Self-Adaptive Financial Fraud Detection System

Aman Kumar
Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada
aman.kumar1@uwaterloo.ca

Manan Raheja
Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada
mraheja@uwaterloo.ca

***Abstract*—With the advancement and easy availability of financial services around the world, the scope of fraudulent transactions has also grown multifold. Financial institutes tackle this challenge by employing fraud detection systems based on advanced machine learning methods. However, financial transaction data observe a significant data drift arising from fraudsters exploiting the existing mechanisms to develop newer methods of committing credit card transaction fraud. As a result, the conventional static machine-learning models, initially effective, experience a notable decline in performance over time. In this paper, we present an implementation of a self-adaptive financial fraud detection system. This system integrates a MAPE-K loop into an MLOps process, enabling continuous monitoring of data drift and autonomous adaptation to emerging patterns. By dynamically updating and retraining the machine learning model, the proposed system effectively maintains high-performance levels, even in the face of significant data drift. The presented results underscore the efficacy of this self-adaptive framework, showcasing its ability to sustain optimal performance and adaptability amidst the ever-changing landscape of financial transaction data.**

***Index Terms*—Self-Adaptive Systems, MAPE-K loop, Machine Learning Operations (MLOps), Financial Fraud Detection**

I. Introduction

With the advancement and easy availability of financial services around the world, the scope of fraudulent transactions has also grown multifold. A fraudulent transaction is one in which the actual account holder didn't approve of the transaction being committed and has potentially incurred some financial loss at the behest of purchasing goods or services or simply transferring money to another account which was deemed to be trustworthy. Beyond financial loss, fraudulent transactions often lead to sale of illegal substances, theft, misrepresentation in criminal cases, and promotion of unfair practices in the society. Some examples of fraudulent transactions include unauthorized sale of goods and services, impersonating the account holder by stealing their credentials, impersonating the person or business entity to which the money was being transferred, transferring more money than what was shown in the invoice, transferring more money than what is allowed or available in the account, and transactions involving sale and purchase of illegal goods and services.[2][1]

Traditionally, financial institutions have been countering fraudulent transactions by employing various means like blocking transactions to or from an identified fraudster's account, imposing appropriate limits on how much money can be transacted in a specific time frame and installing various mechanisms to authenticate the entities to and from which the money is being transferred. However, these measures cannot accurately predict a given transaction as fraudulent or not unless it is reported so. This is where ML-driven models come into play. Since fraudulent financial transactions can have terrible implications for the victim, the margin for error is very less, due to which the ML models need to be highly accurate. [3]

It is nowadays easy to set up a static ML model as financial institutions have huge datasets of transactions which have a large amount of parameters describing the transactions. However, many of these parameters are often strongly correlated and give the fraudsters a wide array of patterns to play around with, and static models find it increasingly difficult to keep up with these changing trends. This is where we have to come up with MLOps - a systematic way of combining state-of-the-art practices and infrastructure to deploy and manage a machine learning model. [6]

Fraudsters are exploiting the existing mechanisms in place to come up with newer methods of committing credit card transaction fraud. In the case of credit card fraud detection, we see that there is a significant data drift arising from new patterns and the ever-changing technological landscape. The problem gets further complicated as financial institutions don't allow transaction datasets to be made public, in order to prevent leaking their users' credentials. Thus, the datasets that are available have to be anonymized implying that (most of) the column names are generic. As a result, financial institutions often resort to developing in-house machine learning and fraud prevention mechanisms, which are inherently difficult to manage and not easily scalable.

