Towards a Technical Debt for AI-based Recommender System

Sergio Moreschini
Tampere University,
University of Oulu
Tampere, Finland
sergio.moreschini@tuni.fi

Ludovik Coba*
Roku Inc.
London, UK
lcoba@roku.com

Valentina Lenarduzzi
University of Oulu
Oulu, Finland
valentina.lenarduzzi@oulu.fi

ABSTRACT

Balancing the management of technical debt within recommender systems requires effectively juggling the introduction of new features with the ongoing maintenance and enhancement of the current system. Within the realm of recommender systems, technical debt encompasses the trade-offs and expedient choices made during the development and upkeep of the recommendation system, which could potentially have adverse effects on its long-term performance, scalability, and maintainability. In this vision paper, our objective is to kickstart a research direction regarding Technical Debt in AI-based Recommender Systems. We identified 15 potential factors, along with detailed explanations outlining why it is advisable to consider them.

KEYWORDS

Technical Debt, Artificial Intelligence, Machine Learning, Recommender System

ACM Reference Format:
Sergio Moreschini, Ludovik Coba, and Valentina Lenarduzzi. 2024. Towards a Technical Debt for AI-based Recommender System. In *International Conference on Technical Debt (TechDebt '24), April 14–15, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3644384.3648574

1 INTRODUCTION

Managing technical debt in recommender systems involves a balance between delivering new features and maintaining and improving the existing system [6, 12]. Regularly reviewing and addressing these debt areas is essential to ensure that the AI-based Recommender Systems continues to provide high-quality recommendations and remains adaptable to changing user needs and market conditions [21].

Technical Debt in the context of AI-based Recommender Systems refers to the shortcuts or compromises made during the development and maintenance of the recommendation system that may negatively impact its long-term performance, scalability, and maintainability. Like in any software development project, Technical Debt in AI-based Recommender Systems can accumulate over time and should be managed carefully to avoid future issues.

The accumulation of Technical Debt in AI-based Recommender Systems can result in degraded user experiences, increased development and maintenance costs, and difficulties in remaining competitive in the fast-evolving field of recommendation technology. Therefore, it's crucial for organizations to prioritize debt reduction and invest in the long-term health of their Recommender Systems.

In this ongoing research paper, we aim at initiating the discussion about a specific Technical Debt for AI-based Recommender Systems. We identified 15 potential factors and we provide explanations for why we recommend taking them into account. Moreover, we define a roadmap with the future steps we want to investigate.

Paper Structure. Section 2 introduces the concept of Technical Debt while Section 3 the recommender systems. Section 4 describes our research vision idea, Section 5 presents our roadmap. Section 6 draws some conclusions.

2 TECHNICAL DEBT

Technical debt is a concept in software engineering that refers to the implied cost of additional work that arises when software is developed or maintained quickly and with shortcuts or suboptimal solutions. It's a metaphorical way of describing the trade-off between getting a piece of software out quickly and dealing with the long-term consequences of taking shortcuts or making compromises in the development process [6, 12].

Technical debt is a concept that highlights the consequences of taking shortcuts or making compromises during software development. It emphasizes the importance of managing and reducing debt over time to ensure the long-term quality and maintainability of software systems [21].

Unfortunately, Technical Debt is an unavoidable but beneficial aspect under certain circumstances, often linked to unforeseeable internal or external business or environmental forces [27].

Evidence about the good and bad effects has been investigated in the last 10 years. Researchers proposed initial approaches to manage and reduce Technical Debt, developing also some tools, but more research and insights are needed [9]. Technical Debt is composed of "technical issues" that include any kind of information derived from the source code and from the software process, such as usage of specific patterns, non-compliance with coding or documentation conventions, and architectural issues [22].

However, just like any other financial debt, every Technical Debt has an interest attached, or else an extra cost or negative impact that is generated by the presence of a sub-optimal solution [23]. When such interest becomes very costly, it can lead to disruptive events, such as development crises [27]. The current best practices employed by software companies include keeping Technical Debt at bay by avoiding it if the consequences are known or refactoring or rewriting code and other artifacts in order to get rid of the

*This work was completed before joining Roku Inc.

