Model Persistence scikit-learn and ONNX

Xavier Dupré

Senior Data Scientist at Microsoft

Professor at ENSAE

Open Source

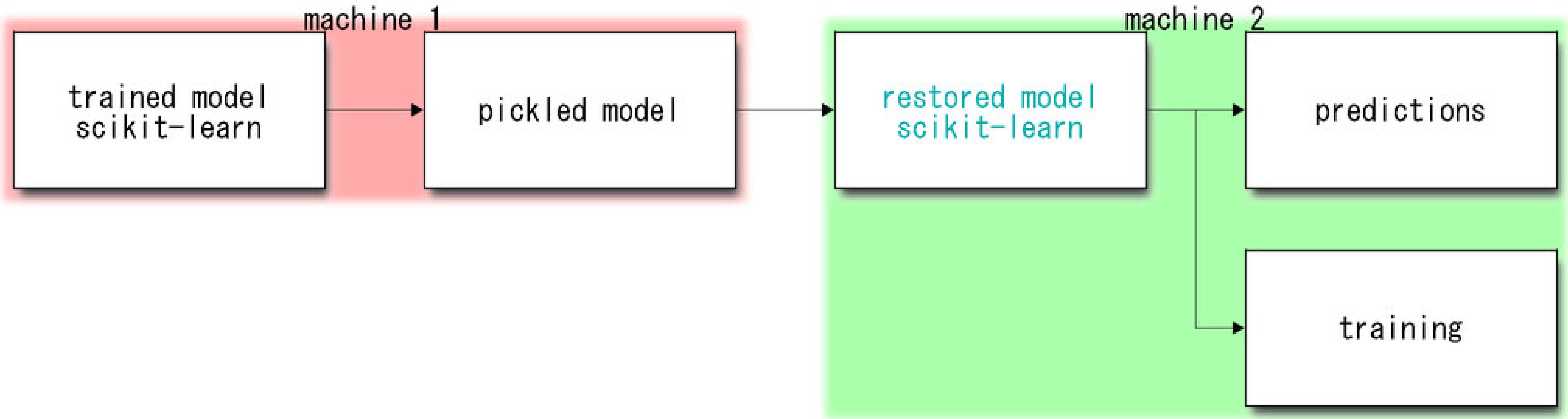
Everything in this presentation
is **open source** (**MIT** license)
and hosted on **github**.

Plan

- Persistence and predictions
- ONNX specifications
- Conversion to ONNX
- Runtime / Benchmark
- Future Plans

