Machine Learning for Software Engineering: Models, Methods, and Applications

Conference or Workshop Item

Machine Learning for Software Engineering

Models, Methods, and Applications

Karl Meinke
KTH Royal Institute of Technology, Sweden
karlm@kth.se

Amel Bennaceur
The Open University, UK
amel.bennaceur@open.ac.uk

ABSTRACT

Machine Learning (ML) is the discipline that studies methods for automatically inferring models from data. Machine learning has been successfully applied in many areas of software engineering ranging from behaviour extraction, to testing, to bug fixing. Many more applications are yet be defined. However, a better understanding of ML methods, their assumptions and guarantees would help software engineers adopt and identify the appropriate methods for their desired applications. We argue that this choice can be guided by the models one seeks to infer. In this technical briefing, we review and reflect on the applications of ML for software engineering organised according to the models they produce and the methods they use. We introduce the principles of ML, give an overview of some key methods, and present examples of areas of software engineering benefiting from ML. We also discuss the open challenges for reaching the full potential of ML for software engineering and how ML can benefit from software engineering methods.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Software and its engineering** → *Model-driven software engineering*;

ACM Reference Format:
Karl Meinke and Amel Bennaceur. 2018. Machine Learning for Software Engineering: Models, Methods, and Applications. In *ICSE '18 Companion: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3183461

1 INTRODUCTION

One can scarcely open a newspaper or switch on the TV nowadays without hearing about machine learning (ML), data mining, big data analytics, and the radical changes which they offer society. However, these technologies have so far had surprisingly little impact on software engineers themselves. By examining the recent literature, we can see small but perhaps significant changes emerging on the horizon for our discipline. Surely one obstacle to the take-up of these exciting technologies in software engineering (SE) is a general lack of awareness of how they might be applied. What problems can ML currently solve? Are such problems at all relevant for software engineers?

Machine learning is a mature discipline with many excellent modern introductions to the subject. However, perspectives on ML from software engineering are less common [2], and an introduction for software engineers that attempts to be both accessible and pedagogic, is a rare thing. Furthermore, at least some of the ML methods currently applied to SE are not widely discussed in mainstream ML. There is much more to ML than deep learning.

With these motivations in mind, we will present in this technical briefing an introduction to machine learning for software engineering researchers and practitioners having little or no experience of ML. This material might also be useful for the AI community to better understand the limitations of their methods in an SE context. To structure our presentation, we will focus on three questions that we think should be addressed before attempting any new ML solution to an existing software engineering problem. These are:

- What class of learned models is appropriate for solving an SE problem?
- For this class of models, are there any existing learning algorithms that will work for typical instances and sizes of my SE problem? Otherwise, is it possible to adapt any fundamental ML principles to derive new learning algorithms?
- Has anyone considered a similar SE problem, and was it tractable to an ML solution?

2 RELEVANCE AND TIMELINESS

There is a strong trend in software engineering towards agile software development, which emphasises incremental and exploratory coding. However, agility de-emphasises the construction of many traditional artefacts such as requirements specifications, models, reports and documentation, which are typically required for software analysis. To meet the needs of changing approaches to software development, future SE techniques and tools will need to be much more automated, lightweight, adaptable and scalable, to keep pace with increased developer productivity. In particular, they will need to construct their own analysis artefacts and models.

Another new line of research concerns using ML to cope with systems integration problems and systems-of-systems such as the Internet of Things. This involves for example statistical learning for service matchmaking and automata learning for emergent middleware. Note that there is currently no single conference for all these topics, so the community is poorly connected. The organisation of thematic workshops such as: (1) *Machine Learning Technologies in Software Engineering* at ASE 2011, (2) *Machine Learning for System Construction* at ISoLA 2011, and (3) *AI meets Formal Software Development* at Schloss Dagstuhl 2012, illustrate the growing interest in the field.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05…$15.00
https://doi.org/10.1145/3183440.3183461

This proposed technical briefing will provide a compact introduction to ML methods and their use in SE, in conjunction with the launch of our new book in early 2018, *Machine Learning for Dynamic Software Analysis: Potentials and Limits*. This book is based on a Dagstuhl Seminar [1], which we co-organised.

3 BACKGROUND

The technical briefing is intended for a broad audience of software engineering researchers and practitioners. No prior knowledge of ML techniques is required, a basic knowledge of modelling is recommended. We will provide the participants with a handout beforehand and we will provide practical examples throughout the presentation. All the material will be available online, and will remain online afterwards. As depicted in Figure 1, the presentation will be structured around three main concepts: *models*, *methods*, and *applications*.

