AutoAI | Part of IBM Watson® Studio Pipeline Notebook

Pipeline 4 Notebook - AutoAI Notebook v1.15.4

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-optimum 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 (https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/autoai-notebook.html)
- 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 a Scikit-learn representation of AutoAI pipeline. This notebook introduces commands for getting data, training the model, and testing the model.

Some familiarity with Python is helpful. This notebook uses Python 3.7 and scikit-learn 0.23.2.

Notebook goals

- Scikit-learn pipeline definition
- Pipeline training
- 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_watson_machine_learning,
- autoai-libs,
- scikit-learn.

In [ ]:

```
!pip install ibm-watson-machine-learning | tail -n 1
!pip install -U autoai-libs==1.12.11 | tail -n 1
!pip install -U scikit-learn==0.23.2 | 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.

In [ ]:

```
from ibm_watson_machine_learning.helpers import DataConnection
from ibm_watson_machine_learning.helpers import S3Connection, S3Location

training_data_reference = [
    DataConnection(
    connection=S3Connection(
        api_key='Wn1mv_wiCAQLb5RNwa9dlxqq33jZuvihrkMYdR_XGSFU',
        auth_endpoint='https://iam.bluemix.net/oidc/token/',
        endpoint_url='https://s3.ap.cloud-object-storage.appdomain.cloud'
    ),
        location=S3Location(
            bucket='aiassistedfarming-donotdelete-pr-2wvfp8awhov9lh',
            path='crop_production.csv'
        )
    ),
]
training_result_reference = DataConnection(
    connection=S3Connection(
        api_key='Wn1mv_wiCAQLb5RNwa9dlxqq33jZuvihrkMYdR_XGSFU',
        auth_endpoint='https://iam.bluemix.net/oidc/token/',
        endpoint_url='https://s3.ap.cloud-object-storage.appdomain.cloud'
    ),
    location=S3Location(
        bucket='aiassistedfarming-donotdelete-pr-2wvfp8awhov9lh',
        path='auto_ml/7ac7b3bd-11ae-4262-bf54-72984a103667/wml_data/4fc35094-932
d-40ce-ab77-d9540f3463ff/data/automl',
        model_location='auto_ml/7ac7b3bd-11ae-4262-bf54-72984a103667/wml_data/4f
c35094-932d-40ce-ab77-d9540f3463ff/data/automl/hpo_c_output/Pipeline1/model.pick
le',
        training_status='auto_ml/7ac7b3bd-11ae-4262-bf54-72984a103667/wml_data/4
fc35094-932d-40ce-ab77-d9540f3463ff/training-status.json'
    )
)
```

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

In [ ]:

```
experiment_metadata = dict(
    prediction_type='regression',
    prediction_column='Production',
    holdout_size=0.15,
    scoring='neg_root_mean_squared_error',
    csv_separator=',',
    random_state=33,
    max_number_of_estimators=2,
    training_data_reference=training_data_reference,
    training_result_reference=training_result_reference,
    deployment_url='https://jp-tok.ml.cloud.ibm.com',
    project_id='c6fb71fa-de3b-4e4e-b8c1-3b927c1b305b',
    drop_duplicates=False
)
```

Pipeline inspection

Read training data

Retrieve training dataset from AutoAI experiment as pandas DataFrame.

Note: If reading data results in an error, provide data as Pandas DataFrame object, for example, reading .CSV file with `pandas.read_csv()`

In [ ]:

```
df = training_data_reference[0].read(experiment_metadata=experiment_metadata)
df.dropna('rows', how='any', subset=[experiment_metadata['prediction_column']],
inplace=True)
```

Train and test data split

In [ ]:

```
from sklearn.model_selection import train_test_split

X = df.drop([experiment_metadata['prediction_column']], axis=1).values
y = df[experiment_metadata['prediction_column']].values

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=experiment_m
etadata['holdout_size'],
                                                    random_state=experiment_meta
data['random_state'])
```

Make pipeline

In the next cell, you can find the Scikit-learn definition of the selected AutoAI pipeline.

Import statements.

In [ ]:

```
from autoai_libs.transformers.exportable import NumpyColumnSelector
from autoai_libs.transformers.exportable import CompressStrings
from autoai_libs.transformers.exportable import NumpyReplaceMissingValues
from autoai_libs.transformers.exportable import NumpyReplaceUnknownValues
from autoai_libs.transformers.exportable import boolean2float
from autoai_libs.transformers.exportable import CatImputer
from autoai_libs.transformers.exportable import CatEncoder
import numpy as np
from autoai_libs.transformers.exportable import float32_transform
from sklearn.pipeline import make_pipeline
from autoai_libs.transformers.exportable import FloatStr2Float
from autoai_libs.transformers.exportable import NumImputer
from autoai_libs.transformers.exportable import OptStandardScaler
from sklearn.pipeline import make_union
from autoai_libs.transformers.exportable import NumpyPermuteArray
from autoai_libs.cognito.transforms.transform_utils import TA1
import autoai_libs.utils.fc_methods
from autoai_libs.cognito.transforms.transform_utils import FS1
from autoai_libs.cognito.transforms.transform_utils import TA2
from sklearn.tree import DecisionTreeRegressor
```

Pre-processing & Estimator.

In [ ]:

