Exploring Anomalies in Authentication Logs with Autoencoders

Hayden Beadles and Jericho Cain, Adobe Inc.
Last updated April 2024

Introduction

Can we use ML to detect cyber security events at Scale?

Problem Statement

- Cyber security events difficult to detect at scale with non-restrictive lookback windows.
- Too much data!
    - ~30k Adobe Actors.
    - ~1.5 million logins per day.
- How to extract the most probable events of interest.
- How to understand Adobe Employee’s Login behavior.

The Plan

- Use ML to detect and understand anomalies.
- Apply rules to anomalies for specific use cases.
- Return prioritized list of anomalies for human review.
- Explore blending data sources.

What Happens Today

How Most Authentication Software Monitors for Anomalies

~1.5 million Logins

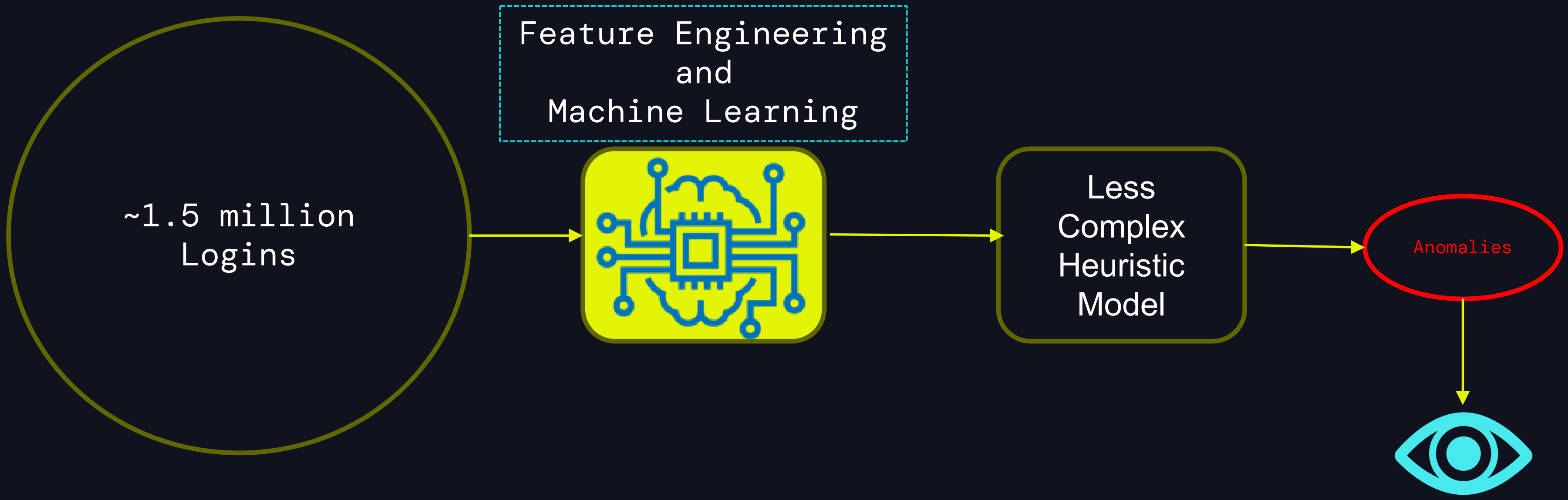
Heuristic Model

- Last 20-50 logins
- Can be increased at cost of compute time
- Hard to capture complex behavior
- Lots of rules, hard to maintain
- Limited to a single data source

Anomalies

Can Machine Learning Help?

Use a model as a filter to eliminate most of the rules.



Machine Learning Challenges

Feature Engineering

Establish User Baselines

- Where do they live?
- What do they use?
- What devices do they have?
- What IP Addresses do they login to?

Encode Violations

- Latitude / Longitude encoding
- How to establish accurate baselines?
- Pyspark / MLFlow to create encodings.

Send them to the model

- Training / Validation Workflow.
- Inference Workflow.
- Model Store.

Feature Engineering

Encoding Multi-dimensional data

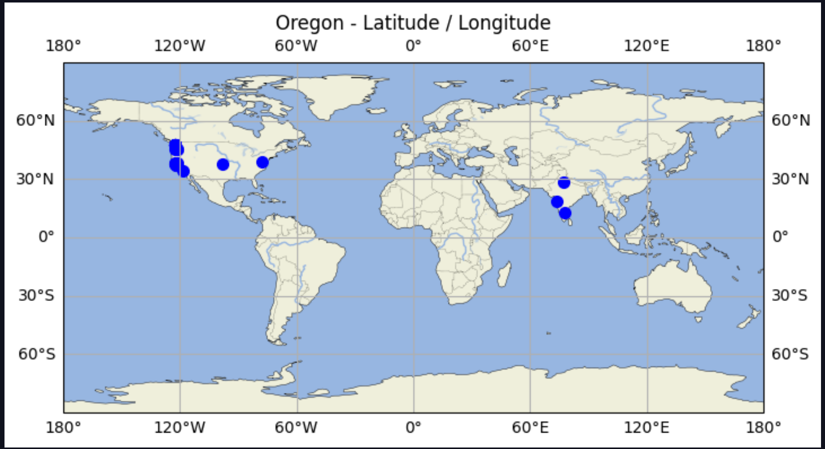
- Break down multi-dimensional features to one dimension
- Need to clean up noise
- Embedding layer may occur too far downstream
- Z-order curve, hashing are good options

