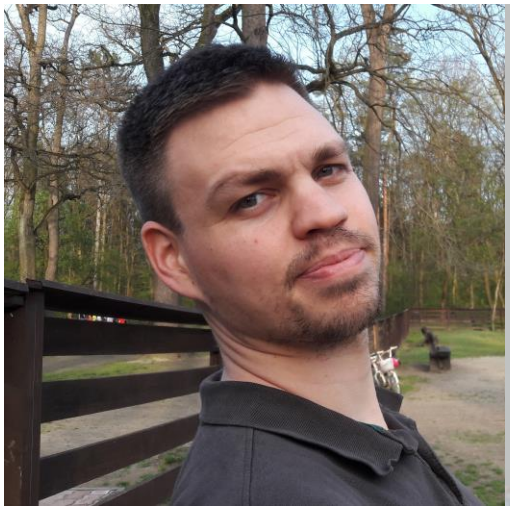


Dr. Hegedűs Péter, Dr. Ferenc Rudolf

Nagyméretű adatbázisok

Jelen tananyag a Szegedi Tudományegyetemen készült az Európai Unió támogatásával.

Projekt azonosító: EFOP-3.4.3-16-2016-00014

Szegedi Tudományegyetem
Cím: 6720 Szeged, Dugonics tér 13.
www.u-szeged.hu
www.szechenyi2020.hu

Spark SQL és MLLib

Összefoglalás

Ez az olvasólecke gyakorlati tudást nyújt a Spark SQL és az MLLib modulok használatáról. Részletesen foglalkozunk a Spark által nyújtott SQL lekérdezési funkciókkal, azok Scala és Python API-n keresztül történő elérésével. Áttekintjuk a gépi tanuláshoz használható MLLib modulok főbb osztályait, és egy konkért gépi tanulási feladaton keresztül áttekintjük az egyes fázisokhoz adott Spark támogatást (pl. feature előállítás, normalizálás, modell tanítás, kiértékelés).
A lecke fejezetei:

- 1. fejezet: **A Spark SQL API ismertetése, Scala és Python API-k bemutatása példákon keresztül (olvasó)**
- 2. fejezet: **A Spark MLLib bemutatása egy összetettebb példa Python-ban történő megoldásán keresztül (olvasó)**

Téma típusa: **gyakorlati**
Olvasási idő: **40 perc**

📖 1. fejezet

Spark SQL API

A Spark részét képezi a Spark SQL modul [1], ami a strukturált adatfeldolgozást támogatja. A Spark SQL integrálja a relációs adat feldolgozást a Spark funkcionális programozási modelljével. Több féle adatforrást is támogat, és az SQL-ben megfogalmazott lekérdezéseket Spark transzformációkká és akciókká alakítja. A klasszikus RDD API-hoz képest a Spark SQL sokkal több információt tartalmaz mind az adatok, mind pedig a rajtuk végzendő műveletek struktúrájáról, amit a Spark végrehajtáskor hatékonyan ki tud használni a végrehajtás optimalizálásához. Az SQL utasítások végrehajtásához a `spark.sql` modul használható Scala, Java, R és Python nyelveken is.

