Challenges in Machine Learning Application Development: An Industrial Experience Report

Md Saidur Rahman, Foutse Khomh, Emilio Rivera
{saidur.rahman,foutse.khomh}@polymtl.ca
emilio.rivera@polymtl.ca
Polytechnique Montreal
Canada

Yann-Gaël Guéhéneuc
yann-gael.gueheneuc@concordia.ca
Concordia University
Canada

Bernd Lehnert
bernd.lehnert@sap.com
SAP Montreal
Canada

ABSTRACT

SAP is the market leader in enterprise application software offering an end-to-end suite of applications and services to enable their customers worldwide to operate their business. Especially, retail customers of SAP deal with millions of sales transactions for their day-to-day business. Transactions are created during retail sales at the point of sale (POS) terminals and those transactions are then sent to some central servers for validations and other business operations. A considerable proportion of the retail transactions may have inconsistencies or anomalies due to many technical and human errors. SAP provides an automated process for error detection but still requires a manual process by dedicated employees using workbench software for correction. However, manual corrections of these errors are time-consuming, labor-intensive, and might be prone to further errors due to incorrect modifications. Thus, automated detection and correction of transaction errors are very important regarding their potential business values and the improvement in the business workflow. In this paper, we report on our experience from a project where we develop an AI-based system to automatically detect transaction errors and propose corrections. We identify and discuss the challenges that we faced during this collaborative research and development project, from two distinct perspectives: Software Engineering and Machine Learning. We report on our experience and insights from the project with guidelines for the identified challenges. We collect developers' feedback for qualitative analysis of our findings. We believe that our findings and recommendations can help other researchers and practitioners embarking into similar endeavours.

CCS CONCEPTS

• **Software and its engineering** → *Programming teams.*

KEYWORDS

Software Engineering for Machine Learning, Error Detection and Correction, Challenges and Best Practices

SE4RAI'22 , May 19, 2022, Pittsburgh, PA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9319-5/22/05…$15.00
https://doi.org/10.1145/3526073.3527593

ACM Reference Format:
Md Saidur Rahman, Foutse Khomh, Emilio Rivera, Yann-Gaël Guéhéneuc, and Bernd Lehnert. 2022. Challenges in Machine Learning Application Development: An Industrial Experience Report. In *Workshop on Software Engineering for Responsible AI (SE4RAI'22), May 19, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3526073.3527593

1 INTRODUCTION

Artificial Intelligence (AI) and Machine Learning (ML) have shown promising prospects for intelligent automation of diverse aspects of business and everyday life [6]. The rapid development of machine learning algorithms and tools, and easier access to available frameworks and infrastructures have greatly fueled this era of the development of AI-Based software solutions for real-world problems. Software companies, big or small, are striving to adopt ML in software applications, in order to improve their products and services. However, the promising potentials of machine learning accompany multifaceted challenges to the traditional software development processes and practices [6, 21]. The use of AI/ML techniques in the development of an application may add challenges to all phases of the development life cycle [1]. The requirements for the ML models are expected to be dynamic in nature; to adapt to the rapidly evolving user requirements and business cases. AI-Based applications are data-driven. Thus, efficient pipelines and infrastructures are required for data-streaming; *i.e.,* data acquisition, data storage, preprocessing and feature extraction, and ingestion by the ML models. Also, model building and deployment have different constraints regarding the type of problem, data, and the target environments. Software engineering for machine learning applications has distinct characteristics that render most traditional software engineering methodologies and practices inadequate [21]. The design of AI/ML applications needs to be flexible to accommodate the rapidly evolving ML components. In addition, the testing of AI/ML applications differ significantly from traditional software testing. So, new guidelines are needed to help AI/ML developers cope with these challenges to develop reliable and high-performance AI/ML applications. Additional challenges are likely to surface when the AI-based applications are developed in collaboration of multiple teams with diverse backgrounds and expertise [20]. In addition, there is an utmost need for guidelines for best practices in collaborative research and development between industry and academia in the context of software engineering for ML applications.

In this paper, we contribute to filling this gap by sharing our experience and insights from a research project on AI-based system development conducted in collaboration between Polytechnique Montreal and SAP. We first present our approach to developing

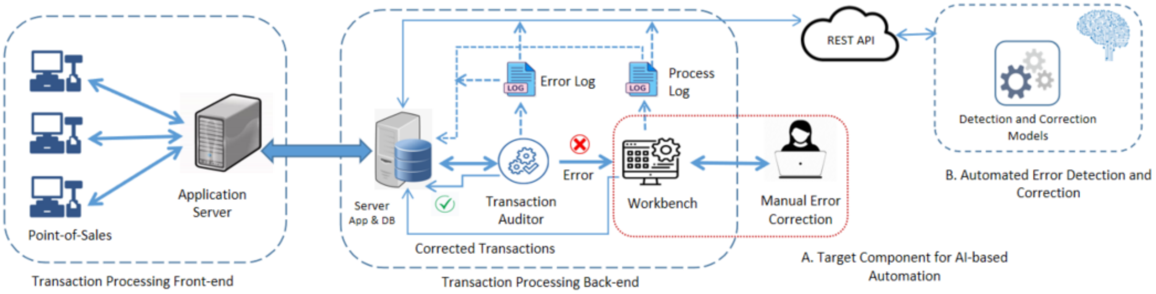
Point-of-Sales
Application Server
Transaction Processing Front-end
Server
App & DB
Error Log
LOG
Process Log
Transaction Auditor
Error
Workbench
Manual Error Correction
Corrected Transactions
Transaction Processing Back-end
A. Target Component for AI-based Automation
REST API
Detection and Correction Models
B. Automated Error Detection and Correction

