onnx_sklearn_functions

February 4, 2023

1 Use function when converting into ONNX

Once a a scikit-learn model is converting into ONNX, there is no easy way to retrieve the original scikit-learn model. The following notebook explores an alternative way to convert a model into ONNX by using functions. In this new method, every piece of a pipeline becomes a function.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

[2]:
```
%matplotlib inline
```

[3]:
```
%load_ext mlprodict
```

1.1 A pipeline

[4]:
```
from sklearn.pipeline import Pipeline
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn import set_config
set_config(display="diagram")

data = load_iris()
X, y = data.data, data.target
steps = [
    ("preprocessing", StandardScaler()),
    ("classifier", LogisticRegression(penalty='l1', solver="liblinear"))]
pipe = Pipeline(steps)
pipe.fit(X, y)
```

[4]:
```
Pipeline(steps=[('preprocessing', StandardScaler()),
                ('classifier',
                 LogisticRegression(penalty='l1', solver='liblinear'))])
```

1.2 Its conversion into ONNX

1.2.1 Without functions

[5]:
```
from mlprodict.plotting.text_plot import onnx_simple_text_plot
from mlprodict.onnx_conv import to_onnx

onx = to_onnx(pipe, X, options={'zipmap': False})
print(onnx_simple_text_plot(onx))
```

```
opset: domain='' version=14
opset: domain='ai.onnx.ml' version=1
input: name='X' type=dtype('float64') shape=[None, 4]
init: name='Su_Subcst' type=dtype('float64') shape=(4,) -- array([5.84333333,
3.05733333, 3.758     , 1.19933333])
init: name='Di_Divcst' type=dtype('float64') shape=(4,) -- array([0.82530129,
0.43441097, 1.75940407, 0.75969263])
init: name='coef' type=dtype('float64') shape=(12,)
init: name='intercept' type=dtype('float64') shape=(3,) -- array([-1.86506089,
-0.89658497, -4.56614529])
init: name='classes' type=dtype('int32') shape=(3,) -- array([0, 1, 2])
init: name='shape_tensor' type=dtype('int64') shape=(1,) -- array([-1],
dtype=int64)
init: name='axis' type=dtype('int64') shape=(1,) -- array([1], dtype=int64)
Sub(X, Su_Subcst) -> Su_C0
  Div(Su_C0, Di_Divcst) -> variable
    MatMul(variable, coef) -> multiplied
      Add(multiplied, intercept) -> raw_scores
        Sigmoid(raw_scores) -> raw_scoressig
          Abs(raw_scoressig) -> norm_abs
            ReduceSum(norm_abs, axis, keepdims=1) -> norm
          Div(raw_scoressig, norm) -> probabilities
        ArgMax(raw_scores, axis=1) -> label1
          ArrayFeatureExtractor(classes, label1) ->
array_feature_extractor_result
            Cast(array_feature_extractor_result, to=11) -> cast2_result
              Reshape(cast2_result, shape_tensor) -> reshaped_result
                Cast(reshaped_result, to=7) -> label
output: name='label' type=dtype('int64') shape=[None]
output: name='probabilities' type=dtype('float64') shape=[None, 3]
```

