

# onnx\_discrepancies

August 14, 2022

## 1 Discrepancies with ONNX

The notebook shows one example where the conversion leads with discrepancies if default options are used. It converts a pipeline with two steps, a scaler followed by a tree.

The bug this notebook is tracking does not always appear, it has a better chance to happen with integer features but that's not always the case. The notebook must be run again in that case.

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

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

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

### 1.1 Data and first model

We take a random datasets with mostly integers.

```
[3]: import math
      import numpy
      from sklearn.datasets import make_regression
      from sklearn.model_selection import train_test_split

      X, y = make_regression(10000, 10)
      X_train, X_test, y_train, y_test = train_test_split(X, y)

      Xi_train, yi_train = X_train.copy(), y_train.copy()
      Xi_test, yi_test = X_test.copy(), y_test.copy()
      for i in range(X.shape[1]):
          Xi_train[:, i] = (Xi_train[:, i] * math.pi * 2 ** i).astype(numpy.int64)
          Xi_test[:, i] = (Xi_test[:, i] * math.pi * 2 ** i).astype(numpy.int64)
```

```
[4]: from sklearn.pipeline import Pipeline
      from sklearn.preprocessing import StandardScaler
      from sklearn.tree import DecisionTreeRegressor

      max_depth = 10

      model = Pipeline([
          ('scaler', StandardScaler()),
          ('dt', DecisionTreeRegressor(max_depth=max_depth))
      ])
```

```
model.fit(Xi_train, yi_train)
```

```
[4]: Pipeline(steps=[('scaler', StandardScaler()),  
                    ('dt', DecisionTreeRegressor(max_depth=10))])
```

```
[5]: model.predict(Xi_test[:5])
```

```
[5]: array([-283.03708629,  263.17931397, -160.34784206, -126.59514441,  
          -150.1963714 ])
```

Other models:

```
[6]: model2 = Pipeline([  
    ('scaler', StandardScaler()),  
    ('dt', DecisionTreeRegressor(max_depth=max_depth))  
])  
model3 = Pipeline([  
    ('scaler', StandardScaler()),  
    ('dt', DecisionTreeRegressor(max_depth=3))  
])  
  
models = [  
    ('bug', Xi_test.astype(numpy.float32), model),  
    ('no scaler', Xi_test.astype(numpy.float32),  
     DecisionTreeRegressor(max_depth=max_depth).fit(Xi_train, yi_train)),  
    ('float', X_test.astype(numpy.float32),  
     model2.fit(X_train, y_train)),  
    ('max_depth=3', X_test.astype(numpy.float32),  
     model3.fit(X_train, y_train))  
]
```

## 1.2 Conversion to ONNX

```
[7]: import numpy  
from mlproduct.onnx_conv import to_onnx  
  
onx = to_onnx(model, X_train[:1].astype(numpy.float32))
```

```
[8]: from mlproduct.onnxrt import OnnxInference  
  
oinfpy = OnnxInference(onx, runtime="python_compiled")  
print(oinfpy)
```

```
OnnxInference(...)  
def compiled_run(dict_inputs):  
    # inputs  
    X = dict_inputs['X']  
    (variable1, ) = n0_scaler(X)  
    (variable, ) = n1_treeensembleregressor(variable1)  
    return {  
        'variable': variable,  
    }
```

```
[9]: import pandas

X32 = Xi_test.astype(numpy.float32)
y_skl = model.predict(X32)

obs = [dict(runtime='sklearn', diff=0)]
for runtime in ['python', 'python_compiled', 'onnxruntime1']:
    oinf = OnnxInference(onx, runtime=runtime)
    y_onx = oinf.run({'X': X32})['variable']
    delta = numpy.abs(y_skl - y_onx.ravel())
    am = delta.argmax()
    obs.append(dict(runtime=runtime, diff=delta.max()))
    obs[-1]['v[%d]' % am] = y_onx.ravel()[am]
    obs[0]['v[%d]' % am] = y_skl.ravel()[am]

pandas.DataFrame(obs)
```

```
[9]:          runtime      diff    v[1583]
0         sklearn    0.000000 -439.590635
1           python  133.641599 -305.949036
2  python_compiled  133.641599 -305.949036
3     onnxruntime1  133.641599 -305.949036
```

The pipeline shows huge discrepancies. They appear for a pipeline *StandardScaler* + *DecisionTreeRegressor* applied in integer features. They disappear if floats are used, or if the scaler is removed. The bug also disappear if the tree is not big enough (*max\_depth=4* instead of 5).

```
[10]: obs = [dict(runtime='sklearn', diff=0, name='sklearn')]
for name, x32, mod in models:
    for runtime in ['python', 'python_compiled', 'onnxruntime1']:
        lonx = to_onnx(mod, x32[:1])
        loinf = OnnxInference(lonx, runtime=runtime)
        y_skl = mod.predict(X32)
        y_onx = loinf.run({'X': X32})['variable']
        delta = numpy.abs(y_skl - y_onx.ravel())
        am = delta.argmax()
        obs.append(dict(runtime=runtime, diff=delta.max(), name=name))
        obs[-1]['v[%d]' % am] = y_onx.ravel()[am]
        obs[0]['v[%d]' % am] = y_skl.ravel()[am]

df = pandas.DataFrame(obs)
df
```

```
[10]:          runtime      diff      name    v[1583]    v[1109]  \
0         sklearn    0.000000    sklearn -439.590635  516.084502
1           python  133.641599         bug -305.949036         NaN
2  python_compiled  133.641599         bug -305.949036         NaN
3     onnxruntime1  133.641599         bug -305.949036         NaN
4           python    0.000029    no scaler         NaN  516.084473
5  python_compiled    0.000029    no scaler         NaN  516.084473
6     onnxruntime1    0.000029    no scaler         NaN  516.084473
7           python    0.000029      float         NaN         NaN
8  python_compiled    0.000029      float         NaN         NaN
9     onnxruntime1    0.000029      float         NaN         NaN
```

```

10         python      0.000003  max_depth=3          NaN          NaN
11  python_compiled  0.000003  max_depth=3          NaN          NaN
12     onnxruntime1  0.000003  max_depth=3          NaN          NaN

```

```

      v[19]      v[4]
0 -549.753386 -97.726497
1         NaN         NaN
2         NaN         NaN
3         NaN         NaN
4         NaN         NaN
5         NaN         NaN
6         NaN         NaN
7 -549.753357         NaN
8 -549.753357         NaN
9 -549.753357         NaN
10         NaN -97.726494
11         NaN -97.726494
12         NaN -97.726494

```

```
[11]: df.pivot("runtime", "name", "diff")
```

```
[11]: name          bug      float  max_depth=3  no scaler  sklearn
runtime
onnxruntime1    133.641599  0.000029    0.000003    0.000029    NaN
python          133.641599  0.000029    0.000003    0.000029    NaN
python_compiled 133.641599  0.000029    0.000003    0.000029    NaN
sklearn                NaN         NaN         NaN         NaN     0.0

```

### 1.3 Other way to convert

ONNX does not support double for TreeEnsembleRegressor but that a new operator TreeEnsembleRegressorDouble was implemented into *mlproduct*. We need to update the conversion.

```
[12]: %load_ext mlproduct
```

```
[13]: onx32 = to_onnx(model, X_train[:,1].astype(numpy.float32))
onx64 = to_onnx(model, X_train[:,1].astype(numpy.float64),
              rewrite_ops=True)
%onnxview onx64

```

```
[13]: <jyquickhelper.jspy.render_nb_js_dot.RenderJsDot at 0x1c394fc1048>
```

```
[14]: X32 = Xi_test.astype(numpy.float32)
X64 = Xi_test.astype(numpy.float64)

obs = [dict(runtime='sklearn', diff=0)]
for runtime in ['python', 'python_compiled', 'onnxruntime1']:
    for name, onx, xr in [('float', onx32, X32), ('double', onx64, X64)]:
        try:
            oinf = OnnxInference(onx, runtime=runtime)
        except Exception as e:
            obs.append(dict(runtime=runtime, error=str(e), real=name))
            continue
    y_skl = model.predict(xr)

```

```

y_onx = oinf.run({'X': xr})['variable']
delta = numpy.abs(y_skl - y_onx.ravel())
am = delta.argmax()
obs.append(dict(runtime=runtime, diff=delta.max(), real=name))
obs[-1]['v[%d]' % am] = y_onx.ravel()[am]
obs[0]['v[%d]' % am] = y_skl.ravel()[am]

```

```
pandas.DataFrame(obs)
```

```

[14]:
      runtime      diff  v[1583]  v[0]  real  \
0      sklearn  0.000000 -439.590635 -283.037086  NaN
1      python  133.641599 -305.949036      NaN  float
2      python   0.000000      NaN -283.037086  double
3  python_compiled  133.641599 -305.949036      NaN  float
4  python_compiled   0.000000      NaN -283.037086  double
5  onnxruntime1  133.641599 -305.949036      NaN  float
6  onnxruntime1      NaN      NaN      NaN  double

                                     error
0                                     NaN
1                                     NaN
2                                     NaN
3                                     NaN
4                                     NaN
5                                     NaN
6  Unable to create InferenceSession due to '[ONN...

