# Adaptation of eXtreme Gradient Boost Algorithm to Credit Card Fraud Data Stream

Thanzin Naing

SID: 861278090

Specialization: Data Science

## Abstract

Credit card fraud has become a growing problem with the rise of online banking. Customers and financial institutions alike face rising financial costs. With this problem emerges new ways of tackling credit card fraud. This study includes one of the ways to tackle one aspect of the credit card fraud detection problem. Credit card fraud detection is a data stream problem, an incremental learning problem, that is fundamentally different from a static learning problem. The problem includes dealing with class imbalance as well as concept drift. Traditional ways of dealing with credit card fraud include rule check-based systems. Such rules are often determined based on statistical analysis of historical data. However, the emergence of machine learning-based methods promises to improve the accuracy of credit card fraud detection. The goal is to not only reduce false alarms but also to detect more true frauds in any data stream. My method is based on eXtreme Gradient Boost which is adapted to work with data streams. It is an ensemble decision tree-based machine-learning algorithm. Instead of using the entire dataset as input, it is capable of using smaller chunks or batches of the incoming data stream. The model is tested using Average Precision and F1 score on 24 chunks of data with each chunk of data representing 2 hours of the stream. The testing reveals an Average Precision score of 0.73 and an F1 score of 0.77. This is a good score, but not an amazing score. My model can achieve good precision across different recall levels and maintains a good balance between precision and recall.


## Introduction

### Project Overview

In this project, I will be performing credit card fraud detection of credit card transactions. I will do this using the eXtreme Gradient Boost(XGBoost) Machine Learning Algorithm. More

specifically, I will be performing two-class classification with the XGBoost. The two-class classification is very good for fraud detection for two reasons. Fraud detection lends itself easily to a classification problem and XGBoost is very good at classification(Chen & Guestrin, 2016). A transaction is either fraudulent or it is not. XGBoost is very good at tackling problems using highly imbalanced datasets. This is very important because the vast majority of credit card transactions are not fraudulent. XGBoost is adept at recognizing non-linear relationships between features. XGBoost is a tree-based ensemble method, which means that it combines the output from multiple trees. In comparison to other ensemble methods like Random Forest, XGBoost possesses a greater ability to capture complex patterns and relationships in the data. This is referred to as model capacity. In Random Forest, multiple decision trees are run in parallel, and the average of the outcomes is produced. However, in Gradient Boost the trees are built sequentially so that each new tree corrects the errors of the previous tree. The model starts with a weak learner - like a decision tree with only one split. Trees are built in a way where each new tree that has been added decreases the loss function – like Binary Cross Entropy for example. The goal is to find the negative gradient in the loss function that will most quickly converse to a minimized solution. In other words, we want to find the Learning Rate that will decrease the Loss Function the fastest. This requires finding the optimal learning rate. A learning rate that is too high will fluctuate around the optimum solution and are less likely to find it while a learning rate that is too low will take too long to converse to the optimum and thus will require more computations. Thus, a chain of these weak learners is combined to produce a solution that is closer to the ground truth than an average of multiple independent trees used in Random Forest. In the real world, non-linear relationships are the norm, and credit card transaction data have particularly complex non-linear relationships. In traditional credit card fraud detection there exist

two parts, learning the specific patterns of the cardholder based on historical data, and learning the general pattern of fraud. These two parts are then combined to produce a more effective detection algorithm. However, due to the unavailability of private financial information, the scope of the project will be limited to general pattern detection based on the transaction information only. This pattern detection will be thus limited to transactions and will not include cardholder profile information to determine the prediction. In addition, I am using a small dataset that only contains two days of data. A larger dataset will be more suitable for testing model performance on recurring and seasonal concept drift. Furthermore, I am utilizing data chunking to create a simulated data stream from a static dataset because I do not have access to a real data stream. Finally, my project assumes that there is readily available labeled data that my model can predict since it is a supervised learning algorithm. However, the availability of labeled data may vary in the real world. When there is a lack of labeled data, then an unsupervised learning algorithm may be more suitable.

**Significance of Project**

There has been a steady rise in financial fraud. A large portion of that financial fraud came in the form of credit card fraud. Most notably, fraudulent financial transactions using stolen credit cards. This is very costly to financial institutions like banks which will have to shoulder most of the burden of credit card fraud. The direct cost to banks is in the form of direct financial loss because banks have to reimburse the victims of unauthorized financial transactions and operational costs associated with resolving cases of credit card fraud. The second cost is indirect in terms of damage to a bank's reputation and loss of customer loyalty. A fraud detection algorithm can do three things. It is a scalable way to detect cases of possible fraud to be investigated(Dal Pozzolo et al., 2014). This is very important for a bank since a bank processes

millions of financial transactions every day thus a bank cannot monitor every transaction. This means that a reliable fraud detection algorithm is necessary. Second, it can reduce the operational costs associated with resolving cases of credit card fraud by preventing the bank from having to investigate false positive cases where it may seem like fraud occurred but did not. Third, a fraud detection algorithm can adapt to new and emerging fraud schemes by learning from historical data and recognizing emerging patterns.

## Literature Review

### Previous Research

In the paper called “Credit card fraud detection: A fusion approach using Dempster–Shafer theory and Bayesian learning” the authors (Panigrahi et al., 2009) proposes a novel approach that combines multiple evidence from past and present behavior. This is to prevent fraudsters from making multiple types of attacks. Their fraud detection system consists of four components: rule-based filter, Dempster-Shafer adder, transaction history database, and Bayesian learner. A rule-based filter measures the extent to which the transaction's behavior deviates from the normal profile behavior of the cardholder. Initial beliefs are assigned to incoming transactions then those beliefs are combined to obtain an overall belief by applying Dempster Shafer's theory. The belief is strengthened or weakened based on its similarity with fraudulent and genuine transaction history using Bayesian Learning. This method was tested on a synthetically generated dataset. The simulation yielded a true positive rate of 98 percent and a false positive rate of less than 10 percent. The simulation shows combining multiple pieces of evidence with Dempster Shafer improves true positive rates. It also shows that adding the Bayesian Learner increases the True Positive rate even further.

In the second paper, the authors (Kundu et al., 2009) propose a method called BLAST-SSAHA hybridization. Sequence alignment is used to quantify and evaluate similarity between two or more sequences. The authors propose a two-stage sequence alignment method containing a profile analyzer(PA) and deviation analyzer(DA). The profile analyzer determines the similarity of the incoming sequence of transactions with sequences of past genuine cardholder spending behavior. Then the incoming sequence is given a Profile Score. Inversely, the sequences are also passed to the deviation analyzer to see how they match up with sequences based on the fraud history database. After that, a Deviation Score is calculated. The difference between these two scores is calculated and checked to see if it meets the threshold by the Final Decision Maker(FDM) which determines if it is normal or an anomaly. It is highly accurate and is fast enough to enable real-time detection of credit card fraud online. It can also counter telecommunication and banking fraud.

