

1. Dataset Sources and Total Size

For this assignment, I selected the **Ecommerce Behavior Data from Multi-Category Store** dataset, available on Kaggle.

- **Link:** <https://www.kaggle.com/datasets/mkechinov/ecommerce-behavior-data-from-multi-category-store>
- **Format:** CSV logs of user activity.
- **Columns:** event_time, event_type, product_id, category_id, category_code, brand, price, user_id, user_session.
- **Size:** Multiple monthly CSV files, each several GB, giving >1 GB of raw text when converted.

This dataset is suitable because it contains large-scale, real-world behavioral text data across multiple product categories (electronics, apparel, etc.), ensuring **domain diversity**.

2. Cleaning Strategies

The raw dataset is structured tabular data. To use it for language model pretraining, I transformed each row into **natural language sentences**, e.g.:

On 2019-11-01, user 123456 viewed a Samsung product in category electronics priced at 199.99 during session abc123.

Cleaning steps applied:

1. **Lowercasing** all text to normalize case.
2. **Whitespace cleanup:** Collapsing multiple spaces to one.
3. **Symbol removal:** Stripped special symbols except alphanumeric, periods, and commas.
4. **Filtering:** Removed very short rows (<5 tokens) and rows missing brand/category.
5. **Deduplication:** Identical rows skipped when chunking.

This produced a clean, textual corpus ready for tokenization.

3. Tokenization Choices

I used Hugging Face's **BERT Base Uncased tokenizer** (bert-base-uncased).

- **Tokenizer type:** WordPiece tokenizer.
- **Vocabulary size:** ~30,000 tokens.
- **Block size:** 128 tokens (longer rows were truncated, shorter ones padded).
- **Reasoning:**
 - BERT-style tokenization is efficient for structured natural sentences.
 - WordPiece handles rare words and subwords well.
 - 128 block size ensures uniform batches while avoiding memory blowups.

The output was stored as **PyTorch tensors** containing:

- input_ids (token indices)
- attention_mask (padding mask)

4. Data Loader Implementation

A custom **PyTorch Dataset** class wrapped the tokenized encodings.

- Implemented `__len__` and `__getitem__` for iterable access.
- Used `torch.utils.data.DataLoader` to batch and shuffle samples.
- Batch size: 16 sequences per batch.

- Padding handled automatically by tokenizer.
- For scalability, data was streamed in **50,000 row chunks** to prevent memory overflow, enabling processing of GB-scale files.

5. Challenges Encountered

- **Memory Bottleneck:** Loading >1 GB CSVs at once caused MemoryError. Solved via chunksize=50000 streaming in Pandas.
- **Tokenization Time:** Large corpora slowed tokenization. Solution: chunking + progress logging.
- **Windows caching warning:** Hugging Face tokenizer cache used symlinks, which Windows restricted. Harmless, ignored.
- **File Format:** .pt files can't be opened directly, only via PyTorch — clarified in documentation.

6. Reflections on Preprocessing Impact

- Converting structured logs into text unlocks them for **language modeling**, preserving semantic patterns of ecommerce behavior.
- Cleaning removed noisy or incomplete entries, increasing **data quality**.
- Proper tokenization and batching ensure models can **learn efficiently** without wasting capacity on formatting noise.
- Chunked streaming preprocessing enables scaling to **industry-size datasets** on modest hardware.

7. Deliverables

1. **Code:** data_collection_preprocessing.py (data collection, cleaning, tokenization, dataloader).
2. **Sample Tokenized Data:** sample_dataset.pt (first chunk of processed dataset).
3. **Report:** This PDF (documenting dataset, cleaning, tokenization, dataloader, challenges, reflections).

8. Steps to Run the Code

1. Install Dependencies

Ensure Python 3.11+ is installed. Then install required packages:

2. pip install torch transformers pandas

(Optional, for streaming support: pip install datasets tqdm)

3. Download Dataset

From Kaggle: [Ecommerce Behavior Data from Multi-category Store](#)

Place CSV files (e.g., 2019-Nov.csv) in the archive/ directory.

4. Run Preprocessing Script

Execute:

5. python data_collection_preprocessing.py

This will:

- Load dataset in chunks
- Clean and normalize text
- Tokenize using BERT tokenizer
- Create a PyTorch DataLoader
- Save sample tokenized batch as sample_dataset.pt

6. Inspect Processed Data

To view decoded text:

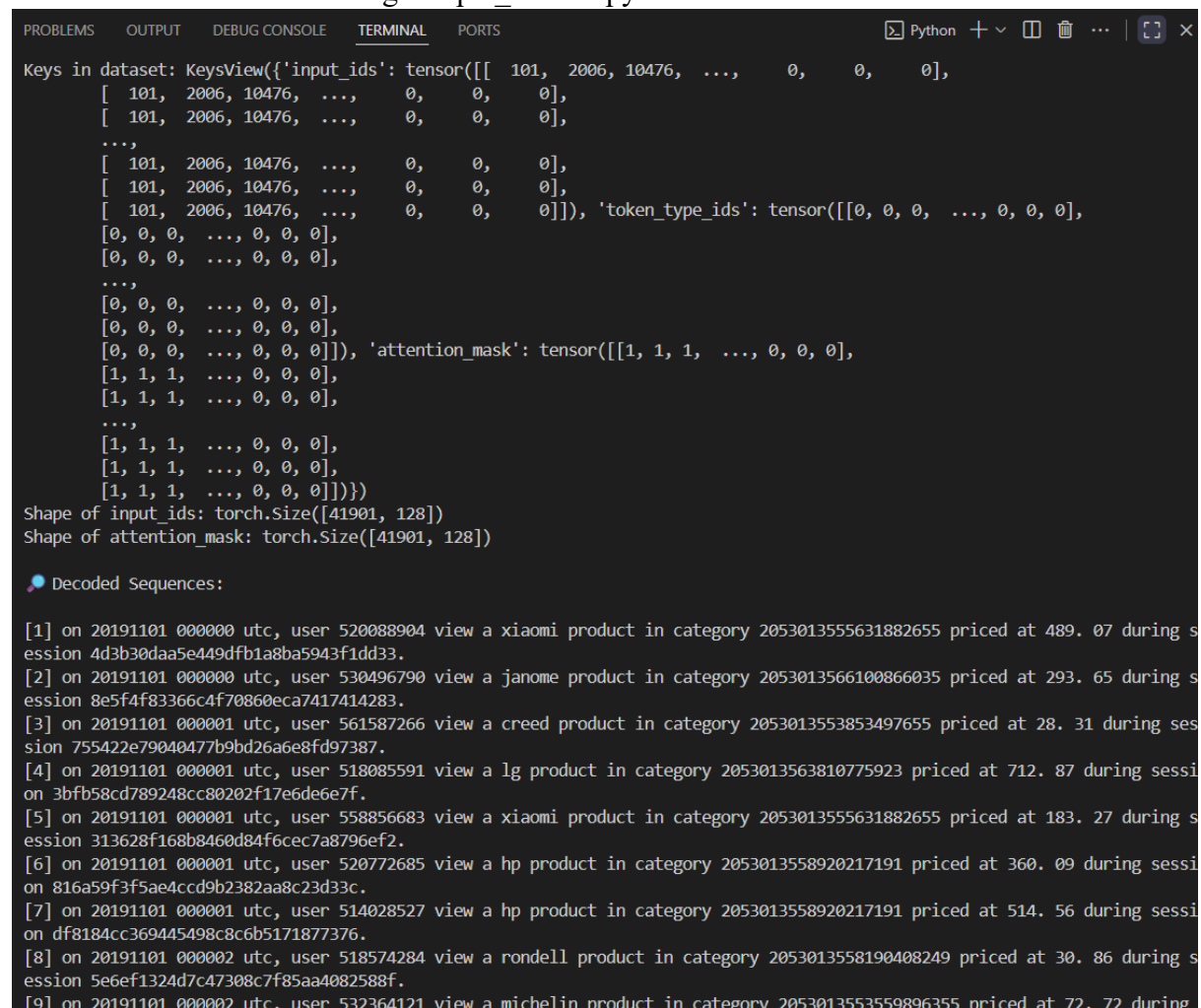
python inspect_dataset.py

Notes

1. In the script `data_collection_preprocessing.py`, the dataset path is currently **hardcoded** to my local directory for testing:
`csv_path = r"C:\Users\mtris\OneDrive\Desktop\MS Admissions Spring 2024\Universities\Northeastern University\Course materials\4th semester\CSYE 7374\archive\2019-Nov.csv"`
When running the code on a different system, this path must be updated to point to the location of the CSV file(s) downloaded from Kaggle. For example:
`csv_path = r"/path/to/archive/2019-Nov.csv" # Linux/Mac`
`csv_path = r"C:\Users\username\Downloads\archive\2019-Nov.csv" # Windows`
2. Currently my code previews the first 20 rows. If you want to view the output for all rows then replace `for i in range(20):` with `for i in range(all_input_ids.shape[0]):`

Screenshots:

The screenshot after executing `sample_dataset.py`



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Keys in dataset: KeysView({'input_ids': tensor([[ 101, 2006, 10476, ..., 0, 0, 0],
[ 101, 2006, 10476, ..., 0, 0, 0],
...,
[ 101, 2006, 10476, ..., 0, 0, 0],
[ 101, 2006, 10476, ..., 0, 0, 0],
[ 101, 2006, 10476, ..., 0, 0, 0]]), 'token_type_ids': tensor([[0, 0, 0, ..., 0, 0, 0],
[0, 0, 0, ..., 0, 0, 0],
...,
[0, 0, 0, ..., 0, 0, 0],
[0, 0, 0, ..., 0, 0, 0],
[0, 0, 0, ..., 0, 0, 0]], 'attention_mask': tensor([[1, 1, 1, ..., 0, 0, 0],
