



# Efficient Synthetic Target Generation for Transaction Data

# Machine learning training

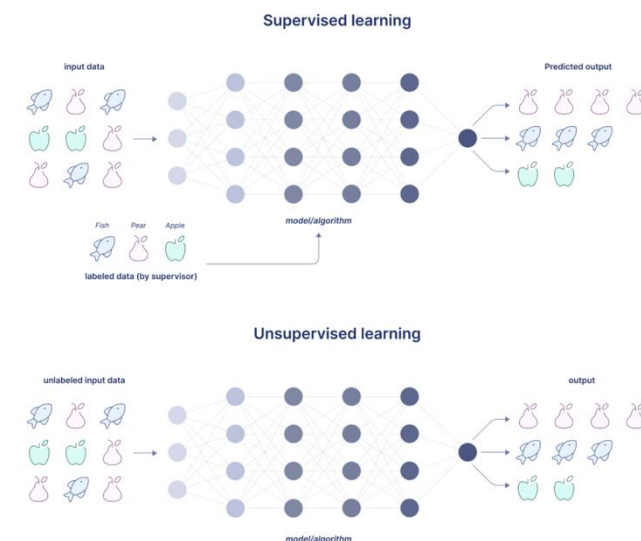
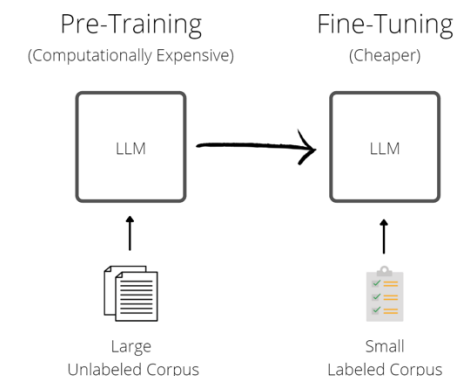
## Supervised and unsupervised learning

**Supervised learning** - training with *labeled* dataset

**Unsupervised learning** - training with *unlabeled* dataset

- Supervised data requires annotations.
- A large amount of data remains unlabeled
- Generative synthetic targets can help solve downstream tasks

Training large models requires a significant amount of data and corresponding labels, which are often unavailable for many datasets



# Financial transaction

## Data structure

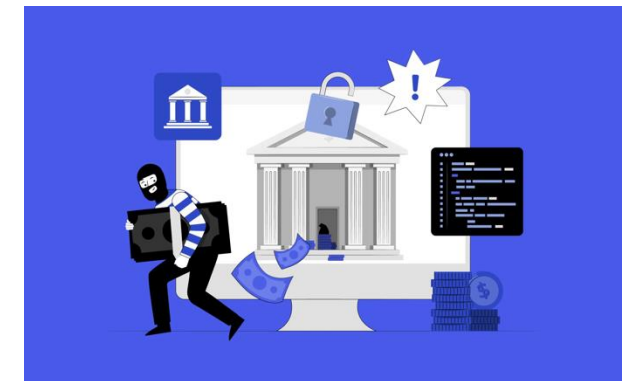
### Features:

- Amount
- Merchant category code (MCC)
- Time of transaction

### Tasks:

- Fraud detection (low target rate)
- Probability of Default (low target rate)

**Synthetic targets are a solution for datasets with limited labeled samples**



Probability Of Default



# Financial transaction

## Synthetic targets

### Synthetic targets examples

- Average transaction amount at drug stores over a week
- Maximum transaction amount at grocery stores during a month
- Minimum transaction amount for rent during a month

	amount	mcc_code	day_of_transaction	target_amount	daily_cumulative_for_target_mcc_5411
0	61.862096	5812	1	0.000000	0.000000
1	112.722485	5942	1	0.000000	0.000000
2	443.690425	5411	2	443.690425	443.690425
3	199.959683	5411	2	199.959683	643.650108
4	72.123675	5411	2	72.123675	715.773784
5	422.050677	5942	2	0.000000	715.773784
6	420.853556	5942	2	0.000000	715.773784
7	211.319643	5942	2	0.000000	715.773784
8	258.552511	5411	3	258.552511	258.552511
9	30.908352	5812	3	0.000000	258.552511

# Synthetic target generation

## Performance bottleneck

### Configurable pipeline:

- Aggregation by different MCC codes
- Time-based feature grouping (transactions per day, month, week)
- Type of aggregation (e.g., sum, mean, max, top 3)
- Requires real-time performance for training ML models (hundreds of users per second)

### Problems:

- Large volume of user transactions (could not be precomputed)
- Need real time performance for training ML model (hundred of users per / sec)
- Should be integrated into Pytorch training pipeline

