A Survey on Machine Learning and Requirements

Author1, Author2, and Authorn

fortiss GmbH. Guerickestraße 25, 80805 München author@fortiss.org

Abstract. The abstract goes here Levi ►you can use labels by using your name as command◄ .

1 Introduction

This is an introduction [7]

1.1 Machine Learning

Machine-Learning (ML) [?] is a range of algorithm to approximate functions and discover patterns in data. Historically, models and heuristics are human-built exhaustive prescriptions of how a system should behave. ML is grounded on different premises: rather than relying on humans to input all the possible cases the system can handle, the field attempts to extrapolate patterns from a representative set of examples that illustrate the expected behaviors. The way in which a learning algorithm operates attempts to emulate the way in which humans learn: from a set of examples, a general model for a behavior is induced.

Many learning algorithms exist, based on different visions of how learning happens in practice [7]. All these algorithms have in common the notion of *features*. Features correspond to characteristics of what is being learned and provide the grounds for the algorithm to abstract from the complexities of the real world. Assume for example that an algorithm should learn, based on a brain scan of a medical patient, to decide whether that patient has brain cancer or not. A number of *features* such as for example the "number of irregular objects in the scan", the "color of such objects", the "disposition of such objects" would be provided to the algorithm. Additionally, the algorithm is fed a number of brain scans together with decisions previously taken on them (cancer found / cancer not found) – the *training data*. The learning algorithm then undergoes a *training phase*. It attempts to find an internal model that allows it to map the decisions to the brain scans, given the training data. The model obtained from the training step is useful if it performs well (generalizes) when applied to new data from outside the training set – in our example, when it can accurately diagnose brain cancer for new brain scans. Such generalization is based on the premise that inputs that are "closer", in terms of the given *features*, should lead to "closer" outputs.

The formal notion of "closeness" is a characteristic of the learner algorithm being employed and determines how the algorithm generalizes the computation from the given examples. Achieving good generalizations is the cornerstone of machine learning and *overfitting* (performing very well on training inputs but very poorly on new inputs) is one of its major challenges. Levi ►cut these last couple of sentences◄

More formally, in textbooks, courses and articles, Machine Learning is often defined following the definition of Tom Mitchell [**?**]:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

Therefore, it is said that to classify some patients into classes (e.g healthy and unhealthy), the task T, one have to define an algorithm that provides a model, such as an artificial neural network. The quality of this model is quantified by a measure P, for instance its accuracy while predicting the classes. This measure is then sent back to the algorithm, a new experience E, in order to choose or improve the model. A machine learning tasks can be discussed and subdivided based on the elements of the following equation:

$$f(\mathbf{X}) = \mathbf{y} + \xi$$

where $\mathbf{X}$ is the $n \times d$ input matrix, containing n samples characterized by d features, $\mathbf{y}$ is the $n \times 1$ target vector containing the classes of the n samples and ξ is a $n \times 1$ vector representing the noise. The goal is to approximate f in order to provide the best mapping between $\mathbf{X}$ and $\mathbf{y}$, given some noise ξ. Indeed, the approximation has to map a $\mathbf{X}$ containing noise, to a $\mathbf{y}$ which may contain noise too. For instance, uncontrolled conditions such as the room temperature and the exposure time to this temperature can induce variations in the information contained in collected blood samples. Moreover, the ξ term also contains the approximation error when, for example, one tries to approximate a non-linear function with a linear function.

The problem presented by the above equation is called *supervised learning*, and can be roughly subdivided in two popular problems: *classification* and *regression*. When the target vector $\mathbf{y}$ is composed of categorical values (i.e. classes), then we have a classification problem. The goal is to learn how to link instances or samples in $\mathbf{X}$ to a certain class (e.g. healthy patient or unhealthy patient). However, if the target vector contains continuous values, we face a regression problem (e.g. predict the body temperature of a patient given some clinical features of the patient).

In some cases, $\mathbf{y}$ is not given and we have to find patterns in $\mathbf{X}$ "blindly." This is called an *unsupervised* learning problem. Finding clusters in $\mathbf{X}$, i.e. finding a $\mathbf{y}$ that has never been given, is such a problem. For instance, one may want to group patients based on symptoms they have.

