DSC 204A: Scalable Data Systems

Programming Assignment 3

Due Date: March 13, 2024

1 Introduction

The goal of this programming assignment is to train and tune a machine learning model with Ray. First, we'll perform feature engineering on the Amazon Reviews dataset. In the second part, we'll train a machine learning model and tune its hyperparameters with Ray. The setup is the same as PA 2, with a 3-node Ray cluster on Datahub. As before, you should make sure to go through the setup instructions in Section 5 before attempting the tasks.

Note: Please get started as early as you can. You have limited time to complete this assignment. Each task in itself should not take long, but given that you might be familiar with Ray Train/Tune APIs, we recommend that you plan early.

2 Preliminaries

2.1 Dataset

The schema for the dataset is given below:

```
1. product (metadata_header.csv)
  |-- asin: string, the product id, e.g., 'A00H8HVV6E'
  |-- salesRank: map, a map between category and sales rank, e.g., {'Home & Kitchen': 796318}
  |  |-- key: string, category, e.g., 'Home & Kitchen'
  |  |-- value: integer, rank, e.g., 796318
  |-- categories: list, list of list of categories, e.g., [['Home & Kitchen', 'Artwork']]
  |  |-- element: list, list of categories, e.g., ['Home & Kitchen', 'Artwork']
  |  |  |-- element: string, category, e.g., 'Home & Kitchen'
  |-- title: string, title of product, e.g., 'Intelligent Design Cotton Canvas'
  |-- price: float, price of product, e.g., 27.9
  |-- related: map, related information, e.g., {'also_viewed': ['B00I8HW0UK']}
  |  |-- key: string, the attribute name of the information, e.g., 'also_viewed'
  |  |-- value: array, array of product ids, e.g., ['B00I8HW0UK']
  |  |  |-- element: string product id , e.g., 'B00I8HW0UK'
2. product_processed (product_processed.csv)
  |-- asin: string, same as above
  |-- title: string, title column after imputation, e.g., 'Intelligent Design Cotton Canvas'
  |-- category: string, category column after extraction, e.g., 'Home & Kitchen'
```

```
3. review (user_reviews_train.csv)
  |-- asin: string, same as above
  |-- reviewerID: string, the reviewer id, e.g., 'A1MIP8H7G33SHC'
  |-- overall: float, the rating associated with the review, e.g., 5.0
```

For Task 1, we will be working mostly with the product information datasets - `product` for Task 1.1 - 1.3 and `product_processed` for Task 1.4.

For Task 2, we'll use the following preprocessed datasets:

```
1. ml_features_train
    |-- feat_0-46: float, features (contiguous) from user and product data
    |-- overall: integer, review rating
2. ml_features_test
    |-- feat_0-46: float, same as above
    |-- overall: integer, same as above
```

All the datasets have been placed in the shared course directory[1]. Thus, you do not have to move, modify or download any of them.

2.2 Deliverables

As before, we'll be using `nbgrader` for this assignment. For each task, there is an expected result with a pre-defined schema. Each output is stored in a dictionary `res`:

```
res
 | -- single_value: int --- an integer number
 | -- list_of_values --- list of values
    | -- element: float --- a float value
```

You can refer to the task-specific notebooks to get a better sense of how this looks. Make sure that the datatypes for each entry matches the given scheme. In some cases, you might need to explicity cast the result to the presented datatype (for example, `np.int64` to `int`).

3 Task 1: Feature Engineering with Modin on Ray

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

1. For the product table, each item in column categories contains a list of lists of hierarchical categories. The schema is `list(list(string))`. We are only going to use the most general category, which is the first element of the nested list: `my_categories[0][0]`. For each row, put the first element of categories in a new column `category`. If categories is null or empty (e.g., `[]`), put a null in your new column.

2. On the other hand, each entry in column salesRank is a key-value pair: (bestSalesCategory, rank). Your task is to flatten it into two columns. Put the key in a new column named `bestSalesCategory` and the value in `bestSalesRank`. Put null if the original entry was null or empty.

