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ドライバーを呼び出し、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と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とネイティブSQLを使ったPyODBCを使ってはどうでしょうか？
2. **JDBC**: Python 3とネイティブSQLを使ったJayDeBeApiはどうでしょうか？
3. **Spark**: PySparkとSQLを使ったら？
4. **IRIS用Pythonネイティブ API**:
   前に使用したCache用Pythonバイディングを拡張してみたらどうでしょうか？
5. **IPtyhon Magic SQL %%sql**とした場合、 それはまだ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？ それらも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、次回の簡易メモで要約することにします。

範囲内:

このクイックデモでは、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:

この簡易メモでは、以下の項目には触れていません。重要な項目であり、具体的なサイトソリューション、デプロ

イ、およびサービスで個別に対処される可能性があります。

エンドツー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とサポート
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コンテナーを「リモート」データベースサーバーとして実行しましたが、
適切な承認済みのアクセス権を持つIRISインスタンスであれば、どれでも使用できます。

```
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]
のセクション4.1に説明
されたセットアップアプローチ、Jupyterノート
ブックについては[こちら]に説明されたセットアップアプローチをノートパソコンで再利用します。 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した。

```
!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
」ネームスペース内にある「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。 以下はちょっとした日常業務的なデータ分析と通常のML
パイプラインの
プリプロセッシングであり、後のデモと比較で何度も触れる内容です。ここでは、利便的に添付しています。

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と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.tonumpy()
  ### 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()

データフレームと、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ユーザーが以下のようにPython
で行うような様々なルーチン分析を試して、Rを置き換えることができます。

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またはテストセットに分割し、一時データベーステーブルに書き込むことができます。

```
  import numpy as np
from matplotlib import pyplot
from sklearn.modelselection import traintestsplit
  # keep e.g. 20% = 30 rows as test data; trained on another e.g. 80% = 120 rows
X = arrayN[:,0:4]
y = arrayN[:,4]
Xtrain, Xvalidation, Ytrain, Yvalidation = traintestsplit(X, y, testsize=0.20, randomstate=1, shuffle=True)
  # make 80% of random rows into a Train set
labels1 = np.reshape(Ytrain,(120,1))
train = np.concatenate([Xtrain, labels1],axis=-1)
  # make 20% of left rows into Test set
lTest1 = np.reshape(Yvalidation,(30,1))
test = np.concatenate([Xvalidation, lTest1],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 tales. You can try to delete old record then insert everytime.
curs.fastexecutemany = 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がわかります。

この記事は非常に短い簡易メモを目的としているため、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

簡易メモを貢献していただける方がいらっしゃらなければ、セクション3.3と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 #Machine Learning (ML) #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