DSC 204A: Scalable Data Systems
Programming Assignment 3

Due: November 17, 2025

1 Introduction

In this assignment, you will (1) perform feature engineering on the Amazon Reviews data, (2) train and tune a machine learning model using Ray, and (3) implement Mixture of Experts with tensor parallel using Ray. The compute setup mirrors PA2 (a 3-node Ray cluster on DataHub).

Update `Python 3 (ipykernel)` by running this in a Jupyter Notebook.

```
!pip install ray==2.50.0
!pip install xgboost==1.7.6
```

Also review the setup in Section 6 before starting.

Start early. Although each task is small, Ray Train/Tune and MoE with tensor parallel may be new to you; plan accordingly.

2 Preliminaries

2.1 Dataset

You will use three datasets:

1. **product** (`metadata_header.csv`)
2. **product_processed** (`product_processed.csv`)
3. **review** (`user_reviews_train.csv`)

Descriptive schema (from the assignment PDF):

```
1) product (metadata_header.csv)
|-- asin: string  -- product id, e.g., 'A00H8HVV6E'
|-- salesRankDict: map<string,int>  -- category -> rank, e.g., {'Home & Kitchen':
    796318}
|   |-- key: string    -- category, e.g., 'Home & Kitchen'
|   |-- value: int     -- rank, e.g., 796318
|-- categories: list<list<string>>  -- hierarchical categories, e.g., [['Home &
    Kitchen','Artwork']]
|   |-- element: list<string>       -- e.g., ['Home & Kitchen','Artwork']
|   |   |-- element: string         -- category name
|-- title: string     -- title, e.g., 'Intelligent Design Cotton Canvas'
|-- price: float      -- price, e.g., 27.9
|-- related: map<string,array<string>> -- e.g., {'also_viewed': ['B00I8HW0UK']}
|   |-- key: string     -- attribute name, e.g., 'also_viewed'
|   |-- value: array<string> -- array of product ids
|       |-- element: string  -- product id, e.g., 'B00I8HW0UK'

2) product_processed (product_processed.csv)
|-- asin: string
|-- title: string     -- title after imputation
|-- category: string  -- extracted top-level category

3) review (user_reviews_train.csv)
|-- asin: string
|-- reviewerID: string  -- e.g., 'A1MIP8H7G33SHC'
|-- overall: float      -- rating, e.g., 5.0
```

```
For Task 2, preprocessed features:
- ml_features_train
  |-- feat_0-46: float  -- contiguous features from user/product data
  |-- overall: int      -- review rating
- ml_features_test
  |-- feat_0-46: float
  |-- overall: int
```

All datasets are available in the shared course directory `~/public/pa3`; you do not need to move or download them.

2.2 Deliverables

We use **nbgrader**. Each task writes a dictionary `res` with the exact schema specified in the task notebooks. Ensure your value types match (e.g., cast `numpy` scalars to Python `int`/`float` where needed).
Generic `res` example:

```
res
|-- single_value: int
|-- list_of_values: list
|   |-- element: float
```

3 Task 1: Feature Engineering with Modin on Ray

3.1 Task 1.1: Flatten `categories` and `salesRank`

1. From `categories` (a list of lists), take the first element of the first list (i.e., `categories[0][0]`) and place it in a new column `category`. If missing/empty, set `null`.
2. `salesRankDict` contains at most one pair of (key, value). From `salesRankDict`, extract the key and the value into new columns `salesCategory` and `salesRank`. If missing/empty, set `null`.

Expected `res` schema:

```
res
|-- count_total: int -- count of rows of the transformed table, including null rows
|-- mean_salesRank: float -- mean value of SalesRank
|-- variance_salesRank: float -- variance of ...
|-- numNulls_category: int -- count of nulls of category
|-- countDistinct_category: int -- count of distinct values of ..., excluding nulls
|-- numNulls_salesCategory: int -- count of nulls of salesCategory
|-- countDistinct_salesCategory: int -- count of distinct values of ..., excluding
    nulls
```

3.2 Task 1.2: Flatten `related`

For each row, compute mean price of products referenced by `related["also_viewed"]`. Ignore ASINs not present in `product` or with `price=null`. Do not impute None (e.g., do not use fillna(0) or similar).
Expected `res` schema:

```
res
|-- count_total: int -- count of rows of the transformed table, including null rows
|-- mean_meanPriceAlsoViewed: float -- mean value of meanPriceAlsoViewed
|-- variance_meanPriceAlsoViewed: float -- variance of ...
|-- numNulls_meanPriceAlsoViewed: int -- count of nulls of ...
```

3.3 Task 1.3: Impute `price`

1. Impute `price` with the *mean* and write to `meanImputedPrice`.
2. Impute `price` with the *median* and write to `medianImputedPrice`.
3. For `title`, replace `null` with the string `"unknown"` and write to `unknownImputedTitle`.

Expected `res` schema:

```
res
|-- count_total: int -- count of rows of the transformed table, including null rows
|-- mean_meanImputedPrice: float -- mean value of meanImputedPrice
|-- variance_meanImputedPrice: float -- variance of ...
|-- numNulls_meanImputedPrice: int -- count of nulls of ...
|-- mean_medianImputedPrice: floats -- mean value of medianImputedPrice
|-- variance_medianImputedPrice: float -- variance of ...
|-- numNulls_medianImputedPrice: int -- count of nulls of ...
|-- numUnknowns_unknownImputedTitle: float -- count of "unknown" of
    unknownImputedTitle
```

4 Task 2: Training and Tuning with Ray

4.1 Task 2.1: Distributed XGBoost with Ray Train

Train a regression model (MSE objective) to predict `overall` using `XGBoostTrainer`

- Objective: regression with squared error (`reg:squarederror`)
- Model params: `max_depth=3`, `eta=0.3`, others default.
- Use Ray Train with a suitable `ScalingConfig`; and you can set 2 workers with 1 CPUs per worker.
- After training, run inference on the test set.

Expected `res` schema:

```
res
|-- test_rmse: float --- RMSE of the test set predictions
|-- train_rmse: float --- RMSE of the train set predictions
```

4.2 Task 2.2: Tuning with Ray Tune

We'll now perform a grid search for 2 Xgboost hyperparameters - max depth and eta. Given our limited computational budget, we'll focus on a small grid of values:

$$\texttt{max\_depth} \in \{3, 5\}, \quad \texttt{eta} \in \{0.3, 0.5\},$$

Your task is as follows:

1. Split the original training set into train/validation (75/25, random).
2. Train Xgboost models with 4 hyperparameter trials over the given grid using Ray Tune.
3. Report the best model's validation and test RMSE. Also report validation RMSEs for specific grid points (as specified in the notebook schema).

Nov 21 Update: For Task 2.2, the original test cases only allowed a 0.01 tolerance around the provided RMSE. For final grading, any submission with RMSE less than 1.13 will receive full credit.

Expected `res` schema:

```
res
|-- test_rmse: float           -- best model's test RMSE
|-- valid_rmse: float          -- best model's validation RMSE
|-- valid_depth_5_eta_0.3: float
|-- valid_depth_3_eta_0.5: float
```

Helpful links

- Distributed XGBoost with Ray Train
- Offline Batch Inference with Ray Data
- Ray Tune
- Ray Tune with XGBoost

5 Task 3. Mixture of Experts with Ray

This assignment implements distributed Mixture of Experts (MoE) using Ray. Before diving into the implementation, here is some background context:

Mixture of Experts (MoE) is a neural network architecture where multiple "experts" (sub-networks) exist in parallel. A gating network selects which experts should process each input. This approach can greatly increase model capacity without a proportional increase in computation per sample. Also, wee Page 43 - 47 of the slides for illustration of MoE.

Tensor Parallelism (TP) is a distributed training technique where a single layer (e.g., a large matrix multiplication) is split across multiple devices. Each device computes a partial result, and the partial results are combined to produce the full output.

5.1 Task 3.1: MoE with Tensor Parallelism (MoE-TP)

- Each expert consists of multiple layers, and the parameters of every layer are **sharded across all workers** (i.e., each worker stores only a slice of the layer's weights).
  In this assignment, we assume there are 10 experts, each containing 2 fully connected layers, for a total of 20 layers. Under tensor parallelism, each worker maintains a shard from every one of these 20 layers.
- Every worker computes its **partial output for each layer** (based on the parameters it stores).
- The partial outputs are then **combined by concatenation** to obtain the full expert outputs. See `ShardedLinear` in `Task3.ipynb` for how a sharded linear layer works.
- This design is well-suited for balanced workloads where all experts need to be computed in parallel.

