Efficient Synthetic Target Generation for Transaction Data

Student: *Artem Zabolotnyi*

Course: High Performance Python Lab 2024

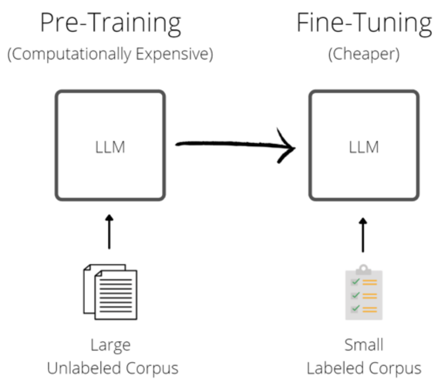
Machine learning training

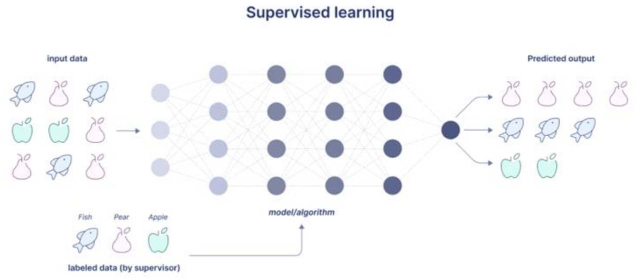
Supervised and unsupervised learning

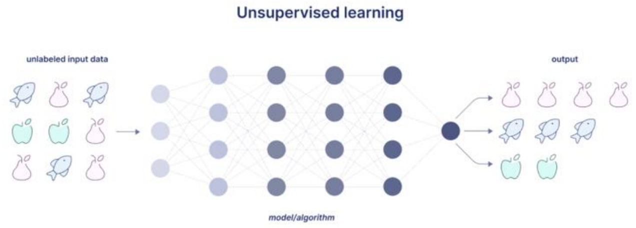
Supervised learning - training with *labeled* dataset
Unsupervised learning - training with *unlabeled* dataset

- Supervised data requires annotations.
- Large amounts of data remain unlabeled
- Generative synthetic targets can help solve downstream tasks

Training large models requires significant amounts of data and corresponding labels, which are often unavailable

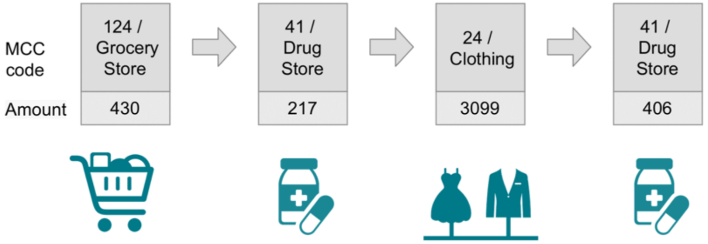






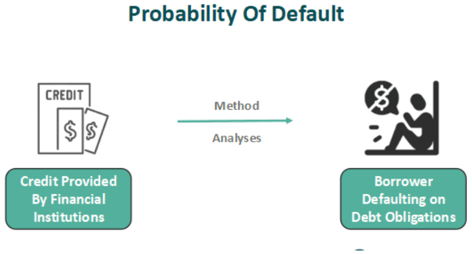
Financial transaction
Data structure

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



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

Synthetic targets provide a solution for datasets with limited labeled samples



Skoltech

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

Challenges:

- High volumes of user transactions cannot be precomputed
- Requires real-time performance for training ML models (hundreds of users per second)
- Should be integrated into Pytorch training pipeline

Skoltech

Synthetic target generation

Baseline

Pure Python + NumPy

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

Performance is insufficient for large-scale ML training

Filter + aggregate

Filter + aggregate

Filter + aggregate

	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

Skoltech

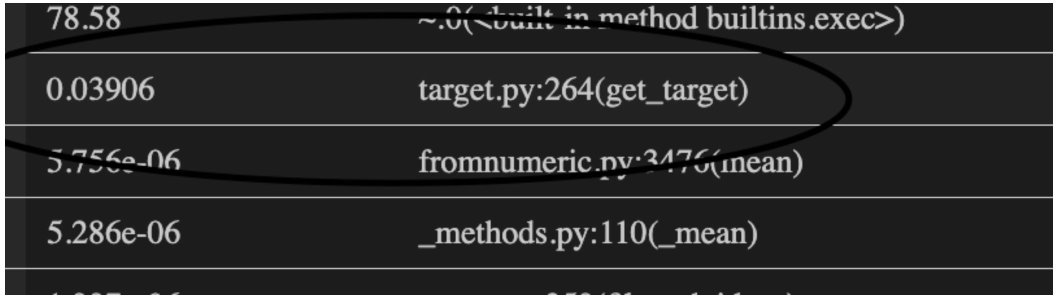
Synthetic target generation
Baseline

Pure Python + NumPy

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

Performance is insufficient for large-scale ML training

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 per user	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:36(generate_targets)
0.004311	target_generator.py:65(generate_target)
0.004159	target_generator.py:122(generate_target_from_intervals)

