

scikit-learn et ONNX

déploiement de modèles

Xavier Dupré

Senior Data Scientist at Microsoft

Professor at ENSAE

Open Source

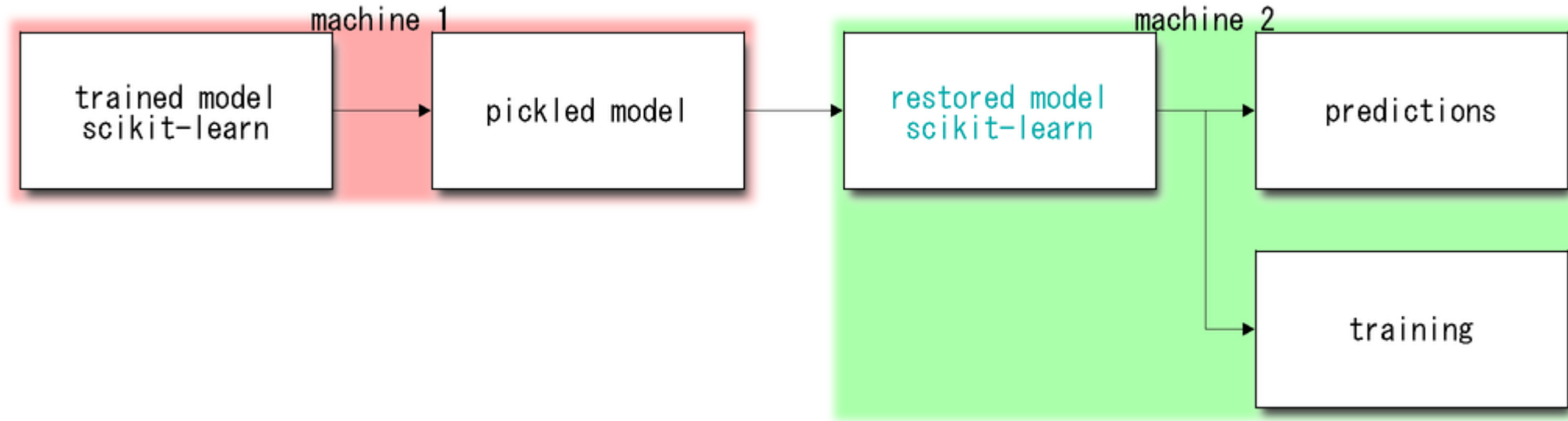
Tout est **open source** (MIT license) et sur **github**.

Plan

- Prédiction en production
- ONNX spécifications
- Conversion to ONNX
- Runtime / Benchmark
- La suite...

Prédictions en production

Sauver un modèle avec pickle

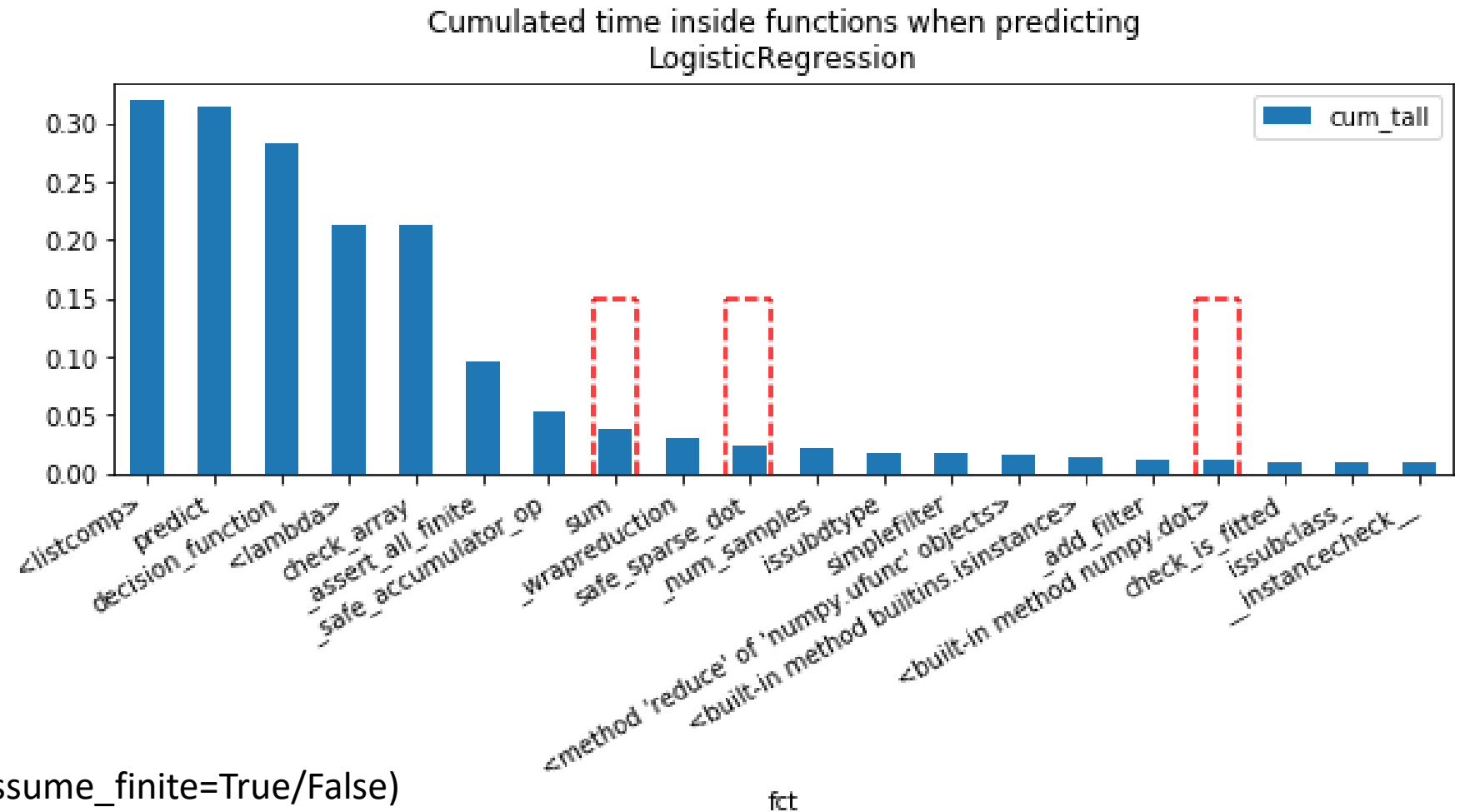


Issues:

- pickle est instable (version de python...)
- Pas mal de monde utilise docker + kubernetes
- Les prédictions ne sont pas rapides (scikit-learn est optimisé pour des « batch predictions » ou prédictions par lot).

Example avec Iris: python >> cython

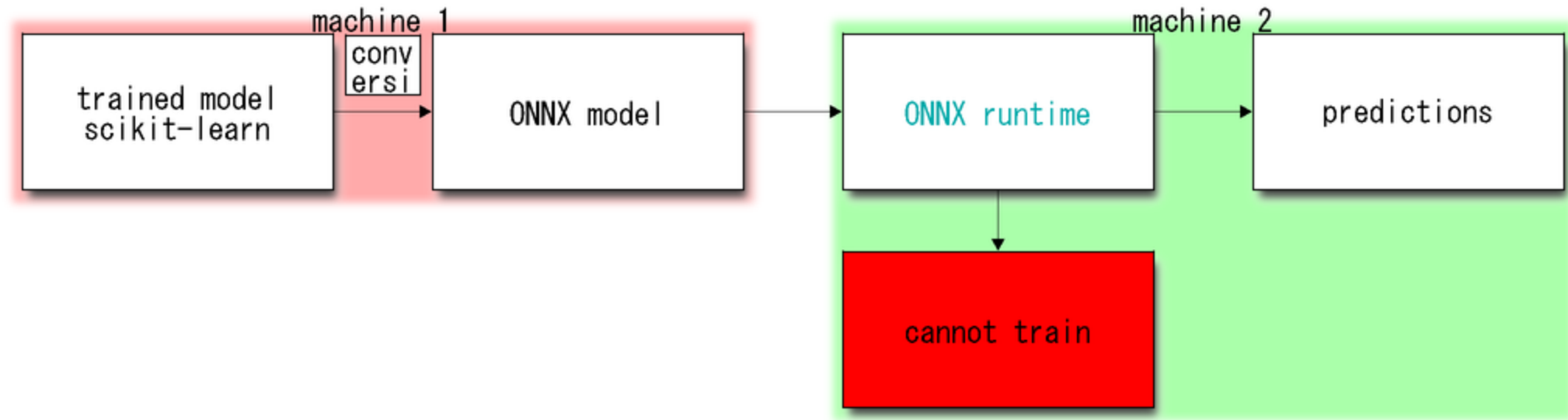
clr.predict(X)



Astuce:

sklearn.config_context(assume_finite=True/False)

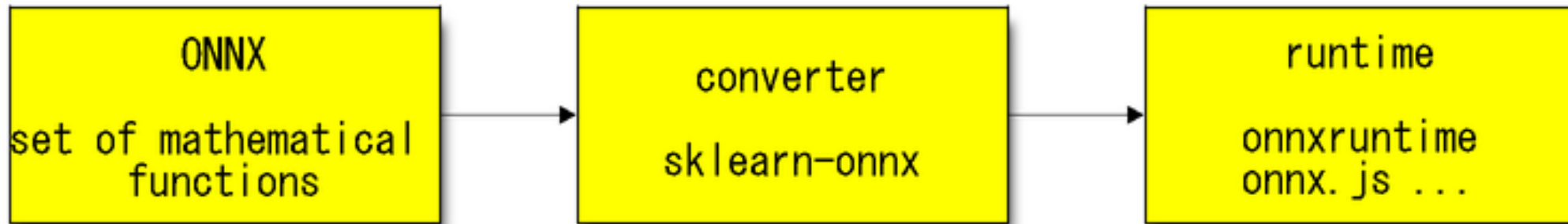
Déploiement avec ONNX...



ONNX est...

- Un format de sérialisation basé sur protobuf
- Une façon de décrire toute fonction de prédiction d'un modèle de machine learning

3 composants pour ONNX



ONNX

- [ONNX](#) = **ensemble d'opérations** mathématiques assemblées dans un **graphe**
- C'est versionné et **stable**: conserve la compatibilité.
- C'est optimisé pour le deep learning, implémenté avec des **single float**

Simple fonction avec 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]])
```

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

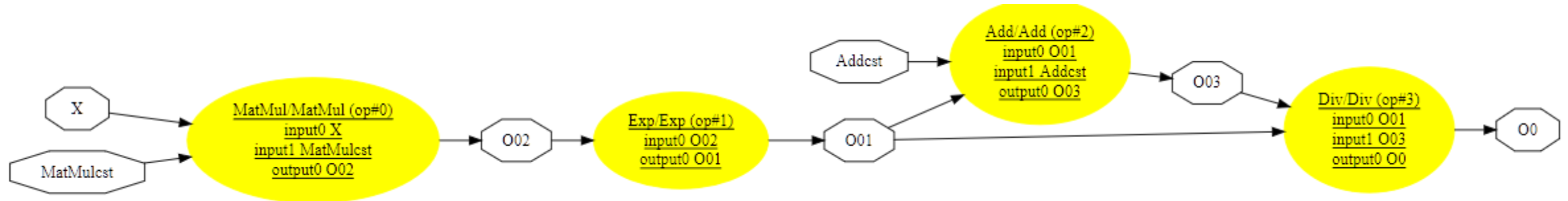
```
onnxExpM = OnnxExp(OnnxMatMul('X', beta32))
```

```
cst = np.ones((1, 3), dtype=np.float32)
```

```
onnxExpM1 = OnnxAdd(onnxExpM, cst) # use of broadcasting
```

```
onnxPred = OnnxDiv(onnxExpM, onnxExpM1)
```

$f(X)=...$



Sérialisation, métadonnées

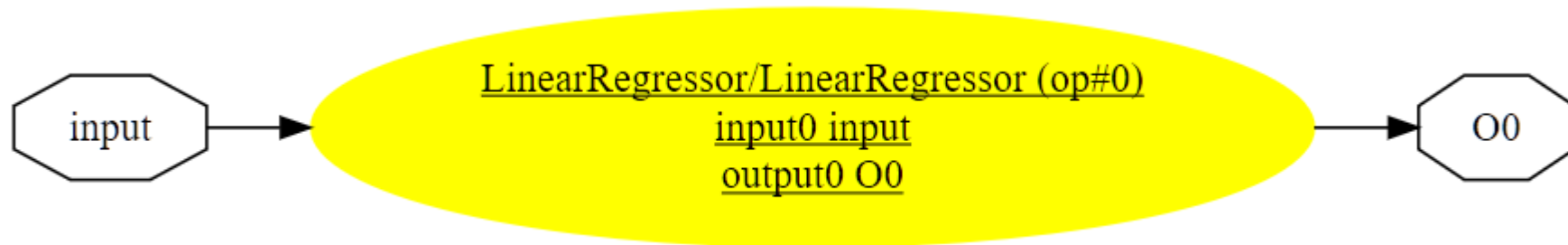
```
In [92]: with open("model-1.onnx", "wb") as f:
         f.write(model_onnx.SerializeToString())
```

```
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 {
    ...
```

Modèle de machine learning

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



Conversion vers ONNX

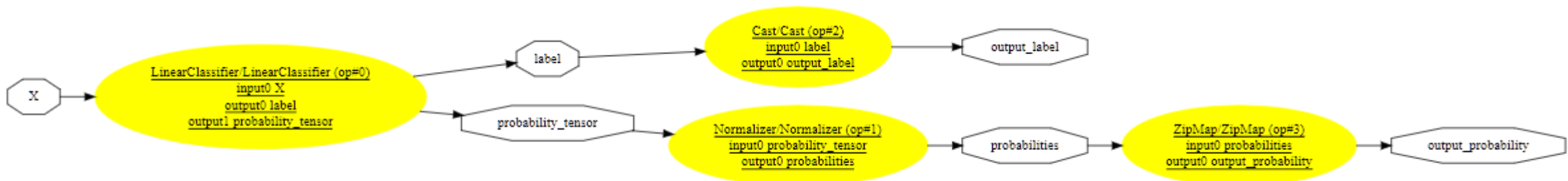
- Chaque librairie a sa librairie de conversion vers ONNX
- **sklearn-onnx** pour **scikit-learn**

Régression Logistique vers 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)
```

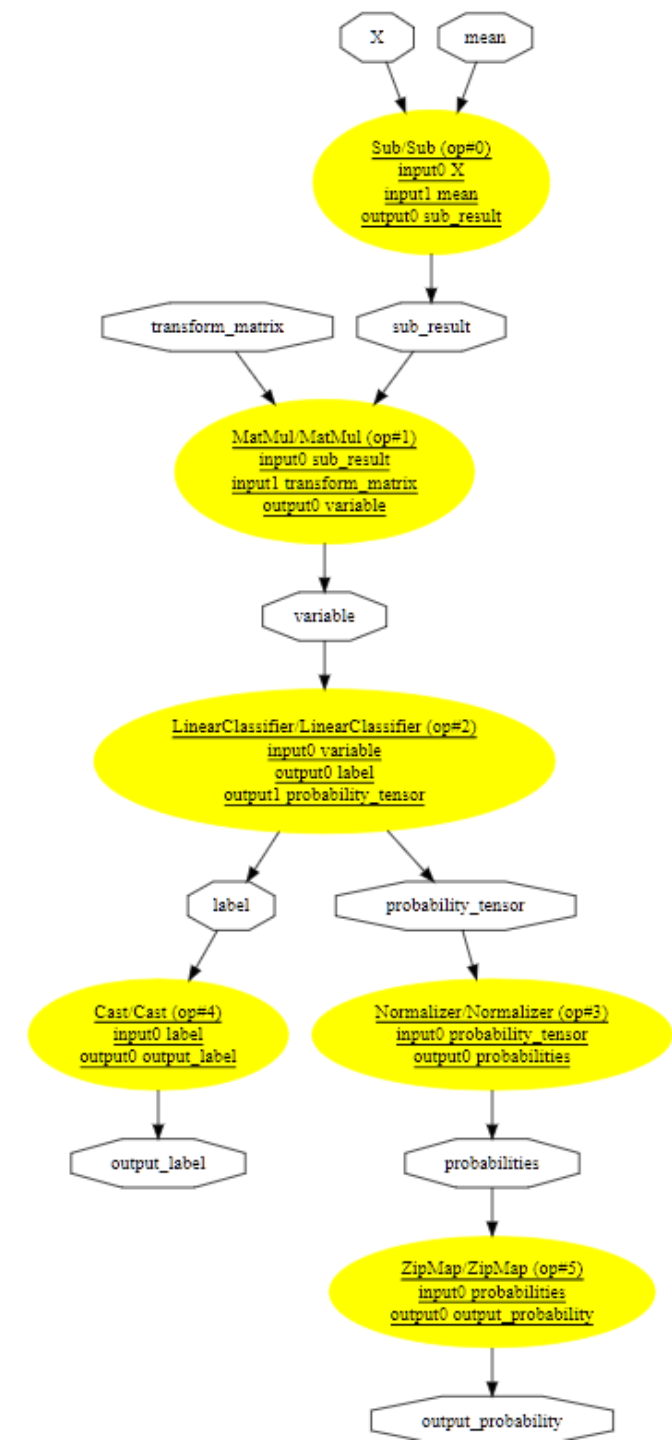
```
In [20]: from skl2onnx import to_onnx
model_onnx = to_onnx(clr, X.astype(np.float32))
```



Pipeline vers 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

- Prédire partout (CPU, GPU, ARM, js, ...)
- Plus de dépendances sur la librairie d'apprentissage
- Un runtime implémente un sous-ensemble de fonctions mathématiques définies par ONNX

onnxruntime (de Microsoft)

- Runtime écrit in C++
- Disponible sur CPU, GPU, ARM
- API pour plusieurs langages C, C++, C#, Python
- Utilise openmp, mkl, dnn, tensorrt, tvm, ngraph...

```
In [23]: 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: prédiction one-off avec LR

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

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

59.7 μ s \pm 4.22 μ s per loop (mean \pm 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 μ s \pm 521 ns per loop (mean \pm std. dev. of 7 runs, 100000 loops each)

Benchmark: prédiction one-off avec RF

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

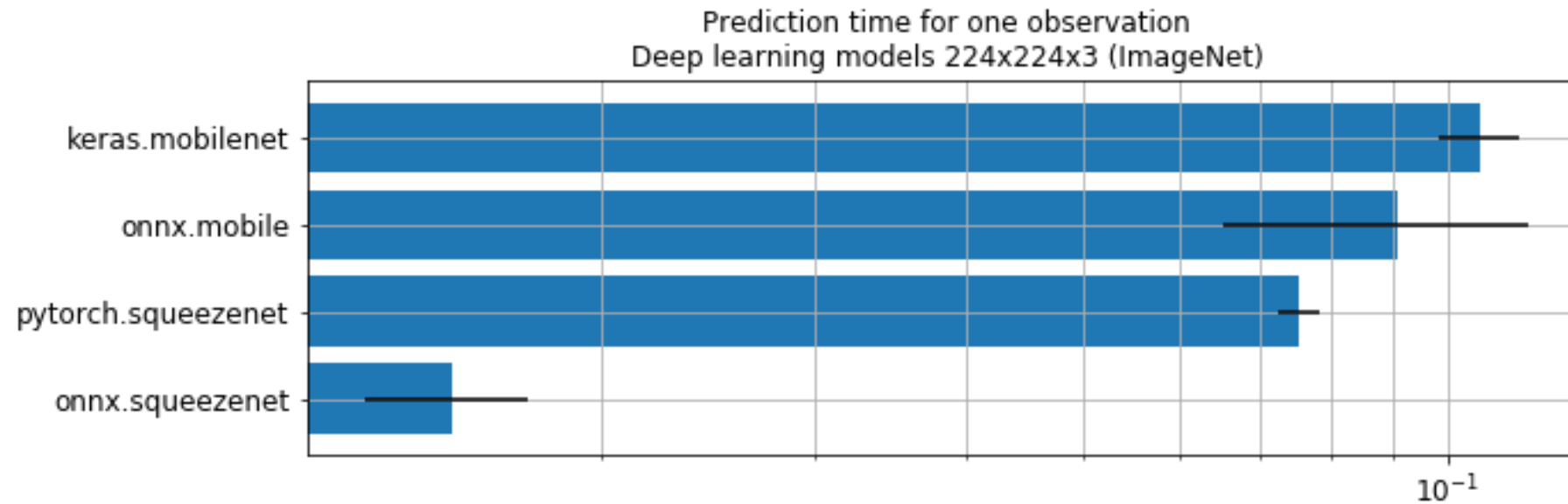
```
In [79]: %timeit clr.predict_proba(X[:1])
```

770 μ s \pm 85.3 μ s per loop (mean \pm 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 μ s \pm 2.79 μ s per loop (mean \pm std. dev. of 7 runs, 100000 loops each)

Benchmark: deep learning (CPU)



La suite...

Aujourd'hui

- Convertisseurs pour la plupart des modèles de machine learning de scikit-learn
- La possibilité d'ajouter son propre convertisseur

Next

- Support des tensors creux (sparse)
- Meilleure performance
- Meilleure documentation

OnnxSklernAdaBoostClassifier	OnnxSklernLabelEncoder	OnnxSklernRandomForestCl
OnnxSklernAdaBoostRegressor	OnnxSklernLasso	OnnxSklernRandomForestRe
OnnxSklernBernoulliNB	OnnxSklernLassoLars	OnnxSklernRidge
OnnxSklernBinarizer	OnnxSklernLinearRegression	OnnxSklernRobustScaler
OnnxSklernCalibratedClassifierCV	OnnxSklernLinearSVC	OnnxSklernSCDClassifier

Merci.

Questions: xadupre@microsoft.com