記事
Toshihiko Minamoto · 2022年2月3日 12m read

Open Exchange

PandasデータフレームをIRISに保存する - 簡易メモ

キーワード: Pandasデータフレーム、IRIS、Python、JDBS

目的

PandasデータフレームはEDA（探索的データ分析）に一般的に使用されるツールです。
MLタスクは通常、データをもう少し理解することから始まります。
先週、私はKaggleにあるこちらのCovid19データセットを試していました。 基本的に、このデータは1925件の遭遇の行と231列で構成されており、タスクは、患者（1つ以上の遭遇レコードにリンク）がICUに入室するかどうかを予測するものです。 つまりこれは、いつものようにpandas.DataFrameを使用して、まず簡単にデータを確認する、通常の分類タスクです。

現在では、IRIS IntegratedML
が提供されています。これには強力な「AutoML」のオプションに関する洗練されたSQLラッパーがあるため、従来型のMLアルゴリズムに対抗して、多様なデータフレームのステージをIRISデータベーステーブルに保存してから、IntegratedMLを実行する方法を頻繁に採用しています。
ただし、dataframe.tosql()
はまだIRISで機能しないため、実際には、ほとんどの時間を他のデータ保存手段をいじることに充てていました。
いわば、土曜の朝の楽しい朝食の時間に素敵なオムレツを作ろうとしていたのに、一日中コンロの下で、ガスとシンクの配管作業をしていたような状況です。
さて、完璧ではありませんが、数週間後に忘れてしまわないように、簡単なメモを残しておくことにします。

範囲

IRISでdataframe.tosql()
を作成しませんでした。残念ながら、まだそこにはたどり着いていませんが、JDBC（JayDeBeApi）を介してデータフレームを動的に直接IRISに保存する簡単なPython関数をできるだけ単純かつ生の状態を維持してさくせいしました。 既知の問題（「MemoryError」）により、PyODBCではまだ機能しない可能性があります。

環境

以下のスクリプトのベースとして、単純なdocker-coposeを介したこちらのIntegratedMLテンプレート
を使って、テストしています。 環境トポロジーはGitHubリポジトリに含まれます。
IRISコンテナーに接続するには、こちらのJDBC Jupyterノートブックを使用しています。

テスト

1. dataframe.tosql() をエミュレートするPython関数を定義する

ノートブックのセルで以下を実行しました。

```
def to_sql_iris(cursor, dataFrame, tableName, schemaName='SQLUser', drop_table=False
):
&lt;span style="color:#999999;">        """
```

```
        Dynamically insert dataframe into an IRIS table via SQL by "excutemany"

        Inputs:
            cursor:      Python JDBC or PyODBC cursor from a valid and estabilised DB
connection
            dataFrame:   Pandas dataframe
            tablename:   IRIS SQL table to be created, inserted or apended
            schemaName:  IRIS schemaName, default to "SQLUser"
            drop_table:  If the table already exsits, drop it and re-
create it if True; othrewise keep it and appen
        Output:
            True is successful; False if there is any exception.
        """&lt;/span>
        if drop_table:
            try:
                curs.execute("DROP TABLE %s.%s" %(schemaName, tableName))
            except Exception:
                pass

        try:
            dataFrame.columns = dataFrame.columns.str.replace("[() -]", "_")
            curs.execute(pd.io.sql.get_schema(dataFrame, tableName))
        except Exception:
            pass

        curs.fast_executemany = True
        cols = ", ".join([str(i) for i in dataFrame.columns.tolist()])
        wildc =''.join('?, ' * len(dataFrame.columns))
        wildc = '(' + wildc[:-2] + ')'
        sql = "INSERT INTO " + tableName + " ( " + cols.replace('-', '_') + " ) VALUE
S" + wildc
        #print(sql)
        curs.executemany(sql, list(dataFrame.itertuples(index=False, name=None)) )
        return True
```

基本的に、上記はIRISテーブルにデータフレームを動的に挿入しようとしています。 テーブルがすでに存在する場合は、完全なデータフレームがその最後にアペンドされますが、存在しない場合は、データフレームの次元（列名と列の型）に基づく新しいテーブルが作成され、その全コンテンツが挿入されます。
executemanyメソッドを使用しているだけです。

2. テスト - 生データファイルをデータフレームに読み込む

ノートブックで以下のコードを実行し、ローカルのドライブからデータフレームに生データを読み込みます。
生データは、こちらのKaggleサイトからダウンロード可能です。

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, roc_auc_score, roc_curve
import seaborn as sns
sns.set(style="whitegrid")
```

