Technical Debt for Citi Bikes ML Pipeline
Sayan Swar

The article titled "Hidden Technical Debt in Machine Learning Systems" explores the concept of technical debt in the context of Machine Learning systems, with a focus on the deployment of ML code. The authors identify various types of problems that require attention and underscore the lack of appropriate tools and resources for maintaining a typical Machine Learning system. The paper emphasizes the need to enhance the management of technical debt in Machine Learning systems, which have certain differences from conventional software systems. The categories of issues are as follows:

1. The first category of issues discussed in the paper is "Complex Models Erode Boundaries," which includes two subcategories. The first one is "Entanglement," which refers to the dynamic nature of inputs in Machine Learning systems, including changes in features, scaling, and distribution. This creates a fundamental challenge known as the CACE principle, which means that changing anything changes everything. To address this issue, one can use ensemble models and tools for high-dimensional visualization to identify changes in predictions.

2. The second subcategory is "Correction Cascades," which deals with models built upon each other to solve different problems, creating dependencies between them. Changes made in the base models can affect the derived models. To mitigate this issue, one can either maintain a single model and modify the features or accept the cost of training separate models.Undeclared consumers: This pertains to the management of access control to the outputs of one model that may serve as features to another model. It is important to establish appropriate restrictions and Service Level Agreements (SLAs) to prevent unauthorized access to these outputs.

3. The issue of undeclared consumers refers to the need to control the access to the outputs of a model that may be utilized as features by another model. It is essential to impose restrictions and define Service Level Agreements (SLAs) to prevent unauthorized access to these outputs.

- Data Dependencies Cost More than Code Dependencies
    1. Unstable Data Dependencies: This refers to the fact that inputs or features of a Machine Learning model can change, and even minor improvements to these features can negatively affect the model's performance because it has been trained on the previous version of the data. To solve this issue, version control can be used to track changes in the input data, allowing better management and control of the training and testing data.

    2. Underutilized Data Dependencies: This section focuses on the problem of underutilized data dependencies, where a model's performance may be negatively affected by redundant features such as legacy, bundled, epsilon, or correlated features. To address this issue, one solution is to conduct exhaustive leave-one-feature-out evaluations, which entails systematically removing each feature and assessing the model's performance to identify which features are truly significant.

    3. Static Analysis of Data Dependencies: The absence of automated tools for static analysis of dependency graphs in machine learning systems creates challenges in managing the features used in the models, unlike traditional software code. One way

to address this issue is by utilizing automated feature management tools that can perform similar dependency analysis and ensure proper feature management in machine learning systems.

- Feedback Loops
    1. Direct Feedback Loops: Bandit algorithms are theoretically correct but impractical for real-world problems with large action spaces, so randomization or isolating specific data from the model may help.

    2. Hidden Feedback Loops: This involves situations where two systems indirectly affect each other through the environment, and improving one system can cause changes in the other. These loops can occur between unrelated systems.

- ML-System Anti-Patterns
    1. Glue Code: Glue code is code specifically written to connect data with existing open-source packages, proprietary packages, or cloud-based platforms. To address this challenge, it may be helpful to create a custom native solution instead of relying on a generic package, as the majority of the code (about 95%) is used for data transformation to fit these packages.

    2. Pipeline Jungles: This code is a variation of glue code and is commonly used in the data preparation stage for tasks like joining and sampling. It is critical to establish an end-to-end pipeline but testing can be difficult and may necessitate extensive integration testing. A comprehensive approach to data collection and feature extraction can help minimize the amount of glue code needed and make testing more manageable.

    3. Dead Experimental Codepaths: This refers to conditional branches added to the main code, leading to complex system dependencies. To address this issue, regularly reviewing and keeping only the relevant branches can help reduce technical debt.

    4. Abstraction Debt: Abstraction Debt occurs when presenting data, models, or predictions in a clear, abstract manner becomes problematic for machine learning systems.

    5. Common Smell: There are certain smells in ML systems that suggest a problem with a component or system, such as plain-old-data type smell, multiple-language smell, and prototype smell.

- Configuration Debt: Configuration debt can arise in large systems due to the multitude of configurable options, leading to challenges in accurately modifying and reasoning about the settings. Adopting sound configuration practices and systems can help alleviate this issue.

- Dealing with Changes in the External World
    1. Fixing Threshold Dynamics: Manually setting thresholds in ML systems can lead to errors, as new data may differ significantly from historical data. To prevent this, thresholds can be adjusted dynamically based on hold-out validation data during runtime.

2. Monitoring and Testing: In a constantly changing environment, evaluating the performance of a machine learning system based only on training metrics is insufficient, and such tests cannot guarantee that the system is functioning as intended. To ensure that everything is operating correctly, it is necessary to keep an eye on and evaluate different components of the machine learning system throughout its entire lifecycle. Some critical areas that require attention include prediction bias, action limits, and upstream producers.

- Other Areas of ML-related debt: Technical debt is a widespread issue in both software engineering and machine learning, as there are different types of debt that can accumulate over time. In machine learning, technical debt can take many forms, including data testing debt, reproducibility debt, process management debt, and cultural debt. It is essential to address these types of debt early on to ensure the long-term success of machine learning systems.

In order to tackle technical debt in our project, we have implemented multiple measures for mitigation. One of these measures involves the use of two models for our modelling purposes. The first model is an existing one that is already in production, while the second is created on-the-go when new data is introduced. This second model is known as the staging model. If the staging model outperforms the production model, we switch to it, otherwise we continue with the production model. This is done using version control, which enables us to keep track of the changes made to the models over time. To ensure accurate representations and prevent overfitting, we keep the training and testing data separate, and report the metrics on the test dataset. As we have used only a single model, we do not face the issue of correction cascades in our project.

We identified several issues discussed in "Hidden Technical Debt in Machine Learning Systems" during our project, which we built from scratch and deployed. We found that using general packages was convenient because it reduced the risk of errors due to missing libraries. Databricks offered various features that made our work more productive and user-friendly. We used "readstream" during the initial phase of our data-intensive application to read our raw data, which solved the problem of modifying data if new data was added. Databricks provided us with a shared storage space where all team members could modify tables, which helped in creating the pipeline. Databricks was also beneficial because it worked with various scripting languages, and we could split the main repository into branches to enable parallel work by each team member. This was feasible because Databricks can integrate with GitHub repositories.

During our exploratory data analysis (EDA), we utilized Databricks' feature that enabled us to create visualizations without having to write code explicitly for them. However, we encountered a problem as these plots weren't saved, and when someone else ran the code, they couldn't see the visualizations. Later, for model development, we used Prophet, and to support MLops, Databricks had an integration called "MLflow". It enabled us to register the model as a staging or production model. This feature provided us with the ability to save a model and enhance it while the existing model was already in production. The primary benefit of this feature was its security. If one team member ran and registered the model in the MLflow registry, others couldn't overwrite or access its outputs. This resolved a significant issue that arose when multiple team members were working on a large pipeline.

Technical Debt for Citi Bikes ML Pipeline
Sayan Swar

We found this project particularly fascinating since we were working with real-world data, but that also brought uncertainties. The project's objective was to predict the net change in bike trips per hour, which presented some challenges. We faced uncertainty since the data we had did not clearly represent the bike station's restocking process, which was a crucial factor that we couldn't account for effectively. Incorporating this factor would have increased the model's complexity significantly, and even after doing so, the model's output might still have shown insignificant changes.

REFERENCES

1. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... & Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, *28*.