Figure 1: Conceptual Overview of the AI-based Transaction Error Detection and Correction System

ML-based components for automatic detection and correction of transaction errors. Here, we apply machine learning on retail transaction data to detect and correct transaction errors. We follow an agile methodology to develop the AI-based solution. We identify the challenges in each step of our software development process. We present these challenges along two distinct perspectives: (1) software engineering, and (2) machine learning. From these perspectives, we explore the relationships and dependencies among the challenges to have better insights into the challenges in software engineering for machine learning. We synthesize our observations from the case study and existing knowledge from the literature on the common challenges and best practices, and formulate a set of guidelines for developing AI/ML applications.

Our key contributions are summarized below:

- We have identified important challenges in software engineering for machine learning. Our insights will help developers building AI-based applications.
- We propose guidelines for researchers and practitioners for best practices in the development of AI/ML applications. We do qualitative analysis of our findings based on developers' feedback.
- Our proposed guidelines have been adopted by SAP as part of its internal guidelines for AI/ML development.

The rest of the paper is organized as follows: Section 2 presents the overview of the transaction processing systems to outline the context of the problem addressed. Section 3 outlines our approach for the detection and correction of transaction errors. We share important lessons that we learned from this case study in Section 4. Section 5 presents qualitative analysis of our findings. Section 6 discusses the related works. Finally, Section 7 concludes the paper.

2 OVERVIEW OF THE TRANSACTION PROCESSING SYSTEM

In this section, we briefly describe the transaction processing system to present the context of the problem addressed in this paper.

2.1 Transaction Processing Workflow

Figure 1 shows a simplified conceptual diagram of the retail transaction processing workflow. A retail transaction generally created in POS terminals represents information about a business activity such as the sale or return of item(s) and the payments. Retail transactions are created using the front-end applications of the transaction processing system. The back-end of SAP's transaction processing system manages the storage, validation and post-processing of transactions to support other downstream business functionalities. At the back-end, all transactions are validated by an automated auditor to verify the consistency of the transactions. Any erroneous transaction is flagged by the audit system and a corresponding error message is added to the error log. These errors are due to different technical (*e.g.,* network issues) and human errors (*e.g.,* incorrect product code). This erroneous transactions are then corrected manually by dedicated employees with necessary domain expertise using workbench application (module A in Fig. 1). Changes made in individual steps of the correction procedure are also saved as change logs (process logs).

2.2 Transaction and Transaction Errors

By transaction (Fig. 2) we refer to a set of records (table rows) in the database related to a single order from customer. A transaction consists of hierarchically organized components including metadata, item-level information (*e.g.,* price, product code, quantity, item-specific discounts) and the transaction-level information (*e.g.,* transaction type, taxes, discounts, payments). Each component can have zero or more sub-component(s). For example, an item can have none or several discounts applied on it. The relationships among the transaction components are governed by well-defined business rules. Each transaction must conform to these rules to pass the validation or audit systems. To mention, we cannot present example transactions due to data privacy agreement.

2.3 The Problem

Transactions are the primary business data in retail industries. However, as mentioned, due to many technical and human errors, transactions can have inconsistencies. These errors need to be corrected for business functionalities (*e.g.,* audits) and data integrity. Currently, due to the lack of an automated or semi-automated tool, these errors are corrected manually by dedicated employees of customer companies of SAP. Manual correction incurs high costs, and it is a serious bottleneck to the business operations. Thus, we address the problem by developing and deploying ML-based components for automatic detection and correction of transaction errors. Here, our objective is to build machine learning models based on transaction data and logs to detect (classify) and to correct (predict actions and the correct values) errors in retail transactions. The goal is to deploy the automated error detection and correction service (module B in Fig. 1) to replace existing manual error correction (module A in Fig.1).

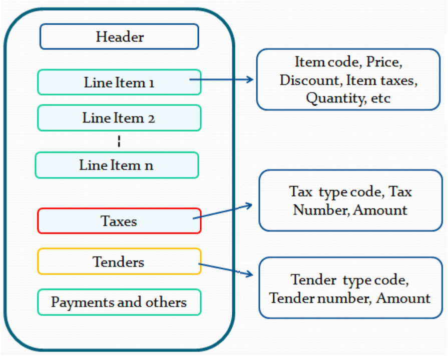
Header
Line Item 1
Line Item 2
Line Item n
Taxes
Tenders
Payments and others
Item code, Price, Discount, Item taxes, Quantity, etc
Tax type code, Tax Number, Amount
Tender type code, Tender number, Amount

Figure 2: Transaction data structure

3 ML-BASED DETECTION AND CORRECTION OF TRANSACTION ERRORS

This section presents our proposed approach for automatic detection and correction of transaction errors.

3.1 Overall Solution Architecture

As shown in Fig. 3, we process transactions data from the database and extract features. Then, we apply machine learning on the features to build ML-models for the components (module B in Fig. 1) for detection and correction of errors. We take a two-step approach. Our first model (Model 1) is a multi-label classifier that detects the types of errors in the erroneous input transaction. Each label in the classification corresponds to a specific error type (and location). Once we have the errors detected, we apply our second model (Model 2) to predict values for the correction of the error. Model 2 is a multi-class classifier that predicts the correction value from the set of possible values for a transaction field.

3.2 Data Acquisition

In our case, transaction data is stored in a large table (TLOG). Another table, the process log table (PLOG) captures all events and changes to transactions that are done either manually or automatically. Although there are millions of transactions in our data set, we are particularly interested in the erroneous transactions that were successfully corrected. Based on the error (task) status of the transactions from PLOG, we select corresponding erroneous transactions from the TLOG table. Again, the TLOG table contains the updated state of the transactions. Thus, once corrected, only the corrected version of an erroneous transaction is available in the TLOG. However, for ML algorithms, we need the erroneous state of the transactions from the TLOG. We apply reverse engineering on transactions in TLOG based on the change-logs in the PLOG to extract the erroneous version of the transactions. The transactions from the TLOG and the change logs in PLOG are the primary sources of data for our analysis.

3.3 Preprocessing Transaction Data

Transaction data is passed through some transformations at SAP customers' end to remove sensitive business and personal information. Not all the transactions with correction logs qualify for our analysis. We filtered out transactions that are missing necessary attributes or that are outliers (with an excessive number of product items).

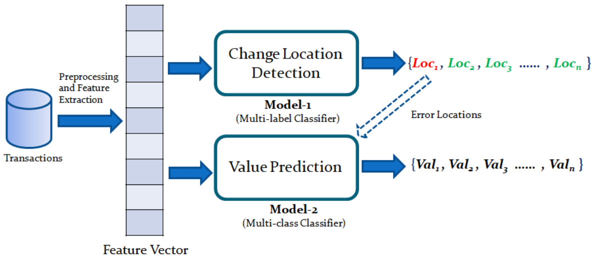
Transactions
Preprocessing and Feature Extraction
Feature Vector
Change Location Detection
Model-1
(Multi-label Classifier)
Value Prediction
Model-2
(Multi-class Classifier)
Error Locations
{Loc₁, Loc₂, Loc₃ ...... , Locₙ }
{Val₁, Val₂, Val₃ ...... , Valₙ }

Figure 3: Error Detection and Correction Inference System Architecture

3.4 Feature extraction

We extract selected transaction field values (*e.g.,* unit price) as base features. We also compute derived features from the extracted base features. Each feature vector is composed of features representing both transaction-level and item-level values of the transaction fields. As transactions are variable in size regarding the number of items, we consider transactions only with maximum 20 retail items based on exploratory analysis to avoid making the feature too sparse. We define a fixed-length feature vector that contains information for a maximum of 20 retail items. Our selection of features are based on the knowledge from domain experts and our understanding of the relationships among the transaction components. We iteratively refined our choice of the transaction fields as features for ML algorithms.

3.5 Error detection approach

A transaction can have multiple errors. Thus, we model our error detection problem as a multi-label classification problem to identify the types and locations of errors. Here, we label the feature vector for each transaction with a binary vector of length equal to the number of possible error classes. The label vector contains 1s in places where there is an error in the corresponding transaction component, and 0s otherwise. We identify errors by examining the manual changes in the process log (PLOG) table. For example, if we detect a change in a tender type code for error correction, we infer that there was an erroneous tender type code. Thus, we set the bit in the label vector corresponding to the tender type code to 1. We apply supervised learning on the labeled dataset to build ML models for multi-label classification for error detection. For error detection, we need transaction data with correction examples. So, we select transactions that were erroneous and have correction histories in PLOG. We exclude erroneous transactions without corrections. Then, we extract features from the erroneous version of the corrected transactions. We also add samples from transactions without any errors in our dataset. These transactions qualify all the selection criteria (*e.g.,* no of items) except they are not erroneous. We split the features into balanced data sets for training, testing, and validation.

3.5.1 Multi-label Classifier. We apply Random Forest algorithm for multi-label classification. Random Forest algorithm can be applied for both classification and regression problems and can handle the over-fitting problem of Decision Tree algorithm. Random Forest algorithm is an ensemble learning method that constructs a multitude of decision trees. The classification and prediction decisions are based on the combination of output of the constructed decision trees. In addition, we also apply Decision Tree, and AdaBoost Classifier to detect errors in transactions. However, RandomForest outperformed both of these algorithms. We also apply neural networks (NN), specifically Multilayer Perceptron (MLP), for error classification. However, because of our limited number (less than 100 to a few hundred for most error types) of error correction samples, MLP could not be trained properly for most of the error types. Finally, we opted for Random Forest which achieved the best performance.

3.6 Error correction approach

By error correction, we refer to predicting the correct values for the erroneous transaction components. We model the value prediction problem as a multi-class classification problem. The predicted class refers to a value from the possible set of values. Each model is trained for a distinct type of error. The features are extracted from the erroneous transactions in TLOG and the corresponding logs in the PLOG. The new value of a changed field of an erroneous transaction is set as the target label for the algorithm to predict. For the categorical field values (*e.g.,* Tender Type Code), we need to predict the correct value from the set of alternative values for a given field. Here, the target label is a one-hot encoded vector of possible values for the associated field. We build our error correction model based on the transactions with corrections available in the transaction database. We split the features into balanced data sets for training, testing, and validation.