```

We see that the use of double removes the discrepancies.

## 1.4 OnnxPipeline

Another way to reduce the number of discrepancies is to use a pipeline which converts every steps into ONNX before training the next one. That way, every steps is either trained on the inputs, either trained on the outputs produced by ONNX. Let's see how it works.

```

[15]: from mlproduct.sklearn import OnnxPipeline

model_onx = OnnxPipeline([
    ('scaler', StandardScaler()),
    ('dt', DecisionTreeRegressor(max_depth=max_depth))
])
model_onx.fit(Xi_train, yi_train)

```

```

C:\xavierdupre\_home_\github_fork\scikit-learn\sklearn\base.py:209:
FutureWarning: From version 0.24, get_params will raise an AttributeError if a
parameter cannot be retrieved as an instance attribute. Previously it would
return None.
  FutureWarning)

```

```

[15]: OnnxPipeline(steps=[('scaler',
                          OnnxTransformer(onnx_bytes=b'\x08\x06\x12\x08skl2onnx\x1a\x
081.7.1076"\x07ai.onnx(\x002\x00:\xf6\x01\n\xa6\x01\n\x01X\x12\x08variable\x1a\x
06Scaler"\x06Scaler*=\n\x06offset=>\xc3.;+=\xc0;=|\xf2\xb0<=\xcd`xf9>=\x89\xad
3\xbd=RL\xab\xbf=V\xc4V\xbe=6<\x9d\xc0=B>\xa0@=\xbb\x93\xea@\xa0\x01\x06*<\n\x05

```

```

scale=ik\b7>=\e8\x17,>)\b5\xa9==\a7\xd5#==Q\x9e\xa1<=\xf5)$<=\x90<\xa2;=(D%
;a\xa8\xa1:= \x9f$:\xa0\x01\x06:\nai.onnx.ml\x12\x1e\mlproduct_ONNX(StandardScaler)Z\x11\n\x01X\x12\x0c\n\n\x08\x01\x12\x06\n\x00\n\x02\x08\nb\x18\n\x08variable
\x12\x0c\n\n\x08\x01\x12\x06\n\x00\n\x02\x08\nB\x0e\n\nai.onnx.ml\x10\x01')),
      ('dt', DecisionTreeRegressor(max_depth=10))]

```

We see that the first steps was replaced by an object *OnnxTransformer* which wraps an ONNX file into a transformer following the *scikit-learn* API. The initial steps are still available.

```
[16]: model_onx.raw_steps_
```

```
[16]: [('scaler', StandardScaler()), ('dt', DecisionTreeRegressor(max_depth=10))]
```

```
[17]: models = [
      ('bug', Xi_test.astype(numpy.float32), model),
      ('OnnxPipeline', Xi_test.astype(numpy.float32), model_onx),
    ]
```

```
[18]: obs = [dict(runtime='sklearn', diff=0, name='sklearn')]
      for name, x32, mod in models:
          for runtime in ['python', 'python_compiled', 'onnxruntime1']:
              lonx = to_onnx(mod, x32[:1])
              loinf = OnnxInference(lonx, runtime=runtime)
              y_skl = model_onx.predict(X32) # model_onx is the new baseline
              y_onx = loinf.run({'X': X32})['variable']
              delta = numpy.abs(y_skl - y_onx.ravel())
              am = delta.argmax()
              obs.append(dict(runtime=runtime, diff=delta.max(), name=name))
              obs[-1]['v[%d]' % am] = y_onx.ravel()[am]
              obs[0]['v[%d]' % am] = y_skl.ravel()[am]

      df = pandas.DataFrame(obs)
      df
```

```
[18]:
```

	runtime	diff	name	v[2276]	v[1109]
0	sklearn	0.000000	sklearn	272.784708	516.084502
1	python	234.930666	bug	37.854042	NaN
2	python_compiled	234.930666	bug	37.854042	NaN
3	onnxruntime1	234.930666	bug	37.854042	NaN
4	python	0.000029	OnnxPipeline	NaN	516.084473
5	python_compiled	0.000029	OnnxPipeline	NaN	516.084473
6	onnxruntime1	0.000029	OnnxPipeline	NaN	516.084473

Training the next steps based on ONNX outputs is better. This is not completely satisfactory... Let's check the accuracy.

```
[19]: model.score(Xi_test, yi_test), model_onx.score(Xi_test, yi_test)
```

```
[19]: (0.6492778377907853, 0.6536515451871481)
```

Pretty close.