Profiler:

Target generation per user: 0.0043s

Implementation	Transactions per user	User per seconds
Python + numpy	4096	25
Numba + numpy	4096	133

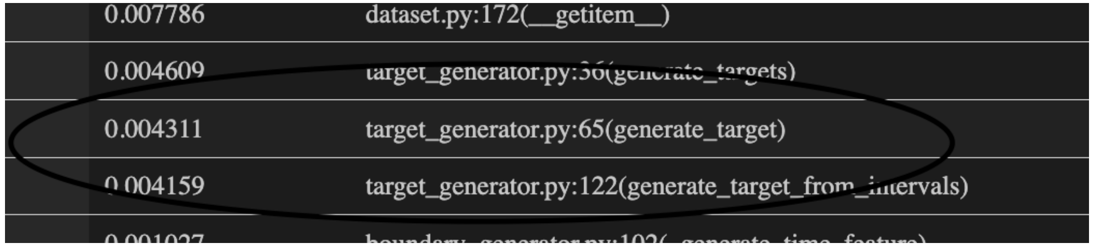
Achieved a 5x speedup in target generation compared to the baseline implementation

Synthetic target generation

Numba optimization with multiprocessing

Pure Python + NumPy + Numba

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



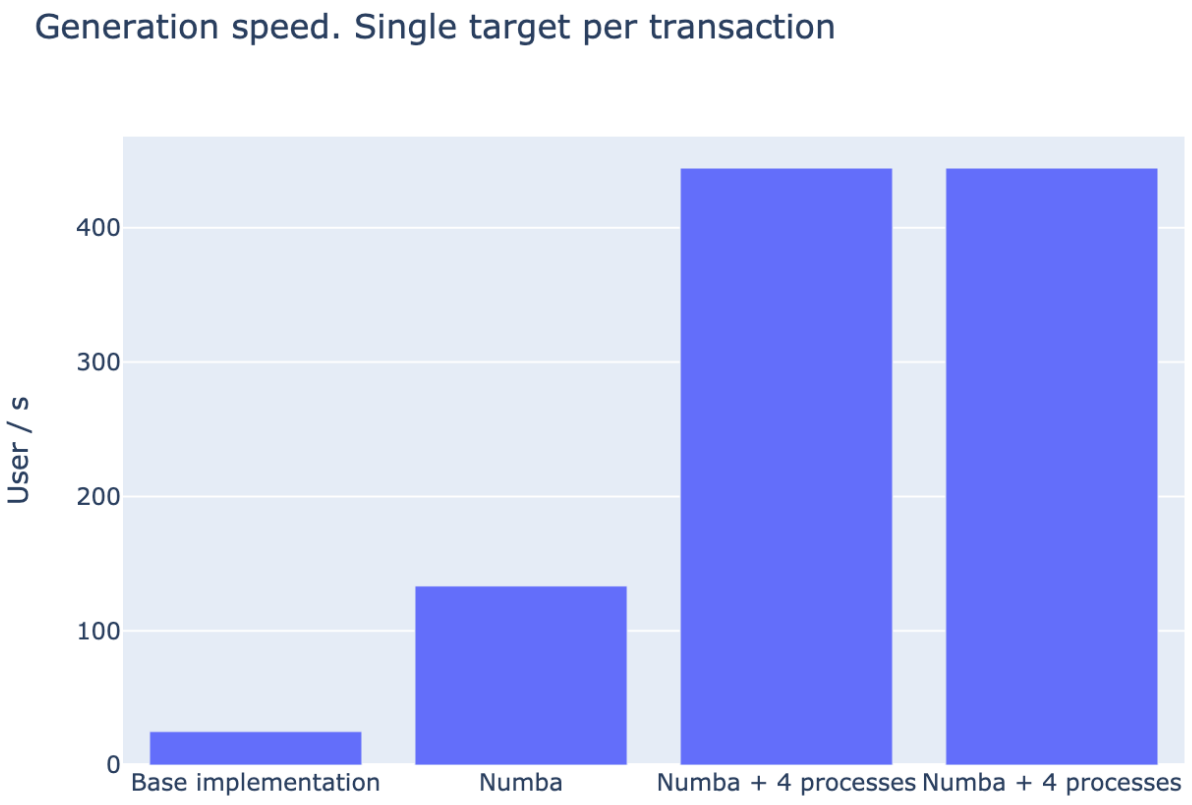
Multiprocessing:

Independent generation for each user

Achieved a 18x speedup in target generation compared to the baseline implementation