Reinforcement Learning can be seen as an intermediate problem where $\mathbf{y}$ is not given but the procedure is guided nevertheless. In RL, an agent has to find a sequence of actions leading to a success. The fact that the sequence leads to a success, i.e what would be in **y**, is not known in advance, but rewards are given to the agent in order for him to know if it follows a path to success. In other words, the goal is, by providing rewards along the way, to find the sequence of actions leading to the desired state. Typical examples can be found in gaming, where an agent receives a reward when he wins the game. From the different chains of actions that led him to a reward, the agent must generalize to find how to win that game.

1.2 Requirements Engineering

Software systems are developed over millions of lines of code, number of modules and documents. The primary goal of the software system is to satisfy users by developing the software that can meet their needs and expectations. This goal is achievable by applying different methodologies and engineering techniques. One of the key factor is to understand and identify the needs of users, also known as, software requirements. Software requirement engineering is the process that helps to determine the requirements in a systematic way to know what functionalities the targeted system should have to fulfil user's needs. Formally RE is defined as [**?**]:

> "Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families."

Software requirements play a key role in the success of a project. In the USA, a survey was conducted over 8380 projects by 350 companies to know the project failure rates. The report overall results showed only 16.2% projects were completed successfully and one-half (52.7%) of projects met with challenges and were completed with partial functionalities, time delays and over budget. Almost 31% of the projects were never completed. The main cause told by the executive managers was the poor requirement. The major problems were the lack of user involvement (13%), requirements incompleteness (12%), changing requirements (11%), unrealistic expectations (6%), and unclear objectives (5%).[**?**] [6]
Software requirement engineering has mainly four phases; requirement elicitation, requirement analysis, requirement documentation and requirement verification [**?**]. Requirement elicitation [**?**,**?**] helps to understand the stakeholders needs, e.g. what features he wants in the software. Requirement elicitation techniques are mostly derived by the social sciences, organizational theory, knowledge engineering and practical experience. For requirements elicitation, different techniques exist in the literature that include interviews, questioners and ethnography etc. Requirement analysis [**?**] is the next step after requirement elicitation. In this phase, software requirements are analyzed to check conflicts and consistency

of requirements. It is also makes sure that the requirements are clear, complete and consistent. Furthermore, the agreed requirements are documented. This documentation has a clear and precise definition of the system functionalities. It also acts as an agreement between stakeholders and developers. These functionalities and requirements are documented usually as diagrams, mathematically formulae or natural languages. These documents are used and iterated until the end of the projects. System requirements are classified into businees requirments, user requirments, functional requirements (FR) and non-functional requirements (NFR),. Functional requirements are the system requirements that include the main features and characteristics of the desired system. Non-functional requirements are the system properties and constraints [?]. NFRs set the criteria for judging the operation of the system e.g. performance, availability and reliability etc.

2 Contributions

l

2.1 Requirements Elicitation and Discovery

[4], [15], [3]

Internal vs External

Prioritization of Requirements

Text mining

2.2 Requirements Analysis and Specification

Non-Functional [22]

Functional [6], [17], [9], [1], [5], [20], [18], [10], [24], [8], [2], [23], [11], [13], [12], [19]

Security Requirements [14], [21], [16]

Contextual Requirements

2.3 Requirements Validation

Traceability [?] [20]

2.4 Requirements Management

Visualization

Structuring Documents

3 Discussion

4 Threats to Validity

The validity of the study might be affected by the coverage of the search results, bias on study selection, and inaccuracy of data extraction.

Study Coverage. Some relevant studies could be missing in our study due to inadequate search strings or missing databases. To cope with this threat, the data preparation was based on a systematic method.

Study Selection Bias. Study assessment might be biased by researchers. To mitigate for this threat, a set of include and exclude criteria was predefined and researchers assessed the title and abstract of the papers based on them to steer the assessment.

Inaccuracy of Data Extraction. Also the data extraction process might be biased by researchers. To mitigate for this threat, the selection of data items was strictly driven by the research questions. Moreover, assignments were marked by the researchers depending on their confidence level. Low-confidence assignments were discussed between the authors until a consensus was reached.

5 Conclusion

References

1. Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider. What works better? A study of classifying requirements. *CoRR*, abs/1707.02358, 2017.
2. A. Casamayor, D. Godoy, and M. Campo. Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. *Inf. Softw. Technol.*, 52(4):436–445, Apr. 2010.
3. C. Castro-Herrera, C. Duan, J. Cleland-Huang, and B. Mobasher. A recommender system for requirements elicitation in large-scale software projects. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 1419–1426, New York, NY, USA, 2009. ACM.
4. J. Cleland-Huang and B. Mobasher. Using data mining and recommender systems to scale up the requirements process. In *Proceedings of the 2Nd International Workshop on Ultra-large-scale Software-intensive Systems*, ULSSIS '08, pages 3–6, New York, NY, USA, 2008. ACM.