### 1.5 Final explanation: StandardScalerFloat

We proposed two ways to have an ONNX pipeline which produces the same prediction as *scikit-learn*. Let's now replace the *StandardScaler* by a new one which outputs float and not double. It turns out that

class *StandardScaler* computes  $X /= self.scale\_$  but ONNX does  $X *= self.scale\_inv\_$ . We need to implement this exact same operator with float32 to remove all discrepancies.

```
[20]: class StandardScalerFloat(StandardScaler):

    def __init__(self, with_mean=True, with_std=True):
        StandardScaler.__init__(self, with_mean=with_mean, with_std=with_std)

    def fit(self, X, y=None):
        StandardScaler.fit(self, X, y)
        if self.scale_ is not None:
            self.scale_inv_ = (1. / self.scale_).astype(numpy.float32)
        return self

    def transform(self, X):
        X = X.copy()
        if self.with_mean:
            X -= self.mean_
        if self.with_std:
            X *= self.scale_inv_
        return X

model_float = Pipeline([
    ('scaler', StandardScalerFloat()),
    ('dt', DecisionTreeRegressor(max_depth=max_depth))
])

model_float.fit(Xi_train.astype(numpy.float32), yi_train.astype(numpy.float32))
```

```
[20]: Pipeline(steps=[('scaler', StandardScalerFloat()),
                      ('dt', DecisionTreeRegressor(max_depth=10))])
```

```
[21]: try:
        onx_float = to_onnx(model_float, Xi_test[:,1].astype(numpy.float))
    except RuntimeError as e:
        print(e)
```

Unable to find a shape calculator for type '<class '\_\_main\_\_.StandardScalerFloat>'

It usually means the pipeline being converted contains a transformer or a predictor with no corresponding converter implemented in *sklearn-onnx*. If the converted is implemented in another library, you need to register the converted so that it can be used by *sklearn-onnx* (function `update_registered_converter`). If the model is not yet covered by *sklearn-onnx*, you may raise an issue to <https://github.com/onnx/sklearn-onnx/issues> to get the converter implemented or even contribute to the project. If the model is a custom model, a new converter must be implemented. Examples can be found in the gallery.

We need to register a new converter so that *sklearn-onnx* knows how to convert the new scaler. We reuse the existing converters.

```
[22]: from skl2onnx import update_registered_converter
from skl2onnx.operator_converters.scaler_op import convert_sklearn_scaler
from skl2onnx.shape_calculators.scaler import calculate_sklearn_scaler_output_shapes

update_registered_converter(
    StandardScalerFloat, "SklearnStandardScalerFloat",
    calculate_sklearn_scaler_output_shapes,
    convert_sklearn_scaler,
    options={'div': ['std', 'div', 'div_cast']})
```

```
[23]: models = [
    ('bug', Xi_test.astype(numpy.float32), model),
    ('FloatPipeline', Xi_test.astype(numpy.float32), model_float),
]
```

```
[24]: obs = [dict(runtime='sklearn', diff=0, name='sklearn')]
for name, x32, mod in models:
    for runtime in ['python', 'python_compiled', 'onnxruntime1']:
        lonx = to_onnx(mod, x32[:1])
        loinf = OnnxInference(lonx, runtime=runtime)
        y_skl = model_float.predict(X32) # we use model_float as a baseline
        y_onx = loinf.run({'X': X32})['variable']
        delta = numpy.abs(y_skl - y_onx.ravel())
        am = delta.argmax()
        obs.append(dict(runtime=runtime, diff=delta.max(), name=name))
        obs[-1]['v[%d]' % am] = y_onx.ravel()[am]
        obs[0]['v[%d]' % am] = y_skl.ravel()[am]

df = pandas.DataFrame(obs)
df
```

```
[24]:
```

	runtime	diff	name	v[1489]	v[1109]
0	sklearn	0.000000	sklearn	378.038116	516.084493
1	python	273.322334	bug	104.715782	NaN
2	python_compiled	273.322334	bug	104.715782	NaN
3	onnxruntime1	273.322334	bug	104.715782	NaN
4	python	0.000020	FloatPipeline	NaN	516.084473
5	python_compiled	0.000020	FloatPipeline	NaN	516.084473
6	onnxruntime1	0.000020	FloatPipeline	NaN	516.084473