3.6.1 Multi-class classifier. We formulate the error correction problem as a multi-class classification problem. Here, the classifier picks one of the values from all possible alternative values as the correct value. The label vector for each data sample represents a binary vector with '1' in the index of the correct value and 0's in all other places. We apply the Logistic Regression (one versus all) algorithm for the value-prediction problem. We also applied Multilayer Perceptron (MLP) neural networks to predict values for error correction. As we have mentioned earlier, we have limited correction data samples for most of the error types. Consequently, NN models performed low compared to simple Logistic Regression. Limited data size and class-imbalance issues are likely to blame for the poor performance of NNs.

3.7 Model Performance

We measure precision, recall, accuracy score and Jaccard similarity score of the models to evaluate model performance.[1,2] Here, *precision* refers to the ratio of the number of correctly predicted or classified cases to the total number of cases and *recall* is the ratio of the total number of correctly predicted or classified cases to the total number of true cases. *Accuracy*, on the other hand, refers to the ratio of correct prediction to the total number of prediction. We also measure *Jaccard Similarity* which is the ratio of the cardinalities of the intersection and the union of two sets of labels (predicted and the true set). In this section, we present the performance of the models that detect "Tender Type code" errors.

3.7.1 Error Detection. Fig. 4 presents the error detection performances of four different model configurations regarding error types. Here, Model1s, Model2s, and Model3s are to predict errors only in the first, second and third tender type code respectively. Model3a is trained to predict all of the first three tender type code errors. We present the error detection performance based on the Random Forest algorithm. Here, the first group of bars in Fig. 4 (Model1s) represents the error detection performance when only errors in the first tender type code were considered. Here, we observe that the detection accuracies of the model are above 90%. However, for the second (Model 2s) and third (Model 3s) models, the detection performance drops. This declination in performance might be related to the comparatively lower number of training samples for each category. Again, when we include all the first three tender type codes in the detection model (Model 3a), we see a slight increase in the detection performance (above 80%). We observe that the size of the training dataset influences the training of the models and their prediction accuracies.

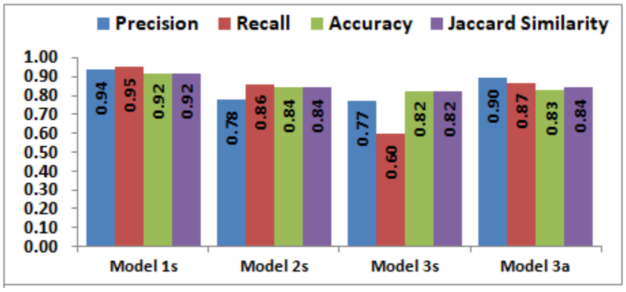

Figure 4: Error detection performance

3.7.2 Value Prediction. For value prediction we compute the accuracy of top-k recommendation of correction values. The performance is based on the results obtained by applying the Logistic Regression Cross-validation (one-vs-rest) algorithm for multi-class classification. Here, the value prediction accuracy of the model is 76% if we consider single predicted value with the highest probability (*i.e.,* k=1). However, we also evaluate the prediction accuracy for top-k (k = [1, 5]) of the model. We see that the accuracy of top-5 prediction containing the expected value is 93%.

3.8 Research and development methodology

We adopted the agile methodology (SCRUM [22]) considering the incremental and iterative nature of the problem. SCRUM accelerates software development with faster delivery cycles while offering flexibility [20]. Our team comprise members from academic and industry partners with diverse expertise on software engineering, machine learning, target software ecosystem and the business model of SAP. In our agile process, the length of each 'sprint' was 30 days with 'daily stand' to update on changes towards the sprint goals. There were weekly meetings to review and to sync progress while there were biweekly meetings for all members from academia and industry to review the progress on the sprint milestones and deliverables. We documented the challenges and discussed solutions to those challenges during our meetings. We later analyzed those documentations to derive lessons on the challenges and best practices in each phases ML application development. Each sprint ended with a 'sprint demo' and review of any 'retrospective' to address backlogs. The whole project spanned over six sprints.

[1] https://en.wikipedia.org/wiki/Precision_and_recall
[2] https://en.wikipedia.org/wiki/Jaccard_index

4 LESSONS LEARNED

Machine learning applications have some distinct characteristics compared to traditional software applications. Thus, software developers and practitioners should be aware of a number of challenges or risk factors regarding ML applications [23]. In this study, we adopted our methodology for a systematic case study based on the guidelines from Wohlin *et al.* [26] and Yin [28]. Here, the proposed error detection and correction system serves as an example case to derive insights for the practitioners. We review existing literature to explore software engineering methodology [1, 29] for ML applications, common challenges [16, 21, 25] and best practices [30] for ML application development. We synthesize our experience from the project with the existing knowledge to list key challenges in ML application development and to formulate recommendations for practitioners. We describe our experience from the case study along two different perspectives: software engineering (S) and machine learning (M). We explore their relationships to have insights into the software engineering principles and practices for machine learning applications.

4.1 Software Engineering Perspectives

In this section, we discuss challenges in different phases of software development life cycle (SDLC) and recommend best practices for ML applications as follows:

4.1.1 ***S1: Requirements Engineering:*** Poor quality of requirements can lead to many issues in the successive phases of the software development [7]. Requirement engineering for ML applications involves both ML specific and traditional requirements engineering activities such as feasibility analysis, requirements gathering, requirement specification and validation. As the requirements in ML applications may change frequently, requirements specification for ML applications is a challenging task [2]. In our project, we gathered requirements from the discussions and demonstration by the domain experts. We analyzed the transaction data structures, example error types, and their correction procedures. We iteratively refined our requirements specification based on the results of our different prototype models and the feedback from the domain experts. In our project, the members from the academic partner did not have direct interactions with the end-users (SAP client). The requirements were gathered and shared by the industry partner. We observed that direct communication with the end-user and the on-site observation of error correction scenarios is important to speed up requirement elicitation. In fact, the round-trip discussions after each model prototyping using the available data would be more efficient if information about the performance of the models and resulting requests for additional data collection were shared directly with users. Iterative refinement of the requirements based on the feedback from the stakeholders can help eliminate the differences in perception of the requirements. Otherwise, delayed identification of requirements and the consequent changes to the ML applications can be costly.