Persistence and Prediction

Persistence with pickle

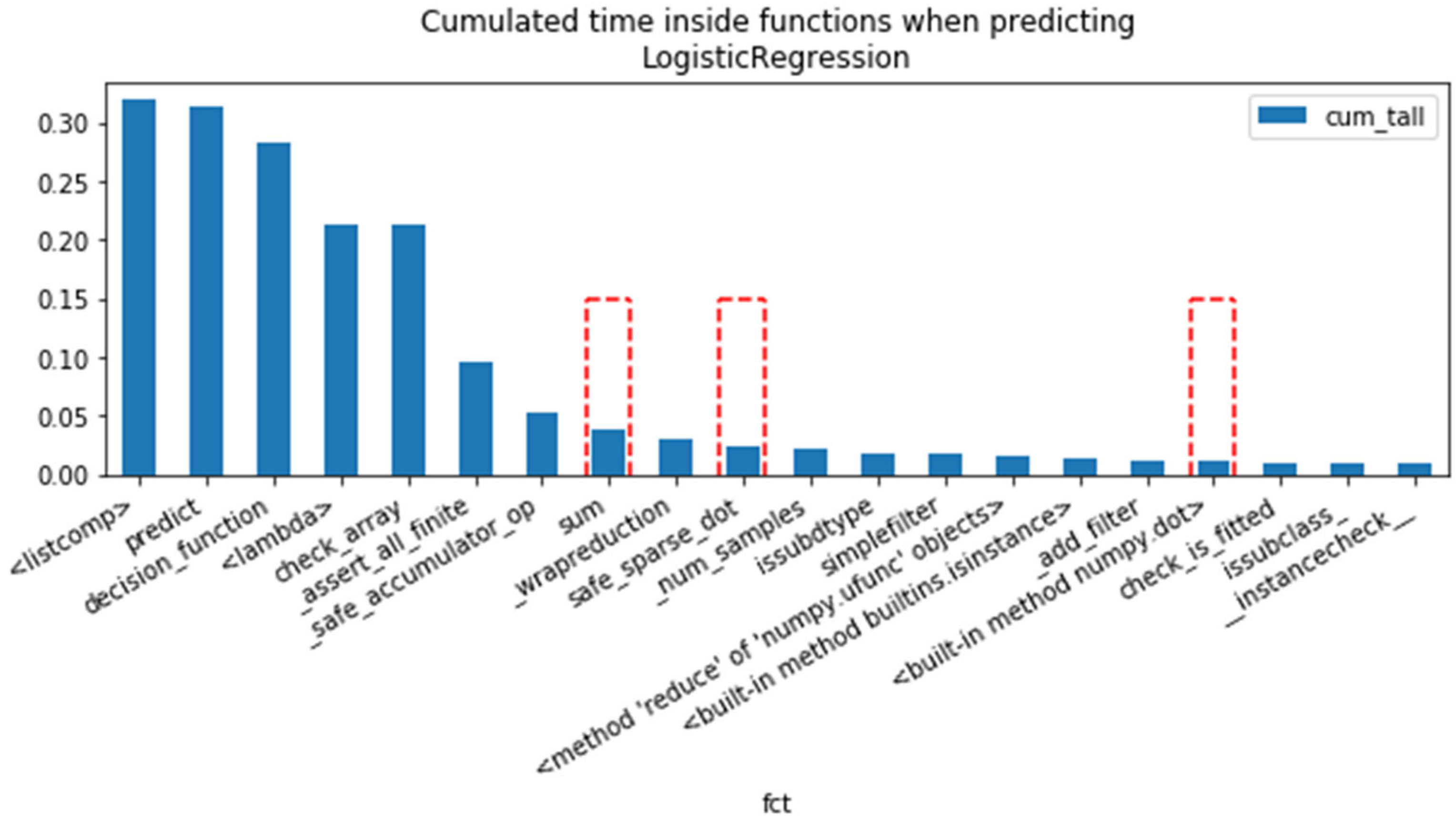


Issues:

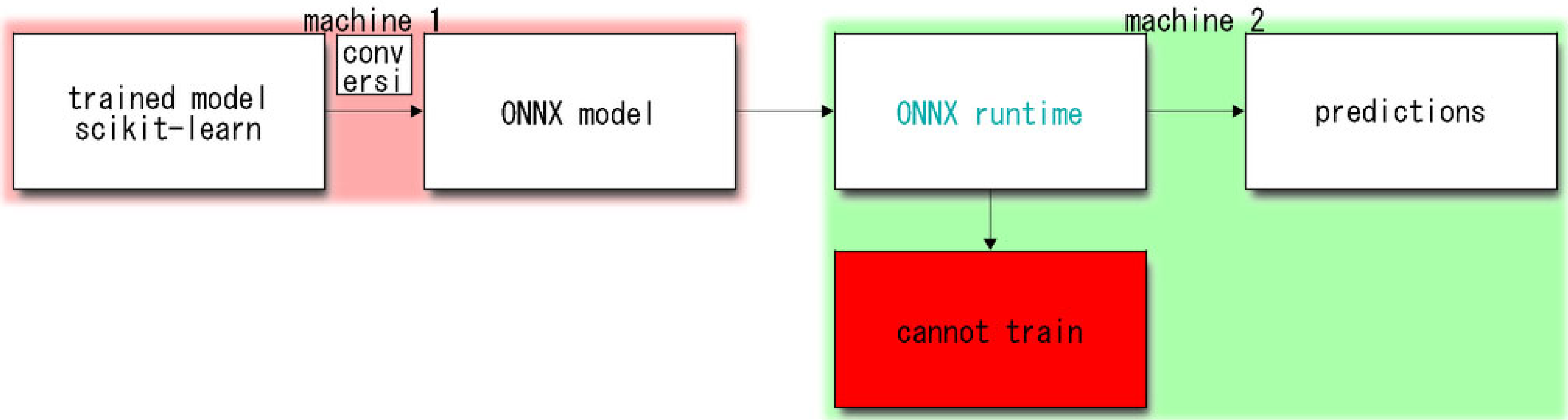
- Unpickle is unstable (python version…)
- Predictions are not fast (scikit-learn is optimized for batch predictions)

