1.SparkSQL

October 11, 2018

1 Dataframes

- Dataframes are a restricted sub-type of RDDs.
- Restircing the type allows for more optimization.
- Dataframes store two dimensional data, similar to the type of data stored in a spreadsheet.
  - Each column in a dataframe can have a different type.
  - Each row contains a `record`.
- Similar to, but not the same as, pandas dataframes and R dataframes

```
In [1]: from pyspark import SparkContext
        sc = SparkContext(master="local[4]")
        sc.version

Out[1]: '2.3.0'

In [2]: import os
        import sys

        from pyspark import SparkContext
        from pyspark.sql import SQLContext
        from pyspark.sql.types import Row, StructField, StructType, StringType, IntegerType
        %pylab inline

Populating the interactive namespace from numpy and matplotlib


In [3]: # Just like using Spark requires having a SparkContext, using SQL requires an SQLConte
        sqlContext = SQLContext(sc)
        sqlContext

Out[3]: <pyspark.sql.context.SQLContext at 0x7f6a30c77e10>
```

1.0.1 Constructing a DataFrame from an RDD of Rows

Each Row defines it's own fields, the schema is *inferred*.

```
In [4]: # One way to create a DataFrame is to first define an RDD from a list of Rows
        some_rdd = sc.parallelize([Row(name=u"John", age=19),
                                   Row(name=u"Smith", age=23),
                                   Row(name=u"Sarah", age=18)])
        some_rdd.collect()
```

```
Out[4]: [Row(age=19, name='John'),
         Row(age=23, name='Smith'),
         Row(age=18, name='Sarah')]
```

```
In [5]: # The DataFrame is created from the RDD or Rows
        # Infer schema from the first row, create a DataFrame and print the schema
        some_df = sqlContext.createDataFrame(some_rdd)
        some_df.printSchema()
```

```
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
```

```
In [6]: # A dataframe is an RDD of rows plus information on the schema.
        # performing **collect()* on either the RDD or the DataFrame gives the same result.
        print(type(some_rdd),type(some_df))
        print('some_df =',some_df.collect())
        print('some_rdd=',some_rdd.collect())
```

```
<class 'pyspark.rdd.RDD'> <class 'pyspark.sql.dataframe.DataFrame'>
some_df = [Row(age=19, name='John'), Row(age=23, name='Smith'), Row(age=18, name='Sarah')]
some_rdd= [Row(age=19, name='John'), Row(age=23, name='Smith'), Row(age=18, name='Sarah')]
```

1.0.2 Defining the Schema explicitly

The advantage of creating a DataFrame using a pre-defined schema allows the content of the RDD to be simple tuples, rather than rows.

```
In [7]: # In this case we create the dataframe from an RDD of tuples (rather than Rows) and pr
        another_rdd = sc.parallelize([("John", 19), ("Smith", 23), ("Sarah", 18)])
        # Schema with two fields - person_name and person_age
        schema = StructType([StructField("person_name", StringType(), False),
                             StructField("person_age", IntegerType(), False)])

        # Create a DataFrame by applying the schema to the RDD and print the schema
        another_df = sqlContext.createDataFrame(another_rdd, schema)
        another_df.printSchema()
        # root
        #  |-- age: binteger (nullable = true)
        #  |-- name: string (nullable = true)
```

```
root
 |-- person_name: string (nullable = false)
 |-- person_age: integer (nullable = false)
```

1.1 Loading DataFrames from disk

There are many maethods to load DataFrames from Disk. Here we will discuss three of these methods 1. Parquet 2. JSON (on your own) 3. CSV (on your own)

In addition, there are API's for connecting Spark to an external database. We will not discuss this type of connection in this class.

1.1.1 Loading dataframes from JSON files

JSON is a very popular readable file format for storing structured data. Among it's many uses are **twitter**, `javascript` communication packets, and many others. In fact this notebook file (with the extension `.ipynb` is in json format. JSON can also be used to store tabular data and can be easily loaded into a dataframe.

```
In [9]: !wget 'https://mas-dse-open.s3.amazonaws.com/Moby-Dick.txt' -P ../../Data/
```

```
--2018-04-09 00:48:07--  https://mas-dse-open.s3.amazonaws.com/Moby-Dick.txt
Resolving mas-dse-open.s3.amazonaws.com (mas-dse-open.s3.amazonaws.com)... 54.231.176.210
Connecting to mas-dse-open.s3.amazonaws.com (mas-dse-open.s3.amazonaws.com)|54.231.176.210|:44
HTTP request sent, awaiting response... 200 OK
Length: 1257260 (1.2M) [text/plain]
Saving to: ../../Data/Moby-Dick.txt.1

Moby-Dick.txt.1     100%[===================>]   1.20M  1.15MB/s    in 1.0s

2018-04-09 00:48:08 (1.15 MB/s) - ../../Data/Moby-Dick.txt.1 saved [1257260/1257260]
```

```
In [10]: # when loading json files you can specify either a single file or a directory contain
         path = "../../Data/people.json"
         !cat $path

         # Create a DataFrame from the file(s) pointed to by path
         people = sqlContext.read.json(path)
         print('people is a',type(people))
         # The inferred schema can be visualized using the printSchema() method.
         people.show()

         people.printSchema()
```

```
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
people is a <class 'pyspark.sql.dataframe.DataFrame'>
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+

root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
```

1.1.2 Excercise: Loading csv files into dataframes

Spark 2.0 includes a facility for reading csv files. In this excercise you are to create similar functionality using your own code.

