

XGBoost Cloud Prediction Invocation Template

Invoke SageMaker Prediction Service

```
In [1]: import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import os

import boto3
import re # python regex module
from sagemaker import get_execution_role
import sagemaker

# SDK 2 serializers and deserializers
from sagemaker.serializers import CSVSerializer
from sagemaker.deserializers import JSONDeserializer

In [2]: # SDK 2
# RealTimePredictor renamed to Predictor
# https://sagemaker.readthedocs.io/en/stable/v2.html

# Create a predictor and point to an existing endpoint
endpoint_name = 'xgboost-bikereental-v1'
predictor = sagemaker.predictor.Predictor(endpoint_name=endpoint_name)

In [3]: predictor.serializer = CSVSerializer()

In [4]: df_all = pd.read_csv('bike_test.csv')

In [5]: df_all.head()
```

Out[5]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	year	month	day	dayofweek	hour
0	2011-01-20 00:00:00	1	0	1	1	10.66	11.365	56	26.0027	2011	1	20	3	0
1	2011-01-20 01:00:00	1	0	1	1	10.66	13.635	56	0.0000	2011	1	20	3	1
2	2011-01-20 02:00:00	1	0	1	1	10.66	13.635	56	0.0000	2011	1	20	3	2
3	2011-01-20 03:00:00	1	0	1	1	10.66	12.880	56	11.0014	2011	1	20	3	3
4	2011-01-20 04:00:00	1	0	1	1	10.66	12.880	56	11.0014	2011	1	20	3	4

In [6]: `df_all.columns[1:]`

Out[6]: Index(['season', 'holiday', 'workingday', 'weather', 'temp', 'atemp',
 'humidity', 'windspeed', 'year', 'month', 'day', 'dayofweek', 'hour'],
 dtype='object')

In [7]: *# Need to pass an array to the prediction*
can pass a numpy array or a list of values [[19,1],[20,1]]
`arr_test = df_all[df_all.columns[1:]].values`

In [8]: `type(arr_test)`

Out[8]: `numpy.ndarray`

In [9]: `arr_test.shape`

Out[9]: (6493, 13)

In [10]: `arr_test[:5]`

```
Out[10]: array([[1.00000e+00, 0.00000e+00, 1.00000e+00, 1.00000e+00, 1.06600e+01,
                1.13650e+01, 5.60000e+01, 2.60027e+01, 2.01100e+03, 1.00000e+00,
                2.00000e+01, 3.00000e+00, 0.00000e+00],
               [1.00000e+00, 0.00000e+00, 1.00000e+00, 1.00000e+00, 1.06600e+01,
                1.36350e+01, 5.60000e+01, 0.00000e+00, 2.01100e+03, 1.00000e+00,
                2.00000e+01, 3.00000e+00, 1.00000e+00],
               [1.00000e+00, 0.00000e+00, 1.00000e+00, 1.00000e+00, 1.06600e+01,
                1.36350e+01, 5.60000e+01, 0.00000e+00, 2.01100e+03, 1.00000e+00,
                2.00000e+01, 3.00000e+00, 2.00000e+00],
               [1.00000e+00, 0.00000e+00, 1.00000e+00, 1.00000e+00, 1.06600e+01,
                1.28800e+01, 5.60000e+01, 1.10014e+01, 2.01100e+03, 1.00000e+00,
                2.00000e+01, 3.00000e+00, 3.00000e+00],
               [1.00000e+00, 0.00000e+00, 1.00000e+00, 1.00000e+00, 1.06600e+01,
                1.28800e+01, 5.60000e+01, 1.10014e+01, 2.01100e+03, 1.00000e+00,
                2.00000e+01, 3.00000e+00, 4.00000e+00]])
```

```
In [11]: result = predictor.predict(arr_test[:2])
```

```
In [12]: result
```

```
Out[12]: b'2.332122325897217\n1.9005593061447144\n'
```

```
In [13]: arr_test.shape
```

```
Out[13]: (6493, 13)
```

Split the input data into chunks

There are thousands of rows in this data set for which need inference.

When communicating over internet, it is a good idea to split the data into chunks to prevent payload and timeout error

```
In [14]: # For large number of predictions, we can split the input data and
# Query the prediction service.
# array_split is convenient to specify how many splits are needed

# Splitting using regular expression as xgboost 1-2-2 is returning
# predicted values with inconsistent delimiters (comma, newline or both)

# pattern looks for one or more of non-numeric characters
```

```
pattern = r'^0-9.+'

predictions = []
for arr in np.array_split(arr_test,10):
    result = predictor.predict(arr)
    result = re.split(pattern,result.decode("utf-8"))

    print (arr.shape)
    predictions += [float(r) for r in result if r != ""] # Thanks, Ionut Barbu!
```

```
(650, 13)
(650, 13)
(650, 13)
(649, 13)
(649, 13)
(649, 13)
(649, 13)
(649, 13)
(649, 13)
(649, 13)
```

```
In [15]: len(predictions)
```

```
Out[15]: 6493
```

```
In [16]: np.expm1(predictions)
```

```
Out[16]: array([ 9.29977784,  5.68963495,  4.11209937, ..., 133.68598006,
                92.35797767,  52.00191752])
```

```
In [17]: df_all['count'] = np.expm1(predictions)
```

```
In [18]: df_all.head()
```

Out[18]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	year	month	day	dayofweek	hour	count
0	2011-01-20 00:00:00	1	0	1	1	10.66	11.365	56	26.0027	2011	1	20	3	0	9.299778
1	2011-01-20 01:00:00	1	0	1	1	10.66	13.635	56	0.0000	2011	1	20	3	1	5.689635
2	2011-01-20 02:00:00	1	0	1	1	10.66	13.635	56	0.0000	2011	1	20	3	2	4.112099
3	2011-01-20 03:00:00	1	0	1	1	10.66	12.880	56	11.0014	2011	1	20	3	3	1.822707
4	2011-01-20 04:00:00	1	0	1	1	10.66	12.880	56	11.0014	2011	1	20	3	4	1.604999

In [19]: `df_all[['datetime','count']].to_csv('predicted_count_cloud.csv',index=False)`

In [20]: `# Delete Endpoint to prevent unnecessary charges
predictor.delete_endpoint()`

In []: