

記事

[Toshihiko Minamoto](#) · 2022年1月6日 9m read

IRISデータベースへのPython JDBC接続 - 簡易メモ

キーワード: Python, JDBC, SQL, IRIS, Jupyterノートブック, Pandas, Numpy, および機械学習

1. 目的

これは、デモの目的で、Jupyterノートブック内でPython 3によってIRIS JDBCドライバを呼び出し、SQL構文でIRISデータベースインスタンスにデータを読み書きする、5分程度の簡単なメモです。

昨年、私は [CacheデータベースへのPythonバインディング](#) (セクション4.7)について簡単に触れました。そこで、Pythonを使ってIRISデータベースに接続し、そのデータをPandasデータフレームとNumPy配列に読み込んで通常の分析を行ってから、事前処理済みまたは正規化されたデータをML/DLパイプラインに通すためにIRISに書き込む作業においてのオプション議論について要約しましょう。

すぐに思い浮かぶ簡単な オプション がいくつかあります。

1. ODBC: Python 3ネイティブSQLを使う:PyODBCを使ってはどうでしょうか?
2. JDBC: Python 3ネイティブSQLを使う:JayDeBeApiはどうでしょうか?
3. Spark: PySpark:SQLを使うは?
4. IRIS用PythonネイティブAPI :
前に使用したCache用Pythonバインディングを拡張してみたらどうでしょうか?
5. IPython Magic SQL %%sql:した場合、 [これ](#) はまだIRISで動作するでしょうか?

ここで見逃されたオプションがほかにありますか? それらを試してみたいですね。

2. 範囲

通常のJDBCアプローチからはじめましょう。ODBC、Spark、PythonネイティブAPIについては、次の簡易メモで要約することにします。

範囲内:

このイックデモでは、以下の一般的なコンポーネントについて触れています。

Anaconda
Jupyterノートブック
Python 3
JayDeBeApi
JPyPe
Pandas
NumPy
IRIS 2019.xのインスタンス

範囲外:

このメモでは、以下の項目には触れていません。重要な項目であり、具体サイトソリューション、デバッグ、およびサービスで個別に対処される可能性があります。

エンドツ・エンドのセキュリティ
機能しないパフォーマンスなど
トラブルシューティングサポート
ライセンス

3. デモ

3.1 IRISインスタンスを実行する

私は単にIRIS 2019.4コンテナを「リモート」データベースサーバとして実行しましたが、適切な権限のアクセス権を持つIRISインスタンスであれば、どこでも使用できます。

```
zhongli@UKM5530ZHONGLI MINGW64 /c/Program Files/Docker Toolbox
$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
NAMES
d86be69a03ab   quickml-demo  "/iris-main"           3 days ago    Up 3 days (healthy)
0.0.0.0:9091->51773/tcp, 0.0.0.0:9092->52773/tcp quickml
```

3.2 AnacondaとJupyterノートブック

Anacondaについては [こちら](#) のセッション4.1に説明されています。Jupyterノートブックについては [こちら](#) に説明されています。Python 3.xはこのステップでインストールされます。

3.3 JayDeBeApiとJPypeをインストールする

私は自分のJupyterノートブックを起動してから、そのセルで以下を実行し、Python-to-JDBC/Javaブリッジをセットアップしました。

```
!conda install --yes -c conda-forge jaydebeapi
```

この記事の執筆時点(2020年1月)では、JeDeBeApiはJPype 0.7を使用していますが、既知のバグによって機能しないため、0.6.3にダウングレードする必要があります。

```
!conda install --yes -c conda-forge JPype1=0.6.3 --force-reinstall
```