The main purpose of this paper (Srivastava et al., 2008) is to propose using a Hidden Markov Model(HMM) to detect credit card fraud. A Hidden Markov Model is a probabilistic model used to describe sequences of data where the underlying process is not directly observable. HMM allows us to model complex systems. It is a double-embedded stochastic process. In statistics, a Stochastic process is a model used to describe random events or systems that evolve. First, a spending profile of the user is established using the k means clustering algorithm. An initial set of probabilities are chosen based on the spending profile. Sequences from the training data are constructed and the model is trained. An incoming transaction is sent to the Fraud Detection System and its sequence is determined as normal or an anomaly. If it is normal, then it is added to the existing sequence. If not, it is classified as an anomaly and an alarm is triggered. This model does not analyze user profiles on a case-by-case basis but instead

maintains a log based on past user data. This model produces a high false positive rate. This is probably due to the model generalizing too much since it is based on a log, not individual user profiles.

In the paper "Detecting Credit Card Fraud by Decision Trees and Support Vector Machines" the authors (Sahin & Duman, 2011) seek to compare the performance of Decision Trees and Support Vector Machines for credit card fraud detection classification. Seven different models are evaluated. There are three different models based on decision trees called CART, C5.0, and CHAID. There are four different models based on SVM called RBF Kernel, Polynomial Kernal, Sigmoid Kernal, and Linear Kernal. The credit card data used for this experiment is real credit card data taken from a national bank's credit data warehouses. It is historical data representing card usage profiles of customers. Each card profile consists of variables that describe the spending characteristics of each card. These are used as inputs to machine learning algorithms. This dataset is highly imbalanced with 20,000 normal transactions to one fraudulent transaction. The data is preprocessed with stratified sampling to reduce the number of nonfraud records while preserving the fraud records. The authors stratified sampled the raw data to produce three different sample sets. The first set has a one-to-one ratio of frauds to non-frauds. The second set has a fraud ratio of one to four. The third set has a fraud ratio of one to nine. Thus, a total of seven models were tested on three different preprocessed sample sets. The conclusion that the authors came to is that decision tree-based models outperform the SVM-based models. SVM often overfit the data and the number of frauds caught by decision tree models often exceeds the amount caught by SVM models. However, as training sets become larger, the accuracy of SVM models becomes comparable to decision tree models. Out of the

decision tree-based models, CART was able to catch the greatest number of frauds. However, the overall accuracy was highest for C5.0.

This paper called “Random Forest for Credit Card Fraud Detection” by the authors (Xuan et al., 2018) details how to make use of the Random Forest machine learning algorithm to tackle the credit card fraud detection problem. They point out that compared to simple decision trees, ensemble decision trees like Random Forest are less likely to overfit and have superior generalizing ability. In addition, ensemble methods are robust to outliers which makes them suitable for problems with highly imbalanced classes, such as credit card fraud data. They trained their model making use of real-world business data. It is a dataset of B2C transactions from November 2016 to January 2017 with 30 million transactions and 62 attributes for each transaction. The dataset has 82,000 transactions labeled as fraud resulting in a ratio of 27 frauds for every 10,000 transactions. The authors say that there are two primary ways of fraud detection: misuse detection and anomaly detection. Misuse detection is when you have incoming transactions that you are trying to determine whether they are fraud or not fraud. Anomaly detection is about building a profile of ‘normal’ behavior for the particular cardholder and seeing whether a new transaction deviates from that behavior. In the real world, both are often used together to make a more robust system against fraud threats. The main focus of this paper is to compare two different Random Forest algorithms to see which one had the better overall performance. They are the Random Tree Random Forest and CART-based random forest. The main difference is how they select features. The Random Tree randomly selects a subset of features when making a split whereas CART tries to find the best split among a random subset of features using metrics such as entropy and gini impurity. What the authors found was that both algorithms performed the same when it comes to precision, but CART-based Random Forest

outperformed Random Tree Random Forest when it comes to recall which led CART-based Random Forest to have a much higher f1 score.

I analyzed a paper called “Credit Card Fraud Detection with Neural Network” in which the authors(Ghosh & Reilly, 1994) propose a new neural network-based fraud detection system that they claim outperforms more primitive rule-based systems. The most primitive and basic systems for credit card detection are rule-based checks. These fraud rules are developed by banks through historical statistical analysis of past fraudulent behaviors. According to the authors, neural networks were able to detect an order of magnitude more fraudulent accounts while simultaneously having the false positive rate reduced by a factor of 20 when compared to rule-based systems. A feasibility study done for Mellon Bank to determine the effectiveness of a neural network-based solution for the bank’s credit card portfolio. The neural network was to be used as a post-processing step to the bank’s current authorization system. The Neural Network used is called the P – RCE. This type of Neural Network was created for pattern recognition problems. It’s a three-layer feed-forward neural network. The trained Neural Network was tested on an unsampled dataset provided by Mellon Bank that included around 2,000,000 transactions which was two months’ worth of transactions. The performance indicator used was a rank curve of the percentage of fraudulent transactions detected by the network plotted against the number of cardholder accounts that are flagged for review each day. Previous attempts by Mellon Bank yielded them around 750 accounts of which 1 of them are detected as fraud every week. The method shows a significant increase in detection performance compared to rule-based methods. However, the more important indicator is economic benefit. To measure cost savings, the authors summed up all the dollars associated with all fraud transactions on each fraud account and then summed up the total dollar cost savings for all of the accounts. The savings are then expressed as

a percentage of total fraud losses. Using this metric to evaluate cost savings, the Neural Network was able to save between 20 to 40 percent of the total fraud losses.

In a paper called “Learned Lessons in credit card fraud detection from a practitioner’s perspective” the authors’ (Dal Pozzolo et al., 2014) purpose in this paper is to summarize lessons learned and common mistakes to avoid when tackling credit card problems. They state the common challenges that a Data Scientist faces when dealing with this type of problem, the common approaches to the problem that do not work, and strong alternatives. Finally, near the end, the authors empirically show the relative performance of various combinations of methods and provide insight into which combination produces the highest performance. It starts by stating that due to the sheer volume of fraudulent credit card transactions, the problem requires the use of an automated system. More specifically, to control costs, a fraud detection team will need to investigate credit card fraud transactions that have the highest probability of being frauds. Due to this, the authors argue that instead of having transactions correctly classified, it is better to rank transactions by their fraud probability. This will naturally lead to the use of a ranking system. This ranking system should have the ability to rank credit card transactions from the most likely to the least likely probability of fraud. Another is the problem of concept drift. Unlike in an academic setting when students work with static datasets, a financial institution like a bank will receive a continuous stream of data. The distribution of this data will change over time. This is due to the evolving and constantly changing nature of fraud. As such, patterns inherent in the data will change and the model will need to be updated automatically to account for this change. The authors discuss how frequently the model should be updated and the most effective methods to go about this. The authors also acknowledge that the lack of access to personal cardholder information by the public is a problem because this means that one is not able to create systems

that consider cardholder behavior in the form of aggregate variables that aggregate aspects of personal spending patterns. The authors believe that information on one transaction is simply not sufficient to detect a fraud occurrence. Private financial information is not allowed to be released to the public due to legality. However, this also means that it severely limits the number of Data Scientists that can work on this problem. Another point of contention is the way classification-based Machine Learning models are traditionally evaluated. The authors argue that due to the unique nature of the problem of fraud detection, traditional performance measures like accuracy, True Positive Rate, True Negative Rate, Balanced Error Rate, and even traditional AUC are misleading. The authors criticize that the AUC(Area under ROC) curve essentially creates the average of the misclassification cost of the two classes. The problem is that this measurement treats the cost of the two misclassifications the same. The authors argue that the ability of the model to correctly classify the minority class is the most important. In total, the authors propose three different methods to measure performance: Ranked AUC, Average Precision(AP), and Precision Rank. A rank-based formulation of AUC assesses the probability that a randomly selected example from the minority class (fraud) ranks higher in the list than a randomly selected example from the majority class (non-fraud). Another metric they propose is average precision(AP), which is used to evaluate a model's ability to rank items in a way that accounts for both precision and recall. If precision is low that means, there are a lot of false positives(misclassification). If the recall is low that means, there are a lot of false negatives(frauds undetected by the algorithm). There is generally a trade-off between precision and recall. This trade-off is illustrated by the precision-recall curve. The AP is the area under the precision-recall curve, with an AP of 1.00 indicating perfect precision and recall. The final measurement is Precision Rank. It measures the precision of retrieved items at a specific rank.