As organizations increasingly turn to AI and ML for data-driven decision-making, the effective deployment and management of models become pivotal in realizing the full potential of these technologies. Once the ML models are in place, deploying them and maintaining them becomes a challenge. This is where cloud-based services come into the picture. [7]

A self-adaptive system is one which satisfies at least one of the following four self-CHOP properties: self-configuring, self-healing, self-optimizing and self-protecting. By making a system self-adaptive, the system becomes more robust and self-managing without the need of human intervention. The process of achieving self-adaptability consists of setting up

a MAPE-K loop over the existing system which will govern the overall system's behaviour. The loop consists of four key components - Monitoring, Analyzing, Planning and Executing. [5]

Monitoring aspect is responsible for constantly detecting changes in the system's performance over time and tracking down various parameters of interest. The observations from Monitoring are then passed onto the Analyzing phase, wherein the system compute the average statistics for those parameters and compares them with pre-determined thresholds, eventually constructing a Utility Function using those parameters and computing its value at a given instance of time. This value has to be optimized as a key objective of the MAPE-K loop. If it is observed that this value is not in optimized state, the system moves to Planning phase wherein it decides the sequence of operations to execute such that the utility function value becomes optimized. Once the planning actions are determined, the system moves onto the Executing phase wherein it brings these changes into effect and completes the MAPE-K loop, circling back to the Monitoring phase.[5]

We are interested in the self-optimizing aspect in this project, as we want to monitor the ML model, analyze its performance on new data, plan and devise strategies to train it for new trends and execute these strategies, completing the MAPE-K loop. We conducted two experiments - one online on IBM Cloud infrastructure and the second one offline on the local system. In both of these experiments, we first trained the model on an initial corpus of credit card financial transaction dataset and then moved onto configuring a MAPE-K loop to make the whole process self-adaptive.

The rest of the paper provides details on the experiments we conducted and their observations and concludes with a recap of the entire process and discusses future work.

II. Methodology

In this project, we have implemented a self-adaptive credit fraud detection system that constantly monitors the drift in the data and, if it observes a data drift, it adapts accordingly to keep performing well in the new environment.

A. Simulating streaming data and data drift

In the real world, automated data streams continuously feed the verified transaction data to data repositories of the financial institute. To simulate such a real-world world scenario, machine learning engineers usually use a live API that provides the MLOps process with a continuous data stream. For our use case, a live API was needed that could stream credit card transaction data and the corresponding fraud label for each transaction. However, such a public API wasn't available. So, we developed a mechanism to simulate a continuous data stream from a static dataset. A separate system was built that would take a static dataset as its source and it would periodically stream financial transactions to a data repository accessible by the detection system. One important design feature of this data streaming system is that it creates a stratified data stream, which means, for example, if the static dataset contained only 1% fraud transactions and the rest of the transactions were genuine, then the data stream would also maintain the same proportions in the data stream. In the area of financial fraud detection, we deal with highly-imbalanced data where only 1-2% of all the transactions correspond to the fraud class therefore, this feature allows us to accurately simulate the real-world conditions.

Another challenge was to simulate data drift in the MLOps process, which is the phenomenon where the new unseen data changes over time having different patterns and different statistical properties than that was used to train the original machine learning model. This drift in the data causes a considerable drop in the performance of the model, especially in financial fraud detection models. This challenge was tackled by selecting two slightly different versions of the same dataset. The first dataset was collected and analyzed during a research collaboration between Worldline and the Machine Learning Group of Université Libre de Bruxelles (ULB) on big data mining and fraud detection [8]. The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) accounts for 0.172% of all transactions. It contains only numerical input variables which are the result of a Principal Component Analysis (PCA) transformation. Unfortunately, due to confidentiality issues, the researchers have not provided the original features and more background information about the data. Features V1, V2,... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction amount. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

The other dataset is an augmented version of the above dataset and it comprises over 550,000 records [4]. This dataset has been augmented in such a way that out of all transactions, half are fraudulent and half are genuine. All the features are also the same, except we have 'id' in place of 'Time'. These slight differences in the datasets allow us to simulate data drift by training the initial machine learning model on one dataset and then creating a continuous data stream using the other dataset as the source.

B. Data engineering and model development

Given the dataset, the next step is to build a data engineering pipeline to prepare the data for training the machine learning models. This pipeline includes performing data transformation, data cleaning, data normalization, and feature selection.