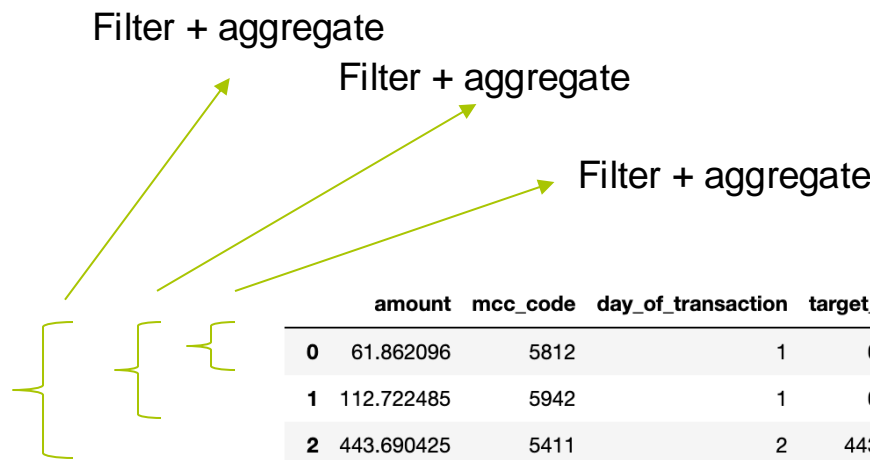
# Synthetic target generation

## Baseline

### Pure Python + NumPy

- Expanding window (Python)
- Filter elements (NumPy)
- Aggregate (NumPy)

**Insufficient speed**



	amount	mcc_code	day_of_transaction	target_amount	daily_cumulative_for_target_mcc_5411
0	61.862096	5812	1	0.000000	0.000000
1	112.722485	5942	1	0.000000	0.000000
2	443.690425	5411	2	443.690425	443.690425
3	199.959683	5411	2	199.959683	643.650108
4	72.123675	5411	2	72.123675	715.773784
5	422.050677	5942	2	0.000000	715.773784
6	420.853556	5942	2	0.000000	715.773784
7	211.319643	5942	2	0.000000	715.773784
8	258.552511	5411	3	258.552511	258.552511
9	30.908352	5812	3	0.000000	258.552511

# Synthetic target generation

## Baseline

### Pure Python + NumPy

- Expanding window (Python)
- Filter elements (NumPy)
- Aggregate (NumPy)

## Insufficient speed

Profiler:

Target generation per user: 0.039s

78.58	~.0(<built in method builtins.exec>)
0.03906	target.py:264(get_target)
5.756e-06	fromnumeric.py:3476(mean)
5.286e-06	_methods.py:110(_mean)

Implementation	Transactions	User per seconds
Python + numpy	4096	25

# Synthetic target generation

## Numba optimization

Pure Python + NumPy + Numba

- Expanding window (Numba + NumPy)
- Filter elements (Numba + NumPy)
- Aggregate (Numba + NumPy)

0.007786	dataset.py:172(__getitem__)
0.004609	target_generator.py:56(generate_targets)
0.004311	target_generator.py:65(generate_target)
0.004159	target_generator.py:122(generate_target_from_intervals)
0.001027	boundary_generator.py:102(generate_time_feature)

Implementation	Transactions	User per seconds
Python + numpy	4096	25
Numba + numpy	4096	230

*Profiler:*

Target generation per user: 0.0043s

**Achieved a 10x speedup in target generation**



# Synthetic target generation

## Numba optimization with multiprocessing

### Pure Python + NumPy + Numba

- Expanding window (Numba + NumPy)
- Filter elements (Numba + NumPy)
- Aggregate (Numba + NumPy)

0.007786	dataset.py:172(__getitem__)
0.004609	target_generator.py:56(generate_targets)
0.004311	target_generator.py:65(generate_target)
0.004159	target_generator.py:122(generate_target_from_intervals)
0.001027	boundary_generator.py:102(generate_time_feature)

### *Multiprocessing:*

Independent generation for each user

**Achieved an  $\approx 26x$  speedup**

Implementation	Transactions	User per seconds
Python + numpy	4096	25
Numba + numpy	4096	230
MP (8 processes) + Numba + numpy	4096	667

# Synthetic target generation

## Numba challenges

- Reimplement slow function using Numba
- Speedup synthetic target generation up to 10x using Numba
- Multiprocessing generation speedup x4
- Combination MP and JIT achieve 50 user per second generation

# Synthetic target generation

## Final speed up

JIT	Num process	Num users	Targets per transaction	Time (s)	Speedup	User per sec
-	0	2000	1	75	1	27
+	0	2000	1	12	6.25	167
+	4	2000	1	4	18.75	500
+	8	2000	1	3	25	667
+	0	2000	20	180	1	11
+	4	2000	20	50	3.6	40
+	8	<b>2000</b>	<b>20</b>	<b>39</b>	<b>4.61</b>	<b>51.28</b>

# Synthetic target generation

## Results

- Reimplemented slow functions using Numba
- Achieved up to a 10x speedup in synthetic target generation with Numba
- Multiprocessing generation speedup 4x from 8 processes
- The combination of multiprocessing and JIT resulted in a throughput of 50 users per second