

# Scikit Model inference in C++

Abhilash Babu

- Using models trained using scikit-learn in a C++ Application
  - Available options
    - ONNX
    - Treelite
    - PMML
    - ...
- Demo

# Available Options

- Use an intermediate format
  - Model persistence options in scikit-learn.
    - ONNX
    - PMML
- ONNX
  - Use ONNX Runtime for inference
  - Focus of this presentation
- PMML
  - Use cPMML library for inference
  - Not being actively developed

# Other Options

- Convert the model into a library.
  - treelite
    - Converts Tree based models to shared library.
    - Works only with tree based algorithms
  - Hummingbird
    - Compiles trained traditional ML models into tensor computations.
    - Works with tree based algorithms
- Use the same underlying library that scikit learn uses
  - liblinear or libsvm
    - No direct mapping available
    - Works only with logistic regression and SVM
- Call python interpreter from C++
  - Use libraries like pybind11, swig etc.
    - Possible loss of performance in conversion between data types of C++ and Python

- Stands for **Open Neural Network Exchange**
- Definition from the official ONNX website
  - ONNX is an open format built to represent machine learning models.
  - ONNX defines a common set of operators - the building blocks of machine learning and deep learning models.
  - ONNX also defines a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers.

```
//LOAD MODEL
Ort::MemoryInfo info("Cpu", OrtDeviceAllocator, 0, OrtMemTypeDefault);
auto input_tensor = Ort::Value::CreateTensor<float>(info, const_cast<float*>(input.data()),
                                                    input.size(),
                                                    _input_shape.data(),
                                                    _input_shape.size());

// RUN INFERENCE
auto ort_outputs = _session.Run(Ort::RunOptions{ nullptr },
                                _input_names.data(),
                                &input_tensor, 1,
                                _output_names.data(), 2);

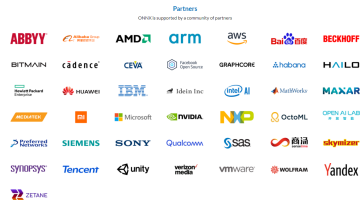
// GET OUTPUT
auto type_info = ort_outputs[0].GetTensorTypeAndShapeInfo();
auto data_length = ort_outputs[0].GetStringTensorDataLength();
std::string result(data_length, '\\0');
std::vector<size_t> offsets(type_info.GetElementCount());
ort_outputs[0].GetStringTensorContent((void*)result.data(), data_length, offsets.data(),
offsets.size());
```

Figure 1: Screenshot

# Why ONNX

- Open source.
- Community project backed by top companies
- Works with models of a wide range of frameworks.
- Works on a variety of platforms

Optimize Inference		Optimize Training									
Platform	Windows	Linux		Mac		Android		iOS		Web Browser (Preview)	
API	Python	C++	C#	C		Java		JS		Obj C	Wicket
Architecture	x64	x86		ARMv84		ARMv82				IBM Power	
Hardware Acceleration	Default CPU	CUDA		DirectML		oneDNN				OpenVINO	
	TensorRT	NNAPI		AGL (Preview)		ArmNN (Preview)				CoreML (Preview)	
	MLGraphX (Preview)	NVIDIA (Preview)		BacklightNPU (Preview)		ViduAI (Preview)					
Installation Instructions	Install Nepti package: <a href="#">Microsoft.ML.Optimize</a>										



## Frameworks & Converters

Use the frameworks you already know and love.