It's good at measuring the performance of a system that deals with a ranked list of items. Another problem that the authors address is the extremely imbalanced nature of the credit card fraud detection problem with the percentage of fraud representing 0.4 percent in this particular dataset. The authors discussed three ways of addressing imbalance: Undersampling, SMOTE, and EasyEnsemble. Undersampling is removing members of the majority class so that there is less of a disparity between the two classes. SMOTE is shorthand for Synthetic Minority Oversampling TEchnique. As the name suggests, it involves generating synthetic replicas of the minority class near the minority class. More specifically, it generates artificial data points in the middle of the distances between the real minority classes to preserve the general pattern of information shown by the minority class. EasyEnsemble on the other hand is a more advanced undersampling technique that involves creating different balanced datasets using undersampling, learning a model for each dataset, and combining all predictions. This technique allows you to learn different aspects of the original majority class. The authors summarized their findings by stating that out of the three Machine Learning methods presented - Neural Networks, Support Vector Machines, and Random Forests – the Random Forest algorithm showed the best results overall when tested with the performance indicators AP, Ranked AUC, and PrecisionRank. Fraud detection tasks should not be described as a classification problem, and it is better to see whether the algorithm can rank the fraud probability of each item. The algorithm needs to be modified to output the ranked probability instead of classification labels. Thus, using AP, Ranked AUC, and PrecisionRank are the best performance indicators for such a task. They also found that breaking the dataset down into chunks, learning from those chunks, and aggregating the outputs yielded better results than using the whole dataset. They have found that constant updates to the algorithm – only possible with a continuous data stream – have yielded better performance

compared to those that were trained on a static dataset. This is because it was able to learn about constantly changing probability distributions and update its model accordingly. To that end, they have found that a daily update frequency yielded better performance than weekly updates. When it comes to techniques used to combat class imbalance, they found that EasyEnsemble and SMOTE consistently outperformed undersampling.

The authors(Gomes et al., 2017) state that ensemble methods are the most widely used techniques for data stream classification due to their performance(accuracy), ability to deal with concept drift, and ease of use in real-world applications. In this instance, a strong learner is difficult to obtain while chains of weak learners are much easier to develop and operate as a strong learner. A weak learner is simply a learner that achieves higher performance than a random guess. An effective ensemble necessitates diverse classifiers that misclassify different instances and thus complement each other. For example, an ensemble of homogeneous classifiers will not achieve a better performance than any single member. However, a diverse set of classifiers probably will as it will more likely be able to explore more of the concept space. Thus, a diverse combination of classifiers can achieve a better performance than any individual classifier. The authors also state that ensemble learners can handle both general machine-learning problems as well as data stream-specific challenges like concept drift, recurrent concepts, and feature drift. The downside is that ensemble algorithms will be computationally(memory) intensive and time-intensive. Unlike batch learning on static datasets, data streams will be non-stationary and ever-evolving. However, many concepts are shared in batch learning and data streams. How does one distinguish between noise and drift? The answer lies in the persistence of concepts. The rate of drift can be abrupt, incremental, gradual, or reoccurring. A sudden change in the prediction error signifies abrupt drift. In gradual drift instances from previous concepts

will become less frequent and instances from new concepts will become more frequent. Incremental drift is about concepts evolving. Recurring drift happens when concepts reemerge periodically. Another concept that is unique to data streams is temporal dependencies between class labels. The features of a dataset are used to provide predictive information. Temporal dependencies are about how the way those features relate to each other change over time. Another problem with data steams is called the stability-plasticity dilemma. A model must have the ability to learn new concepts while retaining previous knowledge. At the same time - due to different types of concept drift - a model has to have a way to get rid of instances that are associated with outdated concepts. Additionally, concepts may reappear, and it is a waste of computational resources to relearn old concepts. Conversely, there may also be instances of concept stability when the algorithm should retain old concepts instead of learning new ones. This makes it a tough challenge for models to deal with. Therefore, it may be helpful to use a drift detector algorithm that acts as an additional mechanism so that a classifier may forget its model and learn a new one when drift does occur. Another problem is related to cardinality. Some may theorize that one can simply add more classifiers to the ensemble to achieve higher accuracy. However, adding too many classifiers creates redundancy which does not affect the final decision quality, but it does affect memory consumption. There are two ways to tackle this issue. The first way is to implement an ensemble that can vary its size dynamically – either by removing redundant classifiers or adding classifiers to cover different parts of the classification space. The second is to simply place a predefined limit on the maximum number of classifiers. The latter method is often chosen not only because it is effective but also because of the complexity of determining a heuristic method for adding and removing classifiers. A source of debate is the concept of dependency. Should the training of members depend on the output of

another, or should classifiers be trained independently? The authors argue that classifiers should be trained independently. The reason is that the former method may lead to overfitting. The authors talk about the concept of Landmark windows. A landmark is defined as a specific number of instances since the last landmark or a specific number of instances in a given time frequency. The authors state that a predefined fixed landmark approach will allow you to apply traditional batch learning algorithms for stream learning. The authors went on to say that batch learners need to be trained on a lot of data to be an accurate model. This means that the window will need to be fairly large. In other words, the model cannot receive inputs that are as small as an instance. Another problem with this is that when concept drift happens, this cannot be considered until the window ends and the new model is generated. This means that the model will be slow to adapt to concept drift. Additionally, when abrupt drift occurs, it is better to have a small number of n instances in each window. Inversely, when drift is gradual then the opposite is true, it is better to have a larger number of n instances trained. The authors warned that if the data has many different types of drift mixed with periods of stability, then any predefined window size will be suboptimal. In other words, they do not recommend using window methods to adapt batch learning to incremental learning. However, it can be done based on how simple and one-dimensional the drift is. Thus, they say it is advisable to specifically use incremental base learners. It is important to maintain an up-to-date model that can deal with data arriving at high speed while also using limited resources. The authors state that the algorithm needs to find the balance between accuracy and computational resource cost. Therefore, the best-performing algorithm in terms of accuracy may not always be the best. Regarding computer resources, the authors mention the limitation of using a single machine even if it is multiple threads. This necessitates the importance of investing in scalable and distributed systems.