That means than the differences between  $\text{float32}(X / Y)$  and  $\text{float32}(X) * \text{float32}(1 / Y)$  are big enough to select a different path in the decision tree.  $\text{float32}(X) / \text{float32}(Y)$  and  $\text{float32}(X) * \text{float32}(1 / Y)$  are also different enough to trigger a different path. Let's illustrate that on example:

```
[25]: a1 = numpy.random.randn(100, 2) * 10
a2 = a1.copy()
a2[:, 1] *= 1000
a3 = a1.copy()
a3[:, 0] *= 1000

for i, a in enumerate([a1, a2, a3]):
    a = a.astype(numpy.float32)
    max_diff32 = numpy.max([
```



```

numpy.abs(numpy.float32(x[0]) / numpy.float32(x[1]) -
          numpy.float32(x[0]) * (numpy.float32(1) / numpy.float32(x[1])))
for x in a]
max_diff64 = numpy.max([
numpy.abs(numpy.float64(x[0]) / numpy.float64(x[1]) -
          numpy.float64(x[0]) * (numpy.float64(1) / numpy.float64(x[1])))
for x in a])
print(i, max_diff32, max_diff64)

```

```

0 1.9073486e-06 7.105427357601002e-15
1 3.7252903e-09 3.469446951953614e-18
2 0.00390625 7.275957614183426e-12

```

The last random set shows very big differences, obviously big enough to trigger a different path in the graph. The difference for double could probably be significant in some cases, not enough on this example.

## 1.6 Change the conversion with option *div*

Option 'div' was added to the converter for *StandardScaler* to change the way the scaler is converted.

```

[26]: model = Pipeline([
      ('scaler', StandardScaler()),
      ('dt', DecisionTreeRegressor(max_depth=max_depth))
    ])
model.fit(Xi_train, yi_train)

```

```

[26]: Pipeline(steps=[('scaler', StandardScaler()),
                      ('dt', DecisionTreeRegressor(max_depth=10))])

```

```

[27]: onx_std = to_onnx(model, Xi_train[:,1].astype(numpy.float32))

%onnxview onx_std

```

```

[27]: <jyquickhelper.jspy.render_nb_js_dot.RenderJsDot at 0x1c3955e75c0>

```

```

[28]: onx_div = to_onnx(model, Xi_train[:,1].astype(numpy.float32),
                      options={StandardScaler: {'div': 'div'}})

%onnxview onx_div

```

```

[28]: <jyquickhelper.jspy.render_nb_js_dot.RenderJsDot at 0x1c3943bd518>

```

```

[29]: onx_div_cast = to_onnx(model, Xi_train[:,1].astype(numpy.float32),
                          options={StandardScaler: {'div': 'div_cast'}})

%onnxview onx_div_cast

```

```

[29]: <jyquickhelper.jspy.render_nb_js_dot.RenderJsDot at 0x1c3955fc2e8>

```

The ONNX graph is different and using division. Let's measure the discrepancies.

```

[30]: X32 = Xi_test.astype(numpy.float32)
      X64 = Xi_test.astype(numpy.float64)
      models = [('bug', model, onx_std),
                ('div', model, onx_div),
                ('div_cast', model, onx_div_cast),]

```

```

obs = [dict(runtime='sklearn', diff=0, name='sklearn')]
for name, mod, onx in models:
    for runtime in ['python', 'python_compiled', 'onnxruntime1']:
        oinf = OnnxInference(onx, runtime=runtime)
        y_skl32 = mod.predict(X32)
        y_skl64 = mod.predict(X64)
        y_onx = oinf.run({'X': X32})['variable']

        delta32 = numpy.abs(y_skl32 - y_onx.ravel())
        am32 = delta32.argmax()
        delta64 = numpy.abs(y_skl64 - y_onx.ravel())
        am64 = delta64.argmax()

        obs.append(dict(runtime=runtime, diff32=delta32.max(),
                        diff64=delta64.max(), name=name))
        obs[0]['v32[%d]' % am32] = y_skl32.ravel()[am32]
        obs[0]['v64[%d]' % am64] = y_skl64.ravel()[am64]
        obs[-1]['v32[%d]' % am32] = y_onx.ravel()[am32]
        obs[-1]['v64[%d]' % am64] = y_onx.ravel()[am64]

df = pandas.DataFrame(obs)
df

```

```

[30]:
      runtime  diff      name  v32[1583]  v64[1246]  v32[1246]  \
0      sklearn  0.0  sklearn -439.590635 -364.555875 -203.438616
1         python  NaN      bug -305.949036 -203.438614          NaN
2  python_compiled  NaN      bug -305.949036 -203.438614          NaN
3   onnxruntime1  NaN      bug -305.949036 -203.438614          NaN
4         python  NaN      div          NaN          NaN -364.555878
5  python_compiled  NaN      div          NaN          NaN -364.555878
6   onnxruntime1  NaN      div          NaN          NaN -364.555878
7         python  NaN  div_cast          NaN          NaN -364.555878
8  python_compiled  NaN  div_cast          NaN          NaN -364.555878
9   onnxruntime1  NaN  div_cast          NaN          NaN -364.555878

      v64[2080]  v64[1109]      diff32      diff64
0  171.604023  516.084502          NaN          NaN
1          NaN          NaN  133.641599  161.117261
2          NaN          NaN  133.641599  161.117261
3          NaN          NaN  133.641599  161.117261
4  329.592377          NaN  161.117261  157.988354
5  329.592377          NaN  161.117261  157.988354
6  329.592377          NaN  161.117261  157.988354
7          NaN  516.084473  161.117261    0.000029
8          NaN  516.084473  161.117261    0.000029
9          NaN  516.084473  161.117261    0.000029

```

The only combination which works is the model converted with option *div\_cast* (use of division in double precision), float input for ONNX, double input for *scikit-learn*.

## 1.7 Explanation in practice

Based on previous sections, the following example builds a case where discrepancies are significant.

```
[31]: std = StandardScaler()
std.fit(Xi_train)
xt32 = Xi_test.astype(numpy.float32)
xt64 = Xi_test.astype(numpy.float64)
pred = std.transform(xt32)
```

```
[32]: from onnxruntime import InferenceSession

onx32 = to_onnx(std, Xi_train[:1].astype(numpy.float32))
sess32 = InferenceSession(onx32.SerializeToString())
got32 = sess32.run(0, {'X': xt32})[0]
d32 = numpy.max(numpy.abs(pred.ravel() - got32.ravel()))
d32
```

[32]: 2.3841858e-07

```
[33]: oinf32 = OnnxInference(onx32.SerializeToString())
gotpy32 = oinf32.run({'X': xt32})['variable']
dpy32 = numpy.max(numpy.abs(pred.ravel() - gotpy32.ravel()))
dpy32
```

[33]: 2.3841858e-07

We tried to cast float into double before applying the normalisation and to cast back into single float. It does not help much.

```
[34]: onx64 = to_onnx(std, Xi_train[:1].astype(numpy.float32),
                    options={id(std): {'div': 'div'}})
sess64 = InferenceSession(onx64.SerializeToString())
got64 = sess64.run(0, {'X': xt32})[0]
d64 = numpy.max(numpy.abs(pred.ravel() - got64.ravel()))
d64
```

[34]: 2.3841858e-07

Last experiment, we try to use double all along.

```
[35]: from onnxruntime.capi.onnxruntime_pybind11_state import InvalidGraph

onx64_2 = to_onnx(std, Xi_train[:1].astype(numpy.float64))
try:
    sess64_2 = InferenceSession(onx64_2.SerializeToString())
except InvalidGraph as e:
    print(e)
```

[ONNXRuntimeError] : 10 : INVALID\_GRAPH : This is an invalid model. Error in Node:Scaler : Mismatched attribute type in 'Scaler : offset'

*onnxruntime* does not support this. Let's switch to *mlpredict*.

```
[36]: onx64_2 = to_onnx(std, Xi_train[:1].astype(numpy.float64))
sess64_2 = OnnxInference(onx64_2, runtime="python")
pred64 = std.transform(xt64)
got64_2 = sess64_2.run({'X': xt64})['variable']
d64_2 = numpy.max(numpy.abs(pred64.ravel() - got64_2.ravel()))
```

```
d64_2
```

```
[36]: 4.440892098500626e-16
```

Differences are lower if every operator is done with double.