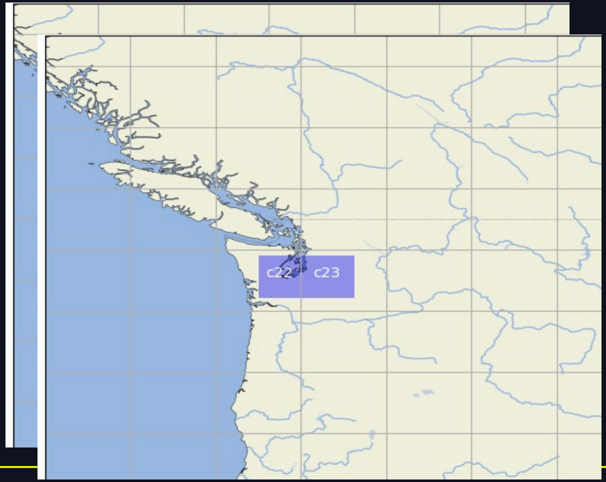
```
import zCurve
Import pyspark.sql.functions as fn

@fn.udf(returnType=DoubleType())
def process_zcurve(input_arr):
    int_process = [int(x) for x in input_arr]
    curve = zCurve.interlace(*int_process, dims=2,
    bits_per_dim=8)
    return float(curve)

df = spark.sql(f"SELECT * FROM {input_table}")
vec_assembler =
    VectorAssembler(inputCols=['device_os_encode',
    'device_browser_encode'], outputCol="features")

silver_df = vec_assembler.transform(df)
# Apply Zcurve to device features
silver_df = silver_df.withColumn(“zcurve_device”,
    process_zcurve(fn.col(“features”))
```

Feature Engineering - Geohash

Encoding Features – Reducing Noise

Features are High Dimensional	Need to widen a range and assign an area to a user
Latitude / Longitude - Too noisy	Geohashing - 3bit precision





Wilson Score Interval

Creating User Baselines

Logins – Samples of Truth

- We want to estimate the ground truth from the sample.
- Retrieve probabilities of successful logins.
- Use Wilson Score Interval to estimate population confidence intervals.

Applying Wilson Score – 80% CI

- Number of Successful Logins from Y.
  - Where Y is a location, application or device.
- $p(x|Y) = \frac{n_s}{s}$
- $z_a = 1.28\ (or\ 1.96\ etc)$
  - Determines CI window (1.28 is 80% CI)
- $w^-, w^+ = \frac{1}{1+\frac{z_a^2}{n}}\left(p + \frac{z_a^2}{2n} \pm \frac{z_a}{n}\sqrt{4np(1-p)} + z_a^2\right)$

Wilson Score Interval

Code – Implement Baselines

Wilson Scoring	Apply Wilson Score Threshold

Wilson Scoring

```
from pyspark.sql.types import ArrayType, DoubleType

@fn.udf(returnType=ArrayType(DoubleType()))
def wilson_score_interval(login_counts, n):
    if n == 0:
        return [0.0, 0.0]
    else:
        p = login_counts / n
        z = 1.28155156554

    upper_left = p + (math.pow(z, 2) / (2 * n))
    upper_right = z * math.sqrt(((p* (1-p))/n) + (math.pow(z, 2)
    / (4 * math.pow(n, 2))))
    lower = 1 + (math.pow(z, 2) / n)

    w_lower_estimate = (upper_left - upper_right) / lower
    w_upper_estimate = (upper_left + upper_right) / lower

    return [w_lower_estimate, w_upper_estimate]
```

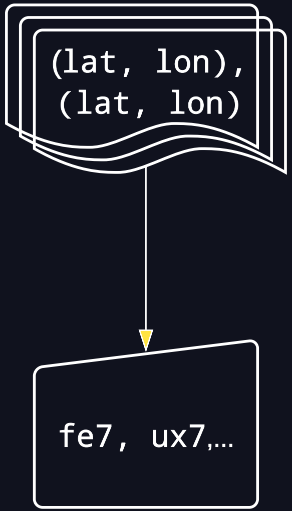
Apply Wilson Score Threshold

```
@fn.udf(returnType=DoubleType())
def wilson_mean(login_counts):
    return sum(login_counts) / len(login_counts)

gold_df =
    df_agg_zcurve_counts.withColumn("wilcox_90%_conf_interval",
    wilson_score_interval(fn.col("device_counts"),
    fn.col("zcurve_count_total")))

gold_df = gold_df.withColumn("mean_frequency",
    wilson_mean(fn.col("wilcox_90%_conf_interval")))

THRESHOLD = .1
final_df = gold_df.filter(f"mean_frequency > {THRESHOLD}")
final_df =
    final_df.groupBy("actor_id").agg(fn.collect_set(fn.col("zcurv
    e_device")).alias("device_set"),
    fn.collect_set(fn.col("device_arr")).alias("device_set_full")
    )
final_df = final_df.withColumn("threshold", fn.lit(THRESHOLD))
```