With Iris: python >> cython

clr.predict(X)



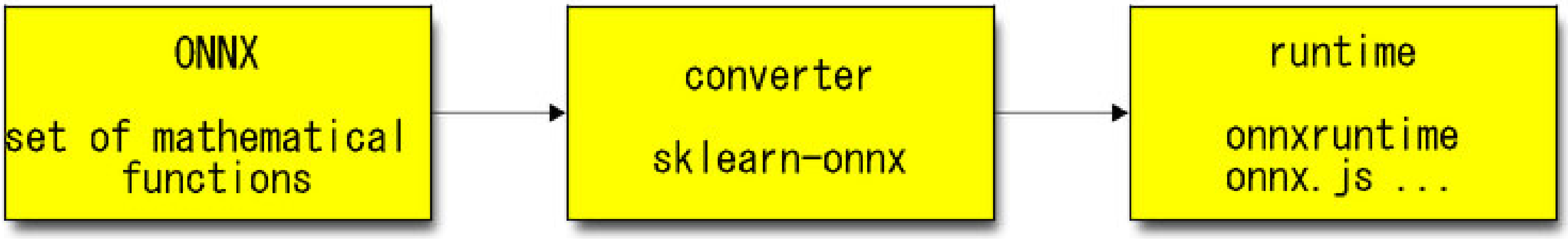
Persistence with ONNX



ONNX…

- Is a serialization format based on protobuf
- Aims at describing any prediction function from machine learned models

Three components for ONNX



ONNX

- ONNX = **Set of mathematical operations** assembled into a **graph**.
- It is versioned and **stable**: backward compatibility.
- It is optimized for deep learning, it works with **single float**

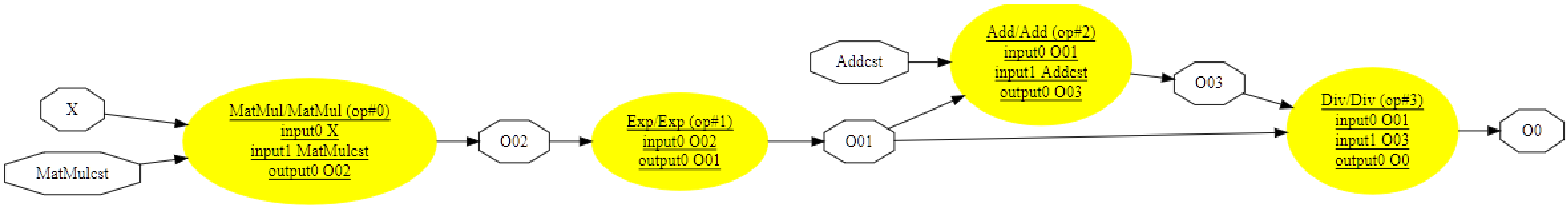
Simple function in ONNX

```
1]: beta = np.random.randn(4, 3)
    M = (X @ beta)
    expM = np.exp(M)
    pred = expM / (expM + 1)
    pred[:5]

1]: array([[0.0022439 , 0.60292776, 0.11036919],
           [0.00474268, 0.46085765, 0.15304197],
           [0.00367439, 0.5859233 , 0.13088156],
           [0.00469139, 0.54574802, 0.15141273],
           [0.00201307, 0.65597864, 0.10384264]])
```

f(X)=...

```
: X32 = X.astype(np.float32)
  beta32 = beta.astype(np.float32)
```

```
onnxExpM = OnnxExp(OnnxMatMul('X', beta32))

cst = np.ones((1, 3), dtype=np.float32)
onnxExpM1 = OnnxAdd(onnxExpM, cst)        # use of broadcasting

onnxPred = OnnxDiv(onnxExpM, onnxExpM1)
```



Serialization, metadata

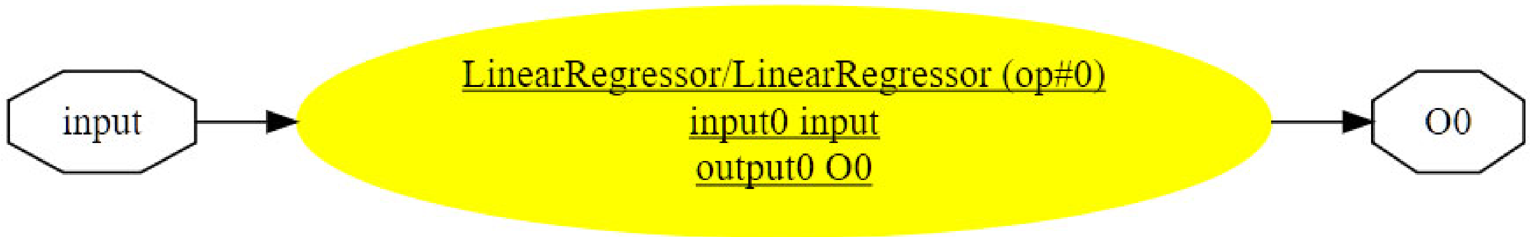
```
In [92]: with open("model-1.onnx", "wb") as f:
             f.write(model_onnx.SerializeToString())
```

```
In [94]: import onnx
         model2 = onnx.load("model-1.onnx")
```

```
ir_version: 5
producer_name: "skl2onnx"
producer_version: "1.4.9999"
domain: "ai.onnx"
model_version: 0
graph {
  node {
    input: "X"
    input: "MatMulcst"
    output: "O02"
    name: "MatMul"
    op_type: "MatMul"
    domain: ""
  }
  node {
...
```

Machine learning functions

```
lin_reg = OnnxLinearRegressor('input',
                              coefficients=beta, targets=2)
```



Conversion to ONNX

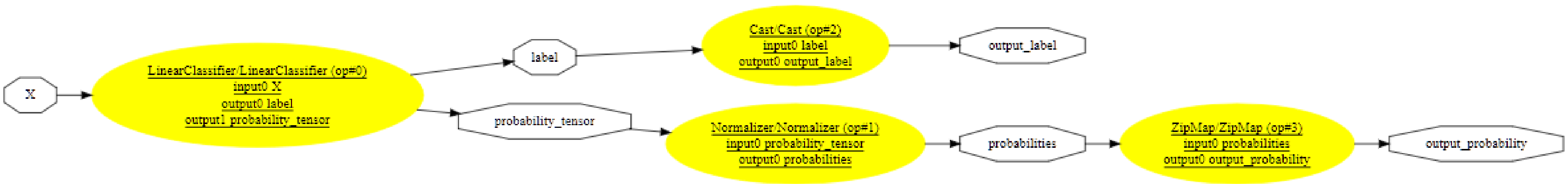
- Each library gets its converter libraries
- **sklearn-onnx for scikit-learn**

Logistic Regression to ONNX

```
In [19]: clr = LogisticRegression(multi_class="auto", solver="liblinear").fit(X, y)
         clr
```

```
Out[19]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                            intercept_scaling=1, l1_ratio=None, max_iter=100,
                            multi_class='auto', n_jobs=None, penalty='l2',
                            random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                            warm_start=False)
```

```
In [20]: from skl2onnx import to_onnx

         model_onnx = to_onnx(clr, X.astype(np.float32))
```