3.4 JDBC経由でIRISデータベースに接続する

公式の [JDBCからIRISへの接続に関するドキュメント](#) はこちらにあります。

JDBCでPyshon SQLを実行するには、以下のコードを例として使用しました。このIRISインスタンスの「USER」名前空間内にある「DataMining.IrisDataset」というデータテーブルに接続します。

```
### 1. Set environment variables, if necessary
#import os
#os.environ['JAVA_HOME']='C: / Progra~1 / Java /jdk1.8.0_241'
#os.environ['CLASSPATH']='C: / interSystems / IRIS20194 / dev / java / lib / JDK18 / intersystems-jdbc-3.0.0.jar'
#os.environ['HADOOP_HOME']='C: / hadoop / bin' #winutil binary must be in Hadoop's Home
```

```
### 2. Get jdbc connection and cursor
import jaydebeapi
url = "jdbc:IRIS://192.168.99.101:9091/USER"
driver = 'com.intersystems.jdbc.IRISDriver'
user = "SUPERUSER"
password = "SYS"
#libx = "C:/InterSystems/IRIS20194/dev/java/lib/JDK18"
jarfile = "C:/InterSystems/IRIS20194/dev/java/lib/JDK18/intersystems-jdbc-3.0.0.jar"
conn = jaydebeapi.connect(driver, url, [user, password], jarfile)
curs = conn.cursor()
### 3. specify the source data table
dataTable = 'DataMining.IrisDataset'

### 4. Get the result and display
curs.execute("select TOP 20 * from %s" % dataTable)
result = curs.fetchall()
print("Total records: " + str(len(result)))
for i in range(len(result)):
    print(result[i])
### 5. Close and clean - I keep them open for next accesses.
#curs.close()
#conn.close()
```

```
Total records: 150
(1, 1.4, 0.2, 5.1, 3.5, 'Iris-setosa')
(2, 1.4, 0.2, 4.9, 3.0, 'Iris-setosa')
(3, 1.3, 0.2, 4.7, 3.2, 'Iris-setosa')
... ..
(49, 1.5, 0.2, 5.3, 3.7, 'Iris-setosa')
(50, 1.4, 0.2, 5.0, 3.3, 'Iris-setosa')
(51, 4.7, 1.4, 7.0, 3.2, 'Iris-versicolor')
... ..
(145, 5.7, 2.5, 6.7, 3.3, 'Iris-virginica')
... ..
(148, 5.2, 2.0, 6.5, 3.0, 'Iris-virginica')
(149, 5.4, 2.3, 6.2, 3.4, 'Iris-virginica')
(150, 5.1, 1.8, 5.9, 3.0, 'Iris-virginica')
```

ここで、JDBCでPythonが機能していることをテストしました。以前はちょっとした日常業務的なデータ分析通常のMLパイプラインのプロトセッシングであり、後のデモと比較で何度も触れる内容です。ここでは、

???に添付しています。

3.5 SQL結果をPandasデータフレーム、そしてNumPy配列に変換する

Pandas/NumPyパッケージがインストールされていない場合は、上記のセクション3.3と同様に、Condaを使ってインストールします。

次に、以下のコードを例として実行しました。

```
### transform SQL results "sqlData" to Pandas dataframe "df", then further to NumPy array "arrayN" for further ML pipelines
import pandas as pd
sqlData = "SELECT * from DataMining.IrisDataset"
```

```

df= pd.io.sql.read_sql(sqlData, conn)
df = df.drop('ID', 1)
df = df[['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth', 'Species']]
# set the labels to 0, 1, 2, for NumPy matrix
df.replace('Iris-setosa', 0, inplace=True)
df.replace('Iris-versicolor', 1, inplace=True)
df.replace('Iris-virginica', 2, inplace=True)
# turn dataframe into Numpy array
arrayN = df.to_numpy()
### 6. Close and clean - if connection is not needed anymore?
#curs.close()
#conn.close()

```

いつものように現在のデータを覗いてみましょう。

```
df.head(5)
```

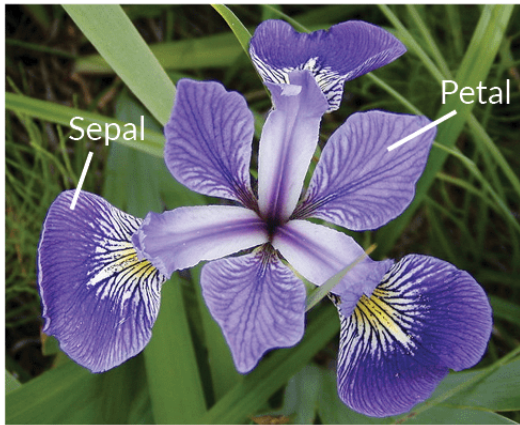
	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
df.describe()
```

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667	1.000000
std	0.828066	0.433594	1.764420	0.763161	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

データフレームは、ソースデータテーブルから正規化されたNumPy配列を取得して、使用できるようになりました。

もちろん、[こちらのリンクのように](#) _____、MLユースケースのようにPythonで行うような様々なマシン分析を試して、Rを置き換えることができます。



Iris Versicolor



Iris Setosa



Iris Virginica

[データソースはこちらから引用されています。](#)

3.6 SQLでデータを分割してIRISデータベースに書き込む

確かに、以降でIRISの刺激的なML機能で使用できるように、データを通常どおりトレーニングセットと検証セットはテストセットに分割し、一時データベースに書き込むことができます。

```
import numpy as np
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
# keep e.g. 20% = 30 rows as test data; trained on another e.g. 80% = 120 rows
X = arrayN[:,0:4]
y = arrayN[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=0.20, random_state=1, shuffle=True)
# make 80% of random rows into a Train set
labels1 = np.reshape(Y_train,(120,1))
train = np.concatenate([X_train, labels1],axis=-1)
# make 20% of left rows into Test set
ITest1 = np.reshape(Y_validation,(30,1))
test = np.concatenate([X_validation, ITest1],axis=-1)
# write the train data set into a Pandas frame
dfTrain = pd.DataFrame({'SepalLength':train[:, 0], 'SepalWidth':train[:, 1], 'PetalLength':train[:, 2], 'PetalWidth':train[:, 3], 'Species':train[:, 4]})
dfTrain['Species'].replace(0, 'Iris-setosa', inplace=True)
dfTrain['Species'].replace(1, 'Iris-versicolor', inplace=True)
dfTrain['Species'].replace(2, 'Iris-virginica', inplace=True)
# write the test data into another Pandas frame
dfTest = pd.DataFrame({'SepalLength':test[:, 0], 'SepalWidth':test[:, 1], 'PetalLength':test[:, 2], 'PetalWidth':test[:, 3], 'Species':test[:, 4]})
dfTest['Species'].replace(0, 'Iris-setosa', inplace=True)
dfTest['Species'].replace(1, 'Iris-versicolor', inplace=True)
dfTest['Species'].replace(2, 'Iris-virginica', inplace=True)
#### 3. specify temp table names
#dataTable = 'DataMining.IrisDataset'
dtTrain = 'TRAIN02'
dtTest = "TEST02"
#### 4. Create 2 temporary tables - you can try drop tables then re-create them every time
curs.execute("Create Table %s (%s DOUBLE, %s DOUBLE, %s DOUBLE, %s DOUBLE, %s VARCHAR(100))" %
(dtTrain, dfTrain.columns[0], dfTrain.columns[1], dfTrain.columns[2], dfTrain.columns[3], dfTrain.columns[4]))
curs.execute("Create Table %s (%s DOUBLE, %s DOUBLE, %s DOUBLE, %s DOUBLE, %s VARCHAR(100))" %
(dtTest, dfTest.columns[0], dfTest.columns[1], dfTest.columns[2], dfTest.columns[3], dfTest.columns[4]))
```

```
### 5. write Train set and Test set into the tables. You can try to delete old record then insert everytime.
curs.fast_executemany = True
curs.executemany( "INSERT INTO %s (SepalLength, SepalWidth, PetalLength, PetalWidth, Species) VALUES (?, ?, ?, ?)" % dtTrain,
    list(dfTrain.itertuples(index=False, name=None)) )
curs.executemany( "INSERT INTO %s (SepalLength, SepalWidth, PetalLength, PetalWidth, Species) VALUES (?, ?, ?, ?)" % dtTest,
    list(dfTest.itertuples(index=False, name=None)) )
### 6. Close and clean - if connection is not needed anymore?
#curs.close()
#conn.close()
```

ここで、IRIS管理コンソールまたはターミナルSQLコンソールに切り替えると、120行のTRAIN02と30行のTEST02という2つの一次テーブルが作成されているのわかります。

この記事は非常に短いメモを目標しているため、本内容はここまでとします。

4. 警告

- 上記のコンテンツは変更または改善される場合があります。

5. 今後の内容

メモを貢献していただける方がいっしょにできれば、セッション3.3:3.4をPyODBC、PySpark、およびIRIS用PythonネイティブAPIに置き換えることにします。貢献していただけるのであれば、大感謝です。

[#JDBC #ODBC #Python #機械学習 #InterSystems IRIS](#)

ソースURL: <https://jp.community.intersystems.com/post/iris%E3%83%87%E3%83%BC%E3%82%BF%E3%83%99%E3%83%BC%E3%82%B9%E3%81%B8%E3%81%AEpython-jdbc%E6%8E%A5%E7%B6%9A-%E7%B0%A1%E6%98%93%E3%83%A1%E3%83%A2>