A modular, scalable and realtime fraud detection system

DISSERTATION

Submitted in partial fulfillment of the requirements of the

MTech Software Engineering Degree programme

By

Bala Dutt
2018ID04610

Under the supervision of

Manish Shah, Distinguished Engineer

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
Pilani (Rajasthan) INDIA

June, 2021

DSE IDZG628T DISSERTATION

A modular, scalable and realtime fraud detection system

Submitted in partial fulfillment of the requirements of the

M. Tech. Software Engineering Degree programme

By

Bala Dutt
2018ID04610

Under the supervision of

Manish Shah, Distinguished Engineer

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI (RAJASTHAN)

June, 2021

ACKNOWLEDGEMENTS

To live for a time close to great minds is the best kind of education.
John Buchan

This work has greatly benefitted by the intellect, knowledge, effort and time of two experts that I had the privilege to work with. I did use more of their time than the process stipulates and they have been gracious to be highly responsive.

First is my supervisor **Manish Shah**. Manish's rich experience building products and technology for more than two decades was very helpful. His ideas and feedback improved the quality of thinking and implementation.

Second is **Prof. Shan Sundar Balasubramaniam**. I have been inspired by the courses that sir took. They have led me to take on this problem and take a system's view. This work has benefitted by his feedback, questions, ideas and support. Getting confirmation and feedback from an expert has been confidence inspiring.

Last but not the least, I am inspired by my wife, **Dr Rumpa Reshmi Munshi**. She wrote her phD thesis recently and that has inspired me to keep a high bar.

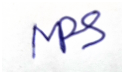
BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

CERTIFICATE

This is to certify that the Dissertation entitled A modular, scalable and realtime fraud detection system

and submitted by Mr./Ms. Bala Dutt IDNo. 2018ID04610

in partial fulfillment of the requirements of DSE ID ZG628T Dissertation, embodies the work done by him/her under my supervision.



Signature of the Supervisor

Name Manish Shah

Designation Distinguished Engineer

Place: Bengaluru

Date: June, 2021

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
FIRST SEMESTER 2020-21

DSE ID ZG628T **DISSERTATION**

Dissertation Title : A modular, scalable and realtime fraud detection system

Name of Supervisor : Manish Shah

Name of Student : Bala Dutt

ID No. of Student : 2018ID4610

Abstract

Key Words: Fraud, Realtime, Streaming, Scale, PAIR, MASSES, Simulator, SBE, Modularity, XACML, EAV, Markov-chain

A fast, adaptive and effective fraud detection system architecture, PAIR, is proposed and demonstrated. It allows multiple transactional data streams and supporting reference data streams with information fusion support. An ensemble of ML algorithms, with a combination of supervised/unsupervised, stream learning/offline learning and rule based are supported. The output is actionable, with multiple delivery mechanisms, to bring human in the loop. The system is responsive in reacting before the transaction completes and also in adapting to evolving situations. Later is enabled by modularity in the design to allow changes on the fly. New stream processing can be defined in a newly designed language. System provides for a set of integrative approaches, ability to define features, maintain history, compare with it and allow multiple separate processing at the same time. System is designed for scale at runtime and scale of development. A MASSES Simulator was built to validate the system from functional and non-functional (scale, response time etc.) point of view. A language for creating multiple simultaneous simulations, on the philosophy of specification by example, was built.

List of symbols & abbreviations used

ML	Machine Learning
MCOD	Micro Cluster based Outlier Detection
STORM	STream. OutlieR Miner
DODDS	Distance-based outlier detection in data streams
ISB	Index Stream Buffer
ATO	Account takeovers
XGBoost	eXtreme Gradient Boosting
ATM	Automated teller machine
OTP	One-time password
XACML	eXtensible Access Control Markup Language
GESD	Generalized Extreme Studentized Deviation
PAIR	Pluggable processing, Actionable, Information fusion and Responsive
SBE	Specification By Example
EAV	Entity Attribute Value
DSL	Domain Specific Language
ATO	Account Take Over
HITL	Human In The Loop
JAAS	Java Authentication and Authorization Service
MOA	Massive Online Analysis
WEKA	Waikato Environment for Knowledge Analysis
ADWIN	ADaptive sliding WINdow

PMML	Predictive Model Markup Language
BDD	Behavior Driven Development
ATDD	Acceptance Test–Driven Development
TDR	Test-Driven Requirements
ORC	Optimized Row Columnar
MASSES	Multi Actor Scenario Stochastic Event Simulation

Table of figures

Table of Tables

Table of Contents

Chapter 1

Background

Digital transformation has been greatly accelerated with recent events of COVID-19. The FinTech industry was on this path as economies were moving to efficient money movement and management technologies. This includes instant pay, micro-payments (with no costs in transactions), opening up banking to applications (OpenBanking in Europe and Aggregator guidelines by RBI in India) and block chain related technologies. New products and services are being introduced like robot advisory and auto-trading, value-added services based on account information and transaction history, or ad-hoc loans in online-banking. Many of these services are facilitated by applications of Artificial intelligence (AI) approaches. As we reduce friction, allow machines to do everything, this hypergrowth can be sustained only by keeping up with security.

Online Frauds

Fraud is wrongful or criminal deception intended to result in financial or personal gain. Thus, bank fraud is commonly described as a criminal act that occurs when a person uses illegal means to receive money or assets from a bank or other financial institution. [2] Bank fraud is distinguished from bank robbery by the fact that the perpetrator keeps the crime secret, in the hope that no one notices until he has gotten away.

Frauds are moving online. In 2016, total fraud involving Single European Payment Area (also known as SEPA: the EU Member States plus Switzerland, Iceland, Lichtenstein, Norway) cards decreased to 1.8 billion euros, which is 0.8% less than in 2015. Card fraud at ATMs dropped by 12.4% and online fraud rose significantly, accounting for 73% of the total value of card fraud in 2016. One Euro for every 2,428 Euros spent on payment cards was lost to fraud. Online card fraud is naturally increasing as digital services develop further and are becoming more and more sophisticated.

The most common types of online fraud reported by the industry are “clean fraud” – where criminals obtain genuine cardholder details including 3D Secure and Address Verification credentials – and “identity theft” – where the fraudster steals the cardholder’s personal data in order to make unauthorized online transactions. There is also a higher risk of insider threats nowadays.

Insider threats

Organizations increasingly face internal threats. SEI’s CERT National Insider Threat Center (NITC) is observing September 2020 as “National Insider Threat Awareness Month” with the theme resilience. A well known example is Snowden-like access, where a valid user with right clearance level accessed resources and performed operations that were legal, but what was not normal was accessing hundreds of thousands of documents over a weekend.

It defines insider threat as "the potential for an individual who has or had authorized access to an organization's assets to use their access, either maliciously or unintentionally, to act in a way that could negatively affect the organization." Insiders can be current or former employees, trusted business partners, contractors, suppliers, or anyone to whom an organization grants access to its critical assets.

This includes wider access given than necessary either due to lack of finer controls or access available even after the Subject leaves the organization.
NITC says, “Insider threat actors have certain behaviors that are observable before a harmful act occurs, such as signs of disgruntlement, financial stress, or an interest in benefiting another organization at the expense of your employer. These activities require different detection and response strategies than threats from external attackers.”

“Insider Threats grew from 2.9% in 2018 to 5.5% today, while Accidental Disclosures increased from 14.7% to 18.2%.” - securitymagazine.com. IBM and the Ponemon Institute estimate that 50% of data breaches are caused by malicious or criminal attacks, while 27% resulted from system glitches and 23% from negligent employees. These statistics reveal that data security strategies need protection against both internal and external threats, including system malfunctions, employee error and outside individuals and groups that intend to steal the data to withdraw money, open lines of credit, commit identity theft, blackmail clients or engage in other types of criminal activity.

In addition, malicious attacks were more expensive: “Companies that had a data breach due to malicious or criminal attacks had a per capita data breach cost of $236, significantly above the mean of $221. In contrast, system glitches or human error as the root cause had per capita costs below the mean ($213 and $197, respectively).”

Bottom line is that perimeter security and merely blocking bad actors at entry (signin) is not enough. The activities have to be monitored and malicious intent checked. This should be coupled with a zero-trust model.

Adversarial attacks

Frauds are getting sophisticated. In many domains, efficient classification systems are being built to fight fraud and other malicious activities. Yet there are adversaries who are able to evade a system’s defenses, resulting in a security breach. Resulting breaches cost businesses billions of dollars every year as well as harm their reputation with customers. In the domain of credit card fraud detection, fraudsters are probing the classification system in order to generate fraudulent transactions that go undetected. For example, fraudsters can purchase thousands of credit card numbers and social security numbers as a means of testing and learning of the current fraud detection systems in use. The performance of the fraud detection system can progressively deteriorate and the amount of time and cost incurred in maintenance is significant. Currently, many fraud detection systems are deployed that are successful in flagging genuine fraudulent transactions, but there are few systems that actively incorporate an adversary’s potential

strategies when attempting to improve defenses. In order to deploy robust systems and create an adaptive model, knowledge of an adversary's most effective strategy is beneficial.

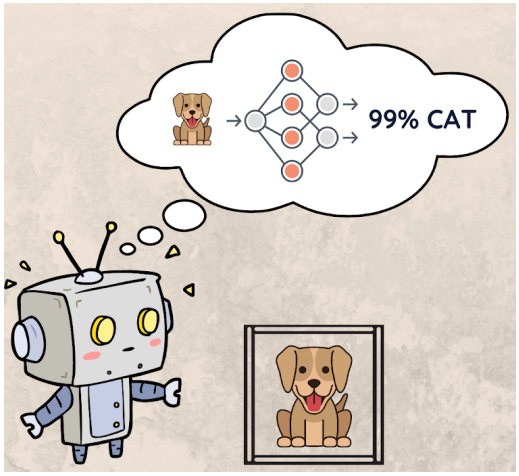


Figure 1. Robot understanding algorithm through samples

Taking into account knowledge of the fraud detection system with the attacker can help create more sophisticated systems that learn online. Few of the ways in which adversarial attacks may happen are,

- Adversarial inputs, - specially crafted inputs to be misclassified to evade detection.
- Data poisoning attacks - pollute training data, feedback weaponization
- Model stealing techniques - blackbox probing

Expectations

Industry typically has used the "keeping money aside" (write-off) approach for fraud. This would have worked at the scale of the past, where online transactions were just a part of all transactions, there was a time delay in settlement and offline mechanisms of investigation were enough. Fraudsters are having an edge today. As labcom fit puts it, "Criminals are creative, collaborative, well-funded and technologically advanced."

Sharing data, having smarter systems and high scale systems that respond very fast is bare minimum expected from an effective system.

To meet such expectations we would use streaming systems and outlier detection mechanisms.

Streaming system

A streaming system may be defined as a type of data processing engine that is designed with infinite datasets in mind. Other terms associated with streaming systems are low-latency, approximate results and unbounded data. A stream is defined as an element-by-element view of the evolution of a dataset over time. The short response time needs of the system point to streaming system design. Transactions are happening and the system makes its calculations as events happen and may wait for some window for remembering history.

Outlier detection

Most of the fraud detection in given situations boils down to outlier detection. Outlier (or anomaly) analysis forms a key mechanism in modern data science and analytics, aiming to detect objects (or points) that appear to be inconsistent with the remainder of the objects in the same dataset. Outlier detection is used in a variety of applications, such as fraud detection, computer network security and medical diagnosis. Because they can mean different things at different times, outliers can be extremely challenging to deal with. On the one hand, the presence of an outlier may point to a hacker who just hacked a system, a thief trying to access a stolen credit card, a rare opportunity in the stock market, or an unhappy customer. On the other hand, outliers may just be statistical errors, random noise of chance that happened upon the dataset. Because of the importance of fast outlier detection on datasets, there have been a big number of techniques for categorizing a data point as an outlier or not.

