A note on anti-fraud mechanisms

Vivek Pandey Robin
Dec 2022

This document explains some of the essentials of our anti-fraud mechanisms. It is a primer to better follow various discussions around anti-fraud efforts. A bit long document, but the read would be worth it if you don't work in the anti-fraud area.

Anti-fraud architecture at Simpl

First let's see where all in our workflow do various fraud filters block the charge calls or the users. There are four dominant such places:

1. At charge call: TS calls AFS at charge call. Some ~10 filters run at the charge call time. If the charge call fails to pass any of these filters then that charge call is blocked.

2. Asynchronously on approval call: Some fraud filters run on approval calls, in async manner. They run immediately after the approval call. If any of the fraud filter fails for the approval call, then the user is transactionally blocked, and the next approval call or charge call for that user will fail.

3. Various batch jobs: Besides the above two, there are various jobs which run at various frequencies: e.g. some jobs run hourly and detect cluster fraud. Some jobs are run in adhoc manner: e.g. VVB, VB blocks.

4. Blocks by fraud monitoring team: Fraud monitoring team manually eyeballs new user transactions (and repeat user transactions in future), calls suspicious users and blocks them if they are satisfied the user is a fraud.

Above AFS actions result in blocking the users in two ways:

1. For some users, approval calls don't succeed. This happens due to points 2, 3, and 4 above.
2. For some users, approval call succeed but then charge call fails. This happens due to point 1 above.

How we think about performance of filters/models

Precision and Recall

For any algorithm to detect fraud, we consider two metrics related to the algorithm: its *precision* and *recall.*

Let's understand these with an example.

Let's say, in a given cycle, 1M users transact, and 10K of them are fraud. One of our anti-fraud models M, declares 20K of the users to be fraud, of which 6K are actually fraud, and 14K are not fraud.

In this case, precision of M = 6K/20K = 30%. So, precision tells us: if M declares a user to be fraud, how likely is the user to be actually a fraud.
In this case, recall of M = 6K/10K = 60%. So, recall tells us: What fraction of overall frauds is M able to catch.

Notes on precision

Next, let's discuss what precision/recall a good model should have.

Ideally, we would like our anti-fraud apparatus to have 100% precision and 100% recall: i.e. all users that we declare to be fraud are actually fraud and we declare all actual fraud users to be fraud users. However, we are currently not close to this ultimate goal.

Since MDR is ~2% it follows that as long as the precision of an anti-fraud algorithm is >2%, it is monetarily beneficial for Simpl.

[To understand this: Assume precision of model is 2%. If the model blocks N users then 0.98 of them are good users, so we lose TPV of 0.98N users from whom we would have earned MDR of 0.98N * 0.02 = ~0.02N. However we also avoid TPV of 0.02N fraudulent users, which we would have lost completely. Thus, at 2% precision, the model breaks even and at higher precisions, bad TPV saved will be larger than good MDR lost]

However, we don't want to be so conservative. At very low precision we hurt growth significantly, so there is another way we look at the question of what is a good precision.

Consider new user delinquency which is ~20% currently. This is achieved with current approval algorithm (RF V4) working and current suite of anti-fraud algorithms. So, if a new anti-fraud algorithm, applied on new users, has a precision of >20%, it will improve overall new user delinquencies, and if it has a precision of <20% it will worsen new user delinquency. Similarly for repeat users, where current delinquency is ~2%, we need algorithms with precision > 2% to improve overall repeat user delinquency.

Model precision, TPV growth, and delinquency

We can draw a simple mathematical relation between a model’s precision and its impact on growth and delinquency.

Let:

T = Total TPV
d = Delinquency %
D = Dlq amount = $d \times T$
r = Net revenue % (MDR – other costs like infra, promotions, etc.)
R = Net revenue = $r \times T$

We always want our $R > D$. If that's not the case, then we can use a set of rules and models to help us achieve it. However, that will impact some growth. **How much?**

Let:
t = % TPV blocked by models
p = precision of the models
Then:

T` = New TPV = $T - t \times T$
D` = New delinquency = $D - p \times t \times T$
R` = New net revenue = $r \times T$`

We can solve for t (impact on growth).
If our constraint is: $R` > D`$

Then: $t > (d - r)/(p - r)$

Note that t is independent of TPV 🙂

Example:
if
T = $500,000,000
d = 5%
D = $d \times T$= $25,000,000
r = 1.5%
R = $r \times T$ = $7,500,000
p = 40%
Then
t **= 9.09%**

Table below shows how the model impacts the new revenue and delinquency based on how much we are willing to sacrifice.

t	T`	D`	R`
4.000%	$480,000,000.00	$17,000,000.00	$7,200,000.00
5.000%	$475,000,000.00	$15,000,000.00	$7,125,000.00
6.000%	$470,000,000.00	$13,000,000.00	$7,050,000.00

7.000%	$465,000,000.00	$11,000,000.00	$6,975,000.00
8.000%	$460,000,000.00	$9,000,000.00	$6,900,000.00
9.000%	$455,000,000.00	$7,000,000.00	$6,825,000.00
9.100%	$454,500,000.00	$6,800,000.00	$6,817,500.00
9.120%	$454,400,000.00	$6,760,000.00	$6,816,000.00
9.130%	$454,350,000.00	$6,740,000.00	$6,815,250.00
9.140%	$454,300,000.00	$6,720,000.00	$6,814,500.00
9.143%	$454,285,000.00	$6,714,000.00	$6,814,275.00
9.144%	$454,280,000.00	$6,712,000.00	$6,814,200.00
9.145%	$454,275,000.00	$6,710,000.00	$6,814,125.00
9.147%	$454,265,000.00	$6,706,000.00	$6,813,975.00
9.148%	$454,260,000.00	$6,704,000.00	$6,813,900.00
9.150%	$454,250,000.00	$6,700,000.00	$6,813,750.00
9.300%	$453,500,000.00	$6,400,000.00	$6,802,500.00
9.500%	$452,500,000.00	$6,000,000.00	$6,787,500.00
10.000%	$450,000,000.00	$5,000,000.00	$6,750,000.00
11.000%	$445,000,000.00	$3,000,000.00	$6,675,000.00
12.000%	$440,000,000.00	$1,000,000.00	$6,600,000.00

Things are a bit more complicated in reality though:

1. We don't have one, but a suite of rules and models, so we need to use the combined precision
2. Some rules act post-transaction, so for even high precision at account level, actual savings will be less
3. Precision decreases as we try to block more users. In the above example, precision will not remain 40% across all values of t.

Precision that we aim for

We currently informally have following precision goals for our algorithms:

User type	Rough Precision goal
New	50%
Repeat	20%

These goals are guided by two factors:
- Precision should be high so that user experience / growth is compromised as less as possible
- There should be a non-trivial recall at the goal precision. If recall is, say like 1%, then we are blocking just 1% overall fraud users, so the system would not have a meaningful contribution in reducing delinquency.

Why do I call it "Rough"? We don't always operate at 50%/20% precision. We take a call on a case-by-case basis, and usually operate at a precision of >50% and >20% respectively.

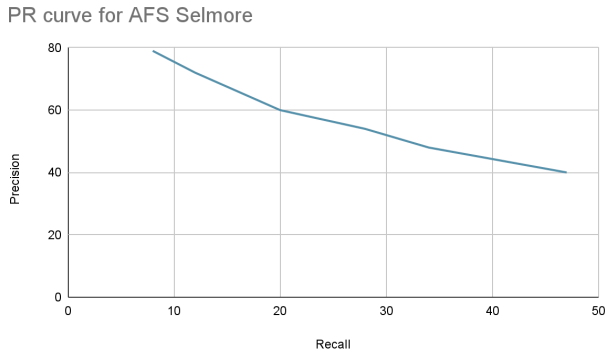
Let's take a moment to understand that non-trivial work is involved in getting to a 50% precision model for new users or 20% precision for repeat users. For repeat users for instance, delinquency is ~2%. So, if you pick 100 users randomly then only 2 of them would go delinquent. However, we have algorithms that are discriminative enough that if they pick 100 users as fraudulent, 20+ of them would be fraudulent.

Both on the new user and repeat user side, we have some models which have 60%+ precision. While we would like precision to be even higher, currently, with higher precisions we have very low recalls.

AFS selmore: a case study

For most of the models that we develop, we can decide where in the PR curve (precision recall curve we want to be). For instance, for the recently developed AFS Selmore (Sequence model for users who have already done 1 transaction), this document presents the analysis.

Below is the precision recall curve when blocking the users on approval call events for 1st cycle transacting users. We can have a precision close to 80%, at a low recall of 8% on one extreme, or a precision of just 40% with a high recall of ~47%. In this case we have decided to operate at a precision of 48% (with recall of 34%)



How much blocking do we do currently?

Let's find answers to following three questions:

- What impact do anti-fraud measures have on approval rates?
- What impact do anti-fraud measures have on the charge call success rate?
- What impact do anti-fraud measures have on delinquency?

Here are the answers to some of the above three questions (the details follow)

- Impact of anti-fraud measures on approval rates
    - ~4pp for new users
    - We don't currently know how much dip in approval rates for repeat users is caused by anti fraud measures
- We block ~0.7% of all transactions, and roughly 5% of new user transactions
- Impact of anti-fraud measures on delinquency
    - Our charge call blocks reduce delinquency by ~1.7pp for new users [This number is back of envelope calculation: real number can be from half to double of this figure.]
    - We currently don't have the delinquency impact of approval call filters.

Details of these calculations are provided in appendices 1 and 2.

Improvements to be made in anti fraud system

In the past quarters we have made significant algorithmic improvements in our anti-fraud sytem. Some of these are:

- Unblocking several users blocked by CTUF
- Developing and deploying SLAM model
- Developing and deploying Selmore model
- Several reactive rules to limit fraud (e.g. rate limit filters, address similarity filters, customer id based cluster identification)
- VVB, VB blocks

In coming quarters, while we continue to improve our algorithms, we need to work on following infrastructure items:

- A more thorough understanding of how much impact our anti-fraud algos are having on approval rates and delinquencies. We have rough ideas around this, but we don't have rigorous measurements around this. A rigorous measurement will help us track impact caused by our algorithms
- We need to track precision and recall of various anti-fraud filters rigorously so that we can tune the filters if their precision / recall goes below a threshold.
- We already track volumes of blocks on a weekly basis. And we have some measure (though not thorough) of precision/recall. We will start tracking precision/recall of filters on a per cycle basis.

Appendix 1: Some calculation details

Approval failing because of blocks:

(Data derived from here)
Till first transaction

Out of ~4.4M distinct users sending us approval calls daily, ~3.3M users fail approval calls (across all merchants). Out of these ~3.3M approval failures, ~175K approvals were because the users had been blocked due to some anti-fraud effort.

Thus, the overall approval rate would increase by 175K / 4.4M = 4% if there were no anti-fraud effort. A 4% dip in approval rates is the penalty we pay to control fraud in the system.

After first transaction
Out of ~660K distinct users sending us approval calls daily, ~176K of them fail. Out of these ~176K approval failures, ~135K are because the users had been blocked due to some reason.

However, we currently don't know how many of these blocks were because of anti-fraud systems and how many were because of other reasons (like D60 blocks)

Charge call failures due to AFS filters

Currently AFS blocks ~2500 transactions per day involving ~1400 users. (Now a days, forced app onboarding blocks account for ~1400 of these blocks involving ~650 users)

Further, there are ~200K unique users doing transactions daily. So, we block around 1.4K/200K = 0.7% of the users on a daily basis.

Most of AFS filters work on new users (rather than repeat users) and assuming 10% of daily transactions to be new user transactions, roughly 5% of new users transactions are being blocked.

Fraud filter impact on new user delinquency

We currently block around ~2500 transactions involving ~1400 users a day. Most of these blocks are for new user transactions. Let's assume we block 1500 transactions a day. (Fraudsters would be trying many times, so if we were not blocking them, the number of transactions in a day would be closer to 1400 rather than 2500.)

Assume Rs 1000 transaction size. So, TPV blocked by anti-fraud systems = Rs 1500 * 1000 * 15 in a cycle = Rs 2.25M in a cycle. Assume 60% precision, so bad TPV blocked = Rs 1.35M.

Total new user TPV in a cycle (Take Dec 2nd cycle for reference) = 76M.

Appendix 2: Relevant queries

Query 1: Number of distinct charge failure events and distinct charge failed users

```
SELECT event_date,
        COUNT(DISTINCT(payload:event_id)) AS distinct_events,
        COUNT(DISTINCT(payload:user.phone_number)) AS distinct_users
FROM vikas.all_events_json
WHERE event_name = 'TransactionChargedFailedEvent'
AND event_date >= '2022-12-20' AND event_date <= '2022-12-28'
GROUP BY event_date
ORDER BY event_date
```

Result 1

#	event_date	distinct_events	distinct_users
1	2022-12-20	2940	1416
2	2022-12-21	3160	1445
3	2022-12-22	2940	1390
4	2022-12-23	3353	1462
5	2022-12-24	3579	1558

6	2022-12-25	3924	1733
7	2022-12-26	3282	1385
8	2022-12-27	3162	1428
9	2022-12-28	3268	1425

Query 2:

```
SELECT
  event_date,
  COUNT(DISTINCT(payload:event_id)) AS event_count,
  COUNT(DISTINCT(payload:user.phone_number)) AS user_count
FROM vikas.all_events_json
WHERE event_date >= '2022-12-20' AND
        event_date <= '2022-12-28' AND
        payload:product.name = 'Simpl Pay Later' AND
        event_name = 'TransactionChargedEvent'
GROUP BY event_date
ORDER BY event_date
```

Result 2:

#	event_date	event_count	user_count
1	2022-12-20	280997	211034
2	2022-12-21	283320	213890
3	2022-12-22	278699	211012
4	2022-12-23	292093	216391
5	2022-12-24	295920	219831
6	2022-12-25	303383	228083
7	2022-12-26	243682	187579
8	2022-12-27	246063	189930
9	2022-12-28	252260	194755

Query 3:

```
SELECT event_date,
        COUNT(DISTINCT(payload:event_id)) AS distinct_events,
        COUNT(DISTINCT(payload:user.phone_number)) AS distinct_users
FROM vikas.all_events_json
WHERE event_name = 'TransactionChargeFailedEvent'
AND event_date >= '2022-12-20' AND event_date <= '2022-12-28'
AND payload:errors[0] = 'afs:approve-transaction:FORCE_APP_ONBOARDING_FILTER'
GROUP BY event_date
ORDER BY event_date
```

Result 3:

#	event_date	distinct_events	distinct_users
1	2022-12-20	1325	602
2	2022-12-21	1447	647
3	2022-12-22	1380	622
4	2022-12-23	1465	664
5	2022-12-24	1652	703
6	2022-12-25	1869	808
7	2022-12-26	1576	636
8	2022-12-27	1495	681
9	2022-12-28	1578	671