielik
ETH Zürich, CH
- Carsten Binnig
TU Darmstadt, DE
- Luc De Raedt
KU Leuven, BE
- Rob DeLine
Microsoft Corporation – Redmond, US
- Joachim Giesen
Universität Jena, DE
- Elena Leah Glassman
Harvard University – Cambridge, US
- Nikolas Göbel
RelationalAI – Zürich, CH
- Jin L.C. Guo
McGill University – Montréal, CA
- Barbara Hammer
Universität Bielefeld, DE
- Fabrice Harel-Canada
UCLA, US
- Ahmed E. Hassan
Queen's University – Kingston, CA
- Steven Holtzen
UCLA, US
- Christian Kästner
Carnegie Mellon University – Pittsburgh, US
- Kristian Kersting
TU Darmstadt, DE
- Miryung Kim
UCLA, US
- Angelika Kimmig
Cardiff University, GB
- Parisa Kordjamshidi
Michigan State University – East Lansing, US
- Vu Le
Microsoft Corporation – Redmond, US
- Rupak Majumdar
MPI-SWS – Kaiserslautern, DE
- Tim Menzies
North Carolina State University – Raleigh, US
- Andreas Metzger
Universität Duisburg – Essen, DE
- Mira Mezini
TU Darmstadt, DE
- Alejandro Molina
TU Darmstadt, DE
- Sandeep Neema
DARPA – Arlington, US
- Siegfried Nijssen
UC Louvain, BE
- Andrea Passerini
University of Trento, IT
- Michael Pradel
Universität Stuttgart, DE
- Christopher Ré
Stanford University, US
- Sameer Singh
University of California – Irvine, US
- Daniel Speicher
Universität Bonn, DE
- Isabel Valera
MPI für Intelligente Systeme – Tübingen, DE
- Guy Van den Broeck
UCLA, US
- Antonio Vergari
UCLA, US
- Laurie Williams
North Carolina State University – Raleigh, US
- Ce Zhang
ETH Zürich, CH
- Jie Zhang
University College London, GB
- Tianyi Zhang
Harvard University – Cambridge, US
- Xiangyu Zhang
Purdue University – West Lafayette, US
- Thomas Zimmermann
Microsoft Corporation – Redmond, US