One of the first approaches for detection is through the use of statistical tests. Based upon a statistical distribution, assume that each data point was generated by such a distribution. From this point of view, outliers are the data points, which have a low probability to be generated by the overall distribution. Another approach is the depth-based one. This technique assumes that the normal data points are collected in the center of the data space while the outliers are located at the edges (border) of the same data space. In order to detect the outliers, one just needs to search the outer layer. Deviation-based approaches also exist in the detection of outliers. In a group of data points, an outlier is the point that does not fit to the general characteristics of the group. For example, if the variance of the group is minimized when removing a point, then that particular point is an outlier. A common technique used for detection is the density-based approach. In this technique the density of each point is computed and compared with the density of its local neighbors. This computation is called the outlier score for each point. This approach is based on the assumption that the density around a normal data point is the same as the density of its neighbors. Finally, one of the most commonly used definitions for outliers is the distance-based one, where an object is an outlier if it does not have more than k neighbors in a distance up to R.

Continuous outlier detection in data streams deals with the problem of keeping an updated list of all outliers after each new object arrives and/or an object expires. Because of the nature of a data stream, all of the data points cannot be stored on the main memory for outlier detection. For that reason, the stream is split into windows and the algorithms are performed based on the data points of the current window.

When data grows large, outlier detection becomes a very challenging task, since applying even a one-pass algorithm to all active data is pro- hibitively expensive. As such the techniques applied on outlier detection algorithms need to be revisited for use in a distributed environment.

Briefly, the relevant state-of-the-art algorithms fall into two categories. The first category contains efficient non-parallel solutions for streaming outlier detection. The second category contains parallel solutions for outlier detection, where, to date, there is a single proposal that

assumes modern distributed computing platforms, such as MapReduce; nevertheless, this solution does not deal with the streaming case.

Devising efficient parallel solutions for this problem involves addressing a series of issues. First, outlier detection algorithms in data streams involve windows that cannot be partitioned into non-overlapping partitions, among which no communication is required. Second, low latency is of high significance in order to deliver results in a timely manner. Third, state information needs to be kept between window slides in order to avoid unnecessary recomputations.

Review of state-of-the art machine learning algorithms

We use the definition,
“An observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism."
In this dissertation we are interested in anomaly or event of interest meaning of outlier and not the noise or unwanted data interpretation as shown below,

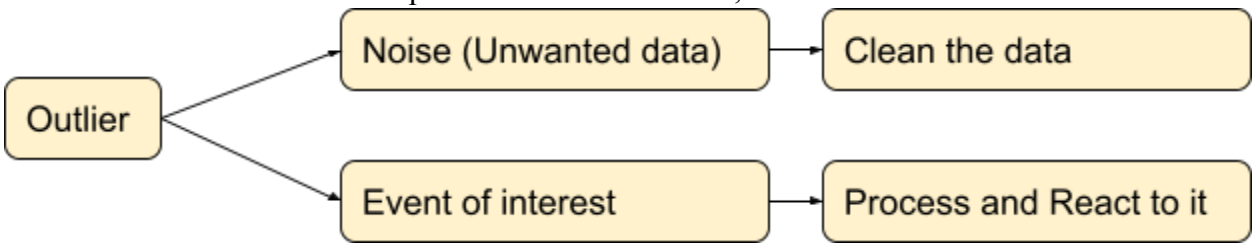


Figure 2. Outliers may mean different depending on usecase

Distance based methods look for an outlier data object o in a generic metric space, if there are less than k objects located within distance R from o.

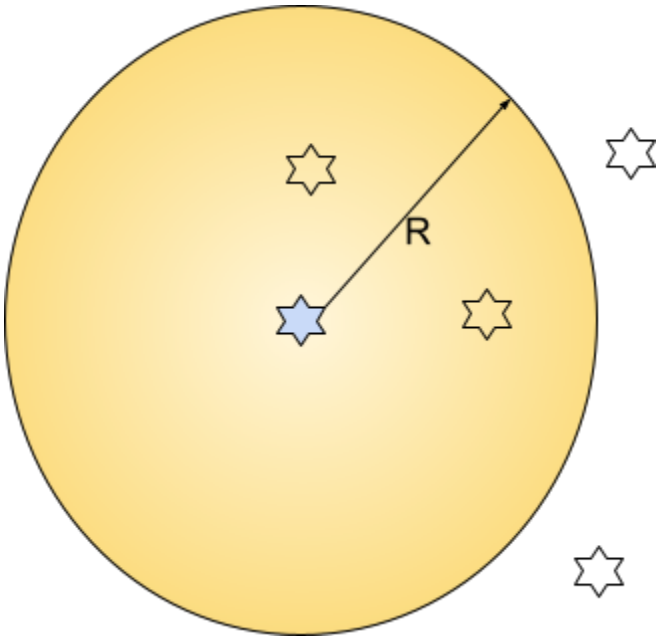


Figure 3. Density based outlier detection

For example here the blue point has 2 neighbours in a distance within R. If k was 3 then this blue point would be an outlier.
Outlier detection can be divided on multiple orthogonal dimensions as shown below. [1]

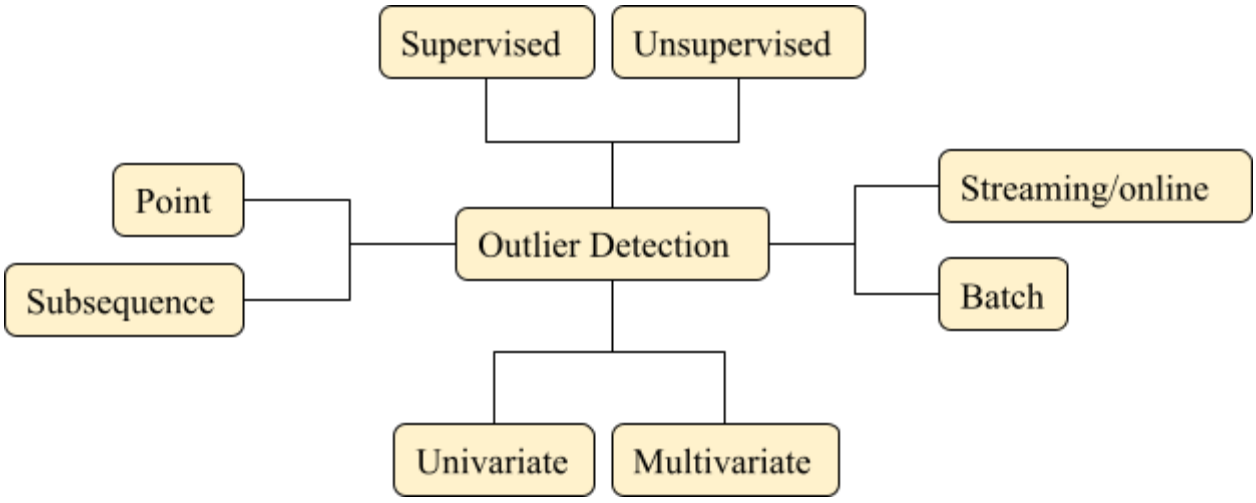


Figure 4. Types of outlier detection approaches.

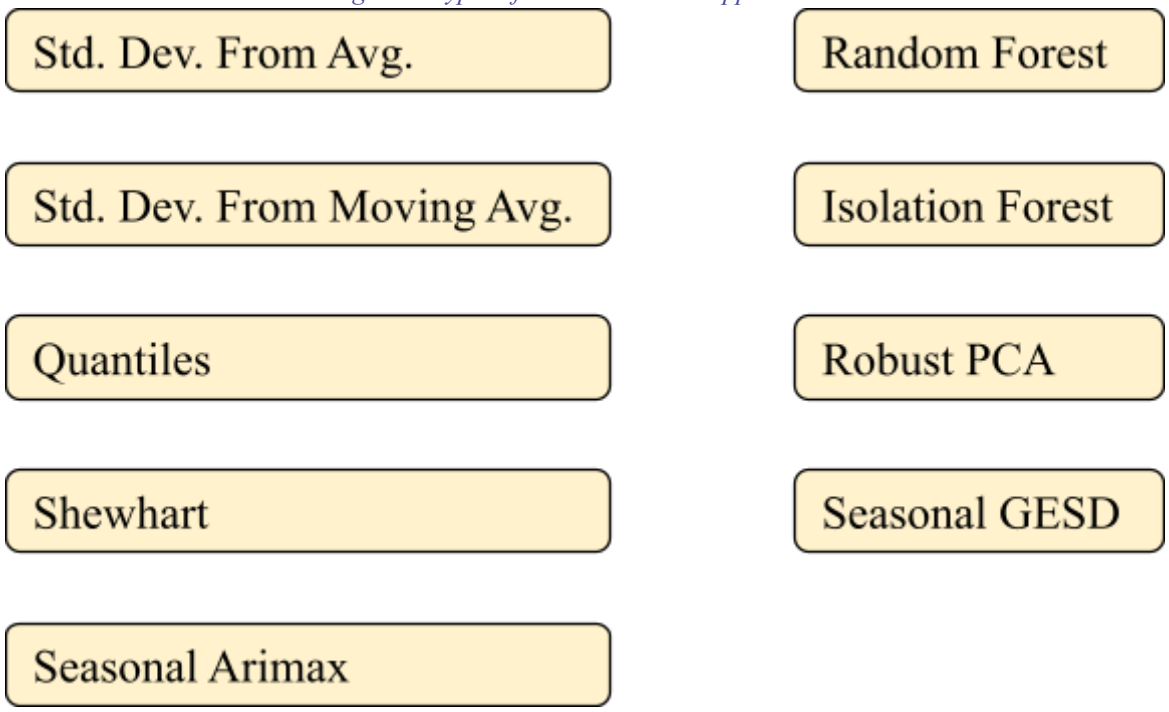


Figure 5. Offline outlier detectors

Here are some of the anomaly algorithms that are used in non-streaming usecases. Most of them are either statistical or decision trees. There are also methods which are prediction based. These are trained offline and deployed for online prediction.

These are cases of point outliers, where a point (data) has to be classified into an outlier or not. Model-based algorithms were discussed just now. Density based algorithms detect local outliers by looking at other points in the vicinity of a point. Histogramming is based on detecting the points whose removal from the univariate time series results in a histogram representation with lower error than the original, even after the number of buckets has been reduced to account for the separate storage of these points.

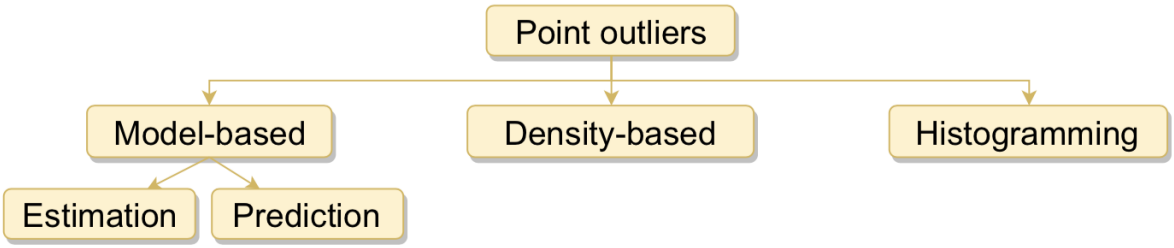


Figure 6. Type of outlier detection algorithms

Sometimes a set of consecutive points jointly behave unusually. This is now a case of subsequence outlier than a point outlier. Since subsequences have multiple points the length becomes important. Methods could be fixed length based on variable length. To also solve for computation complexity the subsequence should be represented adequately. Sometimes these subsequences may be periodic. The subsequence analysis is outside the scope in the rest of the treatment.

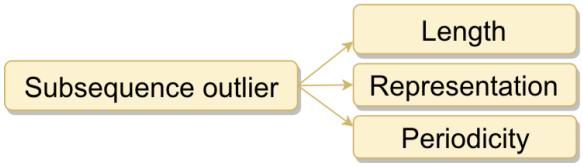


Figure 7. Subsequence outlier detection can depend on length, representation and periodicity

Below are types of subsequence outliers. We will not go into details of them.