TechDebt '24, April 14–15, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0590-8/24/04.
https://doi.org/10.1145/3644384.3648574

accumulated sub-optimal solutions and their negative impact. Companies cannot afford to avoid or repay all the Technical Debt that is generated continuously and may be unknown [27]. The main business goals of companies are to continuously deliver value to their customers and to maintain their products.

3 RECOMMEDER SYSTEMS

RS have become an integral part of modern online platforms and applications, leveraging machine learning techniques to assist users in finding relevant items or content based on their preferences, past behaviors, or similarities to other users. With the vast amount of available options, these systems play a crucial role in enhancing user experience, increasing engagement, and facilitating decision-making [17, 30]. RS can be found across a wide range of online platforms and applications, including e-commerce websites, streaming services, social media platforms, news portals, and more. Their primary goal is to help users navigate through an overwhelming sea of choices and discover new products, movies, music, articles, and other items of interest. Successfully deploying RS in a production environment is not without its challenges. Organizations face various hurdles that need to be overcome to deliver effective and scalable recommendations to users. Some of these challenges include harmonizing multiple data sources, scalability, evaluation, feedback loop, ethical considerations, maintenance and adaptability. Thus, building such large systems could lead to the accumulation of technical debt, which can arise from hurriedly implemented algorithms, sub-optimal data processing methods, or insufficient testing. Over time, this debt can lead to issues such as decreased system performance, increased maintenance efforts, and difficulty in implementing new features or algorithms. Addressing tech debt in RS is crucial to maintaining their effectiveness and ensuring long-term sustainability.

4 TOWARDS A TECHNICAL DEBT FOR AI-BASED RECOMMENDER SYSTEMS

In this section, we share 15 possible Technical Debt factors that we discovered in the literature regarding AI-based Recommender Systems. The identification of these factors is the result of our work in collaboration with practitioners in the field. A rationale for considering each factor is provided. In order to conceptualize a Technical Debt for a AI-based Recommender System (RS) we need first to identify which can be the factors that could characterize this type of Technical Debt (TD).

- **Data Contracts:** Data contracts are very important for defining API data models and data governance [25]. They play a crucial role in ensuring clarity and consistency in data exchange processes between different parties or systems. By defining the structure, format, and exchange rules, they help prevent misunderstandings and ensure seamless communication in distributed data architectures. Particular types of recommenders (e.g. context-aware RS [29]) generate live predictions using data that is extremely volatile (e.g. user features), thus, in such a contract, the client and the service provider define the signals (or features) that the model needs to function properly. Often, data contracts are defined before or during implementation. This may lead to a redefinition of the data pipeline or the signals produced at a later stage, resulting in significant time loss in implementing the hydration service to serve missing features or modeling the RS.
- **Glue code:** There is a large plethora of generic packages for RS [5, 7, 10, 13, 28]. Using generic packages frequently leads to the creation of a glue code system design pattern. This results in the development of a substantial amount of auxiliary code to facilitate data exchange with these packages. Glue code can be costly in the long term because it tends to lock a system into function only with a specific package, making it expensive to explore alternative solutions. Furthermore, it can hinder improvements, as it becomes challenging to leverage domain-specific characteristics or fine-tune the system to meet specific domain goals.
- **Pipeline Jungles:** ML-based RSs, particularly those following MLOps guidelines, are composed of multiple pipelines which provide Continuous Training, Continuous Deployment, and Continuous Integration. This also requires automation of the data scraping, data management, data preprocessing, and monitoring steps, which results in the creation of additional pipelines. This high amount of pipelines can turn into a complex *jungle* of scraping, joining, and sampling procedures. The result is a domino effect caused by any small change.
- **Dead Experimental Codepaths:** As a consequence of the previous two points, in the short term it becomes more convenient to perform code experimentation by introducing experimental codepaths as conditional branches within the primary production code. The cost of experimentation is relatively low from a TD perspective however, the overall cost will increase due to the accumulation of these unused experimental codepaths. Among these unused codepaths only few in the end will be used while most of them are usually tested and abandoned. The main result of this accumulated cost is presented when trying to recover an abandoned branch which might presents compatibility issues due to unfulfilled requirements.
- **Prototype Smell:** Following the previous point, it is a standard procedure to use branches to create new small-scale environment based on prototypes to test new ideas. Maintaining a prototype environment creates new costs to add to the TD, but the main drawback is the feeling that a prototype, and its environment, are ready for production. In some cases, the new environment is not completely compatible with the main branch creating difficulties when deciding to switch between the main branch and a prototype.
- **Inconsistency between offline and online evaluation results:** Most RS are shipped to production using standard evaluation metrics (such as normalized Discounted Cumulative Gain, recall, precision, etc.) and sometimes, ad-hoc KPI-related metrics [32]. Using advanced user models we can execute simulations of users interactions with the system, thus reducing the need for expensive user studies and online testing. However, the approaches are based on simple evaluation proxies that don't map the intricate behaviour