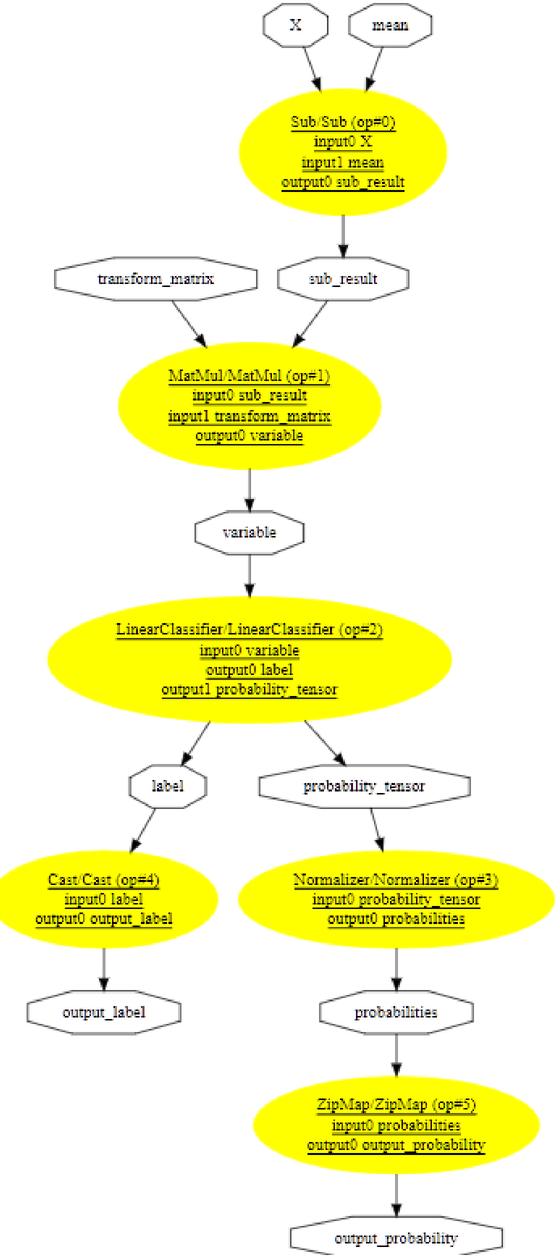


Pipeline to ONNX

```
pipe = Pipeline([('pca', PCA(n_components=2)),
                 ('lr', LogisticRegression(multi_class="auto"))])
pipe.fit(X, y)
```

```
Pipeline(memory=None,
         steps=[('pca',
                 PCA(copy=True, iterated_power='auto', n_components=2,
                     random_state=None, svd_solver='auto', tol=0.0,
                     whiten=False)),
```

```
In [22]: model_onnx = to_onnx(pipe, X.astype(np.float32))
```



Runtime

- Predict anywhere (CPU, GPU, ARM, js, …)
- No dependency on the training framework
- A runtime implements a subset of the mathematical functions defined in ONNX.

onnxruntime (by Microsoft)

- Runtime written in C++
- Available for CPU, GPU, ARM
- Binding for C, C++, C#, Python
- Use openmp, mkldnn, tensorrt, tvm, ngraph…

In [23]:
```python
from onnxruntime import InferenceSession

sess = InferenceSession(model_onnx.SerializeToString())

label, proba = sess.run(None, {'X': X32})
label[:3]
```

Out[23]:
```
array([0, 0, 0], dtype=int64)
```

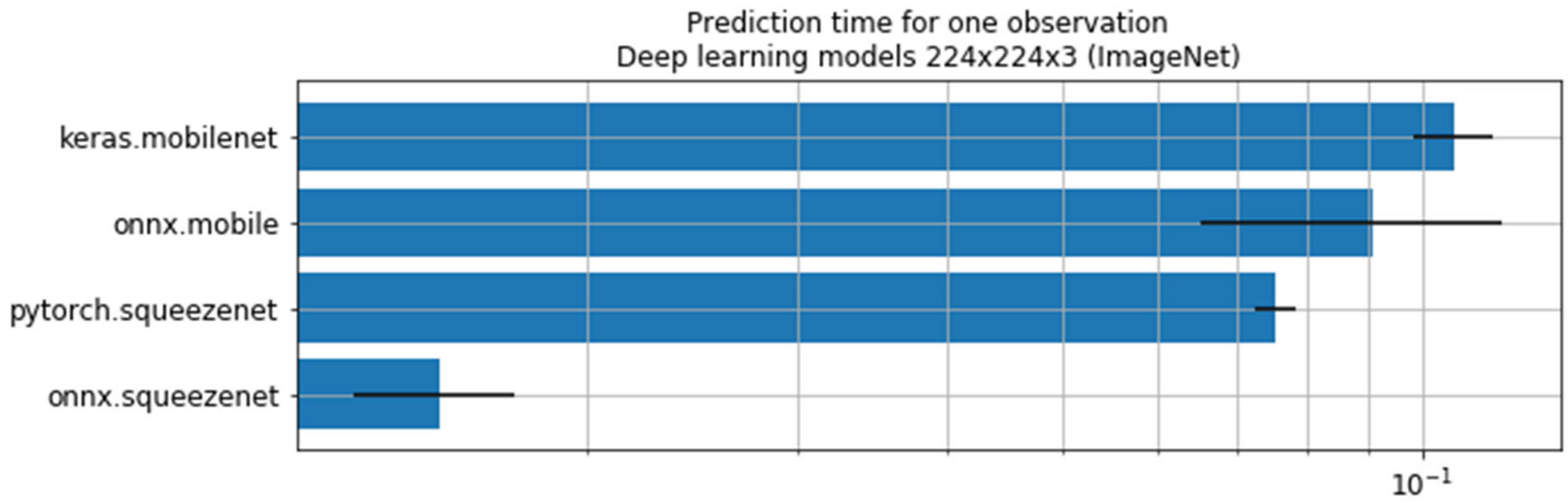
Benchmark: one-off prediction LR

```
In [75]: clr = LogisticRegression(multi_class="auto", solver="liblinear").fit(X, y)

In [76]: %timeit clr.predict_proba(X[:1])

59.7 µs ± 4.22 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)

In [77]: sess = InferenceSession(model_onnx.SerializeToString())
         X32 = X.astype(np.float32)
         %timeit sess.run(None, {'X': X32[:1]})

17.5 µs ± 521 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Benchmark: one-off prediction RF

```
In [78]: clr = RandomForestClassifier(n_estimators=10).fit(X, y)

In [79]: %timeit clr.predict_proba(X[:1])

770 µs ± 85.3 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

In [80]: sess = InferenceSession(model_onnx.SerializeToString())
         X32 = X.astype(np.float32)
         %timeit sess.run(None, {'X': X32[:1]})

18.4 µs ± 2.79 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Benchmark: deep learning (CPU)



Future plans

Today

- Converters for main machine learned models in scikit-learn
- Possibility to add custom converters

Next

- Support for sparse tensors
- Speed improvments
- Better documentation

OnnxSklearnAdaBoostClassifier	OnnxSklearnLabelEncoder	OnnxSklearnRandomForestCl
OnnxSklearnAdaBoostRegressor	OnnxSklearnLasso	OnnxSklearnRandomForestRe
OnnxSklearnBernoulliNB	OnnxSklearnLassoLars	OnnxSklearnRidge
OnnxSklearnBinarizer	OnnxSklearnLinearRegression	OnnxSklearnRobustScaler
OnnxSklearnCalibratedClassifierCV	OnnxSklearnLinearSVC	OnnxSklearnSGDClassifier

Thank you.

Any question: xadupre@microsoft.com