4.1.2 ***S2: Design:*** Software design specifies the scope, functionalities, and interactions of the software components. Unlike traditional software systems, the behavior of the ML applications can be unpredictable and defined by the training data. Again, ML applications require a large volume of data. Thus, their design needs to accommodate the constraints and overhead on data processing. ML application design should be flexible to adopt rapid changes in ML algorithms and frameworks. Again, the performance of the system may degrade due to changes in data pattern (*i.e.,* concept drift) over time without changes in the requirements or without the presence of bugs. Thus, the maintenance requirements of the ML applications may be hard to predict. So, the design need to be flexible to accommodate such changes. In cases of adding AI/ML capabilities to the existing application, the design should consider minimum restructuring of the existing system architecture. Similarly, the design of a new ML application should be flexible to adapt future changes. In our case, we add functionalities to the existing application and we focused on the functional requirements of the modules and the interfaces for interaction with the existing application.

4.1.3 ***S3: Implementation:*** Development of ML applications commonly involves the use of different frameworks and libraries. However, it is hard to put together a diverse set of frameworks and libraries and to ensure compatibility and integrity of the system. Again, ML models are “Black Box" and are hardly explainable [4, 5]. Thus, it is hard to clearly understand why they work and why sometimes they do not. In addition, ML models may exhibit robustness to noises [8] making it harder to verify the implementation. Again, the development environment for ML models might be different from the production environment. So, the implementation of ML applications should consider the target platform requirements while choosing frameworks and libraries. Also, the hardware-software ecosystem for ML applications are rapidly evolving. Thus, the implementation choices should also consider maximizing portability, compatibility, and adaptability of ML applications with lower cost. In our project, we needed exploration and experimentation with different algorithms. So, we implemented a framework that offers flexibility to apply different ML algorithms. Our implementation strategy leverages code reusability. We needed to use different ML frameworks (scikit-learn, TensorFlow, Keras) to implement our ML-based components.

4.1.4 ***S4: Integration:*** Integration process aggregates the components and must also ensure the functional integrity of the system. The integration of ML applications can be viewed as a two-step process; first integrating the sub-components of the ML component and then the integration of the ML components with other non-ML components. Thus, the defined interface between ML components with other components may influence the process and complexity of the integration phase. Again, ML models are expected to evolve continuously. Thus, the ML workflow should facilitate continuous integration of ML models. In our case, we needed to define a new interface for SAP application so that inference from the containerized ML models hosted either locally or on remote cloud can be accessed as services through APIs.

4.1.5 ***S5: Testing:*** As the outcomes of machine learning models are usually stochastic in nature [8, 15], there might not be unique results to compare and verify machine learning applications. Thus, the existing unit testing frameworks (*e.g.,* PyUnit for Python) cannot be readily used to test ML applications. Again, ML models learn from the input data in an adaptive and iterative manner [17]. The rules learned by the ML models depend on many parameters such as the selected features, the model architecture and even the training data. The rules generated by ML systems might even be unknown to the developers [17]. ML models are opaque, and it is hard to explain the model behaviors. Thus, it is harder to identify the erroneous system behaviors and to pinpoint the source of the bugs. Again, ML algorithms may sometimes exhibit robustness to some bugs and produce reasonable outcomes by compensating for noisy data or implementation errors [8]. Thus, bugs in ML application can be tricky to detect and fix. All these issues make the testing and fixing of erroneous behavior in ML applications a very challenging task. For our ML components, we evaluate the model accuracies with the evaluation data set. We also unit-tested individual modules and tested the overall functionality with integration testing.

4.1.6 ***S6: Deployment:*** Deployment phase puts the software system into production by either updating or replacing the existing system. For ML applications, this phase is more likely to add new functional module(s) to the existing system. One important challenge to consider in ML system deployment is that the platform and infrastructure for production system might be very different from the environment the ML model was trained and evaluated. These differences can pose compatibility, portability and scalability challenges and may affect the performance. In our project, we developed a framework to deploy the models and to evaluate their performance. Our design aimed at creating an ML-based service for the existing applications. We tested our model deployment as REST API-based web services for error detection and correction. We took different factors into account for the deployment of our models such as scalability, performance, resource requirements and the ease of model maintenance. The deployment and roll-out strategy should be cautious not to affect the system users.

4.2 Machine Learning Perspectives

ML application development requires a well-defined set of principles and guidelines as recommended by practitioners [30]. Here, we highlight some key challenges and practices for ML application development from ML perspectives.