of a user, often having offline experimental results not well correlated to online results [5, 16, 20, 32].

- **Multiple-Language Smell:** Nowadays, a system can be composed of a combination of several programming languages. Analyzing and maintaining multi-language systems pose significant challenges, and developers consistently encounter difficulties when working with these complex systems [2]. Some of the consequences of this behaviour are code smells within the source code reflecting suboptimal coding choices that manifest as indicators of the existence of code anti-patterns [3].
- **Conflicting KPI goals:** When defying project goals, a well-known approach involves defining precise Key Performance Indicators (KPIs) that align with these objectives. These KPIs act as quantifiable metrics, enabling the developer to monitor progress and ensure that your endeavours effectively advance the developer toward the profit-related targets. Nevertheless, in certain instances, these KPIs may be mutually contradictory, making it impossible to maximize both simultaneously. For instance, on a search engine, engagement results in long-term decreased conversion and vice-versa. However, more sophisticated metrics could interact in non-obvious ways that are invisible before running online experimentation.
  [1, 32].
- **Plain-Old-Data Type Smell:** RSs receive and provide valuable information, but this information is often represented using basic data types such as raw floats and integers. To ensure the correct and meaningful use of both model parameters and predictions, a well-designed RS based on ML should not only produce results, but also provide a clear and informative context. This context is essential to make ML systems transparent, interpretable, and reliable to use in practice [31].
- **Reproducibility Debt:** Reproducibility is a hard constraint in software development, but also it has been an emerging topic in RS [11, 14]. When developing RS software it is important to follow specific guidelines to ensure this and other constraints such as scalability, explainability and interpretability. This is one of the main reasons why MLOps has been emerging nowadays. Compared to classic software development, RS software does not rely only on code but also on data therefore, MLOps has data versioning as one of its foundations to allow reproducibility by reducing to minimum the influence of external factors.
- **Cultural Debt:** Sometimes there is a clear boundary between RS research and engineering, but maintaining this rigid distinction can undermine the long-term health of a system. The cultural debt refers to the debt that is created when different figures (or cultures) cooperate by defining lines and responsibilities. As DevOps has suggested for the last 30 years this is usually not the best approach as attributes should be equally valued and incentivized and not overshadowed by the pursuit of improved results alone. In RS software, it is imperative to cultivate team cultures that not only celebrate advancements in model accuracy but also emphasize the importance of eliminating unnecessary features, reducing complexity, enhancing reproducibility, ensuring system stability, and bolstering monitoring capabilities. As RS are user-feature systems and require the engagement of several teams the cultural debt is therefore more emphasized.
- **Performance Deterioration:** One of the main reasons for performance deterioration are *concept* and *data drift* [15, 24]. Concept drift arises when the initially established relationship between the target and independent variables, on which a model was initially trained, either becomes obsolete or transforms into an unfamiliar configuration for the model. Concept drift leads to a decline in model accuracy because, during production, the model is unaware of these alterations. Data drift refers to the evolving or abrupt changes in the data distribution used to train a machine learning model compared to the data distribution encountered in the actual deployment context. Such changes can be due to a variety of factors, including changes in user actions, changes in the underlying data generation processes, or external events that affect the data collection process. Following the previous example for seasonality, if we would train the RS using a dataset based on a moving window, we would see that when training the model including the data for the holiday season the accuracy of the model would decrease due to data drift. One of the main factors influencing concept drift for RS is seasonality. An example can be related to how during the holiday season, people are looking for seasonal music, flights, or items in a variety of online services such as music streaming, travel companies, or stores.
- **Feedback Loops:** direct feedback loops have received a lot of interest in research the recent years [8, 18, 26]. They arise when the results generated by an RS model start to impact the subsequent input features that the same model uses for training. This could result in algorithmic bias (e.g. popularity bias) which intensifies over time as models, because, in many cases, users will be limited to interact with recommendations, which themselves are later used as part of the historical interaction of users, reinforcing the belief of the model about the validity of these recommendation.
- **Hidden Feedback Loops:** hidden feedback loops are harder to establish and consequentially haven't received a lot of research focus in the RS community. Nowadays companies are using multiple RS in parallel and all models could influence the historical interactions that other models will use in training. For instance, Netflix is personalising the artwork presented to users[4], but also the rows and their content[33], practice that has become an industry standard.