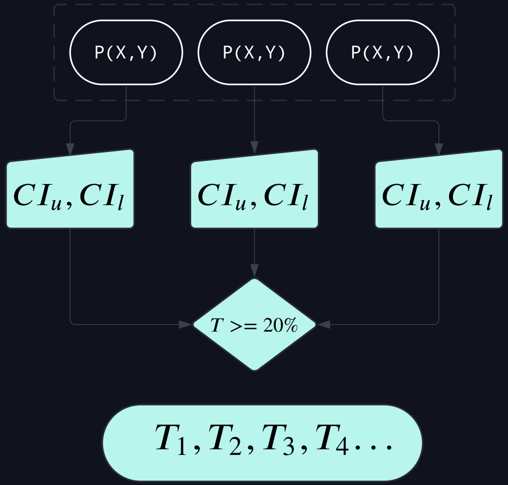
Bringing it together

What it looks like

Apply Hashing

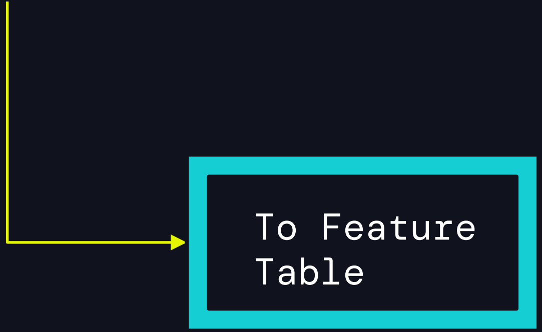


Wilson Score



Baselines

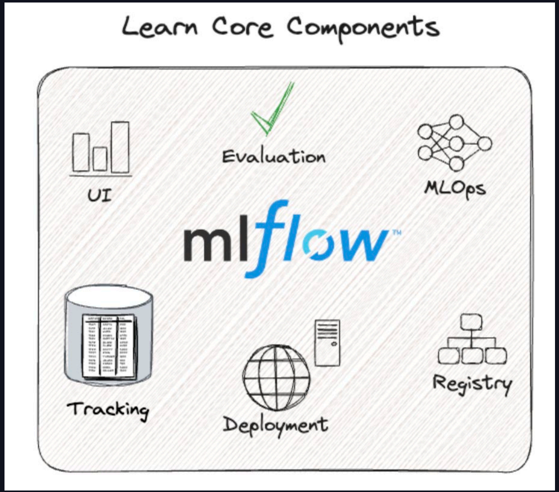
actor_id	geohash_set	threshold
...@adobe.com	["9q9"]	.3
...@adobe.com	["sxf"]	.3
...@adobe.com	["u1x", "dp4"]	.3



ML Challenge - Changing Encodings

How to track Encoding State? When to update?