[6]:
```
%onnxview onx
```

[6]:
```
<jyquickhelper.jspy.render_nb_js_dot.RenderJsDot at 0x193d5fd0ca0>
```

1.2.2 With functions

[7]:
```
onxf = to_onnx(pipe, X, as_function=True, options={'zipmap': False})
print(onnx_simple_text_plot(onxf))
```

```
No CUDA runtime is found, using CUDA_HOME='C:\Program Files\NVIDIA GPU Computing
Toolkit\CUDA\v11.5'
opset: domain='' version=15
opset: domain='sklearn' version=1
input: name='X' type=dtype('float64') shape=[None, 4]
main___Pipeline_1734459081968[sklearn](X) -> main_classifier_label,
```

```
main_classifier_probabilities
output: name='main_classifier_label' type=dtype('int64') shape=[None]
output: name='main_classifier_probabilities' type=dtype('float64') shape=[None,
3]
----- function name=main__preprocessing___StandardScaler_1734202136896
domain=sklearn
----- doc_string: HYPER:{"StandardScaler":{"copy": true, "with_mean": true,
"with_std": true}}
opset: domain='' version=14
input: 'X'
Constant(value=[5.8433333…) -> Su_Subcst
  Sub(X, Su_Subcst) -> Su_C0
Constant(value=[0.8253012…) -> Di_Divcst
  Div(Su_C0, Di_Divcst) -> variable
output: name='variable' type=? shape=?
----- function name=main__classifier___LogisticRegression_1734202137184
domain=sklearn
----- doc_string: HYPER:{"LogisticRegression":{"C": 1.0, "class_weight": null,
"dual": false, "fit_intercept": true, "intercept_scaling": 1, "l1_ratio": null,
"max_iter": 100, "multi_class": "auto", "n_jobs": null, "penalty": "l1",
"random_state": null, "solver": "liblinear", "tol": 0.0001, "verbose": 0,
"warm_start": false}}
opset: domain='' version=13
opset: domain='ai.onnx.ml' version=1
input: 'X0'
Constant(value=[[0.0, 0.0…) -> coef
  MatMul(X0, coef) -> multiplied
Constant(value=[[-1.86506…) -> intercept
  Add(multiplied, intercept) -> raw_scores
    ArgMax(raw_scores, axis=1) -> label1
Constant(value=[0, 1, 2]) -> classes
  ArrayFeatureExtractor(classes, label1) -> array_feature_extractor_result
    Cast(array_feature_extractor_result, to=11) -> cast2_result
Constant(value=[-1]) -> shape_tensor
  Reshape(cast2_result, shape_tensor) -> reshaped_result
    Cast(reshaped_result, to=7) -> label
Constant(value=[1]) -> axis
Sigmoid(raw_scores) -> raw_scoressig
  Abs(raw_scoressig) -> norm_abs
  ReduceSum(norm_abs, axis, keepdims=1) -> norm
  Div(raw_scoressig, norm) -> probabilities
output: name='label' type=? shape=?
output: name='probabilities' type=? shape=?
----- function name=main___Pipeline_1734459081968 domain=sklearn
----- doc_string: HYPER:{"Pipeline":{"memory": null, "steps": [["preprocessing",
"{\"classname\": \"StandardScaler\", \"EXC\": \"Object of type StandardScaler is
not JSON serializable\"}"], ["classifier", "{\"classname\":
\"LogisticRegression\", \"EXC\": \"Object of type LogisticRegression is not JSON
serializable\"}"]], "verbose": false}}
opset: domain='' version=15
opset: domain='sklearn' version=1
input: 'X'
main__preprocessing___StandardScaler_1734202136896[sklearn](X) ->
preprocessing_variable
```

```
  main__classifier___LogisticRegression_1734202137184[sklearn](preprocessing_var
iable) -> classifier_label, classifier_probabilities
output: name='classifier_label' type=? shape=?
output: name='classifier_probabilities' type=? shape=?
```

[8]:
```
%onnxview onxf
```

[8]:
```
<jyquickhelper.jspy.render_nb_js_dot.RenderJsDot at 0x193c63143a0>
```

Based on that, it should be possible to rebuild the original scikit-learn pipeline. Hyperparameters are stored in the attribute `doc_string`.

1.3 A more complex one

[9]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler

data = load_iris()
X, y = data.data, data.target
steps = [
    ("preprocessing", ColumnTransformer([
        ('A', StandardScaler(), [0, 1]),
        ('B', MinMaxScaler(), [2, 3])])),
    ("classifier", LogisticRegression(penalty='l1', solver="liblinear"))]
pipe = Pipeline(steps)
pipe.fit(X, y)
```

[9]:
```
Pipeline(steps=[('preprocessing',
                 ColumnTransformer(transformers=[('A', StandardScaler(),
                                                  [0, 1]),
                                                 ('B', MinMaxScaler(),
                                                  [2, 3])])),
                ('classifier',
                 LogisticRegression(penalty='l1', solver='liblinear'))])
```

[10]:
```
onxf = to_onnx(pipe, X, as_function=True, options={'zipmap': False})
print(onnx_simple_text_plot(onxf))
```

```
opset: domain='' version=15
opset: domain='sklearn' version=1
input: name='X' type=dtype('float64') shape=[None, 4]
main___Pipeline_1734198554880[sklearn](X) -> main_classifier_label,
main_classifier_probabilities
output: name='main_classifier_label' type=dtype('int64') shape=[None]
output: name='main_classifier_probabilities' type=dtype('float64') shape=[None,
3]
----- function name=main__preprocessing__B___MinMaxScaler_1734196938256
domain=sklearn
----- doc_string: HYPER:{"MinMaxScaler":{"clip": false, "copy": true,
"feature_range": [0, 1]}}
opset: domain='' version=14
input: 'X'
Cast(X, to=11) -> Ca_output0
```