5. A. Dekhtyar and V. Fong. Re data challenge: Requirements identification with word2vec and tensorflow. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 484–489, Sept 2017.
6. R. Deocadez, R. Harrison, and D. Rodriguez. Automatically classifying requirements from app stores: A preliminary study. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 367–371, Sept 2017.
7. P. Domingos. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Basic Books, 2015.
8. F. Garzoli, D. Croce, M. Nardini, F. Ciambra, and R. Basili. Robust requirements analysis in complex systems through machine learning. In A. Moschitti and B. Plank, editors, *Trustworthy Eternal Systems via Evolving Software, Data and Knowledge*, pages 44–58, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
9. E. Guzman, M. Ibrahim, and M. Glinz. A little bird told me: Mining tweets for requirements and software evolution. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 11–20, Sept 2017.
10. J. H. Hayes, W. Li, and M. Rahimi. Weka meets tracelab: Toward convenient classification: Machine learning for requirements engineering problems: A position paper. In *2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, pages 9–12, Aug 2014.
11. I. Hussain, O. Ormandjieva, and L. Kosseim. Lasr: A tool for large scale annotation of software requirements. In *2012 Second IEEE International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 57–60, Sept 2012.
12. N. Jha and A. Mahmoud. Mining user requirements from application store reviews using frame semantics. In P. Grünbacher and A. Perini, editors, *Requirements Engineering: Foundation for Software Quality*, pages 273–287, Cham, 2017. Springer International Publishing.
13. W. Jiang, H. Ruan, L. Zhang, P. Lew, and J. Jiang. For user-driven software evolution: Requirements elicitation derived from mining online reviews. In V. S. Tseng, T. B. Ho, Z.-H. Zhou, A. L. P. Chen, and H.-Y. Kao, editors, *Advances in Knowledge Discovery and Data Mining*, pages 584–595, Cham, 2014. Springer International Publishing.
14. R. Jindal, R. Malhotra, and A. Jain. Automated classification of security requirements. In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2027–2033, Sept 2016.
15. H. Kaiya, Y. Shimizu, H. Yasui, K. Kaijiri, and M. Saeki. Enhancing domain knowledge for requirements elicitation with web mining. In *2010 Asia Pacific Software Engineering Conference*, pages 3–12, Nov 2010.
16. E. Knauss and D. Ott. (semi-) automatic categorization of natural language requirements. In C. Salinesi and I. van de Weerd, editors, *Requirements Engineering: Foundation for Software Quality*, pages 39–54, Cham, 2014. Springer International Publishing.
17. Z. Kurtanović and W. Maalej. Automatically classifying functional and nonfunctional requirements using supervised machine learning. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 490–495, Sept 2017.
18. M. Lu and P. Liang. Automatic classification of non-functional requirements from augmented app user reviews. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, EASE'17, pages 344–353, New York, NY, USA, 2017. ACM.

19. R. Pinquié, P. Véron, F. Segonds, and N. Croué. A collaborative requirement mining framework to support oems. In Y. Luo, editor, *Cooperative Design, Visualization, and Engineering*, pages 105–114, Cham, 2015. Springer International Publishing.
20. A. Rashwan. Semantic analysis of functional and non-functional requirements in software requirements specifications. In *Proceedings of the 25th Canadian Conference on Advances in Artificial Intelligence*, Canadian AI'12, pages 388–391, Berlin, Heidelberg, 2012. Springer-Verlag.
21. M. Riaz, J. King, J. Slankas, and L. Williams. Hidden in plain sight: Automatically identifying security requirements from natural language artifacts. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pages 183–192, Aug 2014.
22. J. Slankas and L. Williams. Automated extraction of non-functional requirements in available documentation. In *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*, pages 9–16, May 2013.
23. Y. Wang and J. Zhang. Experiment on automatic functional requirements analysis with the efrf's semantic cases. In *2016 International Conference on Progress in Informatics and Computing (PIC)*, pages 636–642, Dec 2016.
24. G. Williams and A. Mahmoud. Mining twitter feeds for software user requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 1–10, Sept 2017.