Laboratory Manual- 09

for

Fundamentals of Big Data Lab

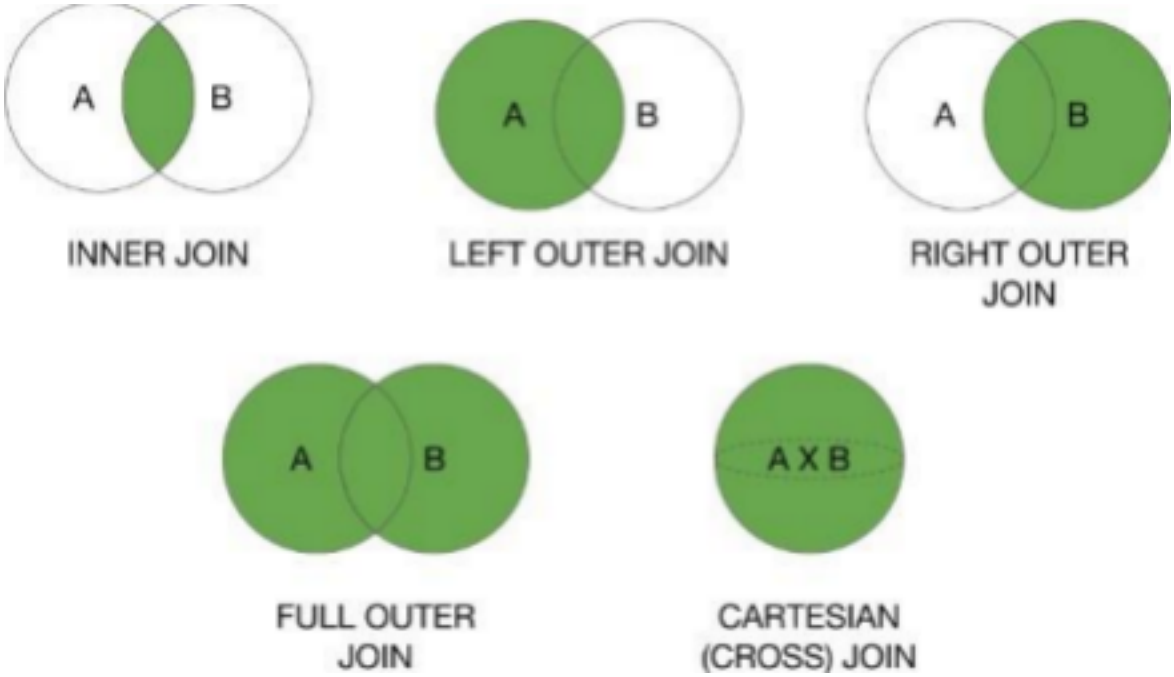
Lab Instructors: Miss Rida Mahmood
Section: BDS-4B,C
Date: 27-April-2024
Semester: Spring 2024

Department of Computer Science

FAST-NU, Lahore, Pakistan

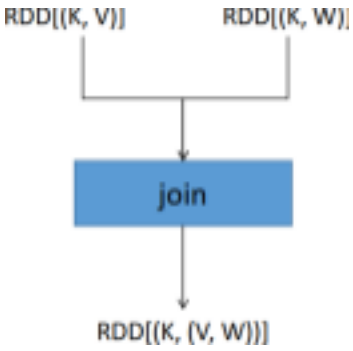
Join Operations in Pyspark

The given below illustration explains the various types of joins.



1. PySpark RDD join/ inner join

   Return an RDD containing all pairs of elements with matching keys in *self* and *other*. Each pair ofelements will be returned as a (k, (v1, v2)) tuple, where (k, v1) is in *self* and (k, v2) is in *other*.