4.2.1 ***M1: Problem Formulation:*** Machine Learning algorithms offer general-purpose solutions. We need to formulate problems appropriately to fit into the ML-based solution space. One key challenge to formulating an ML problem is that one should clearly understand the problem, the algorithms and the mapping of the problem from the original domain to the ML solution space. This requires an ML solutions architect to have diverse set of skills and expertise. The correct formulation of the problem is a prerequisite to the success of other phases of ML application development such as data acquisition, selection and extraction of features, and model building. In this study, we analyzed the transaction processing workflow based on domain knowledge from industry experts. Our problem definition is based on the domain understanding, transaction data structures and organization, types and characteristics of errors and their manual correction process. We also considered the relationships and dependencies among the transaction components.

4.2.2 ***M2: Data acquisition:*** Collection and processing of large volume of data are critical overheads for machine learning [19]. Insufficient data is also a problem for machine learning applications. The data acquisition must focus on the completeness (representative of full range of behaviours), accuracy (correctness of the data), consistency (no contradictory data), and timeliness (relevant to the current state of the system) of data to ensure data quality [10]. However, maintaining data quality requires careful steps in collection, curation, and maintenance of the data. This is often very expensive regarding the time and associated manual labour [10]. In our study, the transaction data was provided by SAP from a client company. Sensitive business and personal information was removed from the transaction. Although SAP solutions have a comprehensive set of features, the clients have the flexibility to customize data structures and functionalities. These customizations add challenges for the ML models to generalize for different clients for the same uses cases. Our analysis suggests that some prior analysis of the data and the problem is important to set appropriate data requirements, especially before the acquisition of a large volume of data. The structure and distribution of the data may evolve over time, the data acquisition process should be flexible to easily adapt such changes in data.

4.2.3 ***M3: Preprocessing:*** For a data-driven system, the fact is "garbage in garbage out". Noisy data is claimed to one of the top challenges for ML practitioners. Noisy data can adversely affect the learning and thus the inference of the models. Thus, raw data need preprocessing to remove noise, to fill in the missing values, and to do other transformations. Data preprocessing is a challenging phase for machine learning and may incur a significant proportion (>50%) of time and effort [12]. In our study, we first performed exploratory analysis of transaction data to understand the attributes of the transactions. We observed that overall structure of the data, the types and range of values of individual fields may demand careful attention and warrant specific preprocessing task to ensure correctness and consistency of the data. We applied a set of heuristics on the transaction data to filter noises. Our preprocessing steps include filling the missing values and the normalization of data fields to correct formatting differences. To reduce data preprocessing overhead, ML workflow should maximize the reusability of the preprocessed data.

4.2.4 ***M4: Feature Extraction:*** Feature extraction performs optimal transformation of input data into feature vectors for the machine learning algorithms [9]. Features should best represent the hidden characteristics of the data. Feature extraction also aims to remove the potential noises and redundancy from the data [9]. This also finds a low-dimensional representation of the data to reduce model complexity, and to improve the speed of the training and inference of ML models. However, it is challenging to extract features by processing a high volume of data. In our case, we considered the domain knowledge and the transaction data structure and relationships as the basis of feature selection. In addition to base features, we also extracted derived features for our ML models. The quality of features greatly influences the performance of the ML models. Thus, ML developers must find the best set of features. There should also be a periodic review of the features as data evolve [30].

4.2.5 ***M5: Model Building:*** ML models are created based on specific ML algorithms depending on the problem and the characteristics of the data. One frequent problem for ML models is 'overfitting'when a model performs well on the training dataset but does not generalize. This might be because the model is too complex resulting from the higher dimensional features and the complex or deeper architecture of the model. The solution is to find the simplest model that does the required job. However, too simple models may fail to capture the hidden patterns in the data causing 'underfitting'. ML developers should start with a simpler solution (build a baseline model) and gradually adopt more complex solutions considering the resource-performance trade-off. Also, data distribution need to be balanced. Otherwise, the inference of the model might be biased towards the dominating class of the training data. In our project, we observed overfitting issues with neural networks of comparatively deeper architecture for error classification. We eliminated feature redundancies and tuned neural network architectures to avoid overfitting. Class-imbalance problem was another important issue as the distribution of error samples were very skewed to a limited class of errors. However, we balanced (stratified) the training, test, and evaluation dataset to improve model performance.

4.2.6 ***M6: Evaluation:*** ML models are evaluated by applying the ML models to the evaluation dataset exclusively separated from the training data set. The evaluation dataset should be complete enough to represent all possible use-case scenarios. Both pre-deployment evaluation and post-deployment performance monitoring are important, as the performance may drop with changes in the input data characteristics. Thus, ML models may need to update (*e.g.,* retrain) to adapt to the changes. So, the evaluation of ML models can be iterative. Also, there might be different interacting models. So, the performance of the individual models may only represent a part of the end-to-end scenarios. Thus, it is important to have both model-level and system-level evaluations. In our study, we evaluated the individual models and also the overall performance after the integration of the models.

4.2.7 ***M7: Model integration and deployment:*** Trained models are integrated into the target application. This involves putting all necessary components (e.g., models, input-output pipelines) together. For multiple models, it may require to define and implement the interface for each model to interact with other models and system components. One common approach is to deploy ML models as services and accessing the services through APIs. The deployment should consider the portability and compatibility of the ML models regarding the target platform. In our project, we develop an interface between the ML-based components and the existing application for model deployment to provide the error detection and correction services. The deployment must ensure a smooth roll-out of the existing system without affecting the user or the business process.