The schema of output is as follows:

[1] ~/public/pa3

```
res
 | -- count_total: int -- count of rows of the transformed table, including null rows
 | -- mean_bestSalesRank: float -- mean value of bestSalesRank
 | -- variance_bestSalesRank: float -- variance of ...
 | -- numNulls_category: int -- count of nulls of category
 | -- countDistinct_category: int -- count of distinct values of ..., excluding nulls
 | -- numNulls_bestSalesCategory: int -- count of nulls of bestSalesCategory
 | -- countDistinct_bestSalesCategory: int -- count of distinct values of ..., excluding nulls
```

3.2 Task 1.2: Flatten `related`

Each entry of related column is a map with four keys/attributes: `also_bought`, `also_viewed`, `bought_together`, and `buy_after_viewing`. Each value of these keys contains an array of product IDs ("attribute arrays"). You need to calculate the length of the arrays and find out the average prices of the products in these arrays.

1. For each row of related: calculate the mean price of all products from the also viewed attribute array. Put it in a new column `meanPriceAlsoViewed`. Remember to ignore the product IDs if they do not match any record in product. Or if they match records in product, but the records have null in the price column. Do not ignore products if they have `price = 0`.

2. Similarly, put the length of that array in a new column `countAlsoViewed`. In this case, you do not need to check if the product IDs in that array are dangling references and do not have matching records in product. Put null (instead of zero) in the new column, if the attribute array is null or empty.

3. The schema of the output dictionary `res` should be -

```
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 ...
 | -- mean_countAlsoViewed: float -- mean value of countAlsoViewed
 | -- variance_countAlsoViewed: float -- variance of ...
 | -- numNulls_countAlsoViewed: int -- count of nulls of ...
```

3.3 Task 1.3: Impute `price`

You may have noticed that there are lots of nulls in the table. Now your task is to impute them with meaningful values that can be used to train machine learning models. You need to impute the numerical column `price` as well as a string column `title`.

1. For column `price`, first cast every entry to float type. Then impute the `nulls` with the mean of all the non-null values. Store the outputs in a new column `meanImputedPrice`.

2. Same as above, but this time impute with the **median** value. Store the imputed data in a new column `medianImputedPrice`. For both these numerical imputation tasks, you may use the `SimpleImputer` from `sklearn.impute`.

3. For column `title`: As for the string-typed columns, we want to impute nulls **and empty strings** simply with a special string `'unknown'`. Store the imputed data in a new column `unknownImputedTitle`.

4. Schema of `res` should look like

```
res
 | -- count_total: int -- count of rows of the transformed table, including null rows
 | -- mean_meanImputedPrice: float or None -- mean value of meanImputedPrice
 | -- variance_meanImputedPrice: float -- variance of ...
 | -- numNulls_meanImputedPrice: int -- count of nulls of ...
 | -- mean_medianImputedPrice: float or None -- mean value of medianImputedPrice
 | -- variance_medianImputedPrice: float -- variance of ...
 | -- numNulls_medianImputedPrice: int -- count of nulls of ...
 | -- numUnknowns_unknownImputedTitle: float -- count of 'unknown' value
 entries in unknownImputedTitle
```

3.4 Task 1.4: Process `title` and one-hot encode `category`

For this task, we will use the imputed dataset `product_processed.csv`. You need to perform the following:

1. For each row, convert `title` to lowercase, then split it by whitespace (`' '`) to an array of strings. Store this array in a new column `titleArray`.

2. Calculate the mean length of the array in `titleArray`.

3. One-hot encode the `category` column. In this case, make sure to use the sorted order of the unique categories to form the one-hot vector. That is, in the resultant one-hot vector, the array indices should correspond to the category names in sorted order.

The output schema is below:

```
res
 | -- count_total: int, count of rows of the transformed table,
 | -- meanLength_titleArray: float, mean length of the array titleArray
 | -- mean_categoryOneHot:  list, mean value of one-hot encoding vectors
```

4 Task 2: Training and Tuning with Ray

In this task, you will train a machine learning model using the preprocessed data using Ray Train. We'll then tune hyperparameters with Ray Tune. We are providing you with the final processed dataframes for training and testing your models.

4.1 Task 2.1: Distributed Xgboost with Ray Train

You need to train an Xgboost model to predict the user rating for a product.

1. Follow the instructions in `Task2.ipynb`. The model should be trained with a regression objective to minimize the mean squared error.

2. The `max_depth` parameter of the model must be set to 3, the `eta` value to 0.3. All other parameters of the model should be left to default values.

3. For the Ray trainer, set the `ScalingConfig` appropriately with a limit of 6 cpus per worker.

4. After training, you also need to write inference code to get predictions for the test dataset.

The schema of the output would be:

```
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 3 Xgboost hyperparameters - `max_depth`, `eta` and `subsample`. Given our limited computational budget, we'll focus on a small grid of values:

- `max_depth`: $[3, 4, 5]$
- `eta`: $[0.3, 0.5]$
- `subsample`: $[0.8, 1.0]$

Your task is as follows:

1. Create a new training and validation set from the original training data - with a random split of 75/25.

2. Train Xgboost models with 12 hyperparameter trials over the given grid using Ray Tune.

3. Select the best model with the lowest validation RMSE.

The schema of the output is as follows:

```
res
  | -- test_rmse: float --- RMSE of the test set predictions for
     the best model
  | -- valid_rmse: float --- RMSE of the validation set
     predictions for the best model
  | -- valid_depth_5_eta_0.3_subsample_0.8: float --- RMSE of the
     validation set predictions for max_depth=5, eta=0.3,
     subsample=0.8
  | -- valid_depth_4_eta_0.3_subsample_1: float --- RMSE of the
     validation set predictions for max_depth=4, ....
  | -- valid_depth_3_eta_0.5_subsample_1: float --- RMSE of the
     validation set predictions for max_depth=3, ....
```

Some helpful resources:

- Distributed Xgboost with Ray train
- Offline Batch Inference with Ray Data
- Ray Tune with Xgboost

5 Development Instructions

We'll be using the same setup on DataHub as PA 2 for this assignment. A quick summary of instructions is given below:

- Select the `ray-notebook` environment with 8 CPUs, 32GB RAM in the head node. We're providing increased memory and CPUs to help with large dataset processing and training for this assignment.
- As before, you should check the Ray dashboard to verify that all the nodes are online.
- You can fetch the assignments under the "Assignments" tab on DataHub. Your task is to complete the notebooks `Task1.ipynb` and `Task2.ipynb` in the appropriate sections.

5.1 Restarting the cluster

If you want to restart your cluster, you should navigate to the "Control Panel" button shown in the top right and click on "Stop My Server". Logging out ("Logout") does not stop your Ray cluster.

Sometimes, you might want to clear out occupied memory from a previous code block, in which case you should run `ray.shutdown()` and, if possible, restart the notebook. With Ray, you should **not** try to stop the Ray instance running on the head node (that is, do **not** run `ray stop` in the terminal). Stick to using `ray.init()` (which connects to the existing Ray instance) and `ray.shutdown()` (which disconnects from the Ray instance) while developing your solutions. If you do end up running `ray stop`, simply restart your cluster as mentioned above.

5.2 Package installations

All the required packages have been installed for you in the Ray cluster. Thus, you do not need to install any libraries yourself for completing this assignment. With DataHub, it is better to exercise caution here even if you want to do some experimentation: libraries you install via `pip` will be a *user* installation, saved in your `private` directory. Installations will thus remain even if you restart your cluster. If you do end up installing a package that breaks your environment, then a simple restart will not solve the issue! In such a case, make sure to review the user installed libraries via `pip list --user` and uninstalling them through `pip uninstall <package>`

5.3 Verifying your solution

For this assignment do **not** use `nbgrader`'s "Validate" button. Instead, to verify the correctness of your submission: Do the following:

- Run all the cells in both the notebooks. This will write out your results to json files. *Verify that all the cells have successfully ran! If your code errors out during grading, you will not receive any points!*
- Use the notebook `PA3Test` to run tests for each task.

6 Grading scheme

We will follow the grading scheme in Table 1. Your code must pass all the tests to be counted pass for a given subtask.

Task No.	Task Description	Score (Pass/Fail)
Task 1.1	Flatten schema and handle list/dict types	15/0
Task 1.2	Flatten schema and perform self-joins	15/0
Task 1.3	Data imputation	10/0
Task 1.4	Apply one hot encoding	10/0
Task 2.1	Distributed Xgboost with Ray Train	25/0
Task 2.2	Tuning with Ray Tune	25/0

Table 1: Grading Rubric for Programming Assignment 3

The runtime for our Task 1 and Task 2 notebooks are about 10 minutes. We will first run your notebooks with a maximum timeout of 40 minutes per task. If your notebook exceeds the given time limit, or if the code errors out, you will not receive any points for that task/subtask.

7 Acknowledgements

This assignment draws extensively from the PySpark programming assignment from DSC 102.