Scala/Java

Első példaként töltsük be (lkoálisan a `spark-master` gépről vagy HDFS-ről) a `2g_BigData-data-transform-SPOC` olvasóleckében bemutatott JSON fájlt (`code/2g_BigData-data-transform-SPOC/source/personal_entries.json`), és végezzünk SQL lekérdezéseket rajta `spark-shell` segítségével (Scala API). Indítsuk el az előző leckékben bemutatott Spark docker stack-et (`docker-hadoop-spark-workbench`), másoljuk át a container-be a JSON fájlt, lépjünk be a `spark-master` container-be és futtassuk a shell-t:

```
$ docker-compose up -d
$ docker cp personal_entries.json spark-master:/spark/personal_entries.json
$ docker cp sales_entries.csv spark-master:/spark/sales_entries.csv
$ docker exec -it spark-master bash
bash-5.0# bin/spark-shell
20/09/01 08:00:22 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
```

```
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
Spark context Web UI available at http://43642cd7fb93:4040
Spark context available as 'sc' (master = local[*], app id = local-
1598947229906).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.0.0
      /_/

Using Scala version 2.12.10 (OpenJDK 64-Bit Server VM, Java 1.8.0_252)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val jsonTable = spark.read.json("file:///spark/personal_entries.json")
df: org.apache.spark.sql.DataFrame = [PID: string, birth_year: bigint ... 4 more
fields]

scala> jsonTable.registerTempTable("personal_json")

scala> val df = spark.sql("SELECT * FROM personal_json")
df: org.apache.spark.sql.DataFrame = [PID: string, birth_year: bigint ... 4 more
fields]

scala> df.show()
+----------+----------+------+-------------------+----------+---------------+
|       PID|birth_year|gender|     last_contacted|      name|   useless_info|
+----------+----------+------+-------------------+----------+---------------+
|1122334455|      1987|     M|2017-03-02 11:43:00|  John Doe|      [1, 2, 3]|
|2233445566|      1965|     F|2019-10-12 15:22:34|  Jane Doe|      [2, 3, 4]|
|3344556677|      1999|      |2020-01-05 08:27:12|John Smith|[1, 2, 3, 4, 5]|
+----------+----------+------+-------------------+----------+---------------+


scala>
```

A `spark.read.json` metódus közvetlenül JSON fájlból tud `DataFrame`-et építeni. Tetszőleges `DataFrame` (nem csak a JSON-ből betöltött) regisztrálható mint SQL tábla (`registerTempTable()` metódus hívás), ami után az SQL API segítségével tetszőleges SQL lekérdezést hajthatunk végre rajta. Töltsük még be a `sales_entries.csv` fájlt is, majd kapcsoljuk össze a két táblát egy lekérdezésben:

```
scala> val csvTable = spark.read.option("header", true).option("delimiter",
";").csv("file:///spark/sales_entries.csv")
csvTable: org.apache.spark.sql.DataFrame = [PID: string, offer_date: string ...
1 more field]

scala> csvTable.registerTempTable("sales_csv")

scala> val join_res = spark.sql("SELECT * FROM personal_json p, sales_csv s
WHERE p.PID = s.PID")
join_res: org.apache.spark.sql.DataFrame = [PID: string, birth_year: bigint ...
7 more fields]
```

```
scala> join_res.select("p.PID", "birth_year", "name", "offer_date",
"offer_text").show()
+----------+----------+----------+----------+--------------+
|       PID|birth_year|      name|offer_date|    offer_text|
+----------+----------+----------+----------+--------------+
|1122334455|      1987|  John Doe|2017-03-02|          null|
|2233445566|      1965|  Jane Doe|2019-10-12|          null|
|3344556677|      1999|John Smith|2020-01-05|terrific offer|
+----------+----------+----------+----------+--------------+
```

Python

Természetesen a Spark SQL API Python-ban is rendelkezésre áll. A fenti példák megoldása `pyspark` shell-ben így néz ki:

```
bash-5.0# bin/pyspark
Python 2.7.18 (default, May  3 2020, 22:58:12)
[GCC 8.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
20/09/01 08:36:57 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
/spark/python/pyspark/context.py:220: DeprecationWarning: Support for Python 2
and Python 3 prior to version 3.6 is deprecated as of Spark 3.0. See also the
plan for dropping Python 2 support at https://spark.apache.org/news/plan-for-
dropping-python-2-support.html.
  DeprecationWarning)
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.0.0
      /_/

Using Python version 2.7.18 (default, May  3 2020 22:58:12)
SparkSession available as 'spark'.
>>> jsonTable = spark.read.json("file:///spark/personal_entries.json")
>>> jsonTable.registerTempTable("personal_json")
>>> csvTable = spark.read.option("header", True).option("delimiter",
";").csv("file:///spark/sales_entries.csv")
>>> csvTable.registerTempTable("sales_csv")
>>> join_res = spark.sql("SELECT * FROM personal_json p, sales_csv s WHERE p.PID
= s.PID")
>>> join_res.select("p.PID", "birth_year", "name", "offer_date",
"offer_text").show()
+----------+----------+----------+----------+--------------+
|       PID|birth_year|      name|offer_date|    offer_text|
+----------+----------+----------+----------+--------------+
|1122334455|      1987|  John Doe|2017-03-02|          null|
|2233445566|      1965|  Jane Doe|2019-10-12|          null|
|3344556677|      1999|John Smith|2020-01-05|terrific offer|
+----------+----------+----------+----------+--------------+
```

📖 2. fejezet

Az MLLib modul használata

Az MLlib [7] (vagy újabban Spark ML) modul a Spark gépi tanulást támogató komponense. Segít abban, hogy a Hadoop klaszteren tárolt adatainkra gyorsan, könnyedén és jól skálázható módon alkalmazhassunk gépi tanuló algoritmusokat. Az MLlib az újabb verzióktól kezdve a DataFrame API-ra épül, noha az RDD alapú API is karbantartás alatt marad.

A fejezet során az alábbi gépi tanulási feladatot végezzük el:

- A bemenő adattáblánk (`code/X_BigData-spark-sql-mllib-SPOC/reprocessed.hungarian.data`) egy nyilvános adathalmaz [3], ahol páciensek 13 különböző laborértéke (a hiányzó adatok -9-es értékkel vannak jelölve) és egy 0-4 fokozatú skála szerepel, ami meghatározza, hogy az adott páciens egy fajta szívbetegségben szenved-e és milyen fokú az (0 - egészséges, 4 - nagyon beteg).
- A 13 oszlopot egy `feature` vektorba szervezzük, amelyet standardizálunk (azaz az értékeket 0 várható értékre és 1 szórásúra alakítjuk az átlag és szórás segítségével).
- Ezután az eredeti adathalmazunkat felosztjuk egy `train` és egy `test` részre 80%-20% arányban.
- A `train` adatokon betanítunk egy klasszikus döntési fa osztályozót, ami több címkés osztályozást végez majd.
- Végül a modellt kiértékeljük a `test` adathalmazon `F1` érték alapján.