As mentioned above, due to confidentiality issues, the researchers have not provided the original features and more background information about the data features. This renders conducting exploratory data analysis pointless because data variables do not represent any real-world features so we will not be able to extract insights about features and their

relationship with each other. Hence, we decided to focus entirely on data engineering and model development.

Therefore, we have performed the following steps in the data engineering pipeline -

1) Data transformation: In this step, we dropped the identifier columns from the datasets: 'Time' and 'id'. Additionally, we separated the 'Class' column which is the target variable, and stores whether a transaction is fraud or genuine.
2) Data cleaning: In this step, we removed all the rows containing missing values in any of the columns. For these datasets, fortunately, there were no missing values.
3) Data normalization: Finally, we performed scaling of the features by performing z-normalization on all the features.

After preparing the data for the next step i.e. model engineering, we conducted an extensive series of experiments to find the best-performing machine learning models and further optimized their performance by tuning their hyperparameters. Finally, we selected a hybrid, custom-built voting classifier, which is a combination of our 3 best performing models: XGBoost, Random Forest, and Gaussian Naive Bayes.

C. System design

To put all the pieces together, we designed a system whose working is illustrated in Figure 1, and we implemented these systems in two ways: entirely on the local system and entirely on the cloud. For reference, the code is available on a GitHub repository.

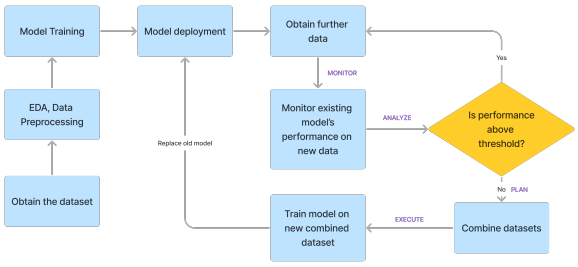

Fig. 1: Workflow of our proposed system

1) Offline Standalone Implementation: The offline implementation consists of three main components -

1) Data Streamer: This component periodically streams a chunk of verified transactions to the data repository. This component requires a source dataset and the address of the data repository. The frequency and the size of each instance of the data stream can also be specified and changed as required.
2) Model Training & Deployment Pipeline: This pipeline takes the transaction data as input. First, it performs the data engineering steps explained in the previous section. Then, it trains a new custom voting classifier model on the latest data and updates the deployment with the new model.
3) Driver Program: This is the main component that orchestrates the complete self-adaptive processes: MAPE-K loop and the MLOps processes. When the system is initiated for the first time, it checks if there exists a trained machine learning. If not, the system triggers the model training pipeline to train the machine learning model on the given dataset. Once the model is trained, the model is deployed and the model is ready to monitor the incoming data stream. The system might not process the received data chunk instantly because it would depend on how often have we configured the system to evaluate its performance. So, when the time is up, and the system observes that there is new data available in the data repository, the system would evaluate its performance on the new data. If the performance is above a set threshold, then the system will continue to work as usual because this indicates that the data has not experienced significant drift, the model performing well as per the requirements and we do not need to waste resources in re-training the model. However, if the performance on the new data is below the threshold, the system would include the new data in the current dataset and it would trigger the model training pipeline to re-train the model on the updated dataset and then deploy the re-trained model. Now, the system will again continue to monitor the incoming data stream with the updated model.

2) Online Experiment: The online experiment was conducted using IBM Cloud Pak for Data, IBM Watson Machine Learning, and IBM Cloud Object Storage. It was conducted on the balanced dataset with 568630 rows in total. We split the dataset into two equal parts keeping a proportional representation of the target class variables. We used the first half of the dataset to train the initial model for deployment, and then sharded the latter half of the dataset for simulating data drift and streaming on which our existing model would be tested every time we have gathered sufficient data. We setup two pipelines in IBM Cloud Pak for Data under our project.

Pipeline 1 was used to initially train the model on the first half of the master dataset and then deploy the same onto the deployment space so that it can be accessed via API calls for getting predictions on subsequent datasets. Pipeline 1 consists of 4 nodes as shown in Figure 2.

