svg image

Pipeline 9 Notebook for training continuation - AutoAI Notebook v2.1.7

Consider these tips for working with an auto-generated notebook:

- Notebook code generated using AutoAI will execute successfully. If you modify the notebook, we cannot guarantee it will run successfully.
- This pipeline is optimized for the original data set. The pipeline might fail or produce sub-optimal results if used with different data. If you want to use a different data set, consider retraining the AutoAI experiment to generate a new pipeline. For more information, see Cloud Platform.
- Before modifying the pipeline or trying to re-fit the pipeline, consider that the code converts dataframes to numpy arrays before fitting the pipeline (a current restriction of the preprocessor pipeline).

Notebook content

This notebook contains code to resume and continue training an AutoAI pipeline partially trained in an AutoAI experiment. If there is additional training data, the notebook retrieves the data in batches and incrementally trains the model, then tests the model.

Some familiarity with Python is helpful. This notebook uses python 3.11 and scikit-learn 1.3.

Notebook goals

This notebook introduces new commands and demonstrates techniques to support incremental learning, including:

- Data reader (read data in batches)
- Incremental learning (`partial_fit`)
- Pipeline evaluation

Contents

This notebook contains the following parts:

Setup

Package installation

Before you use the sample code in this notebook, install the following packages:

- ibm-watsonx-ai,
- autoai-libs,
- scikit-learn,
- snapml

```
!pip install ibm-watsonx-ai | tail -n 1
!pip install autoai-libs~=2.0 | tail -n 1
!pip install scikit-learn==1.3.* | tail -n 1
!pip install -U lale~=0.8.3 | tail -n 1
!pip install snapml==1.14.* | tail -n 1
```

AutoAI experiment metadata

The following cell contains the training data connection details.
Note: The connection might contain authorization credentials, so be careful when sharing the notebook.

```
from ibm_watsonx_ai.helpers import DataConnection
from ibm_watsonx_ai.helpers import ContainerLocation

training_data_references = [
    DataConnection(
        data_asset_id='b965c573-7a52-4991-a4e2-759e0ff933c8'
    ),
]
training_result_reference = DataConnection(
    location=ContainerLocation(

path='auto_ml/7bec33b4-40e1-4925-833e-98edfda8f6ad/wml_data/bbb3633e-
fb09-42a5-8635-211d16e500ec/data/automl',

model_location='auto_ml/7bec33b4-40e1-4925-833e-98edfda8f6ad/wml_data/
bbb3633e-fb09-42a5-8635-211d16e500ec/data/automl/model.zip',
        training_status='auto_ml/7bec33b4-40e1-4925-833e-
98edfda8f6ad/wml_data/bbb3633e-fb09-42a5-8635-211d16e500ec/training-
status.json'
    )
)
```

The following cell contains input parameters provided to run the AutoAI experiment in Watson Studio.

```
experiment_metadata = dict(
    prediction_type='multiclass',
    prediction_column='Fault Type',
    holdout_size=0.1,
    scoring='accuracy',
    csv_separator=',',
    random_state=33,
    max_number_of_estimators=2,
    training_data_references=training_data_references,
    training_result_reference=training_result_reference,
    deployment_url='https://eu-gb.ml.cloud.ibm.com',
    project_id='be285154-f5c5-4d72-846f-1d27f9b3c1ff',
    drop_duplicates=True,

include_batched_ensemble_estimators=['BatchedTreeEnsembleClassifier(Ex
traTreesClassifier)', 'BatchedTreeEnsembleClassifier(LGBMClassifier)',
'BatchedTreeEnsembleClassifier(RandomForestClassifier)',
'BatchedTreeEnsembleClassifier(SnapBoostingMachineClassifier)',
'BatchedTreeEnsembleClassifier(SnapRandomForestClassifier)',
'BatchedTreeEnsembleClassifier(XGBClassifier)'],
    classes=['Line Breakage', 'Overheating', 'Transformer Failure'],
    feature_selector_mode='auto'
)
```

Set `n_jobs` parameter to the number of available CPUs

```
import os, ast
CPU_NUMBER = 1
if 'RUNTIME_HARDWARE_SPEC' in os.environ:
    CPU_NUMBER =
int(ast.literal_eval(os.environ['RUNTIME_HARDWARE_SPEC'])['num_cpu'])
```

watsonx.ai connection

This cell defines the credentials required to work with the watsonx.ai Runtime.

Action: Provide the IBM Cloud apikey, For details, see documentation.

```
import getpass

api_key = getpass.getpass("Please enter your api key (press enter): ")

from ibm_watsonx_ai import Credentials
```

```
credentials = Credentials(
    api_key=api_key,
    url=experiment_metadata['deployment_url']
)

from ibm_watsonx_ai import APIClient

client = APIClient(credentials)

if 'space_id' in experiment_metadata:
    client.set.default_space(experiment_metadata['space_id'])
else:
    client.set.default_project(experiment_metadata['project_id'])

training_data_references[0].set_client(client)
```

Incremental learning

Get pipeline

Download and save a pipeline model object from the AutoAI training job (`lale` pipeline type is used for inspection and `partial_fit` capabilities).

```
from ibm_watsonx_ai.experiment import AutoAI

pipeline_optimizer = AutoAI(credentials,
project_id=experiment_metadata['project_id']).runs.get_optimizer(metad
ata=experiment_metadata)
pipeline_model =
pipeline_optimizer.get_pipeline(pipeline_name='Pipeline_9',
astype='lale')
```

Data loader

Create DataLoader iterator to retrieve training dataset in batches. DataLoader is `Torch` compatible (`torch.utils.data`), returning Pandas DataFrames.

Note: If reading data results in an error, provide data as iterable reader (e.g. `read_csv()` method from Pandas with chunks). It may be necessary to use methods for initial data pre-processing like: e.g. `DataFrame.dropna()`, `DataFrame.drop_duplicates()`, `DataFrame.sample()`.

```
reader_full_data = pd.read_csv(DATA_PATH, chunksize=CHUNK_SIZE)
```

Batch size in rows.

```
number_of_batch_rows = 506

from ibm_watsonx_ai.data_loaders import experiment as data_loaders
from ibm_watsonx_ai.data_loaders.datasets import experiment as
datasets

dataset = datasets.ExperimentIterableDataset(
    connection=training_data_references[0],
    enable_sampling=False,
    experiment_metadata=experiment_metadata,
    number_of_batch_rows=number_of_batch_rows
    )

data_loader = data_loaders.ExperimentDataLoader(dataset=dataset)
```

Continue model training

In this cell, the pipeline is incrementally fitted using data batches (via `partial_fit` calls).

Note: If you need, you can evaluate the pipeline using custom holdout data. Provide the `X_test`, `y_test` and call `scorer` on them.

Define scorer from the optimization metric

This cell constructs the cell scorer based on the experiment metadata.

```
from sklearn.metrics import get_scorer

scorer = get_scorer(experiment_metadata['scoring'])
```

Tuning the incremental learner

For the best training performance set:

- `n_jobs` - to available number of CPUs.

```
pipeline_model.steps[-1]
[1].impl.base_ensemble.set_params(n_jobs=CPU_NUMBER)
```

Set up a learning curve plot

```
import matplotlib.pyplot as plt
from ibm_watsonx_ai.utils.autoai.incremental import
```

```
plot_learning_curve
import time

partial_fit_scores = []
fit_times = []
```

Fit pipeline model in batches

Tip: If the data passed to `partial_fit` is highly imbalanced (>1:10), please consider applying the `sample_weight` parameter:

```
from sklearn.utils.class_weight import compute_sample_weight

pipeline_model.partial_fit(X_train, y_train,
freeze_trained_prefix=True,

sample_weight=compute_sample_weight('balanced', y_train))
```

Note: If you have a holdout/test set please provide it for better pipeline evaluation and replace X_test and y_test in the following cell.

```
from pandas import read_csv
test_df = read_csv('DATA_PATH')

X_test = test_df.drop([experiment_metadata['prediction_column']],
axis=1).values
y_test = test_df[experiment_metadata['prediction_column']].values
```

If holdout set was not provided, 30% of first training batch would be used as holdout.

Filter warnings for incremental training.

```
import warnings

warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split

fig, axes = plt.subplots(1, 3, figsize=(18, 4))

for i, batch_df in enumerate(data_loader):
    batch_df.dropna(subset=experiment_metadata["prediction_column"],
inplace=True)
    X_train =
batch_df.drop([experiment_metadata['prediction_column']],
axis=1).values
    y_train =
```

```
batch_df[experiment_metadata['prediction_column']].values
    if i==0:
        X_train, X_test, y_train, y_test = train_test_split(X_train,
y_train, test_size=0.3)
    start_time = time.time()
    pipeline_model = pipeline_model.partial_fit(X_train, y_train,
freeze_trained_prefix=True)
    fit_times.append(time.time() - start_time)
    partial_fit_scores.append(scorer(pipeline_model, X_test, y_test))
    plot_learning_curve(fig=fig, axes=axes, scores=partial_fit_scores,
fit_times=fit_times)
```

Test pipeline model

Test the fitted pipeline (`predict`).

```
pipeline_model.predict(X_test[:10])
```

Store the model

In this section you will learn how to store the incrementally trained model.

```
model_metadata = {
    client.repository.ModelMetaNames.NAME: 'P9 - Pretrained AutoAI
pipeline'
}

stored_model_details =
client.repository.store_model(model=pipeline_model,
meta_props=model_metadata, experiment_metadata=experiment_metadata)
```

Inspect the stored model details.

```
stored_model_details
```

Create online deployment

You can use the commands below to promote the model to space and create online deployment (web service).

Working with spaces

In this section you will specify a deployment space for organizing the assets for deploying and scoring the model. If you do not have an existing space, you can use Deployment Spaces Dashboard to create a new space, following these steps:

- Click **New Deployment Space**.
- Create an empty space.
- Select Cloud Object Storage.
- Select watsonx.ai Runtime and press **Create**.
- Copy `space_id` and paste it below.

Tip: You can also use the API to prepare the space for your work. Learn more here.

Info: Below cells are `raw` type - in order to run them, change their type to `code` and run them (no need to restart the notebook). You may need to add some additional info (see the **action** below).

Action: Assign or update space ID below.

Prepare online deployment

Test online deployment

Deleting deployment

You can delete the existing deployment by calling the `client.deployments.delete(deployment_id)` command. To list the existing web services, use `client.deployments.list()`.

Summary and next steps

You've successfully completed this notebook! You've learned how to use AutoAI pipeline definition to train the model. Check out the official AutoAI site for more samples, tutorials, documentation, how-tos, and blog posts.

Copyrights

Auto-generated Notebook (License Terms), such agreements located in the link below. Specifically, the Source Components and Sample Materials clause included in the License Information document for Watson Studio Auto-generated Notebook applies to the auto-generated notebooks.

By downloading, copying, accessing, or otherwise using the materials, you agree to the License Terms

---