The goal of this assignment is to implement MoE-TP using Ray. As a refererence implementation, we provide the `SimpleMoE` class. We then provide the skeleton code for the `MoE_TP` class for you to complete. See `Task3.ipynb` for details.

5.2 Grading Rubric

- Efficiency: Achieve a speedup greater than 2×.
- Correctness: Given the same initization of weights and bias, the outputs after one forward pass for `SimpleMoE` and `MoE_TP` should be the same.

6 Development Instructions

6.1 Environment

On DataHub, choose the environment with 8 CPUs and 16GB RAM on the head node.

6.2 Restarting the cluster

To restart, go to *Control Panel* → *Stop My Server*. Logging out does not stop the cluster. If you need to clear state in a session, prefer `ray.shutdown()` and notebook restart over running `ray stop`.

6.3 Shut Down Unused Notebooks

This assignment is memory-intensive. You can close unused Jupyter notebooks or processes to free up CPU memory.

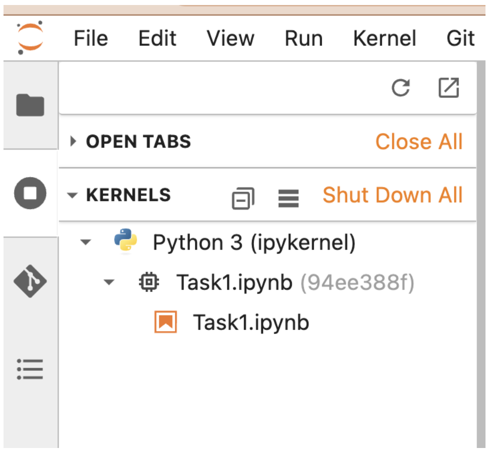
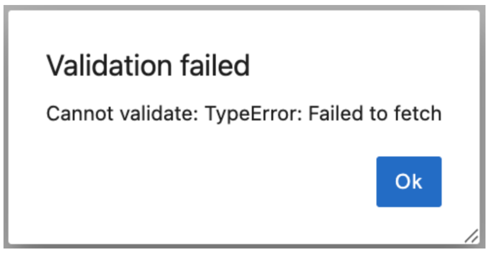


Figure 1: Check opened notebooks, or shut down all notebooks.

6.4 Validate

Sometimes you may encounter a validation error (Figure 2) even if all tests pass when you directly run your notebook.



Validation failed

Cannot validate: TypeError: Failed to fetch

Figure 2: Validation Error.

In such cases, open a terminal and open pa3 folder, then run

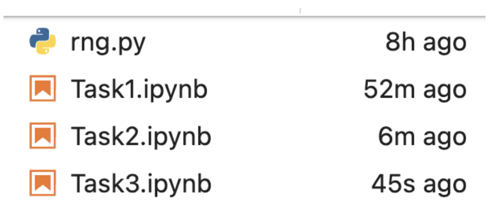
```
nbgrader validate Task1.ipynb
```

You can replace `Task1.ipynb` with the notebook you are validating. This command provides detailed warnings and errors, and is considered more robust than clicking "Validate" button directly. You pass validation if this shows up in the end:

```
Success!  Your notebook passes all the tests.
```

6.5 Submit

Make sure to retain only the four files in Figure 3 in your `pa3` folder when submitting. Do not delete `rng.py`, as it provides the necessary utilities for Task 3.



rng.py	8h ago
Task1.ipynb	52m ago
Task2.ipynb	6m ago
Task3.ipynb	45s ago

Figure 3: List of submitted files.

6.6 Grading

All tasks in PA3 will be graded by Datahub autograder.

Task	Description	Score
1.1	Flatten schema; handle list/dict types	15
1.2	Flatten schema; perform self-joins	15
1.3	Data imputation	10
2.1	Distributed XGBoost with Ray Train	20
2.2	Hyperparameter tuning with Ray Tune	20
3.1	MoE with Tensor Parallel	20

7 Acknowledgements

This assignment draws on prior coursework material (e.g., DSC 102 PySpark assignment).