The "Create data file" node is responsible for copying the dataset from the project assets onto the deployment space.

The "Create AutoAI experiment" node creates an instance of the Watson ML service and configures the various aspects of model architecture and training, like the ML algorithms to use (we selected XG Boost and Gradient Boosting algorithms), target class variable, positive class identifier, parameter to optimize, train-test split ratio and deployment space parameters.

The "Run AutoAI experiment" node is responsible for training the above model on the above dataset. It consists of data pre-processing steps that involve data cleaning, data

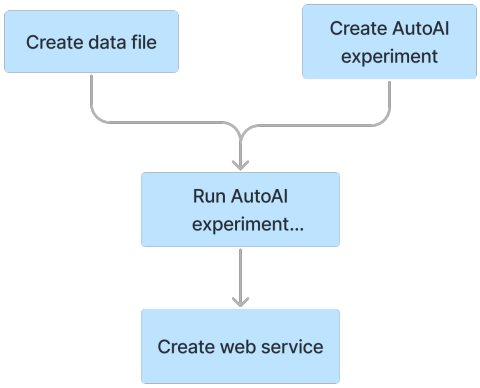

Fig. 2: Pipeline architecture

preliminary analysis, model training, post-training steps including hyperparameter optimization, selecting the best model and deploying it onto the deployment space.

Lastly, the "Create web service" node deploys the model as a web-service so that it can interact with the authorized services/users by means of API calls.

Pipeline 2 was cloned from Pipeline 1, but it would fetch the updated, merged datasets from IBM Cloud Object Storage and train the model on this new corpus. Rest of the steps stay the same.

The MAPE-K loop is realized via an iPynb notebook which acts as the driver code. It begins once the initial model is deployed through Pipeline 1. The monitoring part checks for new available shards of data and tests them on the existing model. The analyzing phase observes the model's average accuracy (we used Recall as a measure to determine the accuracy of the model, and thus our Utility Function for the Self-Adaptation is to maximize this Recall value) over the entire shard, and compares it to the pre-determined threshold. Eventually, there comes an instance where the model's average performance drops below the threshold, which triggers the planning phase. The planning phase determines the sequence of shards on which the model was tested up until the one that caused the accuracy to fall below the threshold, and merges them into an updated dataset file which is then pushed onto the IBM Cloud Object Storage. Lastly, the executing phase kicks in, which triggers the Pipeline 2 for retraining the model onto the new dataset. Once the model gets updated, the loop cycles back to monitoring phase which tests subsequent shards onto the new model.

This way, we don't frequently retrain the model every time it encounters a missed fraudulent transaction – rather we train it at regular intervals of time when sufficient data is made available.

III. Results

A. Online Experiment

We observed that the Pipeline 1 took around 47 minutes to complete and the initial model got successfully deployed. The initial model had test accuracy of more than 99.99%. We sharded the second half of the master dataset into 5 equal parts, each containing 56863 rows and then iteratively tested them on the initial model. In shard number 4, the model's average accuracy drops to around 56%, whereas the threshold set was 99 percent, as can be observed in Figure 3.

At this stage, Pipeline 2 was triggered after merging the initial dataset with the 4 shards and it took around 73 minutes to get the updated model deployed. Lastly, we tested the remaining shard onto the updated model and got average test accuracy of 99.99%, completing the MAPE-K loop, as can be inferred from Figure 4.