5 ROADMAP

Our roadmap includes the following points:

- **AI-based Recommender Systems TD factors validation.** We will validate these factors with TD and Recommender systems experts, to determine if:
  - They consider these factors a real technical issue (and why)
  - How severe they consider them
  - How to fix them (how and when)

- **AI-based Recommender Systems TD factors mining.** We will define a method to collect the factors validated by the experts and we collect them in real projects (open source and industrial ones).
- **AI-based Recommender Systems TD factors impact** We will investigate the impact of the identified factors on:
  - *Software Quality*. We already identified some software quality attributes that can be more negatively impacted by these factors, such as maintainability, readability, and testability.
  - *Development Process*. We will measure some software metrics related to the development process such as process metrics proposed by Kamei [19].

6 CONCLUSION

In this paper, we raised up the conceptualization of Technical Debt in AI-based Recommender Systems. We have identified 15 potential factors and offered explanations for why we advocate considering them. Additionally, we outlined a roadmap outlining the future steps we intend to explore.

REFERENCES

[1] Himan Abdollahpouri, Gediminas Adomavicius, Robin Burke, Ido Guy, Dietmar Jannach, Toshihiro Kamishima, Jan Krasnodebski, and Luiz Pizzato. 2020. Multistakeholder recommendation: Survey and research directions. *User Modeling and User-Adapted Interaction* 30 (2020), 127–158.

[2] Mouna Abidi, Manel Grichi, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2019. Code Smells for Multi-Language Systems. In *European Conference on Pattern Languages of Programs*. Article 12, 13 pages.

[3] Mouna Abidi and Foutse Khomh. 2020. Towards the Definition of Patterns and Code Smells for Multi-Language Systems. In *European Conference on Pattern Languages of Programs 2020*. Article 37.

[4] Fernando Amat, Ashok Chandrashekar, Tony Jebara, and Justin Basilico. 2018. Artwork personalization at Netflix. In *Proceedings of the 12th ACM conference on recommender systems*. 487–488.

[5] Andreas Argyriou, Miguel González-Fierro, and Le Zhang. 2020. Microsoft recommenders: Best practices for production-ready recommendation systems. In *Companion Proceedings of the Web Conference 2020*. 50–51.

[6] Paris Avgeriou, Philippe Kruchten, Ipek Ozkaya, and Carolyn Seaman. 2016. Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162). *Dagstuhl Reports* 6, 4 (2016), 110–138.

[7] Immanuel Bayer. 2016. fastfm: A library for factorization machines. *The Journal of Machine Learning Research* 17, 1 (2016), 6393–6397.

[8] Allison JB Chaney, Brandon M Stewart, and Barbara E Engelhardt. 2018. How algorithmic confounding in recommendation systems increases homogeneity and decreases utility. In *Proceedings of the 12th ACM conference on recommender systems*. 224–232.

[9] Marcus Ciolkowski, Valentina Lenarduzzi, and Antonio Martini. 2021. 10 Years of Technical Debt Research and Practice: Past, Present, and Future. *IEEE Software* 38, 6 (2021), 24–29.

[10] Ludovik Coba, Roberto Confalonieri, and Markus Zanker. 2022. RecoXplainer: a library for development and offline evaluation of explainable recommender systems. *IEEE Computational Intelligence Magazine* 17, 1 (2022), 46–58.

[11] Ludovik Çoba and Markus Zanker. 2017. Replication and reproduction in recommender systems research-evidence from a case-study with the rrecsys library. In *Advances in Artificial Intelligence: From Theory to Practice: 30th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2017, Arras, France, June 27-30, 2017, Proceedings, Part I 30*. Springer, 305–314.