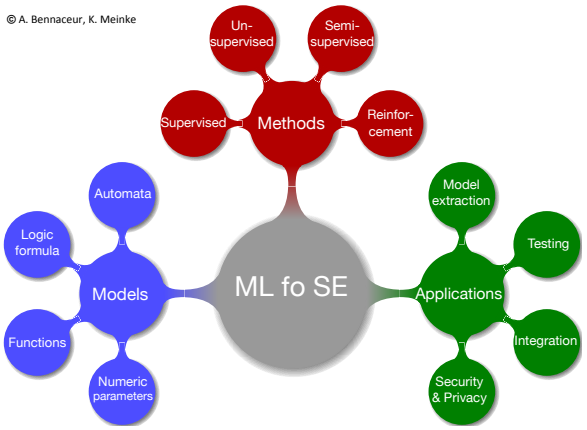

Figure 1: Key concepts of ML for software engineering

3.1 Models

A learning algorithm constructs a *model M* from a given *data set D*. This model represents some sort of synthesis of the facts contained in *D*. Most machine learning algorithms perform *inductive inference*, to extract general principles, laws or rules from the specific observations in *D*. Otherwise, learning would simply correspond to memorisation. So a model *M* typically contains a combination of facts (from *D*) and conjectures which have some predictive power. For software analysis, it is sometimes appropriate to learn an explicit *model of computation* such as an automaton model. At other times, an implicit model such as a function approximation model may be used.

3.2 Methods

The scope and power of algorithms to learn interesting models increases each year, thanks to the extraordinary productivity of the AI community. Therefore, what was technically infeasible five years ago, may have changed or be about to change. This rapid pace of development is reflected in the media excitement. However, the SE community needs to be more aware, on a technical level, of these changes, as well as the fundamental and unchanging theoretical limits. For example, very broad classes of models (e.g. Turing machines) are known to be infeasible to learn.

Such negative results do not necessarily mean that ML cannot be used for your SE problem. Nor does media hype imply that you will succeed. Therefore, we believe that it is beneficial to have some insight into some general principles of learning that go beyond specific algorithms.

3.3 Applications

We also believe it will be beneficial for software engineers to read about success stories in applying ML to SE. Figure 2 illustrates applications of ML techniques to SE activities. In the technical briefing, we will outline some SE applications where ML has already been tried with some degree of success. Scalability in the face of growing software complexity is one of the greatest challenges for SE toolmakers. Therefore, information (however ephemeral) about the state of the art in solved problem sizes is also relevant.

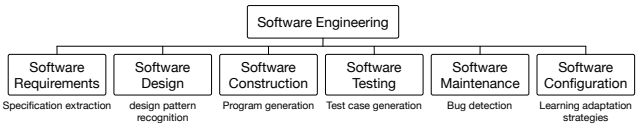

Figure 2: Applications of ML for software engineering

4 PRESENTERS

Prof. Dr. Karl Meinke. In the last ten years Meinke and his group have pioneered new applications of machine learning combined with model checking to the problem of functional requirements testing for software and systems. Group research has studied the learning problem for both procedural and reactive systems. Numerical function approximation as well as symbolic automaton learning methods have been considered. Our main requirements testing tool LBTest[3] (http://www.lbtest.org) has been evaluated in sectors such as automotive, avionics, finance and web, and by major Swedish multinational companies such as SAAB Aerospace, Scania and Volvo. Meinke has a publication track record in the areas of machine learning for finite and infinite state systems, theoretical principles of learning-based testing, and practical tools and case studies for learning-based testing.

Dr. Amel Bennaceur is a Lecturer in Computing at the Open University, UK. She received her PhD degree in Computer Science from the University of Paris VI in 2013. Her research interests include dynamic mediator synthesis for interoperability and collaborative security. She was part of the CONNECT and EternalS EU projects that explored synergies between machine learning and software synthesis [4]. The results of her work have been published in leading conferences and journals such as Middleware, ECSA, and IEEE TSE. She has also been invited to present the results of this work in various scientific events such as Dagstuhl and Shonan seminars.

ACKNOWLEDGMENTS

We acknowledge ERC Advanced Grant no. 291652 (ASAP).

REFERENCES

[1] A. Bennaceur, D. Giannakopoulou, R. Hähnle, and K. Meinke. 2016. Machine Learning for Dynamic Software Analysis: Potentials and Limits (Seminar 16172). *Dagstuhl Reports* 6, 4 (2016), 161–173. https://doi.org/10.4230/DagRep.6.4.161

[2] P. Louridas and C. Ebert. 2016. Machine Learning. *IEEE Software* 33, 5 (2016), 110–115. https://doi.org/10.1109/MS.2016.114

[3] K. Meinke and M. A. Sindhu. 2013. LBTest: A Learning-Based Testing Tool for Reactive Systems. In *Sixth IEEE Int. Conf. on Software Testing, Verification and Validation, ICST 2013, Luxembourg, March 18-22, 2013*. 447–454.

[4] A. Bennaceur *et al.* 2012. Machine Learning for Emergent Middleware. In *Proc. of EternalS'12*. 16–29.