Example:

```
rdd1 = sc.parallelize([("a", 1), ("b", 4)])
rdd2 = sc.parallelize([("a", 2), ("a", 3)])

sorted(rdd1.join(rdd2).collect())

Output = [('a', (1, 2)), ('a', (1, 3))]
```

2. PySpark RDD leftOuterJoin

Perform a left outer join of self and other. For each element (k, v) in self, the resulting RDD will either contain all pairs (k, (v, w)) for w in other, or the pair (k, (v, None)) if no elements in other havekey k.

```
rdd1 = sc.parallelize([("a", 1), ("b", 4)])

rdd2 = sc.parallelize([("a", 2)])

sorted(rdd1.leftOuterJoin(rdd2).collect())

output = [('a', (1, 2)), ('b', (4, None))]
```

3. PySpark RDD rightOuterJoin

Perform a right outer join of self and other. For each element (k, w) in other, the resulting RDD willeither contain all pairs (k, (v, w)) for v in this, or the pair (k, (None, w)) if no elements in self have key k.

```
rdd1 = sc.parallelize([("a", 1), ("b", 4)])

rdd2 = sc.parallelize([("a", 2)])
```



```
sorted(rdd2.rightOuterJoin(rdd1).collect())
Output = [('a', (2, 1)), ('b', (None, 4))]
```

4. PySpark RDD fullOuterJoin

Perform a right outer join of self and other. For each element (k, v) in self, the resulting RDD will either contain all pairs (k, (v, w)) for w in other, or the pair (k, (v, None)) if no elements in other havekey k.

Similarly, for each element (k, w) in other, the resulting RDD will either contain all pairs (k, (v, w))for v in self, or the pair (k, (None, w)) if no elements in self have key k.

```
rdd1 = sc.parallelize([("a", 1), ("b", 4)])

rdd2 = sc.parallelize([("a", 2), ("c", 8)])

sorted(rdd1.fullOuterJoin(rdd2).collect())

output=[('a', (1, 2)), ('b', (4, None)), ('c', (None, 8))]
```

5. RDD cogroup()

cogroup() is another transformation in PySpark that allows you to group the values of multiple RDDsby a common key. It returns an RDD of key-value pairs, where the key is the common key, and the value is a tuple containing all the values from the input RDDs that share the same key.

For each key k in self or other, return a resulting RDD that contains a tuple with the list of values forthat key in self as well as other.


```
rdd1 = sc.parallelize([("a", 1), ("b", 4)])
rdd2 = sc.parallelize([("a", 2)])
```

```
[(x, tuple(map(list, y))) for x, y in sorted(list(rdd1.cogroup(rdd2).collect()))] Output= [('a', ([1], [2])), ('b', ([4], []))]
```

```
# Create RDDs with key-value pairs
rdd1 = sc.parallelize([(1, "A"), (2, "B"), (3, "C")])
rdd2 = sc.parallelize([(2, "X"), (3, "Y"), (4, "Z")])

# Perform cogroup on the common key
grouped_rdd = rdd1.cogroup(rdd2)

# Print the grouped RDD
print(grouped_rdd.collect())
```


Output of the above code: [(1, (['A'], [])), (2, (['B'], ['X'])), (3, (['C'], ['Y'])), (4, ([], ['Z']))]

Broadcast: Broadcasting is a technique used to efficiently share a read-only variable across all the worker nodes in a distributed computing environment. In PySpark, when you perform operations like join or filter on large datasets, Spark may shuffle and transfer data across different nodes, which can be expensive in terms of time and network bandwidth. Broadcasting allows you to send a small amount of data, such as lookup tables or configuration parameters, to all the worker nodes so that they can accessit locally, avoiding unnecessary data shuffling. This can significantly improve the performance of Spark applications.

How does PySpark Broadcast work?

Broadcast variables are used in the same way for RDD, DataFrame.

When you run a PySpark RDD, DataFrame applications that have the Broadcast variables defined and used, PySpark does the following.

- PySpark breaks the job into stages that have distributed shuffling and actions are executed with in the stage.
- Later Stages are also broken into tasks
- Spark broadcasts the common data (reusable) needed by tasks within each stage.
- The broadcasted data is cache in serialized format and deserialized before executing each task.