4.2.8 ***M8: Model management:*** ML model management involving creation, deployment, monitoring, and documentation of ML models is a challenging task in ML workflow [21]. ML models are data-driven and are based on different assumptions on the distributions and patterns of data. However, initial characteristics of data may evolve which may affect the model behaviour. Thus, it is important to monitor the performance of the deployed models, track changes in the data characteristics, and also, retrain and re-validate the models as necessary. These require iterations on ML model life-cycle activities which are often very expensive regarding time and resources. It is also important but hard to keep track of the model versions, dataset and configurations to allow *reproducibility* of ML models [29] and easier management of ML workflow.[3] Model reproducibility helps us to analyze and compare model behavior and performance, and also supports deployment or roll out decisions. We defined policies to version data, keep track of models and configurations for comparative analysis of our ML models. Each phase of the ML life-cycle should be well-documented to support model maintenance.

4.2.9 ***M9: Ethics in AI development:*** Development of ML applications must adhere to AI ethics principles to ensure responsible use of AI "responsible" use of AI.[4] Researchers and practitioners should also adhere to the standard code of ethics for Software Engineering.[5] In our project, we maintained strict data privacy and security policies. Personal information was filtered out from the transactions, and all the team members followed a non-disclosure agreement that protects the privacy and security of personal and business-sensitive information. For ethical use of AI/ML, collective well-being must get priority over business gains.

5 QUALITATIVE ANALYSIS OF THE FINDINGS

We performed qualitative analysis of our findings based on the feedback from anonymous developers from SAP. The aim is to have developers' insights into the identified challenges and proposed recommendations for ML application development regarding the completeness, generalizability, and usability or practical implications. We received feedback from seven developers (P1-P7) from SAP who are not related to this research project. One developer from SAP volunteered for the collection and anonymization of the feedback. Our list of challenges and recommendations were shared with the developers, and they provided feedback by email to the coordinator on two questions, (1) *How well the identified challenges reflect the challenges in your ML application development context?*, and (2) *How actionable the proposed guidelines are regarding your own ML development context and beyond?*.

Most of the developers found our findings relevant to their development context and actionable. P2 mentioned that (the findings) "*reflect very well the challenges that we have experienced in several ML projects*". The findings "*reveals the complexity aspects of ML projects* (P3)" and "*the identified challenges and recommendations are relevant and comprehensive* (P6)". P7 mentioned that the findings "*are relevant and valuable for the daily life of a data scientist and developer.*" while P3 considered the recommendations "*suitable for a whole team that starts with a new ML project*". Regarding usability in practice, P1 commented, "*we could profit substantially from their experiences (recommendations)*". P4 mentioned, "*from the projects I did so far, I find the inputs (recommendations) very usable and it could serve as a good checklist of what to be aware of in a project*". P2 was aware of the recent adoption of the proposed recommendations as part of the internal ML guidelines at SAP and added that the recommendations served as the "*basis for deriving two concrete guidelines: one for developers and architects and one for data scientist which we have applied a couple of times now. So I would say the paper (findings) are actionable*". P3 highlighted the data quality (M2, M3) issue saying that "*in our project, we faced severe data quality issues. Other teams reported the same: garbage in, garbage out*". P5 shared how data privacy concerns posed challenges in developing ML based solutions by their team for banking systems. P7 mentioned that ML

[3] https://code.fb.com/ml-applications/introducing-fblearner-flow-facebook%2Ds-ai-backbone/

[4] https://ai.google/responsibilities/responsible-ai-practices/

[5] https://www.computer.org/web/education/code-of-ethics

aspects such the "*explainability of ML results are currently discussed in our community*". However, some developers emphasized on the guidelines to be more specific (P3, P5, P7) while acknowledging that "*each ML project is different and might need a different focus on the single dimensions (SE or ML) and its importance*" (P3). The developers also mentioned some suggestions from their experiences which we used to refine our recommendations. Thus, our qualitative analysis shows the potential benefits of our recommendations. However, it is not surprising that our recommendations in some cases do not outline the detail execution. This is because our project context does not cover all different aspects of ML development. We thus need to carry out further investigations to have more concrete and context-specific guidelines.

6 RELATED WORKS

Challenges in traditional software engineering process have been widely addressed by the researchers [20]. However, there is a growing need for guidelines and best practices for developing ML applications. Schelter *et al.* [21] focused on ML model management regarding use cases from conceptual, data management, and engineering perspectives. Amershi *et al.* [1] highlighted challenges in AI application development in Microsoft and shared how the teams address those challenges. Zinkevich [30] presented a concise set of guidelines for best practices in ML engineering. Many existing literatures focus on the data acquisition [3, 19], data preprocessing [13, 18], feature extraction [9, 25], model management [21], testing [8, 17] and deployment [21] of ML applications.

Lanubile *et al.* [14] shared practitioners perceived challenges in putting ML models into production. They reported that there is lack of support for developers to adopt software engineering best practices for AI-based system development. Serban *et al.* [24] argued that guidelines for the development of trustworthy ML systems should be adopted as operational best practices for ML development lifecycle. Heck and Schouten [11] suggest that for AI engineering, the software development process needs to be extended with data and model engineering. They highlighted the importance of collection, storage, cleaning and visualization of data and accommodating insights from concerned domain experts. Xie *et al.* in their recent study [27] reported that most of the existing studies did not focus enough on the data management and model production, and further research should address AI life-cycle management. Existing literature shared useful insights on challenges for ML and software engineering separately. However, there is a growing need for a consolidated set of guidelines regarding software engineering for machine learning. We aim to contribute to filling this important gap by reporting about our experience building ML application in an industrial context.