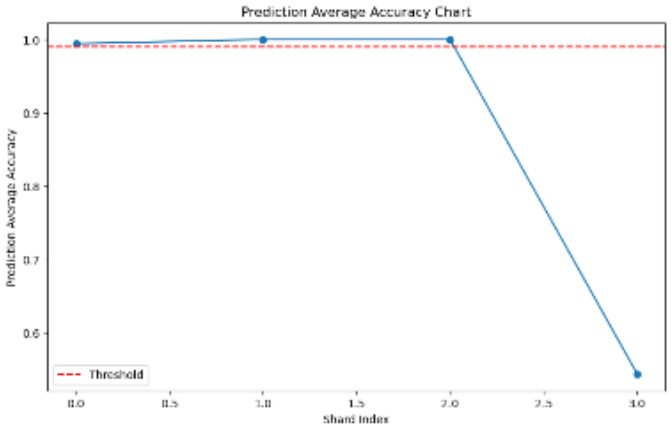

Fig. 3: Plotting shard average predictions on initial model

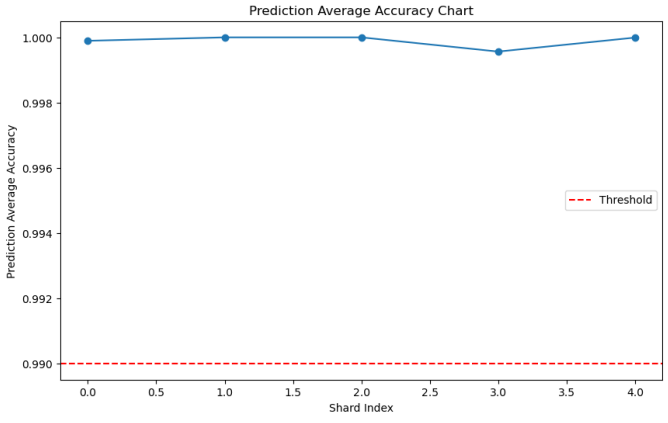

Fig. 4: Plotting shard average predictions on initial model

B. Offline Experiment

We conducted an extensive series of experiments to find the best-performing machine learning models and further optimized their performance by tuning their hyperparameters. One important thing to note is that we have used recall

as our performance metric throughout because the task of fraud detection tackles highly imbalanced datasets where fraud transaction makes only 1-2% of total transactions. So, to accurately capture the performance of the model in predicting the fraud transaction, recall proves to be the most appropriate metric. The results for all the experiments are summarized in the Table I

Classifier	Recall
AdaBoost	74.48%
Bagging Classifier	80.61%
BernoulliNB	63.26%
Calibrated Classifier CV	63.26%
DecisionTreeClassifier	75.51%
ExtraTreeClassifier	76.53%
ExtraTreesClassifier	82.64%
GaussianNB	84.69%
KNeighborsClassifier	80.61%
XGBoost	82%
LinearSVC	78%
SVC	67%
Random Forest	81%
Logistic Regression	65%
Gradient Boosting (max_depth=15)	78%
Custom Voting Classifier	83%

TABLE I: Performance of different classifiers on the Credit Fraud Dataset

By analyzing the performance of all the classifiers, we picked the top 3 performing classifiers with different architectures to build the custom, hybrid voting classifier that would be robust and work optimally in different data distributions. The voting classifier consisted of three models: XGBoost, Gaussian Naive Bayes, and Random Forest. We didn't select any of the hyperparameter optimized because those models were only marginally better.

Further, we also evaluated our system by observing its adaptive behaviour when the system was allowed to run for longer period of time and the data was streamed from another dataset. The performance of the system can be seen in Figure 5. Here, in this graph, we have time on the x-axis and recall score of the model on the y-axis. The first observation on the graph is the model performance on the test set during training to give a reference and draw comparison when model experiences data drift. After the first observation, all the further observations correspond to the performance of the model on the live data stream. As we can see, after the first observation, the model performance drops suddenly because of the data drift faced by the model from another dataset. In this experiment, we had set the threshold to be 75% so since this was below 75%, this triggered the system to self-adapt and the system re-trained the model on the new dataset. As a result, in the next observation, the model performance improved. However, it was still below 75% so it again triggered the system to self-adapt. Finally, the system adapted successfully, the model learned the new data distribution and started to perform well with the new dataset as well.