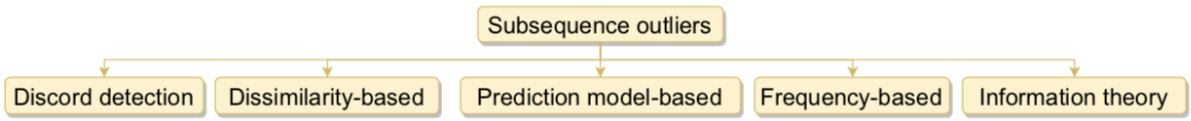


Figure 8. Different type of subsequence outliers

Classification has a wide variety of algorithms that can be included as fraud turns out to be a two class problem. The problem of data highly skewed towards one class can be solved through sample techniques and tuning algorithm thresholds.

We discuss below streaming algorithms that are relevant for the problem at hand.

Outlier detection

Distance-based outlier detection in data streams (DODDS) in unsupervised state and without assumptions on distribution of data is about detecting a data object that is considered an outlier if it does not conform to the expected behavior, which corresponds to either noise or anomaly. [3]

The dataset to be considered for computing this in an unbounded streaming system is data within a window. The window could be count based or time based. Note that in a sliding window the same data point may be part of multiple windows, whereas in a tumbling window they would belong to exactly one window. In a sliding window the point will remember its neighbours from the previous window. [4]

The needs for fraud detection include,

- Not all the data is present as it is streaming of unbounded data. So data present in the time-window should be used.
- Further history should degrade and very old points should not be considered as normal.
- Data coming after the current point should not be depended on to flag outliers. This would prevent flagging the anomaly early.

Exact-Storm

Exact-storm uses Index Stream Buffer (ISB) data structure. It is a pivot-based index where distance of objects from pivots is computed. To find out neighbours of an object o, a close pivot p is used. For each index object o0, if there exists a pivot p, such that $|dist(o, p) - dist(o0, p)| > R$, then, it is known that $dist(o, o0) > R$, and o0 is ignored.

There is an approximate version of this called Approximate-Storm. [5]

AbstractC

Abstract-C uses the information that a data point o will be active for a constant number of slides. Based on that information the algorithm for each data point, instead of storing a list of preceding neighbors and the number of succeeding neighbors, stores the count of its neighbors for each window it is active in a sequence. This saves a lot of space.

MCOD

MCOD (Micro-cluster-based Continuous Outlier Detection) avoids range queries. It creates micro-clusters and assigns data points to them. A micro-cluster has at least $k + 1$ data points all of which are neighbors to each other. Its center is just a point in the metric space and has a radius of $R/2$, implying that the maximum distance between any two objects in the micro-cluster is at most R. Each data point in the cluster is an inlier and does not need to be checked in outlier queries.

Online classifier algorithms

Classifiers are more robust supervised algorithms if the adequate training data is available. The classifiers that train in offline mode are different from ones that train online. In online mode each data arrives as individual and the classifier should train with that be ready to predict. Training and prediction can be simultaneous. Further as time progresses ideally classifiers should lose old knowledge and work on most recent.

Hoeffding tree

The Hoeffding tree is an incremental decision tree learner for large data streams, that assumes that the data distribution is not changing over time. It grows incrementally as a decision tree. Domingos and Hulten proposed the Hoeffding Tree, a very fast decision tree algorithm for streaming data, where we wait for new instances to arrive to cause splits. The most interesting feature of the Hoeffding Tree is that it builds a tree that provably converges to the tree built by a batch learner with sufficiently large data.

A node is expanded as soon as there is sufficient statistical evidence that an optimal splitting feature exists, a decision based on the distribution-independent Hoeffding bound. Here is a sample algorithm,

HOEFFDINGTREE($Stream, \delta$)

Input: a stream of labeled examples, confidence parameter δ

```
1  let HT be a tree with a single leaf (root)
2  init counts n_ijk at root
3  for each example (x, y) in Stream
4       do HTGROW((x, y), HT, δ)
```

HTGROW($(x, y), HT, \delta$)

```
1  sort (x, y) to leaf l using HT
2  update counts n_ijk at leaf l
3  if examples seen so far at l are not all of the same class
4     then
5          compute G for each attribute
6          if G(best attribute) - G(second best) > sqrt((R^2 ln 1/δ) / 2n)
7            then
8                 split leaf on best attribute
9                 for each branch
10                     do start new leaf and initialize counts
```

Figure 9. Hoeffding tree algorithm

The Hoeffding Adaptive Tree is an adaptive extension to the Hoeffding Tree that uses ADWIN as a change detector and error estimator. It has theoretical guarantees of performance and requires no parameters related to change control.

Decision Stump

A decision stump is a machine learning model consisting of a one-level decision tree. That is, it is a decision tree with one internal node (the root) which is immediately connected to the terminal nodes (its leaves). A decision stump makes a prediction based on the value of just a single input feature. Sometimes they are also called 1-rules.

Chapter 2

Problem statement

Intuit has to secure money and sensitive data of users. There are a lot of fraud attacks all the time including account takeovers (ATO). While there are credit card and identity fraud check systems, they have following problems,

1. They are after the fact. Accounts get locked after the fraud has happened. The actor has to be profiled before the attack.
2. They work in silos and do not have external integrations.
3. They work off one algorithm, for e.g. XGBoost. They were written in the era of single algorithm supremacy.
4. They are very hard to change and not pluggable.

Objective of the project : To build a system for fraud detection in real time that is modular for plugging input, reference data, ML algorithm and actions, while being highly scalable

Uniqueness of the project : Being real time and pluggable is the uniqueness of this system. This ideally should be extendable beyond the organization.

Benefit to the organization : Intuit saves money for the customer, itself while also maintaining a reputation for being a good steward of data.

Scope of work

A fraud detection system becomes valuable as it allows more data to be used, multiple ML models to be used and responds before the transaction gets over. This system will enable a new collection of sources to be added on the fly, add new sources of reference information to be added on the fly, add new ML algorithms on the fly, detect fraud in less than 100ms and deliver results to multiple delivery mechanisms including human in the loop. For this, the system will have canonical data formats and pluggable integration mechanisms for components. System will also have a simulator that will generate events through which the rest of the fraud detection system may be tested.

State of the art - background of previous work done

LIP6 and Bleckwen are collaborating to build a real time detection engine, combining explainable AI, behavioural analytics, rules and human in the loop feedback to deliver unparalleled performance. This is supported by French National Agency for Research (ANR). Here are more details,
https://bleckwen.ai/a-new-research-collaboration-to-defeat-financial-crime/.

Money movement is nowadays instantaneous unlike credit card systems where settlement is delayed. People have built systems that come close to realtime, but there aren't systems that are designed for nation scale or web scale including participants from many organizations.

The above link says, "Criminals are creative, collaborative, well-funded and technologically advanced.". The system for the new age has to be much better than siloed effort.

Chapter 3

In this chapter we analyze the problem domain and solution domain. This will then lead us to design the system, described in the next chapter.

Scenarios

Following three scenarios can be useful in problem domain comprehension. The first scenario is of Alice the Grandma who transacts with her bank account only through ATM withdrawals. The money may come into account through small credits or through interest credits. But Alice doesn't use any online facility. Trudy, the fraudster phishes the details and uses it for online shopping or online transfer. Fraud detection system should detect this.

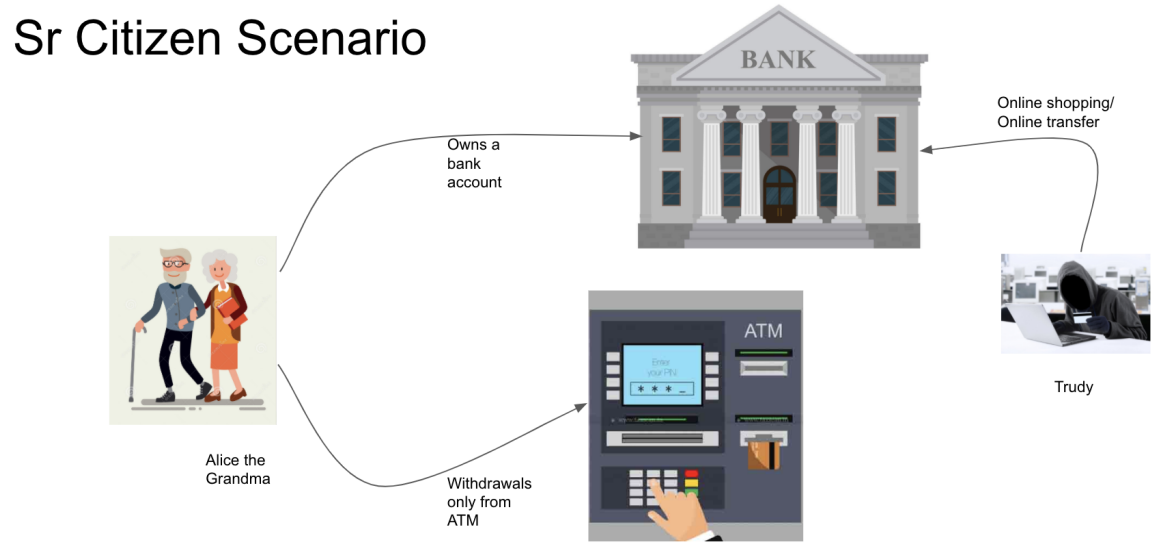


Figure 10. Senior Citizen scenario where Alice only interacts with ATM

Bob, in this scenario, withdraws money from an ATM in Bengaluru, India. Trudy also withdraws money from an ATM in a far away city, Las Vegas in the USA. It is not possible for Bob to travel so far at such short time interval.

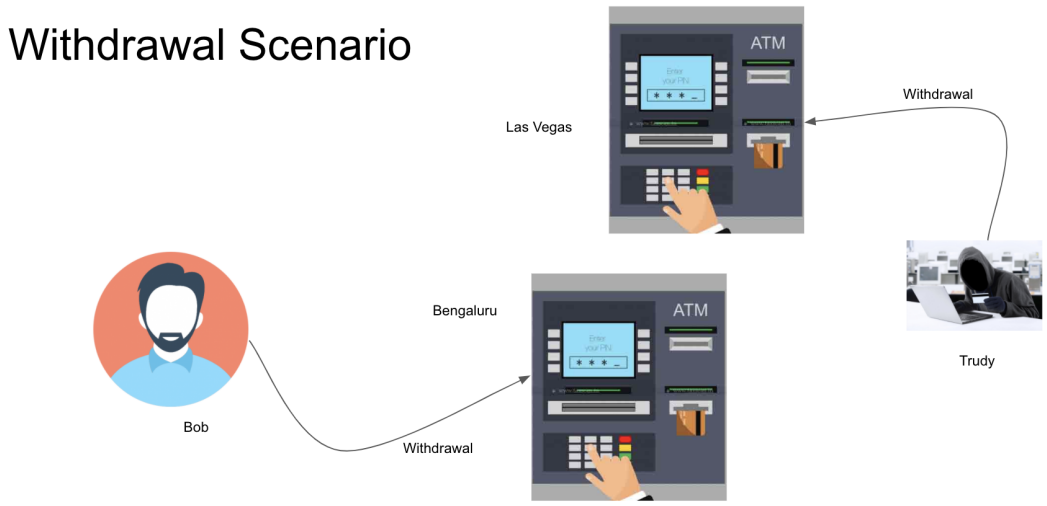


Figure 11. History scenario where hacker transacts from a different location

The last scenario includes cases where the fraudster tricks the payer into believing that money is being received by the payer instead of actually payer paying it. The details of transaction and payee should be easily usable by a fraud detection system. There are also cases of OTP fraud where fraudsters phish for OTP and the account owner is unable to notice this. Can the fraud detection system detect the sequence of activities and safeguard the account owner?

