sklearn_grammar_lr

March 10, 2022

1 Converts a logistic regression into C

The logistic regression is trained in python and executed in C.

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

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

1.1 Train a linear regression

[2]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data[:, :2]
y = iris.target
y[y == 2] = 1
lr = LogisticRegression()
lr.fit(X, y)
```

[2]:
```
LogisticRegression()
```

1.2 Export into C

[3]:
```
# grammar is the expected scoring model.
from mlprodict.grammar_sklearn import sklearn2graph
gr = sklearn2graph(lr, output_names=['Prediction', 'Score'])
gr
```

[3]:
```
<mlprodict.grammar.gmlactions.MLModel at 0x21564d49828>
```

We can even check what the function should produce as a score. Types are strict.

[4]:
```
import numpy
X = numpy.array([[numpy.float32(1), numpy.float32(2)]])
e2 = gr.execute(Features=X[0, :])
print(e2)
```

```
[  0.       -11.264062]
```

We compare with scikit-learn.

[5]:
```
lr.decision_function(X[0:1, :])
```

[5]:
```
array([-11.26406172])
```

Conversion into C:

[6]:
```
res = gr.export(lang='c', hook={'array': lambda v: v.tolist(), 'float32': lambda v:␣
  ↪float(v)})
print(res["code"])
```

```
int LogisticRegression (float* pred, float* Features)
{
    // 2290909222952-LogisticRegression - children
    // 2290909222728-concat - children
    // 2290909222672-sign - children
    // 2290909222616-+ - children
    // 2290909222560-adot - children
    float pred0c0c00c0[2] = {(float)3.3882975578308105,
(float)-3.164527654647827};
    float* pred0c0c00c1 = Features;
    // 2290909222560-adot - itself
    float pred0c0c00;
    adot_float_float(&pred0c0c00, pred0c0c00c0, pred0c0c00c1, 2);
    // 2290909222560-adot - done
    float pred0c0c01 = (float)-8.323304176330566;
    // 2290909222616-+ - itself
    float pred0c0c0 = pred0c0c00 + pred0c0c01;
    // 2290909222616-+ - done
    // 2290909222672-sign - itself
    float pred0c0;
    sign_float(&pred0c0, pred0c0c0);
    // 2290909222672-sign - done
    // 2290909222728-concat - itself
    float pred0[2];
    concat_float_float(pred0, pred0c0, pred0c0c0);
    // 2290909222728-concat - done
    memcpy(pred, pred0, 2*sizeof(float));
    // 2290909222952-LogisticRegression - itself
    return 0;
    // 2290909222952-LogisticRegression - done
}
```

We execute the code with module cffi.

[7]:
```
from mlprodict.grammar_sklearn.cc import compile_c_function
fct = compile_c_function(res["code"], 2)
fct
```

[7]:
```
<function mlprodict.grammar_sklearn.cc.c_compilation.compile_c_function.<locals>
.wrapper_float(features, output=None)>
```

[8]:
```
e2 = fct(X[0, :])
e2
```

[8]:
```
array([  0.      , -11.264062], dtype=float32)
```

1.3 Time comparison

[9]:
```
%timeit lr.decision_function(X[0:1, :])
```

```
64.9 µs ± 5.84 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

[10]:
```
%timeit fct(X[0, :])
```

```
6.17 µs ± 380 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

There is a significant speedup on this example. It could be even faster by removing some Python part and optimizing the code produced by cffi. We can also save the creation of the array which contains the output by reusing an existing one.

[11]:
```
out = fct(X[0, :])
```

[12]:
```
%timeit fct(X[0, :], out)
```

```
6.33 µs ± 430 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

[13]:
```

```