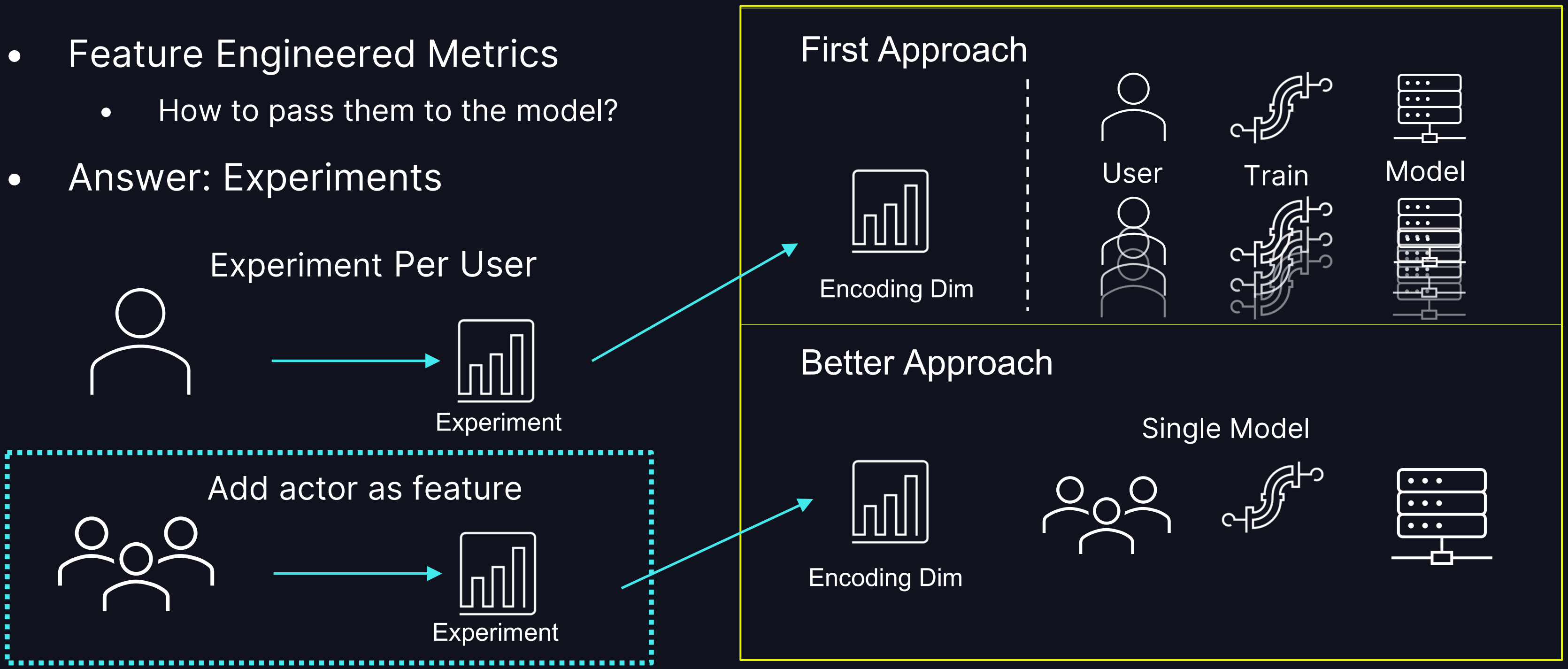
- How do we update encodings for different features?
- Model needs to be trained frequently
- Answer – MLFlow!
    - Pass info via Experiments
    - Metrics



MLFlow - Approaches

Encoding Dimension, testing ideas

- Feature Engineered Metrics
  - How to pass them to the model?
- Answer: Experiments



MLFlow - Code Snippets

Generating Parameters, using them in training

Scala - Generating Parameters	Python - Training

`import org.mlflow.tracking.MlflowContext`
 	
`// Create function to create or replace mlflow run`
`def upsert_run(context: MlflowContext, run_name: String) ={`
`  … // Create or replace mlflow run…`
`}`
`//Create Run, save metrics as json string,`
`val active_run = upsert_run(mlflowContext, cur_date_string)`
`val active_run_id = active_run.getId`
`// Log Encoding Map`
`client.logParam(active_run_id, "encoding_json_map", agg_json_str)`
`client.logParam(active_run_id, "job_new_type", "daily_alpha_job")`
`//Terminate run`
`client.logArtifact(active_run_id, file)`
`client.setTerminated(active_run_id)`
	`client = MlflowClient()`
	`# Retrieve experiment`
	
	`experiment = client.get_experiment_by_name(input_mlflow_path)`
	`experiment_id = experiment.experiment_id`
	`# Pull down run`
	`lookup_run = client.search_runs([experiment_id],`
	`    filter_string=f"attributes.run_name = '{2024-01-22}'")[0]`
	`lookup_run_id = lookup_run.info.run_id`
	`# Download json artifact`
	
	`local_dir = "/tmp/artifact_downloads"`
	`if not os.path.exists(local_dir):`
	`os.mkdir(local_dir)`
	`encoding_map = client.download_artifacts(lookup_run_id,`
	`    "encodings.json", local_dir)`

Databricks - Training Workflow

Journey so far

Hashing | Score Interval | Baselines | MLFlow | Training

Feature Engineering – Key Pre-step

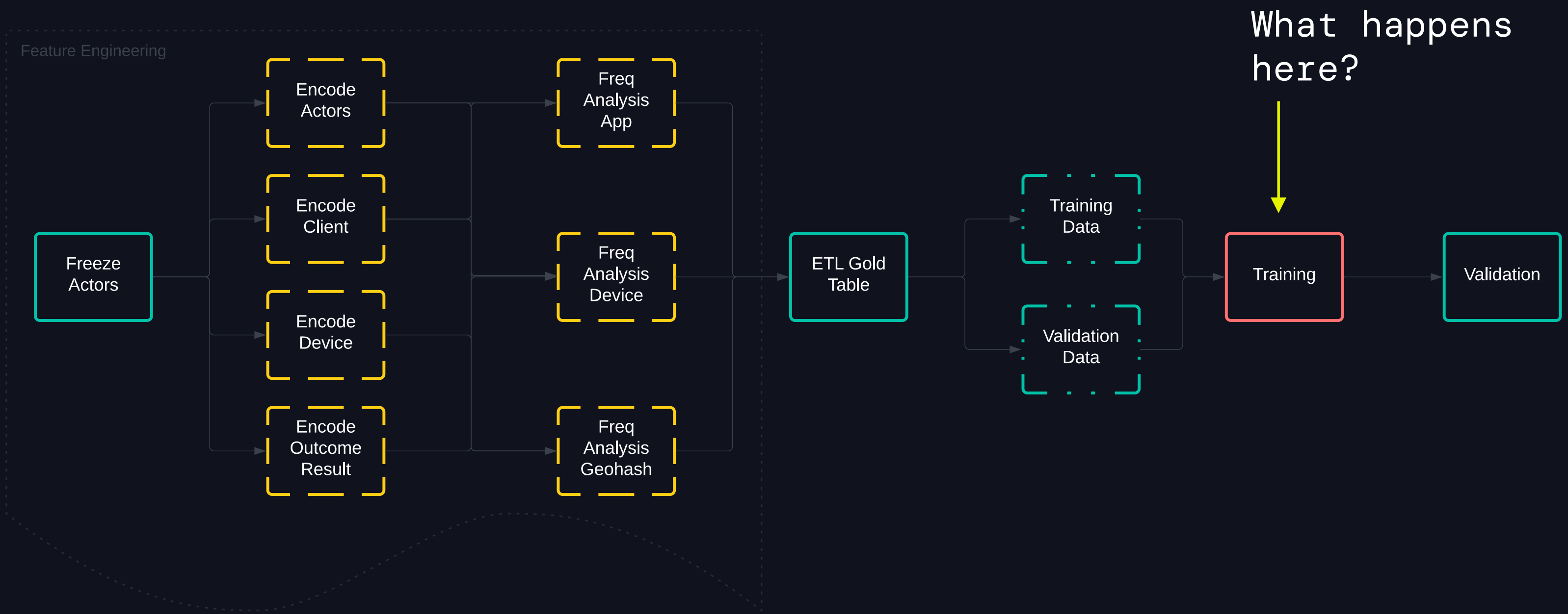
- Baselines help determine AutoEncoder output.
- Allow us to build validation data.
- Store encodings and **version** it.
    - Encodings change if the vocab changes.

Training Param Considerations

- Encoder / Decoder layer sizes?
- Batch Size / Epochs?
- Final Features?
    - actor_id, baseline fields, event date, event hour, login event.

Databricks - Training

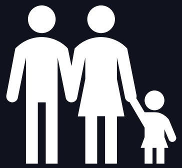
Workflow



Model Considerations

Heuristic Models

- Difficult to maintain for all use cases.
- Lookback restrictions.
- Difficult to scale.
- Difficult to blend with new data sources.
- Difficult to discover new edge cases and anomaly types.



Supervised Models

- Requires Labels.
    - Introduce bias.
- No lookback restrictions.
- Easily blend with new data sources.
- Easy to maintain.

Unsupervised Models

- Does not require labels.
- No lookback restrictions.
- Easily blend with new data sources.
- Easy to maintain.
- Can reveal unexpected patterns.
- Flexible.

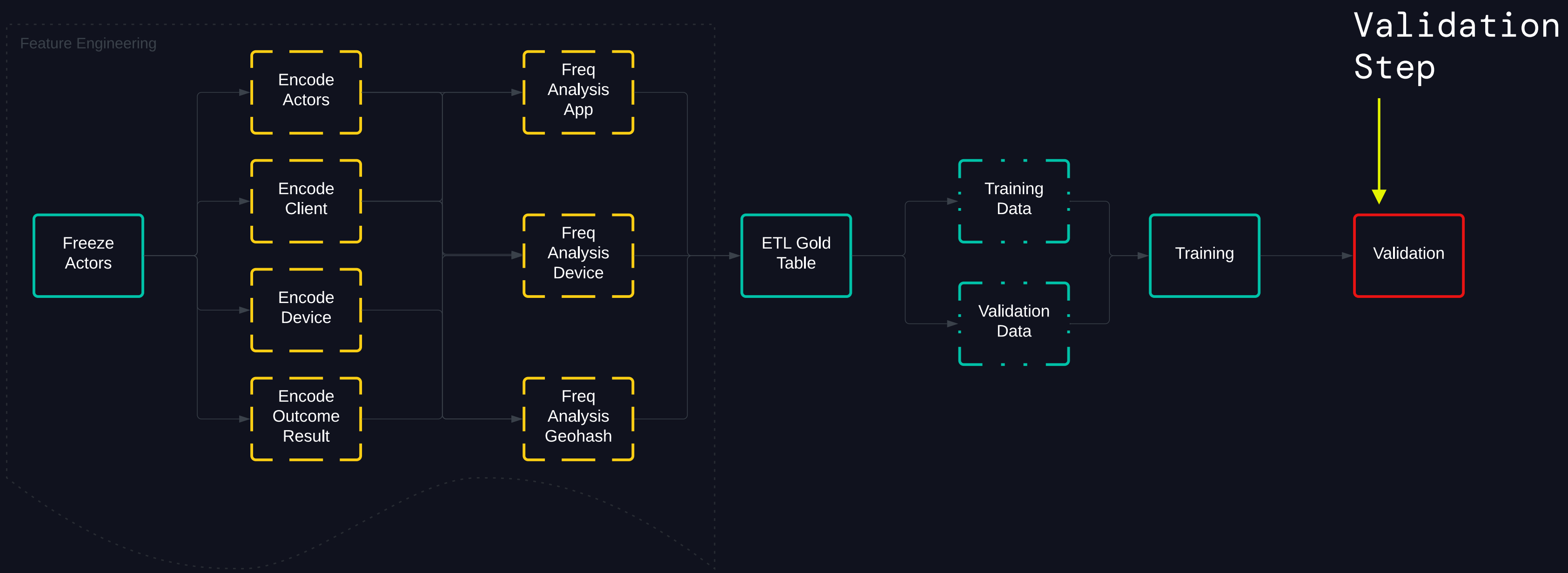
Deep Learning

Autoencoder

- Autoencoder: reconstruct input from latent space representation.
- Feature engineering minimizes historically missed anomalies in training set → small loss.
- Compute average standard deviation across all samples: $\boldsymbol{\sigma}_{train}$.
- Store model and $\boldsymbol{\sigma}_{train}$ in MLFLOW.