A vázolt lépések végrehajtása a `pyspark` shell-ben az alábbi módon néz ki:

```
>>> from pyspark.ml.feature import VectorAssembler
>>> from pyspark.ml.classification import DecisionTreeClassifier
>>> from pyspark.ml.evaluation import MulticlassClassificationEvaluator
>>> data = spark.read.option('delimiter', ' ').option('inferSchema',
True).csv('file:///spark/reprocessed.hungarian.data').withColumnRenamed("_c13",
"label")
>>> data.show()
+---+---+---+---+---+---+---+---+---+---+---+----+----+----+-----+
|_c0|_c1|_c2|_c3|_c4|_c5|_c6|_c7|_c8|_c9|_c10|_c11|_c12|label|
+---+---+---+---+---+---+---+---+---+---+---+----+----+----+-----+
| 40|  1|  2|140|289|  0|  0|172|  0|0.0|  -9|  -9|  -9|    0|
| 49|  0|  3|160|180|  0|  0|156|  0|1.0|   2|  -9|  -9|    1|
| 37|  1|  2|130|283|  0|  1| 98|  0|0.0|  -9|  -9|  -9|    0|
| 48|  0|  4|138|214|  0|  0|108|  1|1.5|   2|  -9|  -9|    3|
| 54|  1|  3|150| -9|  0|  0|122|  0|0.0|  -9|  -9|  -9|    0|
| 39|  1|  3|120|339|  0|  0|170|  0|0.0|  -9|  -9|  -9|    0|
| 45|  0|  2|130|237|  0|  0|170|  0|0.0|  -9|  -9|  -9|    0|
| 54|  1|  2|110|208|  0|  0|142|  0|0.0|  -9|  -9|  -9|    0|
| 37|  1|  4|140|207|  0|  0|130|  1|1.5|   2|  -9|  -9|    1|
| 48|  0|  2|120|284|  0|  0|120|  0|0.0|  -9|  -9|  -9|    0|
| 37|  0|  3|130|211|  0|  0|142|  0|0.0|  -9|  -9|  -9|    0|
| 58|  1|  2|136|164|  0|  1| 99|  1|2.0|   2|  -9|  -9|    3|
| 39|  1|  2|120|204|  0|  0|145|  0|0.0|  -9|  -9|  -9|    0|
| 49|  1|  4|140|234|  0|  0|140|  1|1.0|   2|  -9|  -9|    3|
| 42|  0|  3|115|211|  0|  1|137|  0|0.0|  -9|  -9|  -9|    0|
| 54|  0|  2|120|273|  0|  0|150|  0|1.5|   2|  -9|  -9|    0|
| 38|  1|  4|110|196|  0|  0|166|  0|0.0|  -9|  -9|  -9|    1|
| 43|  0|  2|120|201|  0|  0|165|  0|0.0|  -9|  -9|  -9|    0|
| 60|  1|  4|100|248|  0|  0|125|  0|1.0|   2|  -9|  -9|    1|
| 36|  1|  2|120|267|  0|  0|160|  0|3.0|   2|  -9|  -9|    1|
```

```
+---+---+---+---+---+---+---+---+---+---+----+----+----+-----+
only showing top 20 rows

>>> feature_columns = data.columns[:-1]
>>> assembler = VectorAssembler(inputCols=feature_columns,
outputCol="raw_features")
>>> data_2 = assembler.transform(data)
>>> data_2.show()
+---+---+---+---+---+---+---+---+---+---+----+----+----+-----+-----------------
--+
|_c0|_c1|_c2|_c3|_c4|_c5|_c6|_c7|_c8|_c9|_c10|_c11|_c12|label|
raw_features|
+---+---+---+---+---+---+---+---+---+---+----+----+----+-----+-----------------
--+
| 40|  1|  2|140|289|  0|  0|172|  0|0.0|  -9|  -9|  -9|    0|
[40.0,1.0,2.0,140...|
| 49|  0|  3|160|180|  0|  0|156|  0|1.0|   2|  -9|  -9|    1|
[49.0,0.0,3.0,160...|
| 37|  1|  2|130|283|  0|  1| 98|  0|0.0|  -9|  -9|  -9|    0|
[37.0,1.0,2.0,130...|
| 48|  0|  4|138|214|  0|  0|108|  1|1.5|   2|  -9|  -9|    3|
[48.0,0.0,4.0,138...|
| 54|  1|  3|150| -9|  0|  0|122|  0|0.0|  -9|  -9|  -9|    0|
[54.0,1.0,3.0,150...|
| 39|  1|  3|120|339|  0|  0|170|  0|0.0|  -9|  -9|  -9|    0|
[39.0,1.0,3.0,120...|
| 45|  0|  2|130|237|  0|  0|170|  0|0.0|  -9|  -9|  -9|    0|
[45.0,0.0,2.0,130...|
| 54|  1|  2|110|208|  0|  0|142|  0|0.0|  -9|  -9|  -9|    0|
[54.0,1.0,2.0,110...|
| 37|  1|  4|140|207|  0|  0|130|  1|1.5|   2|  -9|  -9|    1|
[37.0,1.0,4.0,140...|
| 48|  0|  2|120|284|  0|  0|120|  0|0.0|  -9|  -9|  -9|    0|
[48.0,0.0,2.0,120...|
| 37|  0|  3|130|211|  0|  0|142|  0|0.0|  -9|  -9|  -9|    0|
[37.0,0.0,3.0,130...|
| 58|  1|  2|136|164|  0|  1| 99|  1|2.0|   2|  -9|  -9|    3|
[58.0,1.0,2.0,136...|
| 39|  1|  2|120|204|  0|  0|145|  0|0.0|  -9|  -9|  -9|    0|
[39.0,1.0,2.0,120...|
| 49|  1|  4|140|234|  0|  0|140|  1|1.0|   2|  -9|  -9|    3|
[49.0,1.0,4.0,140...|
| 42|  0|  3|115|211|  0|  1|137|  0|0.0|  -9|  -9|  -9|    0|
[42.0,0.0,3.0,115...|
| 54|  0|  2|120|273|  0|  0|150|  0|1.5|   2|  -9|  -9|    0|
[54.0,0.0,2.0,120...|
| 38|  1|  4|110|196|  0|  0|166|  0|0.0|  -9|  -9|  -9|    1|
[38.0,1.0,4.0,110...|
| 43|  0|  2|120|201|  0|  0|165|  0|0.0|  -9|  -9|  -9|    0|
[43.0,0.0,2.0,120...|
| 60|  1|  4|100|248|  0|  0|125|  0|1.0|   2|  -9|  -9|    1|
[60.0,1.0,4.0,100...|
| 36|  1|  2|120|267|  0|  0|160|  0|3.0|   2|  -9|  -9|    1|
[36.0,1.0,2.0,120...|
+---+---+---+---+---+---+---+---+---+---+----+----+----+-----+-----------------
--+
only showing top 20 rows
```

```
>>> scaler = StandardScaler(inputCol="raw_features", outputCol="features",
withStd=True, withMean=True)
>>> scaler = StandardScaler(inputCol="raw_features", outputCol="features",
withStd=True, withMean=True)
>>> scalerModel = scaler.fit(data_2)
>>> data_3 = scalerModel.transform(data_2)
>>> data_3.show()
+---+---+---+---+---+---+---+---+---+---+----+----+----+-----+-----------------
--+--------------------+
|_c0|_c1|_c2|_c3|_c4|_c5|_c6|_c7|_c8|_c9|_c10|_c11|_c12|label|
raw_features|            features|
+---+---+---+---+---+---+---+---+---+---+----+----+----+-----+-----------------
--+--------------------+
| 40|  1|  2|140|289|  0|  0|172|  0|0.0|  -9|  -9|  -9|    0|
[40.0,1.0,2.0,140...|[-1.0018840953055...|
| 49|  0|  3|160|180|  0|  0|156|  0|1.0|   2|  -9|  -9|    1|
[49.0,0.0,3.0,160...|[0.15021730242520...|
| 37|  1|  2|130|283|  0|  1| 98|  0|0.0|  -9|  -9|  -9|    0|
[37.0,1.0,2.0,130...|[-1.3859178945490...|
| 48|  0|  4|138|214|  0|  0|108|  1|1.5|   2|  -9|  -9|    3|
[48.0,0.0,4.0,138...|[0.02220603601068...|
| 54|  1|  3|150| -9|  0|  0|122|  0|0.0|  -9|  -9|  -9|    0|
[54.0,1.0,3.0,150...|[0.79027363449782...|
| 39|  1|  3|120|339|  0|  0|170|  0|0.0|  -9|  -9|  -9|    0|
[39.0,1.0,3.0,120...|[-1.1298953617200...|
| 45|  0|  2|130|237|  0|  0|170|  0|0.0|  -9|  -9|  -9|    0|
[45.0,0.0,2.0,130...|[-0.3618277632328...|
| 54|  1|  2|110|208|  0|  0|142|  0|0.0|  -9|  -9|  -9|    0|
[54.0,1.0,2.0,110...|[0.79027363449782...|
| 37|  1|  4|140|207|  0|  0|130|  1|1.5|   2|  -9|  -9|    1|
[37.0,1.0,4.0,140...|[-1.3859178945490...|
| 48|  0|  2|120|284|  0|  0|120|  0|0.0|  -9|  -9|  -9|    0|
[48.0,0.0,2.0,120...|[0.02220603601068...|
| 37|  0|  3|130|211|  0|  0|142|  0|0.0|  -9|  -9|  -9|    0|
[37.0,0.0,3.0,130...|[-1.3859178945490...|
| 58|  1|  2|136|164|  0|  1| 99|  1|2.0|   2|  -9|  -9|    3|
[58.0,1.0,2.0,136...|[1.30231870015592...|
| 39|  1|  2|120|204|  0|  0|145|  0|0.0|  -9|  -9|  -9|    0|
[39.0,1.0,2.0,120...|[-1.1298953617200...|
| 49|  1|  4|140|234|  0|  0|140|  1|1.0|   2|  -9|  -9|    3|
[49.0,1.0,4.0,140...|[0.15021730242520...|
| 42|  0|  3|115|211|  0|  1|137|  0|0.0|  -9|  -9|  -9|    0|
[42.0,0.0,3.0,115...|[-0.7458615624764...|
| 54|  0|  2|120|273|  0|  0|150|  0|1.5|   2|  -9|  -9|    0|
[54.0,0.0,2.0,120...|[0.79027363449782...|
| 38|  1|  4|110|196|  0|  0|166|  0|0.0|  -9|  -9|  -9|    1|
[38.0,1.0,4.0,110...|[-1.2579066281345...|
| 43|  0|  2|120|201|  0|  0|165|  0|0.0|  -9|  -9|  -9|    0|
[43.0,0.0,2.0,120...|[-0.6178502960619...|
| 60|  1|  4|100|248|  0|  0|125|  0|1.0|   2|  -9|  -9|    1|
[60.0,1.0,4.0,100...|[1.55834123298497...|
| 36|  1|  2|120|267|  0|  0|160|  0|3.0|   2|  -9|  -9|    1|
[36.0,1.0,2.0,120...|[-1.5139291609636...|
+---+---+---+---+---+---+---+---+---+---+----+----+----+-----+-----------------
--+--------------------+