```
numpy_column_selector_0 = NumpyColumnSelector(columns=[0, 1, 2, 3, 4])
compress_strings = CompressStrings(
    compress_type="hash",
    dtypes_list=["char_str", "char_str", "int_num", "char_str", "char_str"],
    missing_values_reference_list=["", "-", "?", float("nan")],
    misslist_list=[[], [], [], [], []],
)
numpy_replace_missing_values_0 = NumpyReplaceMissingValues(
    missing_values=[], filling_values=float("nan")
)
numpy_replace_unknown_values = NumpyReplaceUnknownValues(
    filling_values=float("nan"),
    filling_values_list=[
        float("nan"), float("nan"), float("nan"), float("nan"), float("nan"),
    ],
    missing_values_reference_list=["", "-", "?", float("nan")],
)
cat_imputer = CatImputer(
    strategy="most_frequent",
    missing_values=float("nan"),
    sklearn_version_family="23",
)
cat_encoder = CatEncoder(
    encoding="ordinal",
    categories="auto",
    dtype=np.float64,
    handle_unknown="error",
    sklearn_version_family="23",
)
pipeline_0 = make_pipeline(
    numpy_column_selector_0,
    compress_strings,
    numpy_replace_missing_values_0,
    numpy_replace_unknown_values,
    boolean2float(),
    cat_imputer,
    cat_encoder,
    float32_transform(),
)
numpy_column_selector_1 = NumpyColumnSelector(columns=[5])
float_str2_float = FloatStr2Float(
    dtypes_list=["float_num"], missing_values_reference_list=[]
)
numpy_replace_missing_values_1 = NumpyReplaceMissingValues(
    missing_values=[], filling_values=float("nan")
)
num_imputer = NumImputer(strategy="median", missing_values=float("nan"))
opt_standard_scaler = OptStandardScaler(
    num_scaler_copy=None,
    num_scaler_with_mean=None,
    num_scaler_with_std=None,
    use_scaler_flag=False,
)
pipeline_1 = make_pipeline(
    numpy_column_selector_1,
    float_str2_float,
    numpy_replace_missing_values_1,
    num_imputer,
    opt_standard_scaler,
```

```
    float32_transform(),
)
union = make_union(pipeline_0, pipeline_1)
numpy_permute_array = NumpyPermuteArray(
    axis=0, permutation_indices=[0, 1, 2, 3, 4, 5]
)
ta1 = TA1(
    fun=np.square,
    name="square",
    datatypes=["numeric"],
    feat_constraints=[autoai_libs.utils.fc_methods.is_not_categorical],
    col_names=[
        "State_Name", "District_Name", "Crop_Year", "Season", "Crop", "Area",
    ],
    col_dtypes=[
        np.dtype("float32"), np.dtype("float32"), np.dtype("float32"),
        np.dtype("float32"), np.dtype("float32"), np.dtype("float32"),
    ],
)
fs1_0 = FS1(
    cols_ids_must_keep=range(0, 6),
    additional_col_count_to_keep=8,
    ptype="regression",
)
ta2 = TA2(
    fun=np.add,
    name="sum",
    datatypes1=[
        "intc", "intp", "int_", "uint8", "uint16", "uint32", "uint64", "int8",
        "int16", "int32", "int64", "short", "long", "longlong", "float16",
        "float32", "float64",
    ],
    feat_constraints1=[autoai_libs.utils.fc_methods.is_not_categorical],
    datatypes2=[
        "intc", "intp", "int_", "uint8", "uint16", "uint32", "uint64", "int8",
        "int16", "int32", "int64", "short", "long", "longlong", "float16",
        "float32", "float64",
    ],
    feat_constraints2=[autoai_libs.utils.fc_methods.is_not_categorical],
    col_names=[
        "State_Name", "District_Name", "Crop_Year", "Season", "Crop", "Area",
        "square(State_Name)", "square(District_Name)", "square(Crop)",
        "square(Area)",
    ],
    col_dtypes=[
        np.dtype("float32"), np.dtype("float32"), np.dtype("float32"),
        np.dtype("float32"), np.dtype("float32"), np.dtype("float32"),
        np.dtype("float32"), np.dtype("float32"), np.dtype("float32"),
        np.dtype("float32"),
    ],
)
fs1_1 = FS1(
    cols_ids_must_keep=range(0, 6),
    additional_col_count_to_keep=8,
    ptype="regression",
)
decision_tree_regressor = DecisionTreeRegressor(
    max_depth=5,
    max_features=0.9984040078576402,
    presort=False,
```

```
    random_state=33,
)
```

Pipeline.

In [ ]:

```
pipeline = make_pipeline(
    union,
    numpy_permute_array,
    ta1,
    fs1_0,
    ta2,
    fs1_1,
    decision_tree_regressor,
)
```

Train pipeline model

Define scorer from the optimization metric

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

In [ ]:

```
from sklearn.metrics import get_scorer

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

Fit pipeline model

In this cell, the pipeline is fitted.

In [ ]:

```
pipeline.fit(train_X,train_y)
```

Test pipeline model

Score the fitted pipeline with the generated scorer using the holdout dataset.

In [ ]:

```
score = scorer(pipeline, test_X, test_y)
print(score)
```

Calling the `predict` method

If you want to get a prediction using pipeline model object, call `pipeline.predict()` .

Note: If you want to work with pure sklearn model:

- add the following parameter to `get_pipeline` call: `astype='sklearn'` ,
- or `scikit_learn_pipeline = pipeline.export_to_sklearn_pipeline()`

In [ ]:

```
pipeline.predict(test_X)
```

Next steps

Model deployment as webservice (https://github.com/IBM/watson-machine-learning-samples/tree/master/cloud/notebooks/python_sdk/deployments/autoai)

Run AutoAI experiment with python SDK (https://github.com/IBM/watson-machine-learning-samples/tree/master/cloud/notebooks/python_sdk/experiments/autoai)

Copyrights