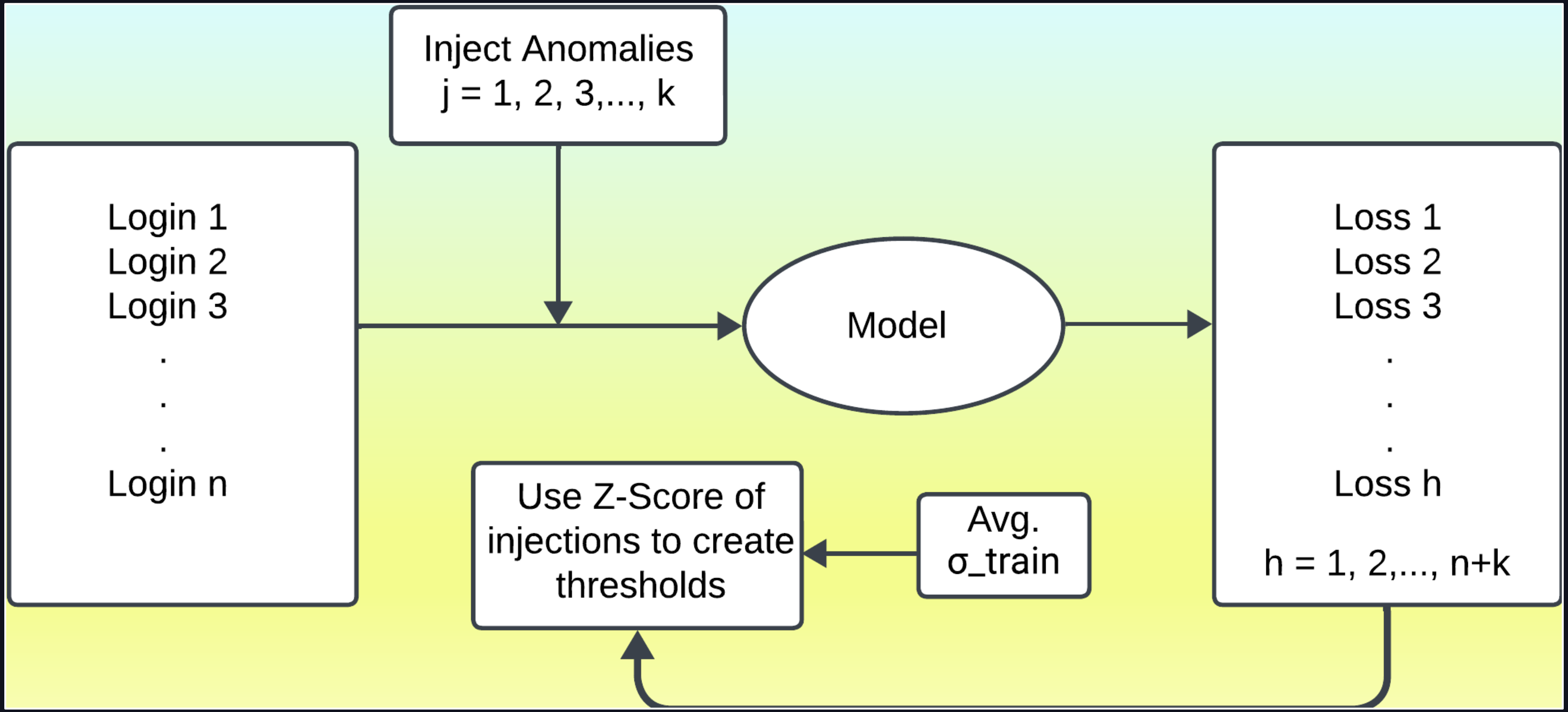
Databricks - Validation

Workflow



Anomaly Detection

Validation Workflow for a Single User



Validation: Use Case of Interest

Impossible Travel

- Injected Anomalies
  - Geohash change, reasonable time of travel.
    - Normal event times.
    - Abnormal event times.
  - Geohash change, unreasonable time of travel.
    - Normal event times.
    - Abnormal event times.
  - No Geohash change.
    - Abnormal event times.
  - Reasonable time of travel?
    - Compute effective travel velocity.
    - Does it exceed speed of commercial passenger jet?

Validation: Use Case of Interest

Impossible Travel

- Considered 100 users
- Test Set
  - 18 injected anomalies
- Recovered 17/18 anomalies
  - All geohash change anomalies recovered
    - high to low loss, highest when event time also abnormal
  - All but 1 non–geohash change anomalies recovered
    - User has broad spectrum of login times

	True Positive	True Negative
Predicted Positive	17	0
Predicted Negative	1	2115

Validation - Scale It Up

Inject anomalies for all 30k users

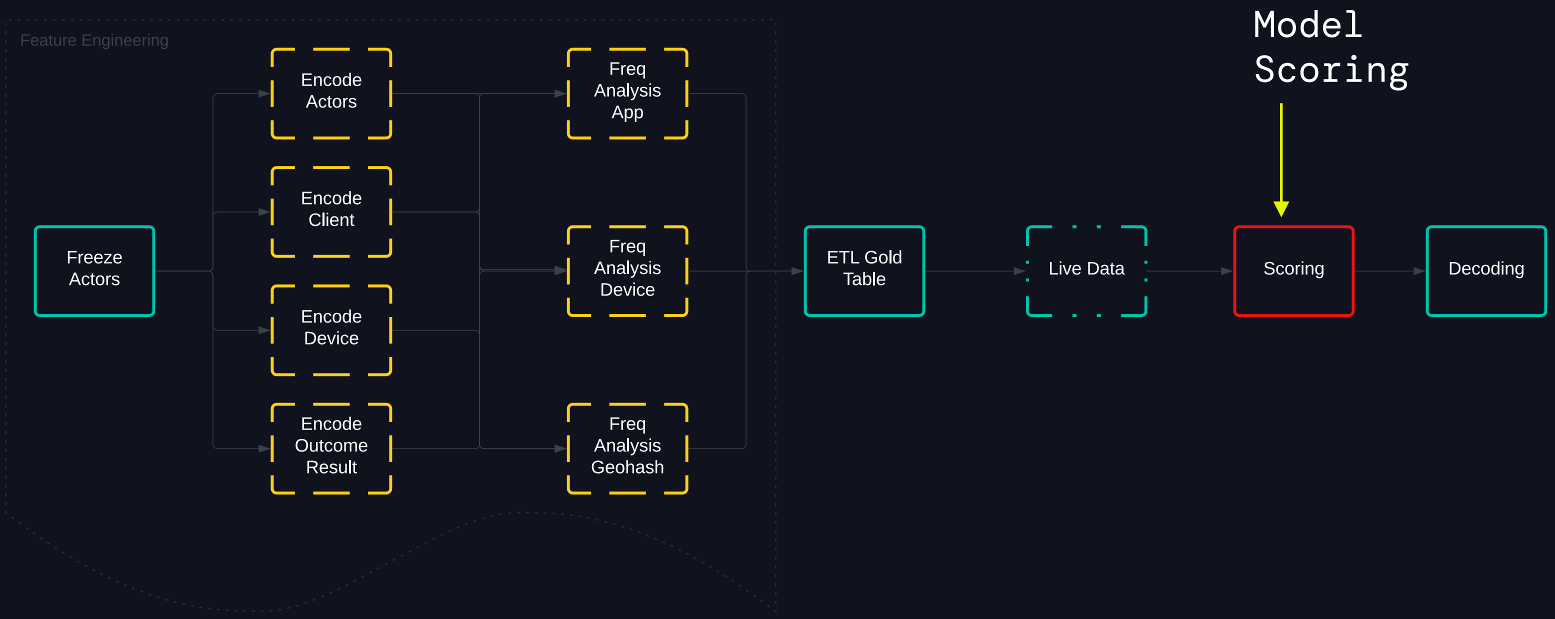
- Compute average standard deviation from training.
- Create validation set for each user.
- User specific mean loss wide distribution – single z-score won't do.
- A z-score exists for each user that captures all injected anomalies.
- This z-score serves as threshold when scoring new incoming data.
- F1 average across all users 98%

User	Data Size	F1	Z-Score	TN	TP
A	613	1	7.1	527	86
B	718	1	6.6	600	118
C	1198	1	7.7	1029	169
D	659	1	7.9	576	83
E	881	1	7.9	746	135
F	1978	1	6.9	1688	290
G	815	1	8	691	124

Table 1: A sample of 7 out of 30k users. Each user data set leads to user specific z-scores to create thresholds capturing each injected anomaly.

Inference

Workflow



PyTorch Modeling

Insights

Training Model

- Test on CPU before CUDA
  - To avoid any CUDA related errors
- Use Parquet tables to store vectors, rather than Delta
- Petastorm Library helps with larger data sets

Testing

- Multiprocessing can be used alongside the GPU
- Be aware of IO bounds when using multiple GPUs
- Use P3 Ec2 instances

Anomalies Found

Types of Anomalies Caught

Session Hijacking

- User uses session on one device in new location.
- Model caught travel over locations in an impossible window.

Impossible Travel

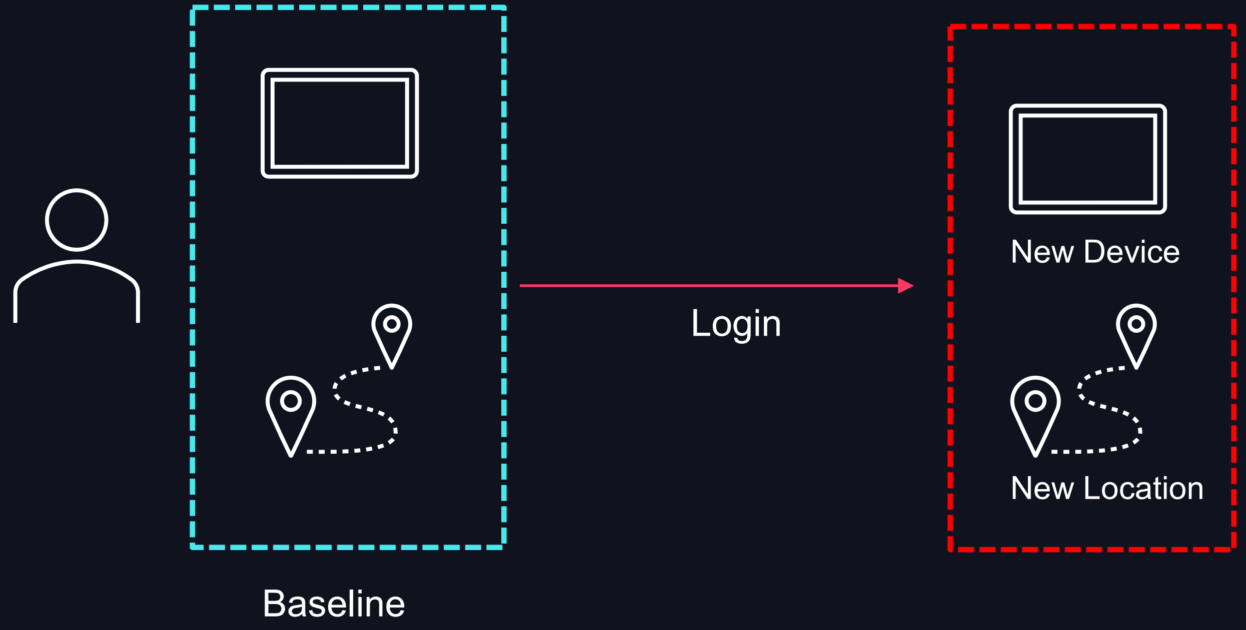
- Someone logs in from X device, location, application in an impossible window of time.
- Can be due to other factors.
  - Third-party VPNs.
  - Registering new device.

Interesting Edge Cases

- Unexplainable things happening at the edge.
  - Odd VPN behaviors.
  - Device registry behaviors.

Edge Cases Found

Anomalies that drive policy change



Explanation: User enrolled a new device. Cloudflare assigned a login ip to the new device when registering, that logged it in a separate location

Edge Cases Found

Anomalies that drive policy change

<u>Explanation:</u> Multiple users registered anomalies to the same external ip or area. Turns out a proxy was being used

Next?

Future Work

- Review raw anomalies prior to use case rules with high loss.
  - Are these malicious?
  - Create anomaly type and edge case type classifications.
    - Could malicious activity look like edge cases – if not can we safely ignore?
  - Can we automatically classify anomaly type without a heuristic?
  - Explore loss associated with other use cases.
- Explore other unsupervised anomaly detection approaches.
- Blend multiple data sources.
  - View user behavior holistically
    - OKTA
    - Entra ID (Azure AD Formerly)

Q/A

Anomalies in Authentication Logs