You are to write a class called `csv_reader` which has the following methods:

- `__init__(self,filepath)`: recieves as input the path to a csv file. It throws an exeption `NoSuchFile` if the file does not exist.
- `Infer_Schema()` opens the file, reads the first 10 lines (or less if the file is shorter), and infers the schema. The first line of the csv file defines the column names. The following lines should have the same number of columns and all of the elements of the column should be of the same type. The only types allowd are `int`,`float`,`string`. The method infers the types of the columns, checks that they are consistent, and defines a dataframe schema of the form:

```
schema = StructType([StructField("person_name", StringType(), False),
                     StructField("person_age", IntegerType(), False)])
```

If everything checks out, the method defines a `self.` variable that stores the schema and returns the schema as it's output. If an error is found an exception `BadCsvFormat` is raised. * `read_DataFrame()`: reads the file, parses it and creates a dataframe using the inferred schema. If one of the lines beyond the first 10 (i.e. a line that was not read by `InferSchema`) is not parsed correctly, the line is not added to the Dataframe. Instead, it is added to an RDD called `bad_lines`. The methods returns the dateFrame and the `bad_lines` RDD.

1.1.3 Parquet files

- Parquet is a popular columnar format.

- Spark SQL allows SQL queries to retrieve a subset of the rows without reading the whole file.

- Compatible with HDFS : allows parallel retrieval on a cluster.
- Parquet compresses the data in each column.

1.1.4 Spark and Hive

- Parquet is a **file format** not an independent database server.
- Spark can work with the Hive relational database system that supports the full array of database operations.
- Hive is compatible with HDFS.

```
In [11]: dir='../../Data'
         parquet_file=dir+"/users.parquet"
         !ls $dir
```

```
Moby-Dick.txt          namesAndFavColors.parquet  users.parquet    US_stations.tsv.gz
Moby-Dick.txt.1  people.json                     US_stations.tsv  Weather
```

```
In [12]: #load a Parquet file
         print(parquet_file)
         df = sqlContext.read.load(parquet_file)
         df.show()
```

```
../../Data/users.parquet
+------+--------------+----------------+
|  name|favorite_color|favorite_numbers|
+------+--------------+----------------+
|Alyssa|          null|  [3, 9, 15, 20]|
|   Ben|           red|              []|
+------+--------------+----------------+
```

```
In [13]: df2=df.select("name", "favorite_color")
         df2.show()
```

```
+------+--------------+
|  name|favorite_color|
+------+--------------+
|Alyssa|          null|
|   Ben|           red|
+------+--------------+
```

```
In [14]: outfilename="namesAndFavColors.parquet"
         !rm -rf $dir/$outfilename
         df2.write.save(dir+"/"+outfilename)
         !ls -ld $dir/$outfilename
```

```
drwxrwxrwx 1 jovyan staff 4096 Apr  9  2018 ../../Data/namesAndFavColors.parquet
```

A new interface object has been added in **Spark 2.0** called **SparkSession**. A spark session is initialized using a `builder`. For example

```
spark = SparkSession.builder \
         .master("local") \
         .appName("Word Count") \
         .config("spark.some.config.option", "some-value") \
         .getOrCreate()
```

Using a SparkSession a Parquet file is read as follows::

```
df = spark.read.parquet('python/test_support/sql/parquet_partitioned')
```