## 1.8 Conclusion

Maybe the best option is just to introduce a transform which just cast inputs into floats.

```
[37]: model1 = Pipeline([
      ('scaler', StandardScaler()),
      ('dt', DecisionTreeRegressor(max_depth=max_depth))
    ])

model1.fit(Xi_train, yi_train)
```

```
[37]: Pipeline(steps=[('scaler', StandardScaler()),
                      ('dt', DecisionTreeRegressor(max_depth=10))])
```

```
[38]: from skl2onnx.sklapi import CastTransformer

model2 = Pipeline([
    ('cast64', CastTransformer(dtype=np.float64)),
    ('scaler', StandardScaler()),
    ('cast', CastTransformer()),
    ('dt', DecisionTreeRegressor(max_depth=max_depth))
])

model2.fit(Xi_train, yi_train)
```

```
[38]: Pipeline(steps=[('cast64', CastTransformer(dtype=<class 'numpy.float64'>)),
                      ('scaler', StandardScaler()), ('cast', CastTransformer()),
                      ('dt', DecisionTreeRegressor(max_depth=10))])
```

```
[39]: X32 = Xi_test.astype(numpy.float32)
models = [('model1', model1, X32), ('model2', model2, X32)]
options = [('-', None),
           ('div_cast', {StandardScaler: {'div': 'div_cast'}})]

obs = [dict(runtime='sklearn', diff=0, name='model1'),
       dict(runtime='sklearn', diff=0, name='model2')]
for name, mod, x32 in models:
    for no, opts in options:
        onx = to_onnx(mod, Xi_train[:1].astype(numpy.float32),
                      options=opts)
        for runtime in ['python', 'python_compiled', 'onnxruntime1']:
            try:
                oinf = OnnxInference(onx, runtime=runtime)
            except Exception as e:
                obs.append(dict(runtime=runtime, err=str(e),
                               name=name, options=no))
                continue

y_skl = mod.predict(x32)
```

```

try:
    y_onx = oinf.run({'X': x32})['variable']
except Exception as e:
    obs.append(dict(runtime=runtime, err=str(e),
                   name=name, options=no))
    continue

delta = numpy.abs(y_skl - y_onx.ravel())
am = delta.argmax()

obs.append(dict(runtime=runtime, diff=delta.max(),
               name=name, options=no))
obs[-1]['v[%d]' % am] = y_onx.ravel()[am]
if name == 'model1':
    obs[0]['v[%d]' % am] = y_skl.ravel()[am]
    obs[1]['v[%d]' % am] = model2.predict(Xi_test).ravel()[am]
elif name == 'model2':
    obs[0]['v[%d]' % am] = model1.predict(Xi_test).ravel()[am]
    obs[1]['v[%d]' % am] = y_skl.ravel()[am]

df = pandas.DataFrame(obs)
df

```

```

[39]:
      runtime      diff  name  v[1583]  v[1246]  v[1109]  \
0      sklearn    0.000000  model1 -439.590635 -162.952888  516.084502
1      sklearn    0.000000  model2 -439.590635 -364.555875  516.084502
2      python    133.641599  model1 -305.949036      NaN      NaN
3  python_compiled  133.641599  model1 -305.949036      NaN      NaN
4      onnxruntime1  133.641599  model1 -305.949036      NaN      NaN
5      python    201.602989  model1      NaN -364.555878      NaN
6  python_compiled  201.602989  model1      NaN -364.555878      NaN
7      onnxruntime1  201.602989  model1      NaN -364.555878      NaN
8      python      0.000029  model2      NaN      NaN  516.084473
9  python_compiled      0.000029  model2      NaN      NaN  516.084473
10     onnxruntime1      NaN  model2      NaN      NaN      NaN
11      python      0.000029  model2      NaN      NaN  516.084473
12  python_compiled      0.000029  model2      NaN      NaN  516.084473
13     onnxruntime1      0.000029  model2      NaN      NaN  516.084473

      options  err
0      NaN      NaN
1      NaN      NaN
2      -      NaN
3      -      NaN
4      -      NaN
5  div_cast      NaN
6  div_cast      NaN
7  div_cast      NaN
8      -      NaN
9      -      NaN
10     -  Unable to create InferenceSession due to '[ONN...
11  div_cast      NaN
12  div_cast      NaN

```

13 div\_cast

NaN

It seems to work that way.

[40]: