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
```

XGBoost Cloud Prediction - Iris Classification

Invoke SageMaker Prediction Service

In [2]:

```
# Acquire a realtime endpoint
endpoint_name = 'xgboost-iris-v1' #DWB# Checked from console - matches
predictor = sagemaker.predictor.Predictor (endpoint_name=endpoint_name)
```

In [3]:

```
predictor.serializer = CSVSerializer()
```

In [4]:

```
# Test predictive quality against data in validation file
df_all = pd.read_csv('iris_validation.csv',
                     names=['encoded_class','sepal_length','sepal_width','petal_length','petal_width'])
```

In [5]:

```
df_all.head()
```

Out[5]:

	encoded_class	sepal_length	sepal_width	petal_length	petal_width
0	1	5.8	2.7	4.1	1.0
1	0	4.8	3.4	1.6	0.2
2	1	6.0	2.2	4.0	1.0
3	2	6.4	3.1	5.5	1.8
4	2	6.7	2.5	5.8	1.8

In [6]:

```
df_all.columns
```

Out[6]:

```
Index(['encoded_class', 'sepal_length', 'sepal_width', 'petal_length',
       'petal_width'],
      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.as_matrix(['sepal_length', 'sepal_width', 'petal_length','petal_width'])
arr_test = df_all[['sepal_length', 'sepal_width', 'petal_length','petal_width']].values
```

In [8]:

```
type(arr_test)
```

Out[8]:

```
numpy.ndarray
```

In [9]:

```
arr_test.shape
```

Out[9]:

```
(45, 4)
```

In [10]:

```
arr_test[:5]
```

Out[10]:

```
array([[5.8, 2.7, 4.1, 1. ],
       [4.8, 3.4, 1.6, 0.2],
       [6. , 2.2, 4. , 1. ],
       [6.4, 3.1, 5.5, 1.8],
       [6.7, 2.5, 5.8, 1.8]])
```

In [11]:

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

In [12]:

```python
arr_test.shape
```

Out[12]:

```
(45, 4)
```

In [15]:

```python
#DWB#  I think that repeat might not be on purpose;
#DWB#+ Let's check instead result.shape
try:
    print(str(result.shape))
except Exception as e:
    print('', file=sys.stderr)
    print("That didn't work.", file=sys.stderr)
    print(f"str(e) is: `{str(e)}`", file=sys.stderr)
    print('', file=sys.stderr)
finally:
    print("That tells us what we need to know.")
##endof:  try/except/finally
```

```
That tells us what we need to know.
```

```
That didn't work.
str(e) is: `'bytes' object has no attribute 'shape'`
```

In [16]:

```python
result
```

Out[16]:

```
b'1.0\n0.0\n'
```

In [22]:

```python
# 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 = []
#DWB# added the next 2 lines
total_row_count = 0
n_columns_and_count = {}
```

```
for arr in np.array_split(arr_test,10):
    result = predictor.predict(arr)
    result = re.split(pattern,result.decode("utf-8"))
    print (arr.shape)
    #DWB# Here is what we can match up
    #DWB# <shape-consistency-check>
    this_chunk_shape = arr.shape
    this_row_count = this_chunk_shape[0]
    total_row_count += this_row_count
    this_col_count = this_chunk_shape[1]
    if this_col_count in n_columns_and_count:
        n_columns_and_count[this_col_count] += 1
    else:
        n_columns_and_count[this_col_count] = 1
    ##endof:  if/else this_col_count in n_columns_and_count
    #DWB# </shape-consistency-check>
    predictions += [int(float(r)) for r in result if r != ""]

#DWB# It's me from here on out.

print()
print("# Looking at the chunks all together #")
print(f"The total number of rows is: {total_row_count}")
print("For each row, I counted the number of columns;")
print("here is the distribution of column counts.")
print(n_columns_and_count)
print()
print("Having inspected that, I can see that")
print("the shape of all the chunks combined is")
print("(45, 4), which matches our original")
print("arr_test.")
```

```
(5, 4)
(5, 4)
(5, 4)
(5, 4)
(5, 4)
(4, 4)
(4, 4)
(4, 4)
(4, 4)
(4, 4)

# Looking at the chunks all together #
The total number of rows is: 45
For each row, I counted the number of columns;
here is the distribution of column counts.
{4: 10}

Having inspected that, I can see that
the shape of all the chunks combined is
(45, 4), which matches our original
arr_test.
```

In [23]:

```
len(predictions)
```

Out[23]:

```
45
```

In [24]:

```
predictions[:5]
```

Out[24]:

```
[1, 0, 1, 2, 2]
```

In [25]:

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'])
```

Out[25]:

```
▾ LabelEncoder
LabelEncoder()
```

In [26]:

```
df_all['class'] = le.inverse_transform(df_all.encoded_class)
```

In [27]:
```
df_all['predicted_class']=le.inverse_transform(predictions)
```

In [28]:
```
df_all.head()
```

Out[28]:

	encoded_class	sepal_length	sepal_width	petal_length	petal_width	class	predicted_class
0	1	5.8	2.7	4.1	1.0	Iris-versicolor	Iris-versicolor
1	0	4.8	3.4	1.6	0.2	Iris-setosa	Iris-setosa
2	1	6.0	2.2	4.0	1.0	Iris-versicolor	Iris-versicolor
3	2	6.4	3.1	5.5	1.8	Iris-virginica	Iris-virginica
4	2	6.7	2.5	5.8	1.8	Iris-virginica	Iris-virginica

In [29]:
```
print('Confusion matrix - Actual versus Predicted')
pd.crosstab(df_all['class'], df_all['predicted_class'])
```

```
Confusion matrix - Actual versus Predicted
```

Out[29]:

predicted_class	Iris-setosa	Iris-versicolor	Iris-virginica
class			
Iris-setosa	16	0	0
Iris-versicolor	0	10	1
Iris-virginica	0	1	17

In [30]:
```
import sklearn.metrics as metrics
print(metrics.classification_report(df_all['class'], df_all['predicted_class']))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        16
Iris-versicolor       0.91      0.91      0.91        11
 Iris-virginica       0.94      0.94      0.94        18

       accuracy                           0.96        45
      macro avg       0.95      0.95      0.95        45
   weighted avg       0.96      0.96      0.96        45
```

In [31]:

```
#DWB#  Still in this second one there's no Endpoint-deletion Code.
#DWB#+ I will put some in, here.
#DWB#+ As Chandra wrote with the previous such code
# Delete Endpoint to prevent unnecessary charges
predictor.delete_endpoint()
```

In [32]:

```
#  I checked the list of endpoints from the AWS Console > Sagemaker ...
#+ and the endpoint that was there is gone.
```

In [ ]: