Practical-09 ML Model implementation and deployment using Sagemaker

1. Set Up

Before executing the notebook, there are some initial steps required for setup. This notebook requires latest version of sagemaker and ipywidgets.

[1]:
```
!pip install sagemaker ipywidgets --upgrade --quiet
```

To train and host on Amazon SageMaker, we need to setup and authenticate the use of AWS services. Here, we use the execution role associated with the current notebook instance as the AWS account role with SageMaker access. It has necessary permissions, including access to your data in S3.

[2]:
```
import sagemaker, boto3, json
from sagemaker import get_execution_role

aws_role = get_execution_role()
aws_region = boto3.Session().region_name
sess = sagemaker.Session()
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml
```

2. Train a Tabular Model on Adult Dataset

In this demonstration, we will train a tabular algorithm on the Adult dataset. The dataset contains examples of census data to predict whether a person makes over 50K a year or not. The Adult dataset is downloaded from UCI Machine Learning Repository.

Below is the table of the first 5 examples in the Adult dataset.

Target	Feature_0	Feature_1	Feature_2	Feature_3	Feature_4	...	Feature_10	Feature_11	Feature_12	Feature_13
0	25	Private	226802	11th	7	...	0	0	40	United-States
0	38	Private	89814	HS-grad	9	...	0	0	50	United-States
1	28	Local-gov	336951	Assoc-acdm	12	...	0	0	40	United-States
1	44	Private	160323	Some-college	10	...	7688	0	40	United-States
0	18	?	103497	Some-college	10	...	0	0	30	United-States

If you want to bring your own dataset, below are the instructions on how the training data should be formatted as input to the model.

A S3 path should contain two sub-directories 'train/', and 'validation/' (optional). Each sub-directory contains a 'data.csv' file (The ABALONE dataset used in this example has been prepared and saved in `training_dataset_s3_path` shown below).

- The 'data.csv' files under sub-directory 'train/' and 'validation/' are for training and validation, respectively. The validation data is used to compute a validation score at the end of each training iteration or epoch. An early stopping is applied when the validation score stops improving. If the validation data is not provided, a fraction of training data is randomly sampled to serve as the validation data. The fraction value is selected based on the number of rows in the training data. Default values range from 0.2 at 2,500 rows to 0.01 at 250,000 rows. For details, see AutoGluon-Tabular Documentation.
- The first column of the 'data.csv' should have the corresponding target variable. The rest of other columns should have the corresponding predictor variables (features).
- All the categorical and numeric features, and target can be kept as their original formats.

```python
from sagemaker import image_uris, model_uris, script_uris

train_model_id, train_model_version, train_scope = (
    "autogluon-classification-ensemble",
    "*",
    "training",
)
training_instance_type = "ml.p3.2xlarge"

# Retrieve the docker image
train_image_uri = image_uris.retrieve(
    region=None,
    framework=None,
    model_id=train_model_id,
    model_version=train_model_version,
    image_scope=train_scope,
    instance_type=training_instance_type,
)
# Retrieve the training script
train_source_uri = script_uris.retrieve(
    model_id=train_model_id, model_version=train_model_version, script_scope=train_scope
)
# Retrieve the pre-trained model tarball to further fine-tune. In tabular case, however, the pre-trained model tarball is dummy
train_model_uri = model_uris.retrieve(
    model_id=train_model_id, model_version=train_model_version, model_scope=train_scope
)
```

2.2. Set Training Parameters

Now that we are done with all the setup that is needed, we are ready to train our tabular algorithm. To begin, let us create a `sageMaker.estimator.Estimator` object. This estimator will launch the training job.

There are two kinds of parameters that need to be set for training. The first one are the parameters for the training job. These include: (i) Training data path. This is S3 folder in which the input data is stored, (ii) Output path: This the s3 folder in which the training output is stored. (iii) Training instance type: This indicates the type of machine on which to run the training.

The second set of parameters are algorithm specific training hyper-parameters.

```python
# Sample training data is available in this bucket
training_data_bucket = f"jumpstart-cache-prod-{aws_region}"
training_data_prefix = "training-datasets/tabular_binary/"

training_dataset_s3_path = f"s3://{training_data_bucket}/{training_data_prefix}"

output_bucket = sess.default_bucket()
output_prefix = "jumpstart-example-tabular-training"

s3_output_location = f"s3://{output_bucket}/{output_prefix}/output"
```

For algorithm specific hyper-parameters, we start by fetching python dictionary of the training hyper-parameters that the algorithm accepts with their default values. This can then be overridden to custom values.

```python
from sagemaker import hyperparameters

# Retrieve the default hyper-parameters for fine-tuning the model
hyperparameters = hyperparameters.retrieve_default(
    model_id=train_model_id, model_version=train_model_version
)

# [Optional] Override default hyperparameters with custom values
hyperparameters["auto_stack"] = "True"
print(hyperparameters)
```

```
{'eval_metric': 'auto', 'presets': 'medium_quality', 'auto_stack': 'True', 'num_bag_folds': '0', 'num_bag_sets': '1', 'num_stac
k_levels': '0', 'refit_full': 'False', 'set_best_to_refit_full': 'False', 'save_space': 'False', 'verbosity': '2'}
```