only showing top 20 rows

>>> dt = DecisionTreeClassifier()
```

```
>>> train, test = data_3.randomSplit([0.8, 0.2])
>>> model = dt.fit(train)
>>> predictions = model.transform(test)
>>> evaluator = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="f1")
>>> f1 = evaluator.evaluate(predictions)
>>> print(f1)
0.573991278909
>>> predictions.select(predictions.columns[13:]).show()
+-----+--------------------+--------------------+--------------------+----------
----------+----------+
|label|        raw_features|            features|       rawPrediction|
probability|prediction|
+-----+--------------------+--------------------+--------------------+----------
----------+----------+
|    0|[29.0,1.0,2.0,140...|[-2.4100080258652...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    0|[32.0,0.0,2.0,105...|[-2.0259742266217...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    1|[34.0,1.0,1.0,140...|[-1.7699516937926...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    0|[35.0,0.0,4.0,140...|[-1.6419404273781...|[11.0,0.0,0.0,0.0...|
[0.91666666666666...|       0.0|
|    0|[35.0,1.0,2.0,120...|[-1.6419404273781...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    0|[37.0,0.0,2.0,120...|[-1.3859178945490...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    0|[37.0,1.0,2.0,130...|[-1.3859178945490...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    1|[37.0,1.0,4.0,140...|[-1.3859178945490...|[0.0,1.0,3.0,2.0,...|
[0.0,0.1666666666...|       2.0|
|    0|[38.0,1.0,2.0,140...|[-1.2579066281345...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    1|[38.0,1.0,4.0,120...|[-1.2579066281345...|[11.0,0.0,0.0,0.0...|
[0.91666666666666...|       0.0|
|    0|[39.0,0.0,3.0,110...|[-1.1298953617200...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    0|[39.0,1.0,2.0,190...|[-1.1298953617200...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    0|[41.0,0.0,2.0,125...|[-0.8738728288909...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    0|[41.0,0.0,2.0,130...|[-0.8738728288909...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    0|[41.0,1.0,4.0,112...|[-0.8738728288909...|[3.0,2.0,0.0,0.0,...|
[0.6,0.4,0.0,0.0,...|       0.0|
|    0|[42.0,1.0,2.0,120...|[-0.7458615624764...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    0|[43.0,0.0,2.0,120...|[-0.6178502960619...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    2|[43.0,1.0,1.0,120...|[-0.6178502960619...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    0|[43.0,1.0,2.0,142...|[-0.6178502960619...|[108.0,2.0,1.0,1....|
[0.96428571428571...|       0.0|
|    4|[43.0,1.0,4.0,140...|[-0.6178502960619...|[0.0,8.0,2.0,0.0,...|
[0.0,0.7272727272...|       1.0|
+-----+--------------------+--------------------+--------------------+----------
----------+----------+
only showing top 20 rows
```

További feladatok

1. Töltsük be és kapcsoljuk a többi adathoz Spark SQL segítségével a `billing_entries.json` fájlt is! ⭐⭐
2. A 2. fejezetben bemutatott gépi tanuló feladat lépéseit szervezzük egy `Pipeline`-ba, és egyben hajtsuk végre a tanítást! ⭐⭐
3. Készítsünk az előző feladatban elkészített megoldásból egy önálló Python alkalmazást, és hajtsuk végre `spark-submit` segítségével! ⭐

Referenciák

[1] https://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes

[2] https://spark.apache.org/docs/latest/ml-guide.html

[3] https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/reprocessed.hungarian.data

[4] https://towardsdatascience.com/apache-spark-mllib-tutorial-ec6f1cb336a9