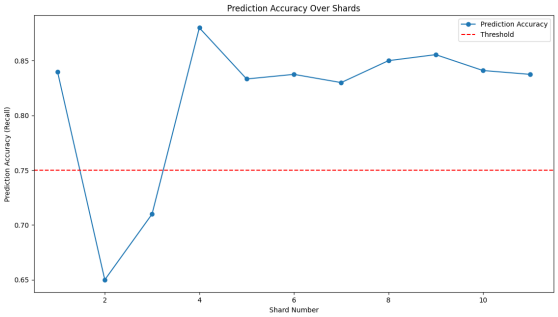

Fig. 5: Performance of our system over time when subject to data drift

IV. Conclusion

From the above two experiments, we can infer that the online experiment achieved better accuracy as it was trained on the balanced dataset using pipelines that include post-training steps such as hyperparameter optimization. This allowed us to keep a high threshold of 99%, which means that the model was able to identify true positives accurately and false negatives were very less. We need the recall value to be as high as possible because we want to make sure that the probability of the model wrongly identifying an actual fraudulent transaction as legitimate should be as close as possible to zero.

The offline experiment allowed us more flexibility to try out different algorithms as well as the possibility to work with highly imbalanced dataset, on which we were able to see as much as 84.96% recall on GaussianNB algorithm, which is slightly more than the Custom Voting Classifier with which it was finally deployed. Further, it was possible to work with highly imbalanced dataset in the offline experiment as well as simulating streaming data.

In both cases, we were able to realize the MAPE-K loop on the MLOps process and achieve the goal of self-optimization of the ML models.

V. Future Work

1) Online Experiment: The online experiment can be extended into a fully automated system that periodically checks for new shards of data, pushes them onto the cloud storage bucket, tests the existing model on these new shards for accuracy, and updates the model if the accuracy thresholds are not met. It can be automated end-to-end given the corresponding accesses and cloud service plans. Further, it can be scaled to have multiple ML models or multiple versions of the same model, and get the best prediction from them, as certain algorithms better estimate specific kinds of patterns. This will make the system more robust and reliable.
2) Offline Experiment: The offline experiment can be extended into offline pipeline which will streamline the MLOps process and include post-training stages pertaining to automated model selection, hyperparameter

optimization, and feature selection, leading to improved accuracy. Further, one avenue where the offline model can outperform the online model is if it is set up within the scope of operations of the financial institutions and is inaccessible publicly and has exclusive access to better quality and more realistic datasets. It may be considered to have such measures installed at the central bank level, which has governing powers over the activities of other financial institutions in the country.

REFERENCES

[1] Katherine J Barker, Jackie D'amato, and Paul Sheridon. "Credit card fraud: awareness and prevention". In: *Journal of financial crime* 15.4 (2008), pp. 398–410.

[2] Tej Paul Bhatla, Vikram Prabhu, and Amit Dua. "Understanding credit card frauds". In: *Cards business review* 1.6 (2003), pp. 1–15.

[3] Andrea Dal Pozzolo et al. "Credit card fraud detection and concept-drift adaptation with delayed supervised information". In: *2015 international joint conference on Neural networks (IJCNN)*. IEEE. 2015, pp. 1–8.

[4] Nidula Elgiriyewithan. *Credit Card Fraud Detection Dataset 2023*. [Online; accessed 3–December–2023]. URL: https://www.kaggle.com/datasets/nelgiriyewithana/credit-card-fraud-detection-dataset-2023/data.

[5] Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. "Machine learning operations (mlops): Overview, definition, and architecture". In: *IEEE Access* (2023).

[6] S Benson Edwin Raj and A Annie Portia. "Analysis on credit card fraud detection methods". In: *2011 International Conference on Computer, Communication and Electrical Technology (ICCCET)*. IEEE. 2011, pp. 152–156.

[7] Cédric Renggli et al. "A Data Quality-Driven View of MLOps". In: *CoRR* abs/2102.07750 (2021). arXiv: 2102.07750. URL: https://arxiv.org/abs/2102.07750.

[8] Machine Learning Group - ULB. *Credit Card Fraud Detection*. [Online; accessed 3–December–2023]. URL: https : / / www . kaggle . com / datasets / mlg - ulb / creditcardfraud/data.