[12] Ward Cunningham. 1992. The WyCash Portfolio Management System. *SIGPLAN OOPS Mess.* 4, 2 (Dec. 1992), 29–30.

[13] Michael D Ekstrand, Michael Ludwig, Joseph A Konstan, and John T Riedl. 2011. Rethinking the recommender research ecosystem: reproducibility, openness, and lenskit. In *Proceedings of the fifth ACM conference on Recommender systems*. 133–140.

[14] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM conference on recommender systems*. 101–109.

[15] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 1–37.

[16] Florent Garcin, Boi Faltings, Olivier Donatsch, Ayar Alazzawi, Christophe Bruttin, and Amr Huber. 2014. Offline and online evaluation of news recommender systems at swissinfo. ch. In *Proceedings of the 8th ACM Conference on Recommender systems*. 169–176.

[17] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. 2010. *Recommender systems: an introduction*. Cambridge University Press.

[18] Ray Jiang, Silvia Chiappa, Tor Lattimore, András György, and Pushmeet Kohli. 2019. Degenerate feedback loops in recommender systems. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*. 383–390.

[19] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. 2012. A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering* 39 (2012).

[20] Karl Krauth, Sarah Dean, Alex Zhao, Wenshuo Guo, Mihaela Curmei, Benjamin Recht, and Michael I Jordan. 2020. Do offline metrics predict online performance in recommender systems? *arXiv preprint arXiv:2011.07931* (2020).

[21] Valentina Lenarduzzi, Terese Besker, Davide Taibi, Antonio Martini, and Francesca Arcelli Fontana. 2021. A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools. *Journal of Systems and Software* 171 (2021), 110827.

[22] Valentina Lenarduzzi, Nyyti Saarimaki, and Davide Taibi. 2019. On the Diffuseness of Code Technical Debt in Java Projects of the Apache Ecosystem. In *International Conference on Technical Debt (TechDebt)*. 98–107.

[23] Zengyang Li, Paris Avgeriou, and Peng Liang. 2015. A systematic mapping study on technical debt and its management. *Journal of Systems and Software* 101 (2015), 193–220.

[24] Zhuoran Liu, Leqi Zou, Xuan Zou, Caihua Wang, Biao Zhang, Da Tang, Bolin Zhu, Yijie Zhu, Peng Wu, Ke Wang, et al. 2022. Monolith: real time recommendation system with collisionless embedding table. *arXiv preprint arXiv:2209.07663* (2022).

[25] Qinghua Lu, Liming Zhu, Xiwei Xu, Jon Whittle, and Zhenchang Xing. 2022. Towards a roadmap on software engineering for responsible AI. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*. 101–112.

[26] Masoud Mansoury, Himan Abdollahpouri, Mykola Pechenizkiy, Bamshad Mobasher, and Robin Burke. 2020. Feedback loop and bias amplification in recommender systems. In *Proceedings of the 29th ACM international conference on information & knowledge management*. 2145–2148.

[27] Antonio Martini, Jan Bosch, and Michel Chaudron. 2015. Investigating Architectural Technical Debt accumulation and refactoring over time: A multiple-case study. *Information and Software Technology* 67 (2015), 237 – 253.

[28] Rama Kumar Pasumarthi, Sebastian Bruch, Xuanhui Wang, Cheng Li, Michael Bendersky, Marc Najork, Jan Pfeifer, Nadav Golbandi, Rohan Anil, and Stephan Wolf. 2019. Tf-ranking: Scalable tensorflow library for learning-to-rank. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2970–2978.

[29] Shaina Raza and Chen Ding. 2019. Progress in context-aware recommender systems—An overview. *Computer Science Review* 31 (2019), 84–97.

[30] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2015. Recommender systems: introduction and challenges. *Recommender systems handbook* (2015), 1–34.

[31] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. *Advances in neural information processing systems* 28 (2015).

[32] Guy Shani and Asela Gunawardana. 2011. Evaluating recommendation systems. *Recommender systems handbook* (2011), 257–297.

[33] Harald Steck, Linas Baltrunas, Ehtsham Elahi, Dawen Liang, Yves Raimond, and Justin Basilico. 2021. Deep learning for recommender systems: A Netflix case study. *AI Magazine* 42, 3 (2021), 7–18.