[1, 1, 1, ..., 0, 0, 0],
...,
[1, 1, 1, ..., 0, 0, 0],
[1, 1, 1, ..., 0, 0, 0],
[1, 1, 1, ..., 0, 0, 0]]]))
Shape of input_ids: torch.Size([41901, 128])
Shape of attention_mask: torch.Size([41901, 128])

Decoded Sequences:

[1] on 20191101 000000 utc, user 520088904 view a xiaomi product in category 2053013555631882655 priced at 489. 07 during session 4d3b30daa5e449dfb1a8ba5943f1dd33.
[2] on 20191101 000000 utc, user 530496790 view a janome product in category 2053013566100866035 priced at 293. 65 during session 8e5f4f83366c4f70860eca7417414283.
[3] on 20191101 000001 utc, user 561587266 view a creed product in category 2053013553853497655 priced at 28. 31 during session 755422e79040477b9bd26a6e8fd97387.
[4] on 20191101 000001 utc, user 518085591 view a lg product in category 2053013563810775923 priced at 712. 87 during session 3bfb58cd789248cc80202f17e6de6e7f.
[5] on 20191101 000001 utc, user 558856683 view a xiaomi product in category 2053013555631882655 priced at 183. 27 during session 313628f168b8460d84f6cec7a8796ef2.
[6] on 20191101 000001 utc, user 520772685 view a hp product in category 2053013558920217191 priced at 360. 09 during session 816a59f3f5ae4ccd9b2382aa8c23d33c.
[7] on 20191101 000001 utc, user 514028527 view a hp product in category 2053013558920217191 priced at 514. 56 during session df8184cc369445498c8c6b5171877376.
[8] on 20191101 000002 utc, user 518574284 view a rondell product in category 2053013558190408249 priced at 30. 86 during session 5e6ef1324d7c47308c7f85aa4082588f.
[9] on 20191101 000002 utc, user 532364121 view a michelin product in category 2053013553559896355 priced at 72. 72 during session 5e6ef1324d7c47308c7f85aa4082588f.
```

