

Run NVIDIA-Merlin MovieLens Example in Jupyter Lab

NVIDIA-Merlin

NVIDIA Merlin is an open source library that accelerates recommender systems on NVIDIA GPUs. The library enables data scientists, machine learning engineers, and researchers to build high-performing recommenders at scale. Merlin includes tools to address common feature engineering, training, and inference challenges. Each stage of the Merlin pipeline is optimized to support hundreds of terabytes of data, which is all accessible through easy-to-use APIs. For more information, see NVIDIA Merlin on the [NVIDIA developer web site](#).

MovieLens

The MovieLens25M is a popular dataset for recommender systems and is used in academic publications. In this example, we will:

- Learn to use NVTabular for using GPU-accelerated feature engineering and data preprocessing.
- Become familiar with the high-level API for NVTabular.
- Use single-hot/multi-hot categorical input features with NVTabular.
- Train a Merlin Model, Export model with PyTorch.

Launch Jupyter Lab, set up the Container Kernel

To launch Jupyter Lab and set up the container kernel, please refer to the following link: <https://doc.oasishpc.hk/books/oasis-user-guide/page/pytorch-with-gpu-in-jupyter-lab-using-container-based-kernel>.

I initiated a job with a GPU configuration of **"1g.10gb" GPU, an 8-core CPU, and 64GB of memory**.

The kernel being used is **"ngc.merlin-pytorch.22.11.sif"** which can be imported using the following path: **"/pfss/containers/ngc.merlin-pytorch.22.11.sif"**

Run example code in Jupyter Lab

To execute NVIDIA-Merlin code in Jupyter Lab (ref: <https://github.com/NVIDIA-Merlin/Merlin/tree/main/examples/getting-started-movielens>), kindly copy the example code and run it in Jupyter Lab.

For the notebook "**01-Download-Convert.ipynb**" please ensure you update the "INPUT_DATA_DIR" to reflect your scratch folder. In my case, I modified it to "/pfss/scratch01/milo/merlin/"

```
# External dependencies
import os

from merlin.core.utils import download_file

# Get dataframe library - cudf or pandas
from merlin.core.dispatch import get_lib
df_lib = get_lib()

INPUT_DATA_DIR = os.environ.get(
    "INPUT_DATA_DIR", os.path.expanduser("/pfss/scratch01/milo/merlin/")
)

download_file(
    "http://files.grouplens.org/datasets/movielens/ml-25m.zip",
    os.path.join(INPUT_DATA_DIR, "ml-25m.zip"),
)

movies = df_lib.read_csv(os.path.join(INPUT_DATA_DIR, "ml-25m/movies.csv"))
movies.head()

movies["genres"] = movies["genres"].str.split("|")
movies = movies.drop("title", axis=1)
movies.head()

movies.to_parquet(os.path.join(INPUT_DATA_DIR, "movies_converted.parquet"))

ratings = df_lib.read_csv(os.path.join(INPUT_DATA_DIR, "ml-25m", "ratings.csv"))
ratings.head()

ratings = ratings.drop("timestamp", axis=1)
```

```
# shuffle the dataset
ratings = ratings.sample(len(ratings), replace=False)

# split the train_df as training and validation data sets.
num_valid = int(len(ratings) * 0.2)

train = ratings[:-num_valid]
valid = ratings[-num_valid:]

train.to_parquet(os.path.join(INPUT_DATA_DIR, "train.parquet"))
valid.to_parquet(os.path.join(INPUT_DATA_DIR, "valid.parquet"))
```

Likewise, for the notebook "**02-ETL-with-NVTabular.ipynb**" please update the "INPUT_DATA_DIR" to match your scratch folder. In my case, I adjusted it to **"/pfss/scratch01/milo/merlin/"**

```
# External dependencies
import os
import shutil
import numpy as np
from nvtabular.ops import *
from merlin.schema.tags import Tags

import nvtabular as nvt

from os import path

# Get dataframe library - cudf or pandas
from merlin.core.dispatch import get_lib
df_lib = get_lib()

INPUT_DATA_DIR = os.environ.get(
    "INPUT_DATA_DIR", os.path.expanduser("/pfss/scratch01/milo/merlin/")
)

movies = df_lib.read_parquet(os.path.join(INPUT_DATA_DIR, "movies_converted.parquet"))
movies.head()

CATEGORICAL_COLUMNS = ["userId", "movieId"]
LABEL_COLUMNS = ["rating"]
```

```

userId = ["userId"] >> TagAsUserID()
movieId = ["movieId"] >> TagAsItemID()

joined = userId + movieId >> JoinExternal(movies, on=["movieId"])

joined.graph

cat_features = joined >> Categorify()

ratings = nvt.ColumnGroup(["rating"]) >> LambdaOp(lambda col: (col > 3).astype("int8")) >>
AddTags(Tags.TARGET)

output = cat_features + ratings
(output).graph

workflow = nvt.Workflow(output)

dict_dtypes = {}

for col in CATEGORICAL_COLUMNS:
    dict_dtypes[col] = np.int64

for col in LABEL_COLUMNS:
    dict_dtypes[col] = np.float32

train_dataset = nvt.Dataset([os.path.join(INPUT_DATA_DIR, "train.parquet")])
valid_dataset = nvt.Dataset([os.path.join(INPUT_DATA_DIR, "valid.parquet")])

%%time
workflow.fit(train_dataset)

# Make sure we have a clean output path
if path.exists(os.path.join(INPUT_DATA_DIR, "train")):
    shutil.rmtree(os.path.join(INPUT_DATA_DIR, "train"))
if path.exists(os.path.join(INPUT_DATA_DIR, "valid")):
    shutil.rmtree(os.path.join(INPUT_DATA_DIR, "valid"))

%time
workflow.transform(train_dataset).to_parquet(

```

```

output_path=os.path.join(INPUT_DATA_DIR, "train"),
shuffle=nvt.io.Shuffle.PER_PARTITION,
cats=["userId", "movieId", "genres"],
labels=["rating"],
dtypes=dict_dtypes
)

%time
workflow.transform(valid_dataset).to_parquet(
    output_path=os.path.join(INPUT_DATA_DIR, "valid"),
    shuffle=False,
    cats=["userId", "movieId", "genres"],
    labels=["rating"],
    dtypes=dict_dtypes
)

workflow.save(os.path.join(INPUT_DATA_DIR, "workflow"))
workflow.output_schema

import glob

TRAIN_PATHS = sorted(glob.glob(os.path.join(INPUT_DATA_DIR, "train", "*.parquet")))
VALID_PATHS = sorted(glob.glob(os.path.join(INPUT_DATA_DIR, "valid", "*.parquet")))

df = df_lib.read_parquet(TRAIN_PATHS[0])
df.head()

```

For the notebook "**03-Training-with-PyTorch.ipynb**" make sure you update the "INPUT_DATA_DIR", "OUTPUT_DATA_DIR" to correspond with your scratch folder. In my case, I changed it to **"/pfss/scratch01/milo/merlin/"**

```

# External dependencies
import os
import gc
import glob

import nvtabular as nvt
from merlin.schema.tags import Tags

INPUT_DATA_DIR = os.environ.get(

```

```

    "INPUT_DATA_DIR", os.path.expanduser("/pfss/scratch01/milo/merlin/")
)

# Output from ETL-with-NVTabular
TRAIN_PATHS = sorted(glob.glob(os.path.join(INPUT_DATA_DIR, "train", "*.parquet")))
VALID_PATHS = sorted(glob.glob(os.path.join(INPUT_DATA_DIR, "valid", "*.parquet")))

import torch
from merlin.loader.torch import Loader

from nvtabular.framework_utils.torch.models import Model
from nvtabular.framework_utils.torch.utils import process_epoch
from nvtabular.framework_utils.torch.utils import DictTransform

BATCH_SIZE = 1024 * 32 # Batch Size

train_dataset = nvt.Dataset(TRAIN_PATHS)
validation_dataset = nvt.Dataset(VALID_PATHS)

train_loader = Loader(
    train_dataset,
    batch_size=BATCH_SIZE,
)

valid_loader = Loader(
    validation_dataset,
    batch_size=BATCH_SIZE,
)

batch = next(iter(train_loader))
batch

del batch
gc.collect()

# ??Model

def extract_info(col_name, schema):
    """extracts embedding cardinality and dimension from schema"""
    return (

```

```

        int(schema.select_by_name(col_name).first.properties['embedding_sizes']['cardinality']),
        int(schema.select_by_name(col_name).first.properties['embedding_sizes']['dimension'])
    )

single_hot_embedding_tables_shapes = {col_name: extract_info(col_name, train_loader.dataset.schema) for
col_name in ['userId', 'movieId']}
mutli_hot_embedding_tables_shapes = {col_name: extract_info(col_name, train_loader.dataset.schema) for
col_name in ['genres']}

single_hot_embedding_tables_shapes, mutli_hot_embedding_tables_shapes

model = Model(
    embedding_table_shapes=(single_hot_embedding_tables_shapes, mutli_hot_embedding_tables_shapes),
    num_continuous=0,
    emb_dropout=0.0,
    layer_hidden_dims=[128, 128, 128],
    layer_dropout_rates=[0.0, 0.0, 0.0],
).to("cuda")
model

optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

%%time
from time import time
EPOCHS = 1
for epoch in range(EPOCHS):
    start = time()
    train_loss, y_pred, y = process_epoch(train_loader,
                                         model,
                                         train=True,
                                         optimizer=optimizer,
                                         loss_func=torch.nn.BCEWithLogitsLoss())
    valid_loss, y_pred, y = process_epoch(valid_loader,
                                         model,
                                         train=False)

    print(f"Epoch {epoch:02d}. Train loss: {train_loss:.4f}. Valid loss: {valid_loss:.4f}.")

OUTPUT_DATA_DIR = os.environ.get(
    "OUTPUT_DATA_DIR", os.path.expanduser("/pfss/scratch01/milo/merlin/")
)

```

```
torch.save(model.state_dict(), OUTPUT_DATA_DIR + "merlin_model.pth")
```

```
batch = next(iter(train_loader))
```

```
transform = DictTransform(train_loader).transform
```

```
x_cat, x_cont, y = transform(batch)
```

```
model.eval()
```

```
torch.onnx.export(model,
```

```
    [(x_cat, x_cont),
```

```
    OUTPUT_DATA_DIR + "merlin_model.onnx"
```

```
    )
```

Revision #2

Created 28 April 2023 06:42:18 by Milo Cheung

Updated 28 April 2023 07:23:53 by Milo Cheung