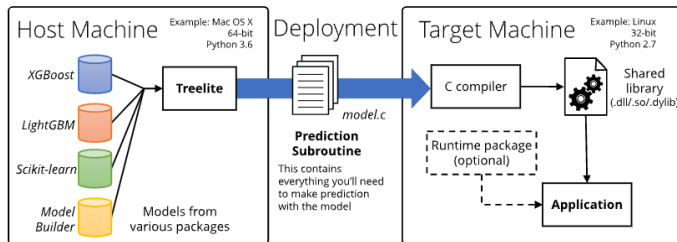


- Stands for Predictive Model Markup Language.
- XML based predictive model interchange format.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PMML xmlns="http://www.dmg.org/PMML-4_4" xmlns:data="http://jpmml.org/jpmml-model/InlineTable" version="4.4">
  <Header>
    <Application name="JPMML-SkLearn" version="1.6.31"/>
    <Timestamp>2021-10-24T03:30:59Z</Timestamp>
  </Header>
  <MiningBuildTask>
    <Extension name="repr">PMMLPipeline(steps=[('classifier', RandomForestClassifier(n_estimators=20, random_state=0))])</Extension>
  </MiningBuildTask>
  <DataDictionary>
    <DataDictionary>
      <MiningModel functionName="classification" algorithmName="sklearn.ensemble._forest.RandomForestClassifier">
        <MiningSchema>
          <MiningSchema>
            <OutputField name="probability(0)" optype="continuous" dataType="double" feature="probability" value="0"/>
            <OutputField name="probability(1)" optype="continuous" dataType="double" feature="probability" value="1"/>
            <OutputField name="probability(2)" optype="continuous" dataType="double" feature="probability" value="2"/>
            <OutputField name="probability(3)" optype="continuous" dataType="double" feature="probability" value="3"/>
            <OutputField name="probability(4)" optype="continuous" dataType="double" feature="probability" value="4"/>
            <OutputField name="probability(5)" optype="continuous" dataType="double" feature="probability" value="5"/>
            <OutputField name="probability(6)" optype="continuous" dataType="double" feature="probability" value="6"/>
            <OutputField name="probability(7)" optype="continuous" dataType="double" feature="probability" value="7"/>
            <OutputField name="probability(8)" optype="continuous" dataType="double" feature="probability" value="8"/>
            <OutputField name="probability(9)" optype="continuous" dataType="double" feature="probability" value="9"/>
          </Output>
          <LocalTransformations>
          </LocalTransformations>
          <Segmentation multipleModelMethod="average">
          </Segmentation>
        </MiningModel>
      </DataDictionary>
    </DataDictionary>
  </MiningModel>
</PMML>
```



- Definition from the website
  - Treelite is a model compiler for decision tree ensembles, aimed at efficient deployment.
  - Treelite overview



# Treelite Example

```
1 import sklearn.datasets
2 import sklearn.ensemble
3 import treelite.sklearn
4
5 X, y = sklearn.datasets.load_boston(return_X_y=True)
6 clf = sklearn.ensemble.RandomForestRegressor(n_estimators=10)
7 clf.fit(X, y)
8
9 treelite_model = treelite.sklearn.import_model_with_model_builder(clf)
10 treelite_model.export_lib(
11     toolchain="gcc",
12     libpath=str(lib_path),
13     params={"parallel_comp": 8},
14     verbose=True,
15 )
```

Figure 2: Treelite code

# Hummingbird

- Hummingbird is a library for compiling trained traditional ML models into tensor computations
- Benefits
  - Leverage optimizations implemented in neural network frameworks
  - Gives native hardware acceleration
  - Both traditional and neural network models use same framework
- Support conversion of sklearn models to Pytorch, ONNX and TVM.

# Hummingbird example

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from hummingbird.ml import convert, load

# Create some random data for binary classification
num_classes = 2
X = np.random.rand(100000, 28)
y = np.random.randint(num_classes, size=100000)

# Create and train a model (scikit-learn RandomForestClassifier in this case)
skl_model = RandomForestClassifier(n_estimators=10, max_depth=10)
skl_model.fit(X, y)

# Use Hummingbird to convert the model to PyTorch
model = convert(skl_model, 'pytorch')

# Save the model
model.save('hb_model')
```

Figure 3: Hummingbird code

- Demo code and slides:  
[https://github.com/abhilb/pydata\\_2021](https://github.com/abhilb/pydata_2021)
- ONNX Runtime: <https://onnxruntime.ai/>
- ONNX: <https://onnx.ai/>
- Treelite: <https://treelite.readthedocs.io/en/latest/>
- PMML Spec:  
<http://dmg.org/pmml/v4-4-1/GeneralStructure.html>
- sklearn2pmml: <https://github.com/jpmml/sklearn2pmml>
- Hummingbird: <https://github.com/microsoft/hummingbird>