```
Constant(value=[0.1694915…) -> Mu_Mulcst
  Mul(Ca_output0, Mu_Mulcst) -> Mu_C0
Constant(value=[-0.169491…) -> Ad_Addcst
  Add(Mu_C0, Ad_Addcst) -> variable
output: name='variable' type=? shape=?
----- function name=main__preprocessing__A___StandardScaler_1734196937584
domain=sklearn
----- doc_string: HYPER:{"StandardScaler":{"copy": true, "with_mean": true,
"with_std": true}}
opset: domain='' version=14
input: 'X'
Constant(value=[5.8433333…) -> Su_Subcst
  Sub(X, Su_Subcst) -> Su_C0
Constant(value=[0.8253012…) -> Di_Divcst
  Div(Su_C0, Di_Divcst) -> variable
output: name='variable' type=? shape=?
----- function name=main__preprocessing___ColumnTransformer_1734520793072
domain=sklearn
----- doc_string: HYPER:{"ColumnTransformer":{"n_jobs": null, "remainder":
"drop", "sparse_threshold": 0.3, "transformer_weights": null, "transformers":
[["A", "{\"classname\": \"StandardScaler\", \"EXC\": \"Object of type
StandardScaler is not JSON serializable\"}", [0, 1]], ["B", "{\"classname\":
\"MinMaxScaler\", \"EXC\": \"Object of type MinMaxScaler is not JSON
serializable\"}", [2, 3]]], "verbose": false, "verbose_feature_names_out":
true}}
opset: domain='' version=15
opset: domain='sklearn' version=1
input: 'X'
Constant(value=[2]) -> init
Constant(value=[4]) -> init_1
Constant(value=[1]) -> init_2
  Slice(X, init, init_1, init_2) -> out_sli_0
    main__preprocessing__B___MinMaxScaler_1734196938256[sklearn](out_sli_0) ->
B_variable
Constant(value=[0]) -> init_3
  Slice(X, init_3, init, init_2) -> out_sli_0_1
    main__preprocessing__A___StandardScaler_1734196937584[sklearn](out_sli_0_1)
-> A_variable
      Concat(A_variable, B_variable, axis=1) -> out_con_0
output: name='out_con_0' type=? shape=?
----- function name=main__classifier___LogisticRegression_1734520717568
domain=sklearn
----- doc_string: HYPER:{"LogisticRegression":{"C": 1.0, "class_weight": null,
"dual": false, "fit_intercept": true, "intercept_scaling": 1, "l1_ratio": null,
"max_iter": 100, "multi_class": "auto", "n_jobs": null, "penalty": "l1",
"random_state": null, "solver": "liblinear", "tol": 0.0001, "verbose": 0,
"warm_start": false}}
opset: domain='' version=13
opset: domain='ai.onnx.ml' version=1
input: 'X0'
Constant(value=[[-2.74108…) -> coef
  MatMul(X0, coef) -> multiplied
Constant(value=[[0.0, -0…) -> intercept
  Add(multiplied, intercept) -> raw_scores
```

```
    ArgMax(raw_scores, axis=1) -> label1
Constant(value=[0, 1, 2]) -> classes
  ArrayFeatureExtractor(classes, label1) -> array_feature_extractor_result
    Cast(array_feature_extractor_result, to=11) -> cast2_result
Constant(value=[-1]) -> shape_tensor
  Reshape(cast2_result, shape_tensor) -> reshaped_result
    Cast(reshaped_result, to=7) -> label
Constant(value=[1]) -> axis
Sigmoid(raw_scores) -> raw_scoressig
  Abs(raw_scoressig) -> norm_abs
  ReduceSum(norm_abs, axis, keepdims=1) -> norm
  Div(raw_scoressig, norm) -> probabilities
output: name='label' type=? shape=?
output: name='probabilities' type=? shape=?
----- function name=main___Pipeline_1734198554880 domain=sklearn
----- doc_string: HYPER:{"Pipeline":{"memory": null, "steps": [["preprocessing",
"{\"classname\": \"ColumnTransformer\", \"EXC\": \"Object of type
ColumnTransformer is not JSON serializable\"}"], ["classifier", "{\"classname\":
\"LogisticRegression\", \"EXC\": \"Object of type LogisticRegression is not JSON
serializable\"}"]], "verbose": false}}
opset: domain='' version=15
opset: domain='sklearn' version=1
input: 'X'
main__preprocessing___ColumnTransformer_1734520793072[sklearn](X) ->
preprocessing_out_con_0
  main__classifier___LogisticRegression_1734520717568[sklearn](preprocessing_out
_con_0) -> classifier_label, classifier_probabilities
output: name='classifier_label' type=? shape=?
output: name='classifier_probabilities' type=? shape=?
```

[11]:
```
%onnxview onxf
```

[11]:
```
<jyquickhelper.jspy.render_nb_js_dot.RenderJsDot at 0x193b6261550>
```

[12]:
```

```