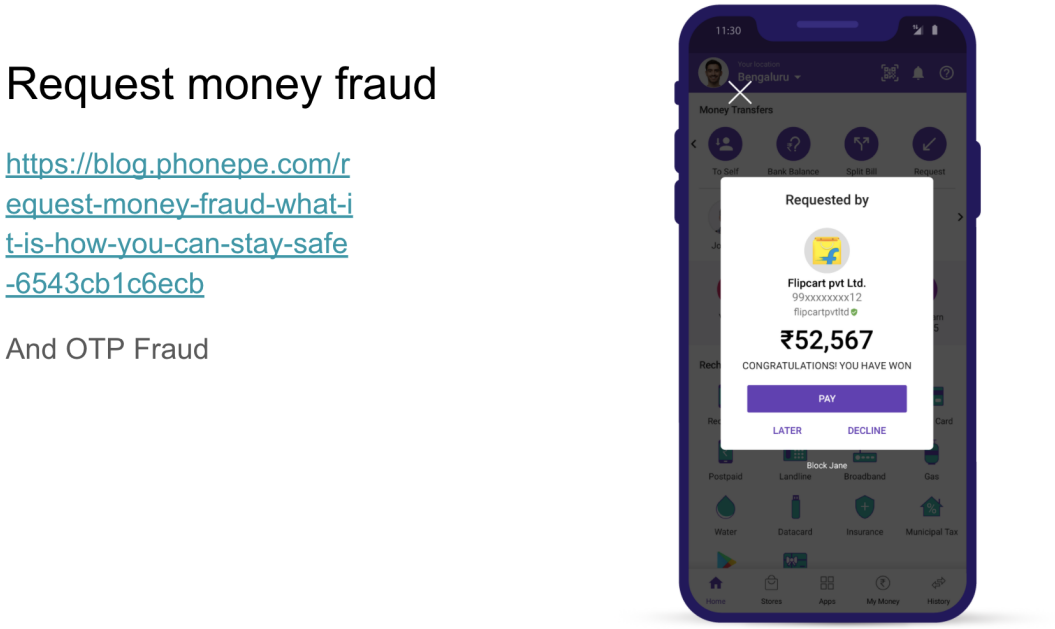


Figure 12. Other types of frauds

Hypothesis

The hypothesis is that systems become more effective if they share data, hence get more data, include humans in loop, respond quickly and rely on the “committee of experts” (ensemble)

approach of ML algorithms (and not rely on a single well tuned algorithm). Also this ensemble can do well in adversarial attacks as it will be difficult for attackers to fool all the systems all the time.

Below is a table showing hypotheses, the capability that is being built and the validation that will be done. Next section talks of validation a little bit more.

Hypothesis	Capability	Validation
A fraud detection system's effectiveness increases with, ● More data (inputs and reference) ● Multiple ML models - committee of experts over an expert, also include user configured thresholds ● Responds before transaction gets over - real time ● Multiple delivery mechanisms including human in the loop (approval vs alerting)	To add new collection sources on the fly, To add new reference sources on the fly, To add new ML algorithms on the fly, To detect fraud in less than 100ms To deliver results to multiple delivery mechanisms including human in the loop Supported by modularity and scale, ● Canonical model - flat data formats, Subject Resource Action Environment model ● Pluggable integration mechanisms for components - decoupled communication. ● Scale strategies ● In future - explainability, auditing, feedback, troubleshooting support	A simulator that will generate events to test the fraud detection system. Adversarial inputs may be generated at a future point using simulator by a relevant adversarial ML algorithm. May be used for scenario based analysis in future

Table 1. Hypothesis for effective system

PAIR Architecture

The hypothesis may be translated into PAIR architecture as shown in the diagram below. There are four aspects,

- Pluggable processing
- Actionable
- Information Fusion
- Responsive

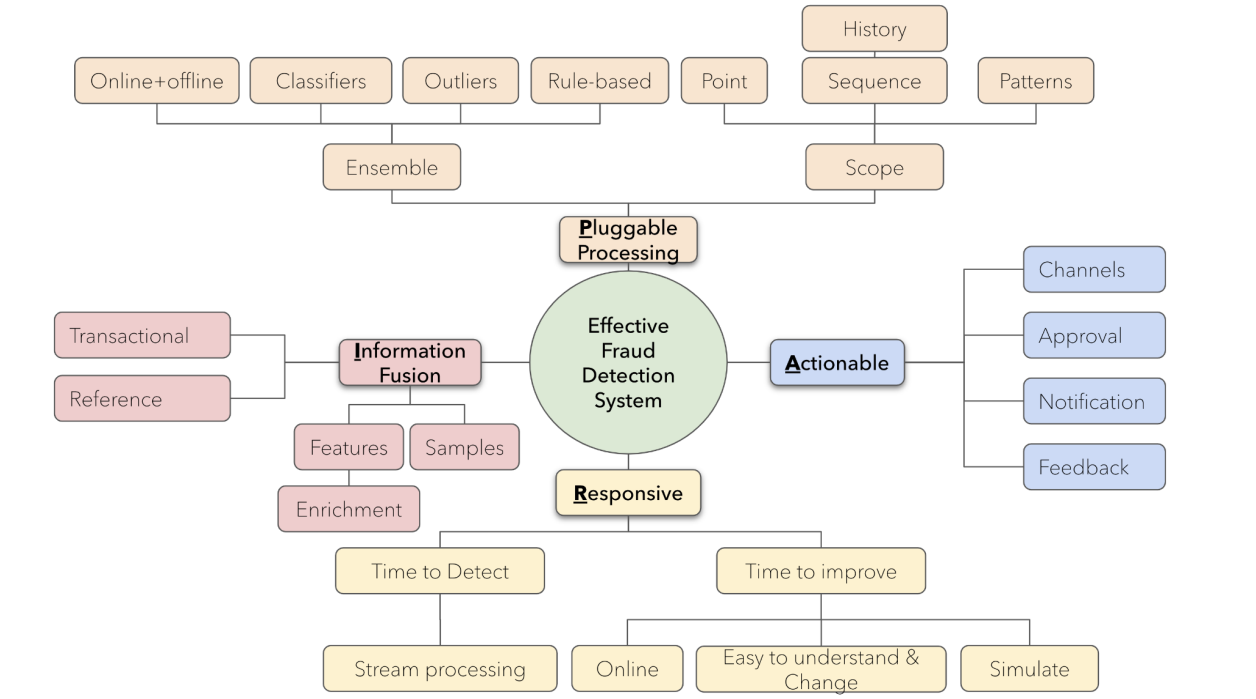


Figure 13. Effective System components

Pluggable Processing

Original progress prize winner (BellKor) for Netflix Grand prize was an ensemble of 107 models [7]. Authors called it as pragmatic chaos and said,

"Our experience is that most efforts should be concentrated in deriving substantially different approaches, rather than refining a simple technique."

"We strongly believe that the success of an ensemble approach depends on the ability of its various predictors to expose different complementing aspects of the data. Experience shows that this is very different than optimizing the accuracy of each individual predictor."

Pluggable processing includes multiple Machine learning algorithms of different types including algorithms that are rule based and configured by humans. These algorithms also look from different perspectives like point analysis of a single data in isolation or a sequence of events on a given resource or broader patterns across large streams. The processing can be composed and added at runtime.

System, then provides for pluggable multiple ML algorithms. The support is for online and offline ML algorithms including classifiers, unsupervised outlier detectors and rule based systems. The ML algorithms could focus on processing a point at a time or can look at history to process a subsequence causing anomaly. Ideal system should also allow for broader patterns to be found beyond one time-series.

Actionable

The results of the ML should be actionable. They should allow actions like approval, or notification and ability to give feedback to the system. Allowing multiple channels with apt choice based on message will provide better user experience and a quick turnaround.

We refer to this as Human in the loop or HITL, which is integration of human in AI pipeline to train and validate in a continuous way. Below is a diagram from an article [9],

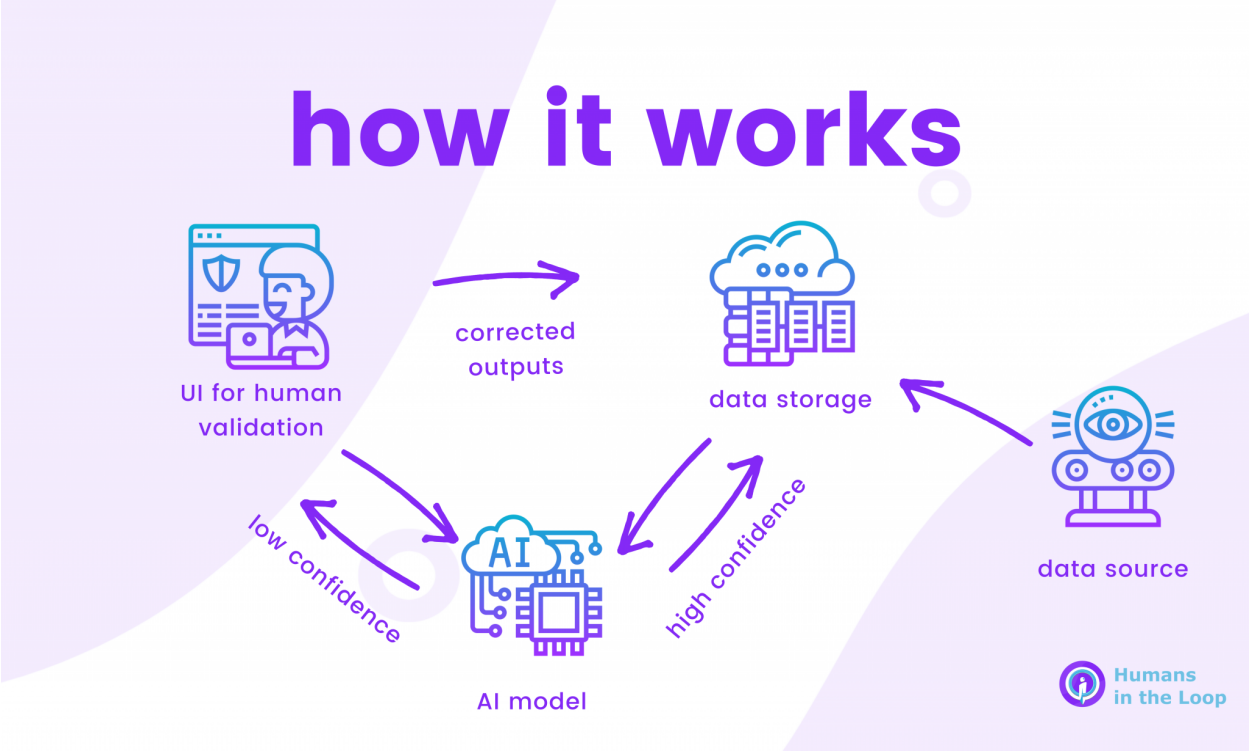


Figure 14. Human in the loop

Information Fusion

Google's Research Director Peter Norvig claimed that "We don't have better algorithms. We just have more data.". This quote is usually linked to the article on "The Unreasonable Effectiveness of Data", co-authored by Norvig. [8]

Not only a variety of transactional (data on which fraud is being detected) but also supporting reference data can be input in the system. The system provides for events to be enriched by reference data. Reference data can be realtime or offline provided. This enrichment causes a larger number of features to be available to ML algorithms. This process is referred to as fusion.

Authors further say,

"Choose a representation that can use unsupervised learning on unlabeled data, which is so much more plentiful than labeled data."

Responsive

Finally, we want the system to be responsive. Responsive as defined in the reactive manifesto is:

"The system responds in a timely manner if at all possible. Responsiveness is the cornerstone of usability and utility, but more than that, responsiveness means that problems may be detected quickly and dealt with effectively. Responsive systems focus on providing rapid and consistent response times, establishing reliable upper bounds so they deliver a consistent quality of service.

This consistent behaviour in turn simplifies error handling, builds end user confidence, and encourages further interaction.”

System detects issues quickly because of stream processing. Because of online algorithms, the system is always learning and can get better as attackers try out new attacks. It uses stream processing to react quickly and also learn quickly. Also, the system is designed to be changed by maintainers easily. For example, the system is easy to understand as it is not monolithic and is modular. The stream processing language allows the system to be seen from a data scientist's abstraction. Further capability to simulate is provided to quickly test the changes. Further by allowing multiple stream processing at same time, A/B experiments may be done for rapid development.