In the next paper – “On Incremental Learning for Gradient Boosting Decision Trees” - the authors (Zhang et al., 2019) state that the main problem is that most boosting algorithms were designed for batch learning. However, they propose a modification of the gradient boosting decision tree(GBDT) that they call iGBDT. Their model is capable of incrementally learning a model without restarting with every batch. There is a difference between online learning and data stream classification. Data stream techniques often need to ‘forget’ the old data since there is a fixed size of the latest data to obtain analytical results in real-time. For online learning, new instances and old instances are equally important therefore there is a need to keep the information from old instances. Gradient Boosting Decision Tree (GBDT) is a boosting algorithm. Just like XGBoost, it uses gradient descent to gradually decrease the loss function with each decision tree that is built down the chain. The loss function represents the difference between the predicted value and the true value. Thus, the loss function shows how bad the model is and this will gradually decrease with gradient descent representing an incremental improvement in the model. The iGBDT algorithm they propose will use least squares as the loss function and Classification and Regression Tree(CART) as the weak learners. The authors said that a common way GBDT is applied to real-world data streams is that newly arrived data is combined with a previous data set. Then the model learns from this combined dataset from scratch. This method is known as straightforward GBDT. The downside is that the model can result in very long computation times. Since you are repeating the process of combining datasets and retraining the model from the ground up with every new batch of data, it’s a very inefficient brute-force method. The iGBDT method on the other hand processes all of the new batches of data at the same time and does not process new inputs 1 by 1. Thus, iGBDT lazily updates the old model instead of building it from scratch. The authors made separate comparisons of iGBDT

with GBDT on running time based on three factors: batch size, learning rate, and tree depth. The authors are using of various sizes datasets – Abalone, Winequality white, Page Blocks, and Covertype - from the UCI repository. When the batch size is small the GBDT method rebuilds the model more frequently slowing it down in comparison to iGBDT. When testing varying batch sizes, we see iGBDT be 2-16 faster. Naturally, increasing the learning rate decreased the running time of both GBDT and iGBDT. The running time of GBDT decreased faster than iGBDT. Overall, iGBDT is 1.2 – 12 times faster at various learning rates. The performance difference between GBDT and iGBDT is very wide. With varying tree depths, iGBDT is 1.2 – 12 times faster. In all experiments, the accuracy results are the same since the same tree is being built. Therefore, on an incremental learning problem, iGBDT can achieve the same classification performance while dramatically decreasing computation cost.

In this paper - Adaptive XGBoost for Evolving Data Streams - , the authors(Montiel et al., 2020) show their adaptation of eXtreme Gradient Boosting algorithm for evolving data streams. They call their method Adaptive eXtreme Gradient Boost(AXGB). Their method involves creating new members of the ensemble from mini-batches of data from the incoming data stream. Their ensemble is of a fixed size but is still updated on the new data to make sure that it is keeping up with concept drift. Their method also incorporates a concept drift detection mechanism that signals the algorithm to update itself automatically. The authors state that, in stream learning there are two different types of algorithms: instance-incremental methods, and batch-incremental. In an instance incremental method, a single instance is used at a time as input while in a batch incremental method uses sequential batches of data as input. In most cases, instance incremental methods not only have higher performance but are also more memory efficient. For stream learning, the model should have the ability to provide a prediction at any

time. Since the authors propose a model with a fixed ensemble size, there has to be a way of handling updates once the ensemble is full. They propose two methods of updating strategies: push and replacement. The Push updating strategy consists of creating new models as they are appended to the ensemble and when the ensemble is full then older models are removed to make way for new models. The Replacement updating strategy consists of replacing older members of the ensemble with newer ones. With the replacement updating strategy, newer models carry more weight than older models, which is the opposite of the case for the push updating strategy. Although there is a fixed ensemble size, the authors do not use a fixed window size because it would result in a sub-optimal performance at the beginning of the stream. The model instead uses a dynamic window size that doubles in size with each iteration starting from a minimum window size. The dynamic window size W increases until reaching the maximum window size. Therefore, base functions trained on small windows are replaced with newer ones that are trained on more data. The authors state that although the model can deal with concept drift, it may be too slow to deal with fast drifts. Therefore, they propose using a drift detector called ADWIN to track changes in the performance(classification accuracy) of AXGB. The way ADWIN works is that if a change is detected, the worst-performing member of the ensemble is replaced with a new one. AXGB receives a change detection signal from ADWIN which it uses to update the ensemble. The mechanism works by resizing the window to the minimum window, training, and adding members to the ensemble either through the push or replacement update method.

The authors used both real-world and synthetic datasets to evaluate their model for a total of 6 datasets in total. The authors are comparing 4 versions of their model: AXGB with ADWIN using push update, AXGB without ADWIN using push update, AXGB with ADWIN using replacement, and AXGB without ADWIN using replacement. They are comparing the four

versions of their model with two batch incremental methods called BXGB and AWE-J48. The authors will also compare the four versions of their models with instance incremental methods: Adaptive Random Forest(ARF), Hoeffding Adaptive Tree (HAT), Leverage Bagging with Hoeffding Tree as base learner(LBHT), Oza Bagging with Hoeffding Tree as base learner(OBHT), self-adjusting memory with kNN (SMKNN), and Ensemble of Restricted Hoeffding Trees (RHT). Batch eXtreme Gradient Boost(BXGB) is a batch incremental implementation of the base XGB algorithm. Batches of samples are incrementally used to update the model. XGB is used as the base learner to learn an ensemble for each batch thus new base models are trained independently on Windows (batches of data). When the ensemble gets full then older models are replaced with new ones. Predictions are formed using a majority vote. We can also consider using the same batch incremental method except by using Accuracy Weighted Ensembles with Decision Trees(AWE-J48) as the base batch learner instead of XGB.

They found that the best overall performer is the AXGB without ADWIN using replacement. However, when comparing push strategy versions, AXGB with ADWIN comes out on top. This correlated with what the authors talked about earlier. Since the model adapts too slowly to fast concept drift, ADWIN helps speed it up. However, the same cannot be said for the version that uses the replacement strategy. When comparing AXGB to instance incremental methods, Adaptive Random Forest(ARF) came out on top. The authors state this is to be expected since research consistently shows that instance-incremental methods often outperform batch-incremental methods.

The authors also explored the effect hyperparameter tuning has on the relative performance of various methods. To see the effect of hyperparameter tuning on AXGB, the model is trained and validated on the first 30 percent of the data stream using different

hyperparameter combinations in a grid. The best one is chosen based on classification accuracy. The best-performing hyperparameter combinations during this phase are then carried over and applied to the remaining 70 percent of the stream. In this phase, the ensemble is trained from scratch. The authors note that while optimizing hyperparameters benefits all methods, BXGB received the largest boost in performance and is the best-performing method. It also had a faster training time compared to both AXGB versions that had an ADWIN drift detector. The authors found out that the best parameter options for BXGB favor a small max window size, small learning rate, and small max depth. The authors theorize that a smaller window size updates the model faster and thus allows the model to better deal with concept drift.

When evaluating AXGB, the authors found that AXGB without ADWIN benefits from a smaller ensemble size while the opposite is true for AXGB with ADWIN. This means that drift-aware methods benefit from larger ensembles and thus can build more complex models. When comparing training times between different AXGB versions, it is found that AXGB methods with not ADWIN had the lowest training times implying that the drift detector adds overhead training time. Therefore, the authors conclude that AXGB with replacement updates had the best combination of performance and memory cost. The authors also conclude that BXGB may be considered the better model if the user is prioritizing predictive performance over memory consumption.

**Findings and Unanswered Questions**

Based on all the previous research I have reviewed so far; several conclusions can be made. Traditional methods of credit card fraud detection involving rule-based detection methods are less capable than machine learning methods. The advent of machine learning methods revolutionized credit card fraud detection. We learned from Ghosh(Ghosh & Reilly, 1994), that

traditional rule-based methods fall short when compared to a neural network algorithm. However, we also learned that ensemble decision trees, like those of random forests, are even more powerful for credit card classification compared to neural networks(Dal Pozzolo et al., 2014). In addition, credit card fraud detection cannot be treated as a traditional machine learning problem since we are dealing with a data stream and not a static dataset(Dal Pozzolo et al., 2014). Data streams have their own set of challenges such as concept drift. Thus, traditional machine learning algorithms have to be specifically adapted to handle data streams to work in the real world. Data streams can arrive in batches or instances and algorithms have been adapted to work with one type of data stream or the other. Research on ensemble learning for data streams suggests that while algorithms for instance-based data streams can be more effective on average compared to an algorithm adapted for batch-based data streams, batch-based data streams can also be just as effective when a data stream does not contain multiple types of concept drift at once(Gomes et al., 2017).

While there is research that attempts to independently address the most common problems of credit card fraud detection like concept drift, extreme class imbalance, suitable performance evaluation metrics, and limitations of transaction data, there are not any that attempt to address these problems at once using an ensemble decision tree algorithm adapted for data stream batch learning. I propose utilizing data chunking to break down a real static dataset into a data stream, while simultaneously utilizing eXtreme Gradient Boost in a batch-based data steam learning context and evaluating using suitable performance metrics.

**My Preliminary Work on the Research Topic**

For my preliminary work on the topic of credit card fraud, I ran a Random Forest algorithm on credit card fraud data that I found on Kaggle. Since Random Forest is a decision

tree-based ensemble machine learning algorithm, I believe that it would serve as a good benchmark for what this type of algorithm is capable of when it comes to credit card fraud detection. In addition, it might serve as a way to experiment before applying them to my model. According to the description of the dataset, it was a PCA-transformed dataset of European bank transaction data. I ran my own Random Forest algorithm with a little tuning on the entire dataset and was able to achieve precision, recall, and F1 scores of 0.94, 0.80, and 0.84 respectively. I performed by algorithm on PCA-transformed data. PCA is a feature transformation method that creates new and more relevant features from the old ones. Therefore, it may improve my model's ability to generalize the data. I have also attempted preprocessing steps like under-sampling, oversampling, and hyperparameter tuning. However, I was unable to achieve a higher performance using those methods. Based on additional research I have done; under-sampling causes a loss of information in the dataset which possibly reduces model performance(Liu et al., 2009). Additionally, hyperparameter tuning is not usually recommended for those who lack domain knowledge since it is difficult to determine ranges of parameters to input for a hyperparameter tuning algorithm (Yang & Shami, 2020). I have some success with very mild oversampling when it comes to improving the performance of the algorithm. However, when oversampling is too severe, the performance goes down. I believe that this is due to overfitting that data. Based on my research, the one downside of oversampling techniques like SMOTE is that it has the potential to make the algorithm overfit the data (Chawla et al.,2018). While my model's performance overall was not too terrible, recall is a bit low compared to precision. If I can increase recall, it will substantially improve the F1 score.

**Remaining Questions**

I believe that the areas of knowledge that have been overlooked are real credit card data, data streaming algorithms, data imbalance, and suitable evaluation metrics. First, some research papers did not utilize real credit card data to evaluate their algorithms. Due to a lack of availability or access, many researchers opted to use simulations or synthetic data to evaluate their algorithms and methods. This is problematic because synthetic data cannot replicate all of the nuances of real data(Fang et al.,2022). Therefore, one must be careful of coming to conclusions based on the results of synthetic data. Second, some research papers trained and evaluated their algorithm on a static dataset. This will not work in actuality because, in a real-life setting, the data will come in the form of a stream. Many machine learning algorithms are adapted for learning on static datasets. Therefore, for those algorithms to work well they must be adapted to the data steaming context. Third, some papers have not considered whether an algorithm is adept at handling highly imbalanced data before applying them to credit card fraud detection problems. From my point of view, any algorithm that cannot sufficiently handle highly imbalanced data can be dismissed as a potential solution to this problem since class imbalance is so extreme for credit card data. According to my research, decision trees and their variations are well suited to handling class imbalance. Finally, many papers used accuracy as an evaluation metric to determine the performance of their algorithm. Accuracy is not a suitable metric for accessing the performance of an algorithm on credit card data(Dal Pozzolo et al., 2014). This is because credit card fraud data has an extreme degree of class imbalance. For example, an algorithm that simply lists every transaction as ‘not fraud’ will have a very high degree of accuracy since there may be one fraud for every five hundred non-frauds or an even more extreme ratio. Therefore, this raises the importance of a more suitable evaluation metric. By

combining knowledge from different areas of research, I came up with a method that attempts to address all of these issues.

# Methodology

## Approach

My general approach is that I take a static credit card fraud detection dataset and turn it into multiple chunks of data based on the time feature. The time feature is the number of times elapsed since the first transactions are recorded. This is a real credit card dataset that was originally a data stream but was archived and turned into a static dataset. Through the use of data chunking, I was able to turn it back into a data stream albeit in the form of batches not instances. I will choose an appropriate time interval based on my testing. However, research indicates that a greater frequency of iterations of the model – which in my context translates to shorter time intervals - tends to yield better performance. This is because greater frequency makes the batch data stream algorithm more adept at capturing concept drift. The logic is that in the absence of real streaming data, I am using the time feature to simulate a data stream from the static dataset. The idea is to create an algorithm that takes apart the static dataset one interval at a time and those chunks of data are used as inputs for my XGB algorithm. The XGB algorithm is nested inside of my primary algorithm which I will call my model. The model has a data chunking component, training component, and evaluation component.

XGB is initialized and the model iteratively updates the algorithm with each successive data chunk to sequentially build out its tree with every loop of the algorithm. Every time the model iterates it makes a prediction and checks those predictions with the test data to reveal average precision and ROC AUC score, which I chose as my evaluation metrics. Not every

chunk of data is the same. Some chunks will have higher or lower scores on the evaluation, and this is meant to keep track of which chunks are contributing positively or negatively to my overall model performance

**Data Collection**

Since I do not have direct access to financial institution servers and such organizations are required by law to uphold data privacy laws and keep user data a closely guarded secret. I used a publicly available online dataset called “Credit Card Fraud Detection” by the Free University of Brussels. The dataset consists of labeled transactions made by credit cards in September 2013 by European cardholders. The interval of this dataset is two days. It contains 492 frauds out of 284807 transactions. The ratio is 1 fraudulent transaction for every 579 nonfraudulent transactions showing extreme class imbalance. The positive instances(frauds) are the most important thing that the algorithm will need to predict but those instances are highly limited in information.

To protect data privacy, the original features and background information cannot be shown. To make it publishable the dataset authors performed feature extraction in the form of Principle Component Analysis(PCA) transformation on the dataset. The new features are thus the principal components that were extracted from the original features. The only original features that are not PCA transformed are ‘Time’ and ‘Amount’ since they do not provide any user-specific information. The feature ‘Time’ contains the number of seconds that have elapsed between each transaction and the initial transaction in the dataset. The feature called ‘Amount’ indicates the transaction amount. The feature ‘Class’ is the target class that is either 1 or 0; 1 means fraud and 0 means non-fraud.

**Analytic methods**

For my project, I will use the Python programming language to create my algorithm. I will utilize the following Python packages for my method: ‘pandas’, ‘xgboost’, ‘sklearn.metrics’, and ‘matplotlib.pyplot’. The purpose of ‘pandas’ is to turn my dataset into a data frame which allows me to manipulate the data more easily. The purpose of the ‘xgboost’ package is so that I can embed xgboost into my algorithm. This will modify the general xgboost algorithm to be more suited to data steam learning. The package ‘sklearn.metrics’ will be where I will obtain my various evaluation metrics. I will then use ‘matplotlib.pyplot’ to create a graphical representation of my results.

I will load my dataset using pandas and assign it to a variable. I will then use that variable to separate the target feature(‘class’) from all other features to create two data frames. I will further separate the two data frames into their training and testing versions. The next step would be to create my model. The general idea behind my model is that it will be a for-loop algorithm where in each loop, several processes occur. First, it will use the time feature to process data into chunks based on a specific pre-defined time interval. For each temporal chunk, features and labels are extracted which are then used to train the xgb model. Next, a prediction is made on the test dataset. These predictions are used to evaluate the average precision score and f1 for each chunk. This process is repeated several times until all of the chunks in the dataset are processed. With each loop, the model is updated. The tree is constructed gradually from the ground up. This means not only adding new branches but also pruning previous branches to the tree. The best branch to be pruned is based on its contribution to reducing the loss function. I expect that my model will achieve a medium to high average precision score and F1 score. If everything works as I intended, then AP and F1 will be high – above 80 percent. Decision tree-based ensemble

learners are well adapted to imbalanced learning problems and gradient boosting algorithms which are known for their high prediction accuracy, xgboost is both.

**Plans for interrupting results**

There are currently four different outcomes from the result of my project. The first outcome is that both the F1 score and Average Precision are high. This means that my algorithm is working as intended and does a good job of classifying in a data stream environment. Not only is my model good at detecting fraudulent transactions but also good at minimizing false alarms. Machine Learning models often involve the tradeoff between precision and recall. The model has to be sensitive(recall) enough to detect the true frauds but not so sensitive that it triggers too many false alarms(precision). It also means that my model can maintain a high precision across different recall levels.

The second outcome is that F1 score is high, but the Average Precision is low. In this scenario, my model is doing well in terms of overall precision and recall. This means my model can correctly identify instances of the positive class and minimize false positives and false negatives. However, the precision of the model is dropping quickly as the recall gets high enough. This means that while my model is good at classifying predictions in general, it is not so accurate where positive predictions are concerned. As the model tries to capture more positive instances, it is making more mistakes. This could mean that my model is not dealing well with class imbalance. This is because Average Precision is more sensitive to class imbalance.

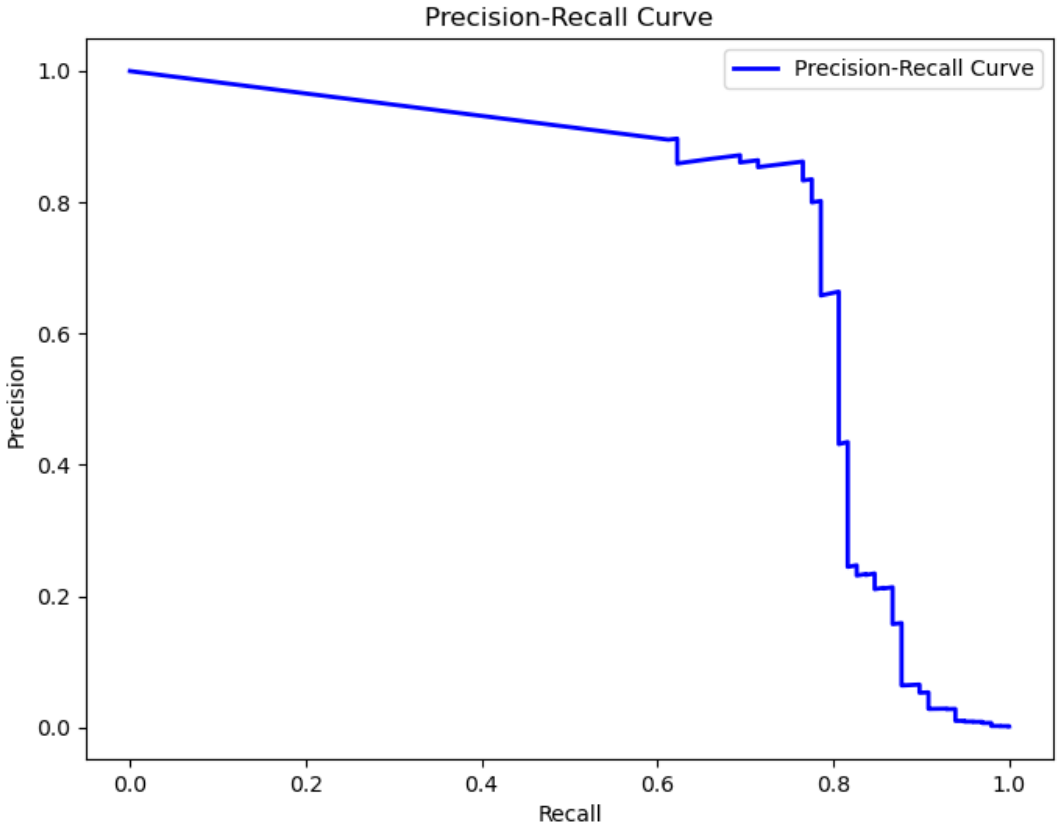
The third outcome is that F1 is low but Average Precision is high. A low F1 score means that the model does not do a good job of finding the optimal balance between precision and recall. However, the model has a relatively high precision at different recall levels. This means

that while there are few false alarms, the model is struggling to correctly classify the positive class. This result could mean either class imbalance or suboptimal model parameters.

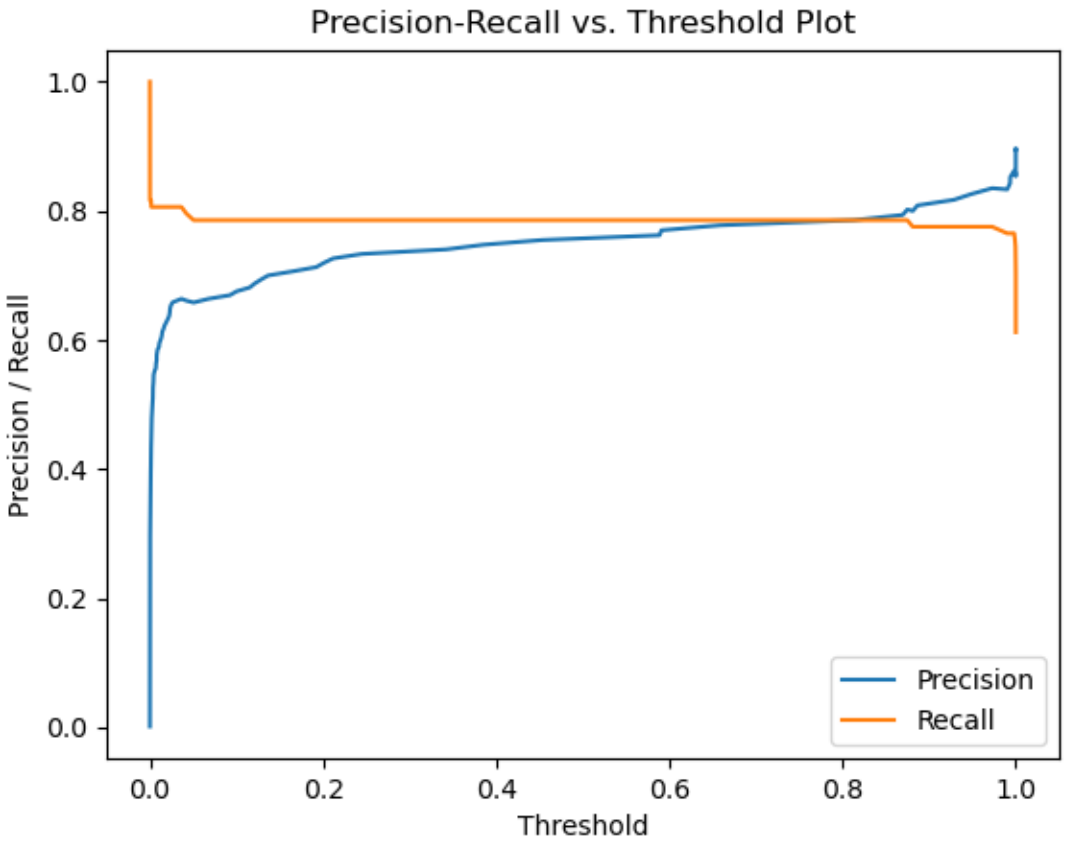
The fourth outcome is that both F1 and Average precision are low. This would mean that my algorithm is incorrectly suited to the task. Further Hypermeter tuning or feature engineering might improve the algorithm but are unlikely to yield high evaluation scores. In this case, another algorithm other than xgboost will be better suited to the task. Based on my research, I do not expect this to happen. However, my evaluation scores might be lower than I anticipated due to several factors. It may be because converting the static dataset into a simulated data stream is not sufficient to emulate all the unique aspects of a real data stream. It may be because I used a batch data stream algorithm instead of an instance-based data stream. Minimal hyperparameter tuning could also lead to sub-optimal evaluation scores. It may indicate that my model is struggling to deal with class imbalance. This is unlikely since gradient boosting algorithms are robust against this in addition, I can configure xgboost to add greater weights to minority instances.

## Results

The following graph shows how well my model balances the tradeoff between precision and recall.




As to be expected, precision and recall are inversely correlated. However, this changes when we hit the 0.8 recall mark. As the decision threshold is adjusted to achieve higher recall, a critical threshold is reached where the model becomes too lenient in classifying instances as positive, which may abruptly lead to a sudden increase in the number of false positives. This is when precision declines dramatically. This means that my model performs very well for most recall levels except those higher than 0.8 when the precision and recall trade off becomes very unfavorable.




The threshold on the x-axis is a decision threshold between 0 and 1 that separates instances into positive and negative classes. When the predicted probability is greater than the threshold, then the instance is positive and vice versa. We see a clear relationship between precision and recall in this graph. The higher the threshold the higher the precision but the lower the recall. Ideally, we would want the highest possible precision and recall. In other words, we would want to prevent as many false alarms as possible(high precision) to enhance the customer experience, reduce investigation costs, and reduce the risk of blocking customers' access to funds. However, the model should be sensitive enough to pick out a large enough pool of potential frauds for the fraud team to investigate. If employing a conservative strategy focused on cost savings, the threshold should be set high to prevent false alarms. This is because when

the threshold is set high, the model has a very high degree of certainty or confidence in a prediction. This allows an institution to maintain cost-efficient fraud investigation operations. At the 0.8 threshold, the precision and recall curves intersect. This is the point where beyond which precision is prioritized over recall.

| Time Interval chunk(seconds) | Average Precision | F1 score |
|---|---|---|
| 0 - 7200 | 0.65 | 0.72 |
| 7200-14400 | 0.70 | 0.74 |
| 14400-21600 | 0.70 | 0.75 |
| 21600-28800 | 0.71 | 0.68 |
| 28800-36000 | 0.69 | 0.68 |
| 36000-43200 | 0.73 | 0.74 |
| 43200-50400 | 0.76 | 0.73 |
| 50400-57600 | 0.80 | 0.70 |
| 57600-64800 | 0.79 | 0.67 |
| 64800-72000 | 0.77 | 0.61 |
| 72000-79200 | 0.63 | 0.48 |
| 79200-86400 | 0.73 | 0.50 |
| 86400-93600 | 0.67 | 0.68 |
| 93600-100800 | 0.64 | 0.64 |
| 100800-108000 | 0.72 | 0.68 |
| 108000-115200 | 0.75 | 0.74 |
| 115200-122400 | 0.71 | 0.71 |

| | | |
|---|---|---|
| 122400-129600 | 0.73 | 0.67 |
| 129600-136800 | 0.77 | 0.68 |
| 136800-144000 | 0.74 | 0.63 |
| 144000-151200 | 0.73 | 0.70 |
| 151200-158400 | 0.73 | 0.74 |
| 158400-165600 | 0.73 | 0.76 |
| 165600-172800 | 0.73 | 0.77 |
| Final | 0.73 | 0.77 |

The Average Precision(AP) score is the area under the precision-recall curve and its scalar value. It shows the ability of a model to rank positive instances over negative ones. An AP score of 0.5 would indicate a random model. The Average Precision score for my model is 0.73 which shows my model is effectively prioritizing fraud transactions. The F1 score is the harmonic mean of precision and recall. In other words, it is a metric that assumes a balance between precision and recall. A random guessing model would achieve a 0 F1 score. The F1 score for my model is 0.77 which means it has a good balance of precision and recall.

## Discussion

I believe that my Average Precision and F1 scores are lower than I expect them to be. While they were not terrible, they were not high either. I believe that my model might have suboptimal hyperparameters. An ensemble decision tree-based machine learning model has a very large number of hyperparameters. Tree algorithms like XGBoost are one of the most complex ML algorithms to be tuned because not only do they have many hyperparameters, but they also have many different types of hyperparameters(Yang & Shami, 2020). As the number of

hyperparameters linearly increases, there is an exponential increase in the number of hyperparameter combinations that have to be tried. Different hyperparameters may not be independent of each other. Additionally, I cannot use the brute force solution of trying every combination of hyperparameters for set ranges like Grid Search and similarly Randomized Search. Instead, I will have to use hyperparameter optimization(HPO) methods like hyperband, genetic algorithm, or Particle Swarm Optimization (Yang & Shami, 2020) to truly find the best hyperparameters. These are outside the scope of my project.

Another explanation for mediocre performance scores may be because my algorithm is a batch algorithm that is adapted for the data steaming task. However, for a true online learning algorithm, it has to have the ability to process the data instance by instance. There is evidence to suggest that instance algorithms are better performing and more memory efficient than batch algorithms. They are better performing because they are better able to adapt to concept drift especially complex drift where there are multiple types of drift in an interval period. While batch algorithms can mitigate this somewhat while having higher update frequencies, they may still be insufficient in some cases(Gomes et al, 2017).

## Conclusion

I learned that the credit card fraud detection task is different from other machine learning tasks in that the data source is a data stream. Having the data source be a data stream means that one does not have access to the whole data set. This means that the machine learning model has to be built with a continuous trickle of data instead of one large dataset input. Since most traditional machine learning models are not adapted to data stream learning, they have to be modified to fit this new paradigm. In addition, they have to consider different types of concept

drift. The dynamic nature of drift means that the model must adapt to the new data while simultaneously retaining some of the old concepts that are still relevant. Extreme class imbalance is one of the factors that must be considered in the credit card fraud task. The extreme nature of class imbalance disqualifies machine learning algorithms that are sensitive to class imbalance like Naïve Bayes or K Nearest Neighbors. In addition, class imbalance makes certain traditional evaluation procedures ineffective like accuracy, ROC-AUC, and confusion matrix due to being highly misleading in this context.

The shortcomings of my project are that my method does not take into consideration customer behavior. Anomaly detection in the case of cardholder behavior would provide an important piece of information that would enhance the performance of my model. Anomaly detection can be used as a precursor layer to a two-layer fraud detection system that will filter out transactions that deviate significantly from normal patterns or it can be incorporated as part of the main algorithm by aggregating cardholder data and using it as an additional feature column in the training dataset. Another shortcoming is that of the data itself. The dataset only contains two days' worth of data. This is an insufficient amount of time to train an algorithm properly for a task of this type because it cannot learn long-term or seasonal patterns in fraud. If the data spanned a period of two years, then the model would be forced to contend with many more unique types of fraud behavior patterns and types of concept drifts. The data distribution of the stream may not change significantly in a two-day time frame.

In addition, such a small amount of data could make the model overfit due to noise or randomness which could throw off my model's ability to generalize.

The strength of my project is that I came up with an approach that comprehensively seeks to tackle many of the challenges and problems that would undermine a more traditional or

simplistic model from accurately classifying fraud instances. For example, I chose an XGBoost algorithm that is insensitive to class imbalance, able to learn complex patterns from data, can handle large volumes of data sufficiently, and utilize ensemble learning to improve the reliability of predictions, and modified it for use in a data-streaming context. I made sure to simulate the model being updated frequently so that it has a greater ability to deal with concept drift. In addition, I chose to create a simulated data stream from a real dataset because I believed that it would be closer to a real data stream compared to a completely synthetic data stream. In the end, I created a model that can perform moderately well with room to improve.

## References

Chen, T., & Guestrin, C. (2016). XGBoost: a Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 785–794. https://doi.org/10.1145/2939672.2939785

Dal Pozzolo, A., Caelen, O., Le Borgne, Y.-A., Waterschoot, S., & Bontempi, G. (2014). Learned lessons in credit card fraud detection from a practitioner perspective. *Expert Systems with Applications*, *41*(10), 4915–4928. https://doi.org/10.1016/j.eswa.2014.02.026

Fang, K., Vaikkunth Mugunthan, Ramkumar, V., & Lalana Kagal. (2022). Overcoming Challenges of Synthetic Data Generation. *2022 IEEE International Conference on Big Data (Big Data)*. https://doi.org/10.1109/bigdata55660.2022.10020479

Fernandez, A., Garcia, S., Herrera, F., & Chawla, N. V. (2018). SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary. *Journal of Artificial Intelligence Research*, *61*, 863–905. https://doi.org/10.1613/jair.1.11192

Ghosh, S., & Reilly, D. L. (2011, April 11). Credit card fraud detection with a neural-network. *Roceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=323314

Gomes, H. M., Barddal, J. P., Enembreck, F., & Bifet, A. (2017). A Survey on Ensemble Learning for Data Stream Classification. *ACM Computing Surveys*, *50*(2), 1–36. https://doi.org/10.1145/3054925

Kundu, A., Panigrahi, S., Sural, S., & Majumdar, A. K. (2009). BLAST-SSAHA Hybridization for Credit Card Fraud Detection. *IEEE Transactions on Dependable and Secure Computing*, *6*(4), 309–315. https://doi.org/10.1109/tdsc.2009.11

Montiel, J., Mitchell, R., Frank, E., Bernhard Pfahringer, Talel Abdessalem, & Bifet, A. (2020, July 1). Adaptive XGBoost for Evolving Data Streams. *Research Commons (the University of Waikato)*. 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK. https://doi.org/10.1109/ijcnn48605.2020.9207555

Panigrahi, S., Kundu, A., Sural, S., & Majumdar, A. K. (2009). Credit card fraud detection: A fusion approach using Dempster–Shafer theory and Bayesian learning. *Information Fusion*, *10*(4), 354–363. https://doi.org/10.1016/j.inffus.2008.04.001

Şahin, Y. G., & Duman, E. (2011). Detecting credit card fraud by decision trees and support vector machines. *Proceedings of the International MultiConference of Engineers and Computer Scientists*, *1*, 442–447. https://openaccess.dogus.edu.tr/xmlui/bitstream/handle/11376/2366/eduman_2011.pdf?sequence=1&isAllowed=y

Srivastava, A., Kundu, A., Sural, S., & Majumdar, A. K. (2008). Credit Card Fraud Detection Using Hidden Markov Model. *IEEE Transactions on Dependable and Secure Computing*, *5*(1), 37–48. https://doi.org/10.1109/tdsc.2007.70228

Xuan, S., Liu, G., Li, Z., Zheng, L., Wang, S., & Jiang, C. (2018). Random forest for credit card fraud detection. *IEEE Xplore*, 1–6. https://doi.org/10.1109/ICNSC.2018.8361343

Yang, L., & Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, *415*, 295–316. https://doi.org/10.1016/j.neucom.2020.07.061

Zhang, C., Zhang, Y., Shi, X., Almpanidis, G., Fan, G., & Shen, X. (2019). On Incremental Learning for Gradient Boosting Decision Trees. *Neural Processing Letters*, *50*(1), 957–987. https://doi.org/10.1007/s11063-019-09999-3

Liu, X.-Y., Wu, J., & Zhou, Z.-H. (2009). Exploratory Undersampling for Class-Imbalance Learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 39(2), 539–550. https://doi.org/10.1109/tsmcb.2008.2007853

## Appendix

Since my computer code is too lengthy and would take up too much space in my paper, I decided to convert my Jupyter Notebook into markdown and Latex formats. The following links contain the computer code and the outputs of my Machine Learning algorithm.

Google Drive

https://drive.google.com/drive/folders/1jMjyCY6Y5jlgDhpkyaFJZdOs5tmuDeCK?usp=drive_link

Jupyter Notebook

workspace6.ipynb