```
import os
for dirname, _, filenames in os.walk('./input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

df = pd.read_excel("./input/raw_data_kaggle_covid_icu.xlsx")

df
```

./input/datasets605991272346KaggleSirioLibanesICUPrediction.xlsx
Out[2]:

	PATIENT_VISIT_IDENTIFIER	AGE_ABOVE65	AGE_PERCENTIL	GENDER	DISEASE GROUPING 1	DISEASE GROUPING 2	DISEASE GROUPING 3	DISEASE GROUPING 4	DISEASE GROUPING 5	DISEASE GROUPING 6	...	TEMPERATURE_DIFF	OXYGEN_SATURATION_DIFF	BLOOD_PRESSURE_DIASTOLIC_DIFF_REL	BLOOD_PRESSURE_SISTOLICDIFF_REL	HEART_RATE_DIFF_REL	RESPIRATORY_RATE_DIFF_REL	TEMPERATURE_DIFF_REL	OXYGEN_SATURATION_DIFF_REL	WINDOW	ICU
		1	60th		0.0	0.0	0.0	0.0	1.0	1.0	...	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	0-2	
1		1	60th		0.0	0.0	0.0	0.0	1.0	1.0	...	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	2-4	
2		1	60th		0.0	0.0	0.0	0.0	1.0	1.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4-6	
3		1	60th		0.0	0.0	0.0	0.0	1.0	1.0	...	-1.000000	-1.000000	NaN	NaN	NaN	NaN	-1.000000	-1.000000	6-12	
4		1	60th		0.0	0.0	0.0	0.0	1.0	1.0	...	-0.238095	-0.818182	-0.389967	0.407558	-0.230462	0.096774	-0.224222	-0.814433	ABOVE12	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1920	384		50th	1	0.0	0.0	0.0	0.0	0.0	0.0	...	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	0-2	
1921	384		50th	1	0.0	0.0	0.0	0.0	0.0	0.0	...	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	2-4	
1922	384		50th	1	0.0	0.0	0.0	0.0	0.0	0.0	...	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	4-6	
1923	384		50th	1	0.0	0.0	0.0	0.0	0.0	0.0	...	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	6-12	
1924	384		50th	1	0.0	0.0	1.0	0.0	0.0	0.0	...	-0.547619	-0.838384	-0.701863	-0.585967	-0.763868	-0.612903	-0.551337	-0.835052	ABOVE12	

1925 rows × 231 columns

3. テスト - Python over JDBCでIRIS DBに接続する

```
import jaydebeapi
url = "jdbc:IRIS://irisimlsvr:51773/USER"
driver = 'com.intersystems.jdbc.IRISDriver'
user = "SUPERUSER"
password = "SYS"
jarfile = "./intersystems-jdbc-3.1.0.jar"



conn = jaydebeapi.connect(driver, url, [user, password], jarfile)
curs = conn.cursor()
```

4. テスト - データフレームをIRISテーブルに保存する

```
iris_schema = 'SQLUser'
iris_table = 'Covid19RawTableFromCSV'



to_sql_iris(curs, df, iris_table, iris_schema, drop_table=True)   # save it into a ne
w IRIS table of specified name
#to_sql_iris(curs, df, iris_table)  # append dataframe to an exsiting IRIS table
```

Out[4]: True

```
import pandas as pd
from IPython.display import display



df2 = pd.read_sql("SELECT COUNT(*) from %s.%s" %(iris_schema, iris_table),conn)
display(df2)
```

	Aggregate1
0	1925

したがって、全データが「Covid19RawTableFromCSV」というIRISテーブルに挿入されました。IRIS管理ポータルにログインすると、レコードが含まれるそのテーブルも表示されます。

5. テスト - 簡単なベンチマークを実行する

このデータフレームをたとえば10回挿入して、1つのJDBCセッションでこの基本的なCE
dockerに掛かった時間を確認しましょう。

```
from tqdm import tqdm
import pandas as pd
import time
from IPython.display import display
```

```
start = time.clock()
for i in tqdm(range(0, 10)):
    to_sql_iris(curs, df, iris_table)

print("Total time elasped: ", time.clock()-start, " for importing total records:")
df2 = pd.read_sql("SELECT COUNT(*) from %s.%s" %(iris_schema, iris_table),conn)
display(df2)


100%|??????????| 10/10 [00:14<00:00,  1.42s/it]

Total time elasped:  12.612431999999998  for importing total records:
```

	Aggregate1
0	19250

以上です。非常に基本的ではありますが、少なくともデータ分析パイプラインに沿って操作されたデータフレームを保存し、多少本格的なMLを試すために、SQLインターフェースを介してIntegratedMLを呼び出すことができるようになりました。

言及すべき**警告**:
データフレームの文字列は、「
オブジェクト」として解釈されることがあるため、
df['column'].astype(str)などを使用して、IRISテーブルに挿入される前に文字列に変換する必要があります。
「DROP TABLE」は前のテーブルを上書きするために使用されます。 「DROP
VIEW」は前のビューを削除するために使用できます。

#JDBC #Python #Machine Learning (ML) #InterSystems IRIS
InterSystems Open Exchangeで関連アプリケーションを確認してください

ソースURL:
https://jp.community.intersystems.com/post/pandas%E3%83%87%E3%83%BC%E3%82%BF%E3%83%95%E3%83%AC%E3%83%BC%E3%83%A0%E3%82%92iris%E3%81%AB%E4%BF%9D%E5%AD%98%E3%81%99%E3%82%8B-%E7%B0%A1%E6%98%93%E3%83%A1%E3%83%A2