Validation

This system will be validated for functional and non-functional aspects using a simulator. A simulation would consist of a scenario and can have scale activities to reflect real world situations.

Online and offline ML

For the reasons discussed earlier like,

- Instant transactions - time window to fail a transaction getting narrower
- Sophisticated attacks that learn detector (Adversarial attacks)
- Insider threats
- Zero-day attacks - where a customer creates a new account and suffers fraud attack

we need an online system that learns even after its first training. It will be supported by offline ML algorithms that train on batch data and then augment online algorithms. The focus of this dissertation will be online algorithms.

Domain Model

The model is an extension of Entity–attribute–value model (EAV). EAV is a data model to encode, in a space-efficient manner, entities where the number of attributes (properties, parameters) that can be used to describe them is potentially vast, but the number that will actually apply to a given entity is relatively modest. Such entities correspond to the mathematical notion of a sparse matrix.

EAV is also known as object–attribute–value model, vertical database model, and open schema.

The domain model for the system consists of Subject, Resource, Action and Environment entities as provided by the XACML standard. An Entity is defined as a container for attributes, where each attribute is a key-value pair. Entity is,

- Representation of any data handled by the system
- Has a flat model as list of attributes each with key and a value
- Attributes may be called as fields

Each event that is sent to the system is about a Subject accessing a Resource to perform an Action in an Environment.

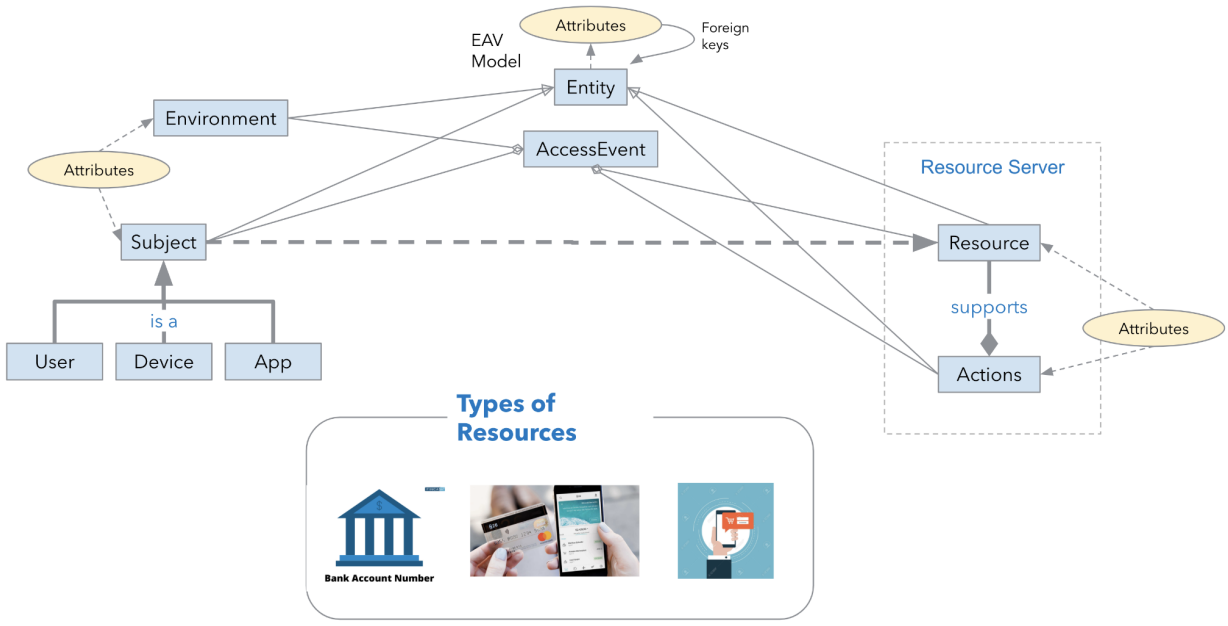


Figure 15. Domain Model for data

A resource is what is to be protected or what is accessed. It is also called an object in some models. It supports a set of actions. Id is mandatory for a resource as it identifies the resource.

A subject is simply an actor accessing resources. It defines an entity that can model multiple kinds of actors like users, devices or applications. In a security context, a subject is any entity that requests access to an object. Principal is a subset of a subject that is represented by an account, role or other unique identifier. User is a subset of principal usually referring to a human operator. In JAAS - A Subject consists of a set of Principals, where each Principal represents an identity for that user. For example, a Subject could have a name Principal ("Susan Smith") and a

Social Security Number Principal ("987-65-4321"), thereby distinguishing this Subject from other Subjects.

An event, called an access event, has a resource, subject, action and environment.

Integrative aspects

Having a common domain model of Subject, Resource, Action and Environment enables multiple collection data sources to be integrated. It also enables a common processing framework where multiple ML algorithms can be plugged in.

In addition to this reference data is needed to enrich the data processing. The enrichment is also built on a flatter attribute based model. An attribute in an entity like Subject may refer to another entity like a foreign key. This would help in enrichment.

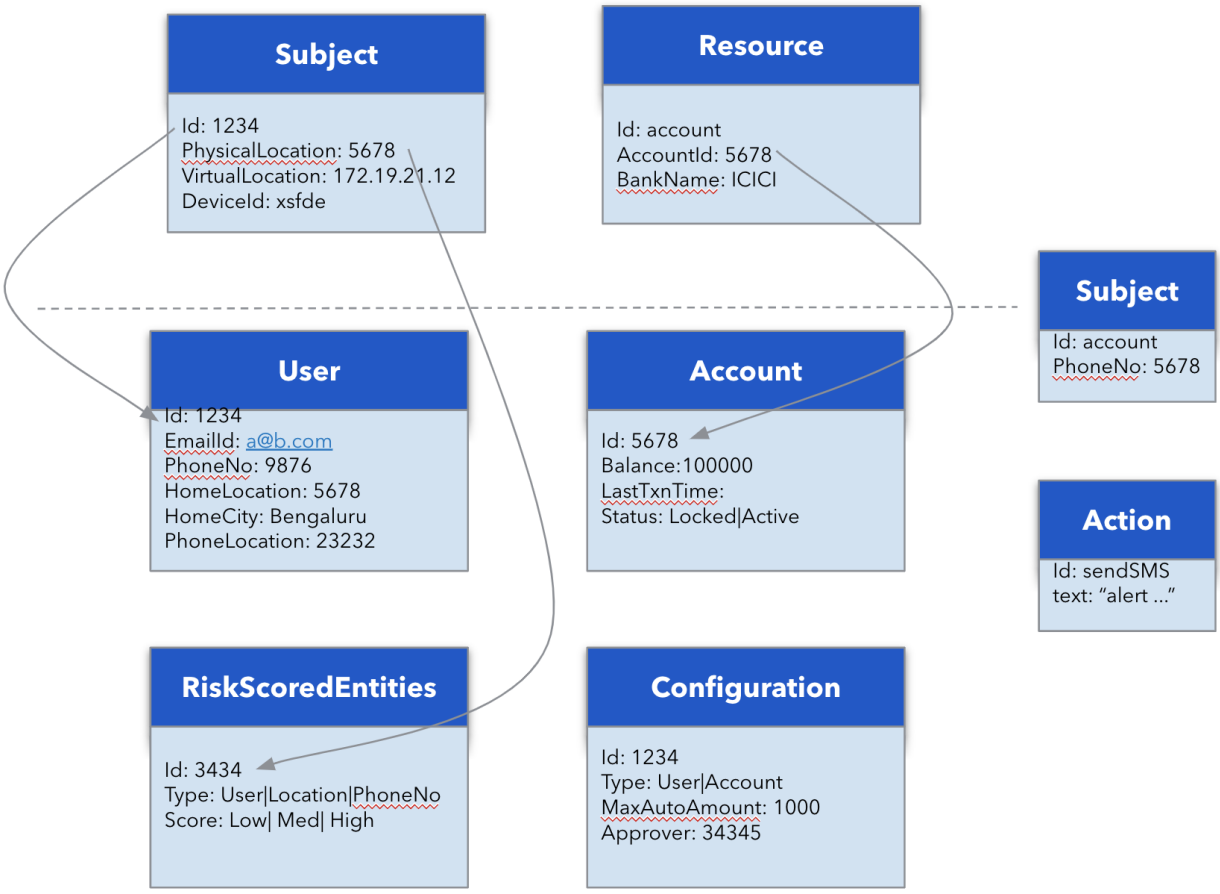


Figure 16. Integration through referencing

To represent substructure in EAV, one incorporates a special EAV table where the value column contains references to other entities in the system (i.e., foreign key values into the objects table). To get all the information on a given object requires a recursive traversal of the metadata, followed by a recursive traversal of the data that stops when every attribute retrieved is simple (atomic). Recursive traversal is necessary whether details of an individual class are represented in conventional or EAV form. We use a similar approach for reference data based enrichment, with the end result that end result is again a flattened entity ready to be given to ML.

Data processing phases

Diagram below shows a linear view of how an event may be processed. A user action causes a trigger which generates an access event. This data is enriched by matching it reference data or even offline trained ML algorithms. The enrichment is guided by enrichment specifications. Once this is done, security aspects may be handled by anonymizing, masking or dropping data that is security sensitive. Not all the attributes may be needed as features and we may sometimes like to derive attributes that can become features. For example, the date of a transaction may not be a good signal compared to time since the last transaction. Once features are ready they need to be transformed from string to numerical (including categorical) so that they can be fed to ML algorithms. We call this step Tensorify. Then the list of algorithms to be applied and their hyper tuned parameters are specified. Different ML algorithms will generate fraud events. These need to be post processed before they are ready for delivery. Post processing may involve steps like formatting the message for human consumption, enriching the event with the address of the recipient and action based on configuration of the user. Once this is done, the events may be delivered based on delivery specifications.

Multiple pipelines may work on the same event and may key streams based on different aspects. For example at times the keying might be based on resource and at times on subject.

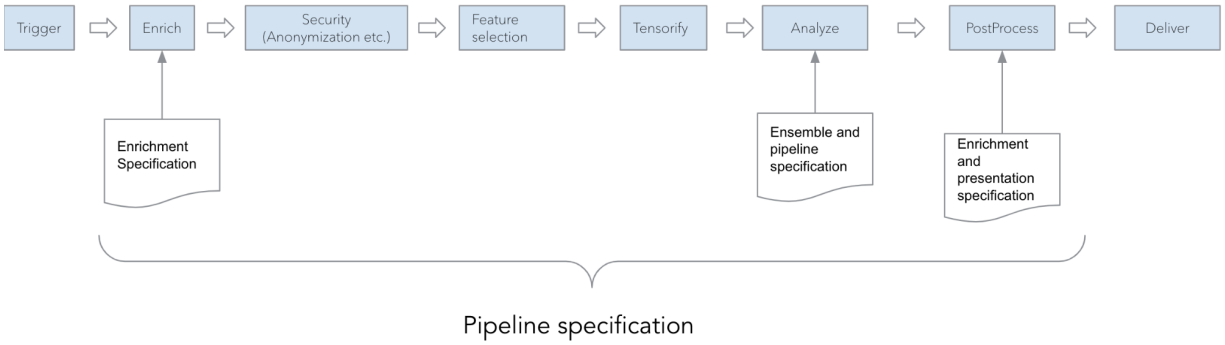


Figure 17. Stream processing pipeline stages

Solution architecture

The generalized stream processing architecture is specified below. Simulator running multiple scenarios is in the collection tier. It generates events that Fraud detector configured in Analysis tier processes and generates alerts or action to be taken for delivery tier.

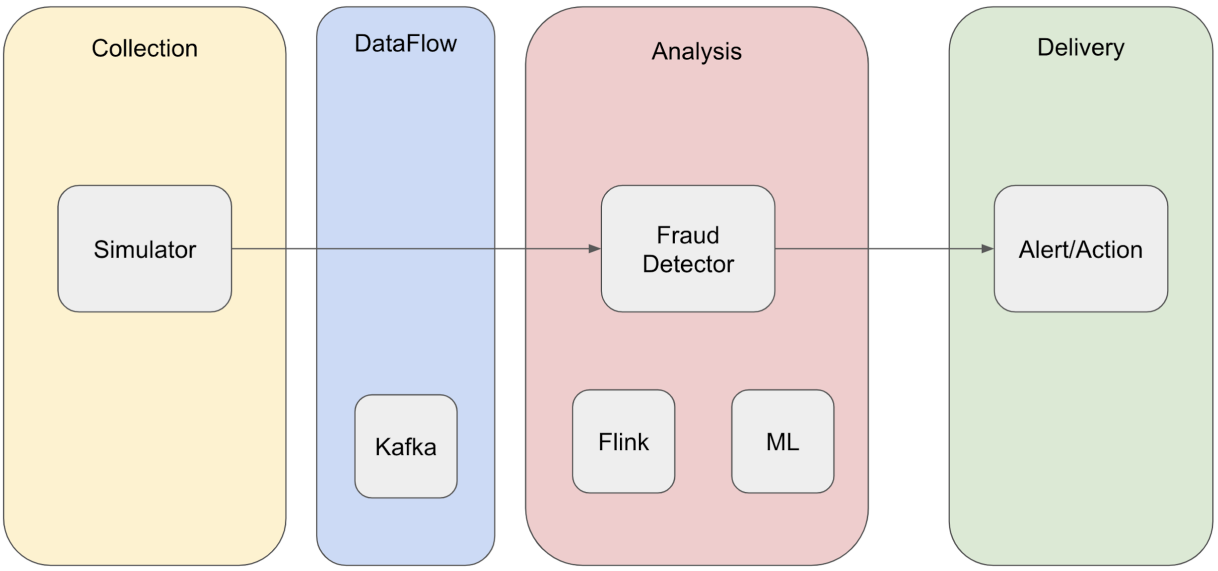


Figure 18. High level general stream processing architecture

The below diagram shows the modularity aspect where multiple collectors, analysis components and delivery components would work together to solve a real world scenario.

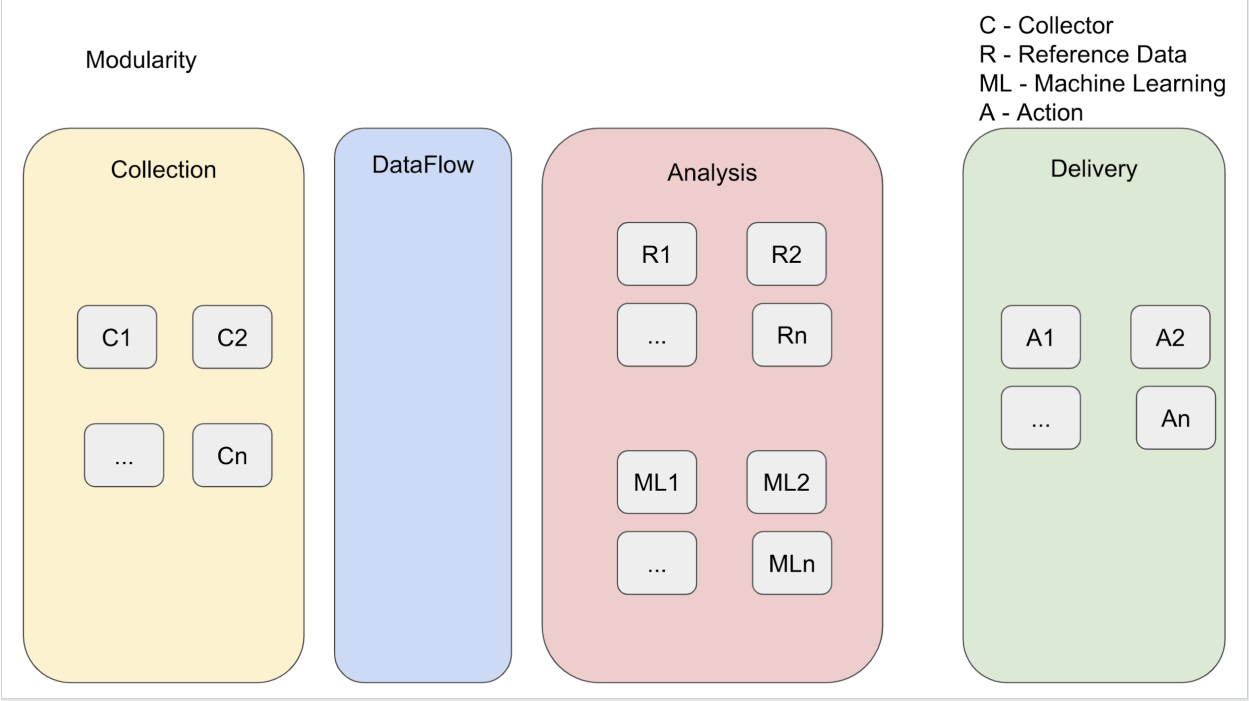


Figure 19. Pluggable general stream processing architecture

A detailed architecture diagram shows resource servers generating access events which are fed to the access event pipeline. These are then picked by multiple ML algorithms processed with help of multiple reference data and then fraud events are generated to the delivery pipeline. From there multiple delivery mechanisms can work in parallel to perform action needed. As part of the human in loop a feedback event may be generated which has already been classified. This feedback can go through the same path and train the ML algorithm in real time in online mode.

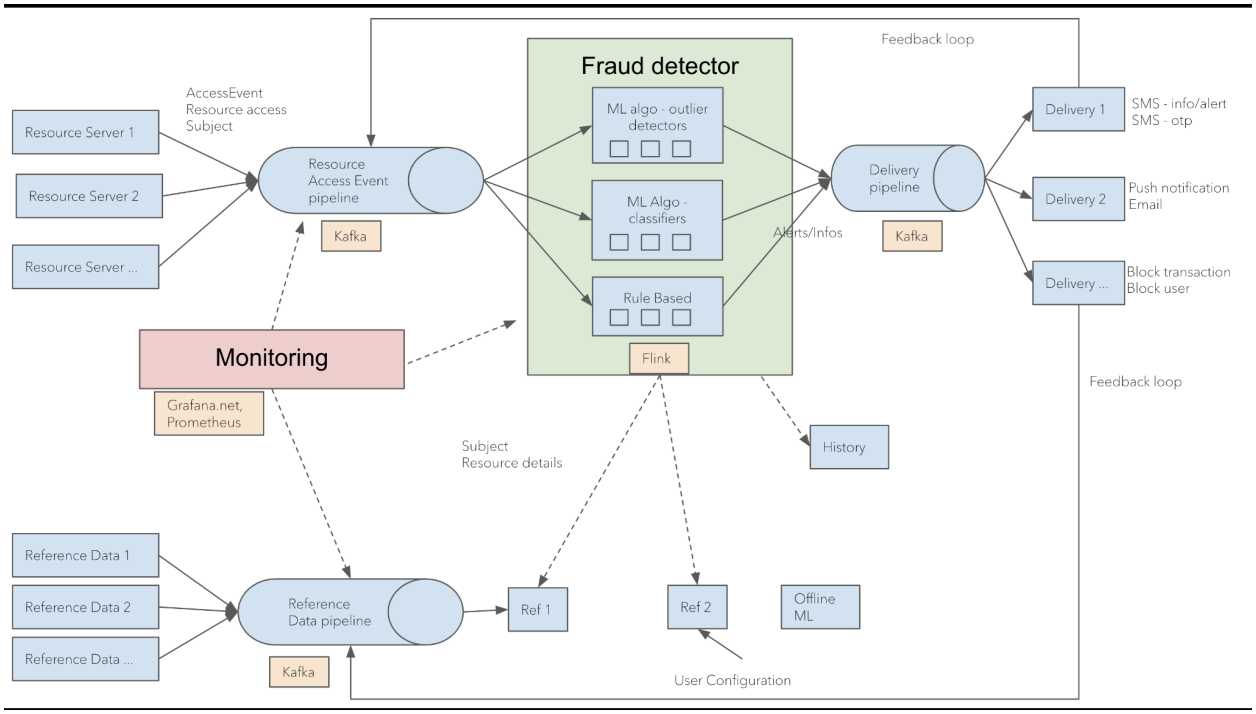


Figure 20. Architecture of Fraud Detection System

Modularity

Ability to integrate components on the fly, Modularity is a key part of the architecture. Modularity is supported by system by,

- A common interface for communication
- Decoupled integration using Kafka
- A GitOps model described later that allows system configuration

Scale

The system scales because of,

1. Using software like Kafka and Flink that can scale
2. With the capability to partition data in Kafka and also by keying different streams in Flink

The scale in the system is in terms of,

- Number of resources
- Number of subjects
- Number of algorithms in each processing
- History to be managed
- Reference data to be managed

By allowing stream processing to be arbitrarily defined over a swimlane specified using selectors the throughput each swimlane has to handle can be tuned. In extreme cases we can define a swimlane for each resource or each subject. This will be too many swimlanes but each swimlane will be limited to throughput of messages to be processed.

Queuing systems like Kafka could scale by partitioning by resource id or if use case demands, subject id.

Similarly history can be defined by a key attribute. Later we may provide capability to give composite key attributes including multiple attributes to arbitrarily tune the granularity. The size of history could also be specified per key attribute. This model can create lots of small histories, which are easier to process.

Multiple ML algorithms can be run in parallel so that a large number of them, although pretty fast, would not impact the overall processing time. This will help BellKor like system where 107 ML algorithms were part of the ensemble.

Reference data can grow to be large, but we want it to be manageable to be loaded into the in-memory DB of the processor. It can be later backed as a chained cache to a bigger far cache.

Realtime

By using Flink and Kafka, we provide for realtime processing. This system is built on streaming architecture. We use Kappa architecture to provide realtime capability. This is different from systems that take data, store it first in hadoop or a data store like Vertica and then process it. Further using streaming principles of windowing and acting on unbounded data we have the system designed for realtime.

Technology choice

Kafka is being used as a data flow tier. All events land up at Kafka and stream processing pipelines consume from it. Kaka provides for scale, performance and resilience.

Flink is used for streaming. Flink provides true streaming with very low latency and very high scale.

MOA - Massive Online Analysis is the most popular open source framework for data stream mining, with a very active growing community. Related to the WEKA project, MOA is also written in Java, while scaling to more demanding problems. [6]

RocksDB
RocksDB is a persistent key-value store for fast storage environments. It is an embeddable persistent key-value store for fast storage. This is used as an implementation for reference data.

Spring - The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform. We use Spring expression language for defining Domain Specific Languages (DSLs).

Prometheus is a monitoring tool used with  Grafana to show pretty monitoring dashboards.

Jython is an implementation of the Python programming language designed to run on the Java platform.

Predictive Model Markup Language is a platform agnostic language to export models post training. PMML-Java evaluator is integrated to use offline trained models that are exported into PMML support.

Chapter 5

Machine Learning

The brain of the system is a Machine learning system that detects fraud. Here is an excerpt from the architecture diagram provided earlier.

Figure 23. Fraud detector ensemble of different types of ML

There are following parts to the ensemble,

1. Outlier detectors - these look for anomalies and are only interested in a window of data. There is no permanent learning for these algorithms. Following outlier detectors are included,
    a. ExactSTORM
    b. ApproxSTORM
    c. MCOD
    d. SimpleCOD
    e. AbstractC
2. Classifiers - these only learn when they are trained. Note that these are online algorithms that train from a stream of data as they are also making decisions. Both things work in parallel.
    a. HoeffdingTree
    b. HoeffdingAdaptiveTree
    c. NaiveBayes
    d. DecisionStump
    e. AdaHoeffdingOptionTree
3. Rule based - sometimes the actors (Subjects) may be in the deny list where they have been already known to cause fraud. Or the customer may configure the system to notify if a transaction was beyond a certain limit or fail a transaction if the limit exceeds or trigger an approval flow.