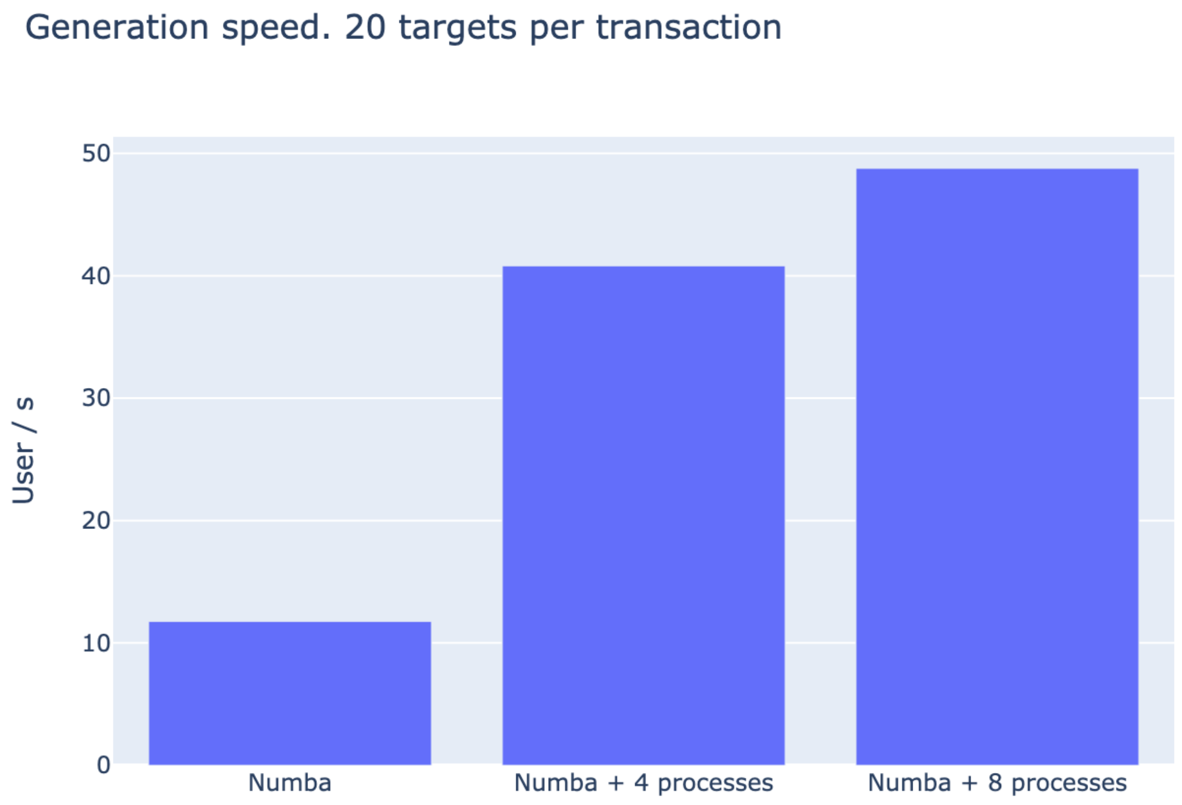
Implementation	Transactions per user	User per seconds
Python + numpy	4096	25
Numba + numpy	4096	133
MP (8 processes) + Numba + numpy	4096	444

Synthetic target generation
Single-Target Per Transaction Speedup



Skoltech

Synthetic target generation
20-Target Per Transaction Speedup



Skoltech

Synthetic target generation
Numba challenges

- JIT-compiled classes are not serializable (needed for PyTorch dataloader)
- Static typing creates challenges when Python dictionaries are used to store features
- For some functions, you must manually provide signatures with static types to enforce proper data types
- Not all operations are supported by Numba, which sometimes requires rewriting code

Skoltech

Synthetic target generation
Results

- Reimplemented slow functions using Numba
- Achieved up to a 10x speedup in synthetic target generation with Numba
- Achieved a 4x speedup in multiprocessing generation with 8 processes
- Combining multiprocessing and JIT resulted in a throughput of 50 users per second for 20 targets per transaction

Skoltech

Synthetic target generation
Final speed up

Performance Metrics for single target per transaction generation

JIT	Num Process	Num Users	Transactions per User	Num Targets per Transaction	Time (s)	Speedup	User per Sec
-	0	2000	4096	1	80	1	25
+	0	2000	4096	1	15	5	133
+	4	2000	4096	1	4.5	18	444
+	8	2000	4096	1	4.5	18	444

Performance Metrics for 20 targets per transaction generation

JIT	Num Process	Num Users	Transactions per User	Num Targets per Transaction	Time (s)	Speedup	User per Sec
+	0	2000	4096	20	212	1	9
+	4	2000	4096	20	67	3	30
+	8	2000	4096	20	45	5	44

Skoltech