7 CONCLUSION

In this paper, we share our experience on the development of ML-based automatic detection and correction of errors in retail transactions. We outline our detail approach involving research and development to solve this important real-world problem in the retail business. We identify challenges in ML application development from software engineering and machine learning perspectives. Based on our experience building these ML components and the principles and practices in software engineering and machine learning, we report some important insights and highlight some recommendations that we believe will be useful to researchers and practitioners embarking in engineering ML applications.

ACKNOWLEDGMENTS

We gratefully acknowledge the funding support by Mitacs Canada and SAP Canada for this research.

REFERENCES

[1] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. 2019. Software Engineering for Machine Learning: A Case Study. In *Proc. ICSE-SEIP*. 291–300.

[2] M. Attarha and N. Modiri. 2011. Focusing on the importance and the role of requirement engineering. In *Proc. ICIS*. 181–184.

[3] J. Attenberg, P. Melville, F. Provost, and M. Saar-Tsechansky. 2011. *Selective data acquisition for machine learning*. 101–155.

[4] A. Bibal and B.ît Frénay. 2016. Interpretability of machine learning models and representations: an introduction.. In *ESANN*.

[5] F. Doshi-Velez and B. Kim. 2017. Towards A Rigorous Science of Interpretable Machine Learning. arXiv:1702.08608

[6] F. Khomh and G. Antoniol. 2018. Bringing AI and machine learning data science into operation. https://www.redhat.com/en/blog/bringing-ai-and-machine-learning-data-science-operation/.

[7] D. Firesmith. 2007. Common Requirements Problems, Their Negative Consequences, and the Industry Best Practices to Help Solve Them. *J. Object Technol.* 6, 1 (2007), 17–33.

[8] R B Grosse and D K Duvenaud. 2014. Testing mcmc code. *arXiv preprint arXiv:1412.5218* (2014).

[9] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh. 2008. *Feature extraction: foundations and applications*. Vol. 207. Springer.

[10] P. Hebron. 2016. *Machine Learning for Designers*. O'Reilly Inc.

[11] P. Heck and G. Schouten. 2021. Lessons Learned from Educating AI Engineers. In *Proc. WAIN*. 1–4.

[12] F. Herrera, S. Ventura, R. Bello, C. Cornelis, A. Zafra, D. Sánchez-Tarragó, and S. Vluymans. 2016. *Multiple Instance Learning*. Springer.

[13] G. E. Hinton and R. R. Salakhutdinov. 2006. Reducing the Dimensionality of Data with Neural Networks. *Science* 313, 5786 (2006), 504–507.

[14] F. Lanubile, F. Calefato, L. Quaranta, M. Amoruso, F. Fumarola, and M. Filannino. 2021. Towards Productizing AI/ML Models: An Industry Perspective from Data Scientists. In *Proc. WAIN*. 129–132.

[15] C. Murphy, G. E. Kaiser, and M. Arias. 2007. An approach to software testing of machine learning applications. (2007).

[16] A. Paleyes, R. Urma, and N. D. Lawrence. 2020. Challenges in Deploying Machine Learning: a Survey of Case Studies. *CoRR* (2020). https://arxiv.org/abs/2011.09926

[17] K. Pei, Y. Cao, J. Yang, and S. Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *Proc. SOSP*. 1–18.

[18] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Wozniak, and F. Herrera. 2017. A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing* 239 (2017), 39 – 57.

[19] Y. Roh, G. Heo, and S. E. Whang. 2021. A Survey on Data Collection for Machine Learning: A Big Data-AI Integration Perspective. *TKDE* 33, 4 (2021), 1328–1347.

[20] A. B. Sandberg and I. Crnkovic. 2017. Meeting Industry-Academia Research Collaboration Challenges with Agile Methodologies. In *Proc. ICSE-SEIP*. 73–82.

[21] S. Schelter, F. Bießmann, T. Januschowski, D. Salinas, S. Seufert, and G. Szarvas. 2018. On Challenges in Machine Learning Model Management. *IEEE Data Eng. Bull.* 41 (2018), 5–15.

[22] K. Schwaber and M. Beedle. 2002. *Agile software development with Scrum*. Vol. 1. Prentice Hall Upper Saddle River.

[23] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young. 2014. Machine Learning: The High Interest Credit Card of Technical Debt. In *SE4ML (NIPS 2014 Workshop)*.

[24] A. Serban, K. van der Blom, H. Hoos, and J. Visser. 2021. Practices for Engineering Trustworthy Machine Learning Applications. In *Proc. WAIN*. 97–100.

[25] D. Storcheus, A. Rostamizadeh, and S. Kumar. 2015. A survey of modern questions and challenges in feature extraction. In *Feature Extraction: Modern Questions and Challenges*. 1–18.

[26] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in software engineering*. Springer.

[27] Yuanhao Xie, Luís Cruz, Petra Heck, and Jan S. Rellermeyer. 2021. Systematic Mapping Study on the Machine Learning Lifecycle. In *Proc. WAIN*. 70–73.

[28] R. K. Yin. 2017. *Case study research and applications: Design and methods*. Sage publications.

[29] M. Zaharia. 2018. MLflow: A platform for managing the machine learning lifecycle. https://www.oreilly.com/content/mlflow-a-platform-for-managing-the-machine-learning-lifecycle/.

[30] M. Zinkevich. 2017. Rules of machine learning: Best practices for ML engineering. https://developers.google.com/machine-learning/guides/rules-of-ml.