1.2 Lets have a look at a real-world dataframe

This dataframe is a small part from a large dataframe (15GB) which stores meteorological data from stations around the world. We will read the dataframe from a zipped parquet file.

```
In [22]: from os.path import split,join,exists
         from os import mkdir,getcwd,remove
         from glob import glob

         # create directory if needed

         notebook_dir=getcwd()
         data_dir=join(split(split(notebook_dir)[0])[0],'Data')
         weather_dir=join(data_dir,'Weather')

         if exists(weather_dir):
             print('directory',weather_dir,'already exists')
         else:
             print('making',weather_dir)
             mkdir(weather_dir)

         file_index='NY'
         zip_file='%s.tgz'%(file_index) #the .csv extension is a mistake, this is a pickle fil
         old_files='%s/%s*'%(weather_dir,zip_file[:-3])
         for f in glob(old_files):
             print('removing',f)
             !rm -rf {f}
```

```
directory /home/jovyan/work/Sections/Data/Weather already exists
removing /home/jovyan/work/Sections/Data/Weather/NY.parquet
removing /home/jovyan/work/Sections/Data/Weather/NY.tgz
```

```
In [23]: command="wget https://mas-dse-open.s3.amazonaws.com/Weather/by_state/%s -P %s "%(zip_
         print(command)
         !$command
         !ls -lh $weather_dir/$zip_file

wget https://mas-dse-open.s3.amazonaws.com/Weather/by_state/NY.tgz -P /home/jovyan/work/Sectio
--2018-04-09 00:52:22--  https://mas-dse-open.s3.amazonaws.com/Weather/by_state/NY.tgz
Resolving mas-dse-open.s3.amazonaws.com (mas-dse-open.s3.amazonaws.com)... 52.218.144.50
Connecting to mas-dse-open.s3.amazonaws.com (mas-dse-open.s3.amazonaws.com)|52.218.144.50|:443
HTTP request sent, awaiting response... 200 OK
Length: 23182008 (22M) [application/x-tar]
Saving to: /home/jovyan/work/Sections/Data/Weather/NY.tgz

NY.tgz              100%[===================>]  22.11M  1.47MB/s    in 15s

2018-04-09 00:52:37 (1.51 MB/s) - /home/jovyan/work/Sections/Data/Weather/NY.tgz saved [231820

-rwxrwxrwx 1 jovyan staff 23M Mar 16 20:25 /home/jovyan/work/Sections/Data/Weather/NY.tgz
```

```
In [24]: #extracting the parquet file
         !tar zxvf {weather_dir}/{zip_file} -C {weather_dir}

NY.parquet/
NY.parquet/_SUCCESS
NY.parquet/part-00000-8342bcf4-7fc2-4183-8e11-aefdb4915fbb-c000.snappy.parquet
NY.parquet/part-00001-8342bcf4-7fc2-4183-8e11-aefdb4915fbb-c000.snappy.parquet
NY.parquet/part-00002-8342bcf4-7fc2-4183-8e11-aefdb4915fbb-c000.snappy.parquet
NY.parquet/part-00003-8342bcf4-7fc2-4183-8e11-aefdb4915fbb-c000.snappy.parquet
NY.parquet/part-00004-8342bcf4-7fc2-4183-8e11-aefdb4915fbb-c000.snappy.parquet
NY.parquet/part-00005-8342bcf4-7fc2-4183-8e11-aefdb4915fbb-c000.snappy.parquet
NY.parquet/part-00006-8342bcf4-7fc2-4183-8e11-aefdb4915fbb-c000.snappy.parquet
NY.parquet/part-00007-8342bcf4-7fc2-4183-8e11-aefdb4915fbb-c000.snappy.parquet
NY.parquet/part-00008-8342bcf4-7fc2-4183-8e11-aefdb4915fbb-c000.snappy.parquet
NY.parquet/part-00009-8342bcf4-7fc2-4183-8e11-aefdb4915fbb-c000.snappy.parquet
NY.parquet/part-00010-8342bcf4-7fc2-4183-8e11-aefdb4915fbb-c000.snappy.parquet
NY.parquet/part-00011-8342bcf4-7fc2-4183-8e11-aefdb4915fbb-c000.snappy.parquet
NY.parquet/part-00012-8342bcf4-7fc2-4183-8e11-aefdb4915fbb-c000.snappy.parquet
NY.parquet/part-00013-8342bcf4-7fc2-4183-8e11-aefdb4915fbb-c000.snappy.parquet
```

```
In [28]: weather_parquet = join(weather_dir, zip_file[:-3]+'parquet')
         print(weather_parquet)
         df = sqlContext.read.load(weather_parquet)
         df.show(1)

/home/jovyan/work/Sections/Data/Weather/NY.parquet
+-----------+-----------+----+--------------------+-----+
|    Station|Measurement|Year|              Values|State|
+-----------+-----------+----+--------------------+-----+
|USC00303452|       PRCP|1903|[00 7E 00 7E 00 7...|   NY|
+-----------+-----------+----+--------------------+-----+
only showing top 1 row
```

```
In [26]: #selecting a subset of the rows so it fits in slide.
         df.select('station','year','measurement').show(5)
```

```
+-----------+----+-----------+
|    station|year|measurement|
+-----------+----+-----------+
|USC00303452|1903|       PRCP|
|USC00303452|1904|       PRCP|
|USC00303452|1905|       PRCP|
|USC00303452|1906|       PRCP|
|USC00303452|1907|       PRCP|
+-----------+----+-----------+
only showing top 5 rows
```

1.3 Summary

- Dataframes are an efficient way to store data tables.
- All of the values in a column have the same type.
- A good way to store a dataframe in disk is to use a Parquet file.
- Next: Operations on dataframes.