4. Offline ML - PMML (Predictive Model Markup Language) based java evaluator is part of the system. So models trained in any programming language and exported to PMML can be part of the pipeline.
5. Test ML - in addition a RandomBinaryClassifier is provided for testing purposes.

The combination of outlier detectors and classifiers works as complementary system that,

- Has long term memory in classifiers as they are trained with fraud feedback loop and short term memory with outlier detectors
- Deals with known and unknown situations. Classifiers deal with known situations and flag when they see the signals observed in the past. Outlier detectors, in contrast, work very well with new unknown situations. They are only interested in knowing if something changed in the data stream.

The combination of decisions will be through ensemble options like voting, a veto vote and may be score based addition.

While outlier detectors work best in cold starts and new situations in which ML could not be trained, classifiers learn fraud patterns and work best in persistent fraud attacks. Rule based algorithms also work like classifiers with feedback loop input, they are however very strict with detection.

Note that a feedback loop is provided. So events with known class may be ingested into the pipeline the same way with appropriate attributes to show that these are meant for training the ML.

History

History feature is provided for a look back into events. Events may be identified by a set of attributes. Once a key attribute is used to id the event, history for that id would be kept to configurable size, say 5. A circular buffer is used for that.

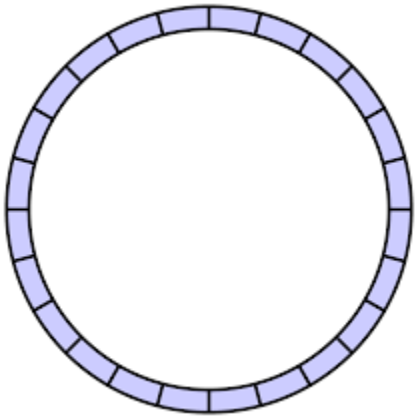


Figure 24. History buffer

A Stream processing is configured with a set of attributes that will act as keys. Also a size is provided for each of them. An event may be part of multiple histories depending on how may keys match. Each new event goes through following phases,

1. Enrich with history - the event is matched to find if a history has been maintained. Then the event is compared with the history for attributes for which enrichment is needed. The comparison can be of different types ranging from subtraction to distance computation

etc. Once comparison is done the result may be put into a configured attribute. This attribute may then be input to ML and it would be used to detect anomalies.
2. Save history - after the event is processed it may be included in the history. For this again it will be matched with histories. And added to history, knocking off the oldest data if the size of the queue gets too full.

History is different from the window in streaming. Windows are time or count based but look at events coming in. History is much more durable and is based on the identity of the event defined by keys. So objects coming in the same window/stream can belong to different histories. One way to think is to take count based windows, over a long period of time, coupled with streams partitioned by key attributes and ability to make operations over the whole window.

Reference Data

Reference data is the non-transactional data that is used by the system. That is, this is data that is not checked for fraud. But it is used to detect fraud in other transactional data. This data can be used by rule based ML, or as configuration or as enrichments to the incoming event. The incoming event is matched with reference data to enrich it before it is passed to ML.

To assist this a database called RocksDB is used to demonstrate. The reference data consists of entities with a flat set of attributes. There is no fixed set of attributes, so it is a heterogeneous store. Each entity has a type and id. This is used by enrichment configuration.

When an incoming event matches the id of the configured entity type in enrichment configuration, the attributes of that entity are taken and used to enrich the event. So, if Subject id was matched with entity type user with identifier id, then attributes of user are added to Subject attributes.

This simple model allows us to plugin any kind of data using a very simple approach.

The reference data doesn't always have to be seeded offline. There is a reference data pipeline and it could be used to seed the data. This may be useful in scenarios like,

- there is a feedback loop where we need to update the deny user with a new fraudster detected
- there is the location of the owner of the account that also has been continuously tracked using say a mobile app. And when a transaction like ATM withdrawal happens then it could be matched to the location of the mobile to further ascertain that the right owner is transacting. Similarly we may need to track live data while transactions are happening.

Fraud Detection Stream Processing Language

The system has been made to be completely configurable and specifiable using a domain specific language "Fraud Detection Stream Processing" language.

This configures a fraud detection stream processing. Following can be done,

- enable monitoring
- add name
- add selectors
- add enrichments
- add any security processing
- add swimlane

- add history
- add history enrichment
- add feature
- add ML algorithms
- add pre-delivery enrichments

Here is a sample instance,

```
fraudDetectionStreamProcessing

    //The attributes based on which events will be selected. Rest of the events
will be ignored.
    .addSelector("resource", "tenant", "Random")

    //Attributes for swimlane. Each swimlanes analyzes events in it, learns from it
and maintains state.
    .addSwimlane("resource", "id")

    //The attributes that will be taken as features for ML. Rest will be ignored.
    .addML("RandomBinaryClassifier");
```

Figure 25. Simplest stream processing example

The above sample creates a stream processing that randomly calls events as fraud or not. Also it only looks at events that has a tenant attribute of resource set to Random. This shows that multiple such stream processing can be defined where they may look at exclusive data sets or subset of streams.
Here is a more detailed example,

```
fraudDetectionStreamProcessing

    //The attributes based on which events will be selected. Rest of the events will be ignored.
    //.addSelector("resource", "tenant", "A1Bank")

    //Enable monitoring through prometheus
    //TODO: also specify which events to be monitored.
    .setMonitoring(true)

    //Add enrichments. Reference data corresponding to this will be used to enhance the event.
    .addEnrichment("subject", "id", "denyUsers", "id")
    .addEnrichment("subject", "id", "user", "id")
    .addEnrichment("subject", "id", "userConfiguration", "id")
    .addEnrichment("subject", "IPAddress", "denyIPAddress", "id")
    .addEnrichment("resource", "accountId", "account", "id")

    //Security handling
    .addSecurityProcessing("subject", "social-security-number", "remove")
    .addSecurityProcessing("subject", "employeeId", "anonymize")
    .addSecurityProcessing("subject", "credit-card", "mask-till-last-four")

    //Attributes for swimlane. Each swimlanes analyzes events in it, learns from it and maintains state.
    .addSwimlane("resource", "id")
    .addSwimlane("resource", "accountId")

    //Attributes for keeping history. These keys decide the identity of the event. Same event may have two histories based on two keys.
    .addHistory("resource", "id", "5")
    .addHistory("subject", "id", "5")

    //How to use history with current event to enrich. Simplest approach is to diff with last state.
    .addHistoryEnrichment("environment", "location", "diff", "environment", "location-diff")
    .addHistoryEnrichment("environment", "time", "diff", "environment", "time-diff")
    .addHistoryEnrichment("resource", "amount", "diff", "resource", "amount-diff")

    //The attributes that will be taken as features for ML. Rest will be ignored.
    .addFeature("resource", "id", "categorical")
    .addFeature("action","id", "categorical")
    .addFeature("resource","amount", "double")
    .addFeature("environment","location-diff", "double")
    .addFeature("resource","amount-diff", "double")
    .addFeature("environment","location-diff", "double")
    .addFeature("resource","amount-diff", "double")

    //The attributes that will be taken as features for ML. Rest will be ignored.
    .addML("HoeffdingTree")
    .addML("HoeffdingAdaptiveTree")
    .addML("NaiveBayes")
    .addML("DecisionStump")
    .addML("AdaHoeffdingOptionTree")
    .addML("ExactSTORM")
    .addML("ApproxSTORM")
    .addML("MCOD")
    .addML("SimpleCOD")
    .addML("AbstractC")

    //Add post procesing enrichments. Reference data corresponding to this will be used to enhance the event.
    .addPreDeliveryEnrichment("subject", "id", "denyUsers", "id")
    .addPreDeliveryEnrichment("subject", "id", "user", "id");
```

Figure 26. Moderately complex stream processing example

Chapter 6

MASSES

Multi Actor Scenario Stochastic Event Simulation (MASSES) is designed to support multiple actors working in a scenario independently as per their own clock. Multiple of these scenarios can run in parallel as separate processes, all generating events into the same event pipeline. A Scenario is like a play where multiple actors are given their script and they are playing their part independent of each other. The actors have capability to make random state transitions instead of a deterministic script. Actors have the option to wait between states. Each state transition results in an event being fired, which is the goal of the simulation. This event is consumed by the Fraud Detection System.

Simulator

A DSL (Domain specific language) is defined to specify a simulation. This language is a JavaScript based language that uses a vocabulary and model that the executor simulator provides. The simulator connects to Kafka as a producer and generates events.

The simulation defines actors with each having a probabilistic Markov chain based state transition spec. As instances of actors are created and run they go through these state transitions based on probability and delays configured. At each state transition an access event is generated and sent to the access event pipeline. To make simulation realistic a notion of pool of subjects and resources are available for actors to impersonate. So an actor instance may behave like a user Alice (Subject) with a particular bank account (Resource), fixed for its lifetime. And it does operation on it continuously.

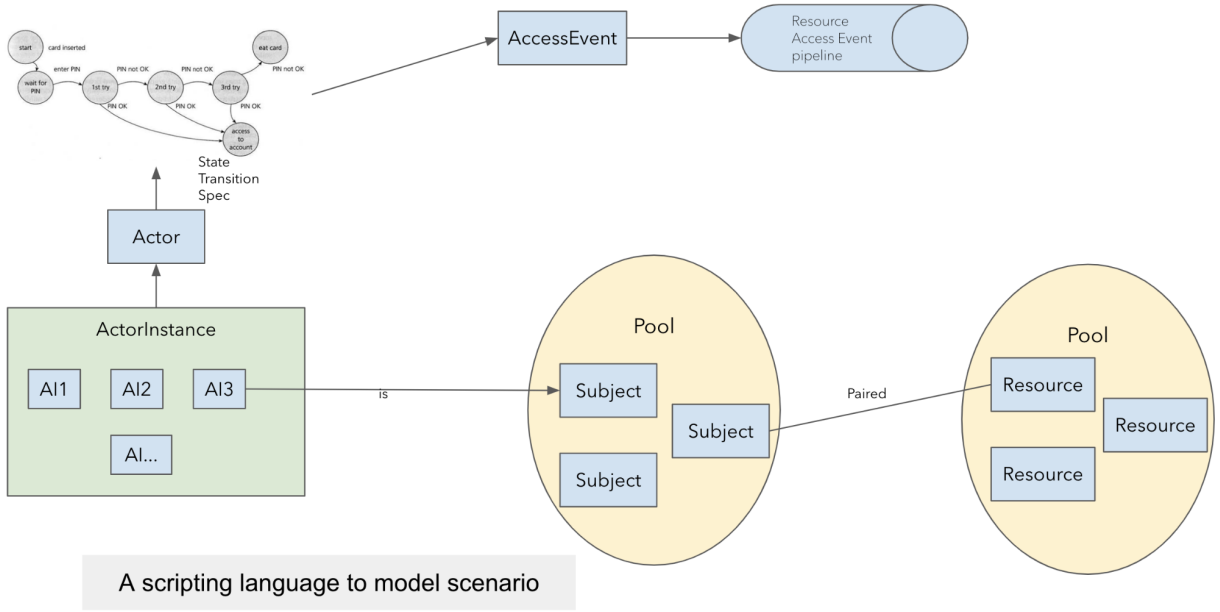


Figure 27. Simulator architecture

Simulation Language

A simulation language is a domain specific language (DSL) provided by the MASSES system. It is used to specify a simulation. Here is a simple example,

```
logger.info("Running the Random simulation")

var importIt = new JavaImporter(java.lang.String,java.util,java.io,java.time,com.stream.simulation,com.stream.fraud.model);
with (importIt) {
  var simulator = new Simulator();

  var state = simulator.defineState("SingleState")
        .addResource("id", "account")
        .addResource("tenant", "Random")
        .addResource("amount", "100")
        .addAction("id", "atmWithdrawal");

  var markovChain = [
        new StateTransition("SingleState", "SingleState", 1 , 1000)
  ];

  var resourceTemplate = new Resource();
  resourceTemplate.setAttribute("accountId","123");
  resourceTemplate.setAttribute("simulation","random");
  var resourcePool = simulator.definePool(1, resourceTemplate);
  var subjectTemplate = new Subject();
  subjectTemplate.setAttribute("id","123");
  subjectTemplate.setAttribute("simulation","random");
  var subjectPool = simulator.definePool(1, subjectTemplate);
  var subjectResourcePool = simulator.pair(resourcePool,subjectPool);
  simulator.defineActor("123").stateTransition("SingleState", markovChain);

  simulator.startActors("123", subjectResourcePool, 100, 100);
  simulator.sleepInMilliSecs(300000);
  simulator.pauseAndPrompt("End the simulation?");
  simulator.end()
}
result=simulator;
logger.info("Done - simulation script")
```

Figure 28. An example simulation script

We use a Simulator as a platform to configure the simulation. We define states and then a markov chain based on it. Markov chain specifies the state transitions (from and to state), probability with which state transition happens and time delay after which the transition would happen. The system would run it perpetually if the chain is configured that way.

Once this is done we define resource and subject templates. These are used to create pools of resources and subjects. A pairing is done in this specific use case where a subject and resource are matched, like a bank account owner and bank account. We also provide the size of the pool.

We then start simulation by defining actors. We define what percentage of the pool will be used and what percentage of it will be active at any point of time. This translates to the actual count of subjects and resources being used in the simulation. This represents reality where we have a large set of users of a system, but only a fraction of them are active at any point.

The simulation can go on for a duration, or wait for user to end it through a response on prompt or end it the way one wants in the script.

Operation

To allow for this whole system to be constructed on the fly we would use the GitOps operation model. A developer may specify all aspects of the system as a git repo. Interacting with git repo is the main interface for this system deployed in the cloud. Once a developer makes changes, the repo is picked by a build process that validates the input and then provisions/updates the system. A virtual view is shown for a developer. Virtually it would appear to the developer that an instance of the whole system has been done for the developer. However, in reality it configures the system so that specifications in git repo are enforced.

So, the interface to developer is,

- Git repo to make changes to input to the system
- Monitoring virtual view to monitor how the system is doing

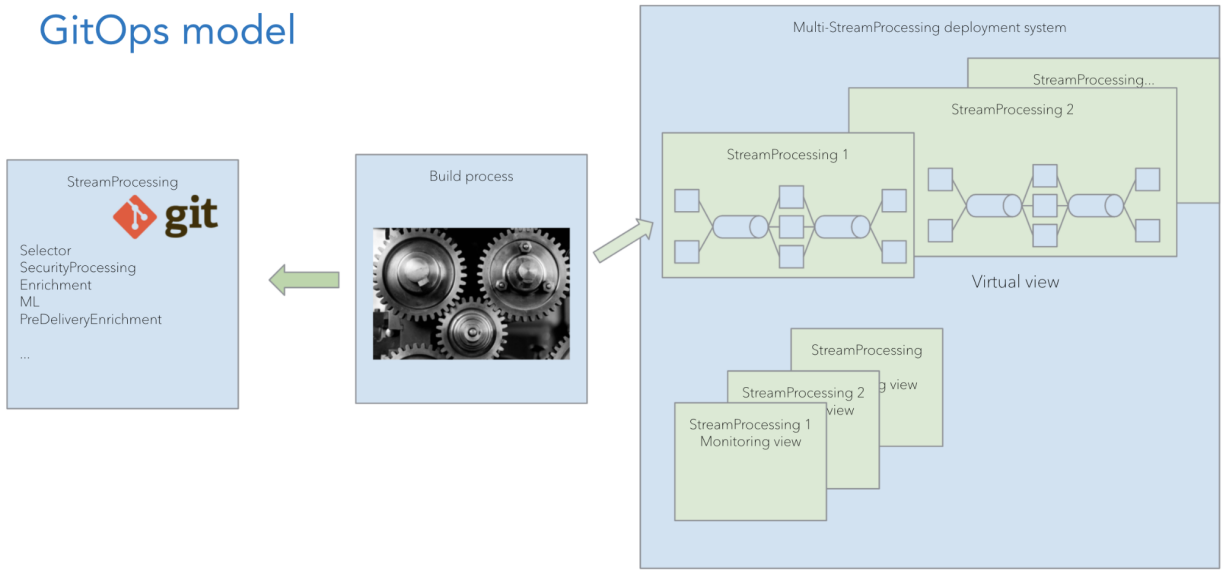


Figure 29. Git ops model for defining new stream processing at runtime

Chapter 7

MASSES Simulations

Please note that demo video mentioned in appendix and code link provided at the same place can be used to get details of these simulations. Following simulations are done to show the capability of the system,

1. A random ML algorithm solution to show a simple pipeline. A Random Binary Classifier is used that without looking at input does a coin toss to decide whether it is fraud. While the system is practically of no use, it has value in making the learning curve smooth. It shows how different parts work and one can see something working end to end.
2. A tenant using a complete pipeline to show all the capabilities. The pipeline shows almost all phases of processing starting from naming the stream processing, selector specification, enrichment, history, ML and pre-delivery enrichment. A tenant named "A1 Bank" is used as an example.
3. A senior citizen scenario showing outlier detection to detect new operations done as fraud. Alice is the good actor whereas Trudy is the bad actor. This is also enhanced to show a feedback loop to train online classifier algorithms which make fraud detection more robust. The classifier learns the actions of Trudy and is easily able to flag them as fraud.
4. A deny user scenario showing rule based fraud detection. Subjects, IP addresses etc. may be put in the deny list. Also, a configuration by the end user might trigger the flow, for example crossing of a certain transaction amount. This bypasses the ML and uses rules to make decisions.
5. A history scenario where ability to maintain history and compare with history to make ML more robust. This allows a sudden change in an attribute compared to previous is passed as a signal instead of the absolute value of the attribute.

These simulations are based on the scenarios mentioned in Chapter 3.

These simulations are picked up for,

- ease of understanding the system by a new developer
- demonstrating the capabilities of the system
- creating a set of tests to validate functional and non-functional aspects of the system

These simulations would be made part of continuous testing of the system so that enhancements are quality assured. These simulations were used to drive the requirements of the system much like behaviour driven development, we used scenario driven development.

Specification by Example

Behavior Driven Development (BDD) is an approach that consists of defining the behavior of a feature through examples. These examples are defined before the development starts and are used as acceptance criteria. They are also part of the definition of done. That is they are executable requirements.

This is also referred to as Specification by Example (SBE). It is an emerging practice for creating software based on realistic examples, bridging the communication gap between business stakeholders and the dev teams building the software. A key aspect of specification by example is creating a single source of truth about required changes from all perspectives. Specification by

example is also known as example-driven development, executable requirements, acceptance test–driven development (ATDD or A-TDD), or Test-Driven Requirements (TDR).

The simulations here are an enhancement in the sense that it does not describe a situation, but carries out a simulation over a time period with tests firing events in a decoupled way from the system. This is evident from the way simulations are specified in contrast to SBE, which has Given, When and Then structure. Simulations on the other hand have states and state transitions at the core.

Conclusions / Recommendations

PAIR architecture fulfills the hypothesis laid out for an effective fraud detection system and meets the goals of a modular, scalable and realtime fraud system.

By providing a unified representation of inputs, multiple data sources can be plugged into the system. The stream processing can be agnostic of the number of data sources. This is true for both transactional and reference data.

This is also aided by reference data that can be updated in online and offline ways. Ability to easily use diverse data (Information fusion) to support ML will greatly increase chances of detecting fraud.

Multiple ML models instead of one makes it not only better but makes it work in a variety of situations. They are better as a committee of experts is better over an expert. Further a combination of outlier detectors and classifiers works both in cold start situations or Day 0 situations where no training has been given and also in situations where sufficient training may be provided. We used online algorithms so that this training can be provided simultaneously as the models are being used. This makes the system more responsive. We have the option to use offline trained models as well. We have rule based models that work when the fraudsters are known or the user wants to configure thresholds that should be treated as anomalies. This makes the system much more practical for real world use.

A MASSES system was built to demonstrate a simulation language that can make it easy to simulate scenarios and test the system. Fraud training data is too small usually (and highly skewed) and becomes ineffective as hackers evolve their techniques like adversarial fraud or new systems are put in place. It is also easy to understand scenarios and systems through play of scenarios using this simulation. Feedback is also included in simulation.

We have demonstrated that multiple stream processing systems can coexist. There is a language designed to specify such stream processing. This includes ability to specify events processing is interested in, security processing, enrichment, ability to maintain and use history, select features, ML algorithms, pre-delivery processing and delivery. This allows development organizations to scale by owning their own processing specification and also allowing data to be pooled together and with shared-usage by systems.

The recommendation is to use this as a real system in a use case, learn from it and evolve the system.

Future work

Following were out of scope for the current work and may be taken up as natural extensions.

The hypothesis needs to be validated by measurement. That is the benefit of bringing multiple ML models and more data in terms of improvement in fraud detection should be measured. For this one would need real data, a baseline model and then compare a well-defined measure of accuracy between the baseline model and this system.

This system should be hooked up with real delivery mechanisms. This system should be tested for multi-tenancy where multiple git repo configs can be provided and all this works together.

The system can be tuned by using data formats like ORC, Avro, Parquet etc. to work efficiently at scale. Further to improve real world performance needs, edge, hybrid and global deployments may be considered. The histories can be persisted so that system restarts do not erase history.

Support for providing metrics in A/B experiments will help where multiple stream processing may be used for the same data and measure of success be computed automatically.

The simulation language can be extended to also assert the results to verify if the system was able to catch tests. These simulations then can become part of continuous tests.

Bibliography / References

1. A submission to ACM Computing Surveys (CSUR):
Ane Blázquez-García, ‘A review on outlier/anomaly detection in time series data’, arXiv:2002.04236 [cs.LG], https://arxiv.org/abs/2002.04236, Feb 2020.

2. A Journal Paper:
Alexander Diadiushkin, Kurt Sandkuhl and Alexander Maiatin , ‘Fraud Detection in Payments Transactions: Overview of Existing Approaches and Usage for Instant Payments’, in Complex Systems Informatics and Modeling Quarterly (CSIMQ), September/October 2019, pp 72–88.

3. A Journal Paper:
Luan Tran, Liyue Fan and Cyrus Shahabi, ‘Distance-based Outlier Detection in Data Streams’, Proceedings of the VLDB Endowment, Vol. 9, No. 12, Aug. 2016.

4. A Conference Paper:
Jian Zhang, Richard Wydrowski, Zeqiang Wang, Sai Sankalp Arrabolu, Keiji Kanazawa, Lech Gudalewicz, Hong Gao, Roman Batoukov and Souren Aghajanyan, ‘Möbius: Online Anomaly Detection and Diagnosis’, in KDD’2018, August, 2018.

5. A MSc Dissertation:
A.C.G. Viera, ‘Parallel Continuous Monitoring of Distance-Based Outliers in Streaming Data’, MSc. Dissertation, ARISTOTLE UNIVERSITY OF THESSALONIKI, Feb. 2018.

6. A Journal Paper:
Albert Bifet, Geoff Holmes, Richard Kirkby and Bernhard Pfahringer, ‘MOA: Massive Online Analysis’, Journal of Machine Learning Research, 2010, pp 1601-1604.

7. An Article:
Yehuda Koren, ‘The BellKor Solution to the Netflix Grand Prize’, https://netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf , Aug. 2009.

8. A Journal Paper:
Alon Halevy, Peter Norvig, and Fernando Pereira, ‘The Unreasonable Effectiveness of Data’, IEEE INTELLIGENT SYSTEMS, 2009.

9. An article:
https://humansintheloop.org/what-is-a-human-in-the-loop/

Appendices

The entire code with setup instructions are present here - https://github.com/fraud-detection-system/Fraud-Detection-System

Note that src/main/resources directory contains stream processing specifications and also simulation scenarios as .sim files.

The video of the demo is here - https://youtu.be/qvt7pgCnSHU.

List of Publications/Conference Presentations, if any.