2.3. Start Training

2.3. Start Training

We start by creating the estimator object with all the required assets and then launch the training job. Note. We do not use hyperparameter tuning for AutoGluon models because AutoGluon succeeds by ensembling multiple models and stacking them in multiple layers rather than focusing on model/hyperparameter selection.

```
from sagemaker.estimator import Estimator
from sagemaker.utils import name_from_base

training_job_name = name_from_base(f"jumpstart-example-{train_model_id}-training")

# Create SageMaker Estimator instance
tabular_estimator = Estimator(
    role=aws_role,
    image_uri=train_image_uri,
    source_dir=train_source_uri,
    model_uri=train_model_uri,
    entry_point="transfer_learning.py",
    instance_count=1,
    instance_type=training_instance_type,
    max_run=360000,
    hyperparameters=hyperparameters,
    output_path=s3_output_location,
)
```

3. Deploy and Run Inference on the Trained Tabular Model

In this section, you learn how to query an existing endpoint and make predictions of the examples you input. For each example, the model will output the probability of the sample for each class in the model. Next, the predicted class label is obtained by taking the class label with the maximum probability over others. Throughout the notebook, the examples are taken from the Adult test set. The dataset contains examples of census data to predict whether a person makes over 50K a year or not.

We start by retrieving the artifacts and deploy the `tabular_estimator` that we trained.

```
inference_instance_type = "ml.m5.2xlarge"

# Retrieve the inference docker container uri
deploy_image_uri = image_uris.retrieve(
    region=None,
    framework=None,
    image_scope="inference",
    model_id=train_model_id,
    model_version=train_model_version,
    instance_type=inference_instance_type,
)
# Retrieve the inference script uri
deploy_source_uri = script_uris.retrieve(
    model_id=train_model_id, model_version=train_model_version, script_scope="inference"
)

endpoint_name = name_from_base(f"jumpstart-example-{train_model_id}-")

# Use the estimator from the previous step to deploy to a SageMaker endpoint
predictor = tabular_estimator.deploy(
    initial_instance_count=1,
    instance_type=inference_instance_type,
```

Next, we download a hold-out Adult test data from the S3 bucket for inference.

```
jumpstart_assets_bucket = f"jumpstart-cache-prod-{aws_region}"
test_data_prefix = "training-datasets/tabular_binary/test"
test_data_file_name = "data.csv"

boto3.client("s3").download_file(
    jumpstart_assets_bucket, f"{test_data_prefix}/{test_data_file_name}", test_data_file_name
)
```

]:
```
newline, bold, unbold = "\n", "\033[1m", "\033[0m"

import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

# read the data
test_data = pd.read_csv(test_data_file_name, header=None)
test_data.columns = ["Target"] + [f"Feature_{i}" for i in range(1, test_data.shape[1])]

num_examples, num_columns = test_data.shape
print(
    f"{bold}The test dataset contains {num_examples} examples and {num_columns} columns.{unbold}\n"
)

# prepare the ground truth target and predicting features to send into the endpoint.
ground_truth_label, features = test_data.iloc[:, :1], test_data.iloc[:, 1:]

print(f"{bold}The first 5 observations of the data: {unbold} \n")
test_data.head(5)
```

The following code queries the endpoint you have created to get the prediction for each test example. The `query_endpoint()` function returns an array-like of shape (num_examples, num_classes), where each row indicates the probability of the example for each class in the model. The num_classes is 2 in above test data. Next, the predicted class label is obtained by taking the class label with the maximum probability over others for each example.

4. Evaluate the Prediction Results Returned from the Endpoint

We evaluate the predictions results returned from the endpoint by following two ways.

- Visualize the predictions results by plotting the confusion matrix.
- Measure the prediction results quantitatively.

]:
```
# Visualize the predictions results by plotting the confusion matrix.
conf_matrix = confusion_matrix(y_true=ground_truth_label.values, y_pred=predict_label)
fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i, s=conf_matrix[i, j], va="center", ha="center", size="xx-large")

plt.xlabel("Predictions", fontsize=18)
plt.ylabel("Actuals", fontsize=18)
plt.title("Confusion Matrix", fontsize=18)
plt.show()
```

]:
```
# Measure the prediction results quantitatively.
eval_accuracy = accuracy_score(ground_truth_label.values, predict_label)
eval_f1 = f1_score(ground_truth_label.values, predict_label)

print(
    f"{bold}Evaluation result on test data{unbold}:{newline}"
    f"{bold}{accuracy_score.__name__}{unbold}: {eval_accuracy}{newline}"
    f"{bold}F1 {unbold}: {eval_f1}{newline}"
)
```

Next, we delete the endpoint corresponding to the trained model.

]:
```
# Delete the SageMaker endpoint and the attached resources
predictor.delete_model()
predictor.delete_endpoint()
```