You should be creating and using broadcast variables for data that shared across multiple stages and tasks.

Note that broadcast variables are not sent to executors with `sc.broadcast(variable)` call instead, they will be sent to executors when they are first used.

Use case

Let me explain with an example when to use broadcast variables, assume you are getting a two-letter country state code in a file and you wanted to transform it to full state name, (for example CA to California, NY to New York e.t.c) by doing a lookup to reference mapping. In some instances, this data could be large and you may have many such lookups (like zip code e.t.c).

Instead of distributing this information along with each task over the network (overhead and time consuming), we can use the broadcast variable to cache this lookup info on each machine and tasks use this cached info while executing the transformations.

```
import pyspark
from pyspark.sql import SparkSession
```

```
states = {"NY":"New York", "CA":"California", "FL":"Florida"}
broadcastStates = spark.sparkContext.broadcast(states)

data = [("James","Smith","USA","CA"),
    ("Michael","Rose","USA","NY"),
    ("Robert","Williams","USA","CA"),
    ("Maria","Jones","USA","FL")
  ]

rdd = spark.sparkContext.parallelize(data)

def state_convert(code):
    return broadcastStates.value[code]

result = rdd.map(lambda x: (x[0],x[1],x[2],state_convert(x[3]))).collec
print(result)
```


Accumulator: An accumulator is a distributed variable that is used to accumulate values across different worker nodes in a distributed computing environment. It is a write-only variable that can be updated in parallel by multiple tasks running on different worker nodes. Accumulators are used for aggregating information or collecting statistics during the execution of a Spark job.

```
from pyspark import SparkContext
sc = SparkContext("local", "Accumulator app")
num = sc.accumulator(10)
def f(x):
  global num
  num+=x
rdd = sc.parallelize([20,30,40,50])
rdd.fosreach(f)
final = num.value
print("Accumulated value is -> %i" % (final))
```

Introduction to DataFrames in PySpark

There are limitations of RDDs. So, dataframes overcome that limitations of Rdds. In this manual you go through how to use data frames in pyspark.

For detailed information, go through the documentation of Pyspark.

How to create DataFrames?

There are multiple ways to create DataFrames in Apache Spark:

- DataFrames can be created using an existing RDD
- You can create a DataFrame by loading a CSV file directly
- You can programmatically specify a schema to create a DataFrame

```
# spark is an existing SparkSession
df = spark.read.csv("examples/src/main/resources/people.csv")
# Displays the content of the DataFrame to stdout
df.show()
```



```
# +----+-------+
# | age| name|
# +----+-------+
# |null|Michael|
# | 30| Andy|
# | 19| Justin|
# +----+-------+
```

Some basic operations of dataframes

In Python, it's possible to access a DataFrame's columns either by attribute (df.age) or by indexing (df['age']). While the former is convenient for interactive data exploration, users are highly encouraged to use the latter form, which is future proof and won't break with column names that are also attributes on the DataFrame class.

```
# spark, df are from the previous example
# Print the schema in a tree format
df.printSchema()
# root
# |-- age: long (nullable = true)
# |-- name: string (nullable = true)

# Select only the "name" column
df.select("name").show()
# +-------+
# | name|
# +-------+
# |Michael|
# | Andy|
# | Justin|
# +-------+

# Select everybody, but increment the age by 1
df.select(df['name'], df['age'] + 1).show()
# +-------+---------+
```

```
# | name|(age + 1)|
# +-------+---------+
# |Michael| null|
# | Andy| 31|
# | Justin| 20|
# +-------+---------+

# Select people older than 21
df.filter(df['age'] > 21).show()
# +---+----+
# |age|name|
# +---+----+
# | 30|Andy|
# +---+----+

# Count people by age
df.groupBy("age").count().show()
# +----+-----+
# | age|count|
# +----+-----+
# | 19| 1|
# |null| 1|
# | 30| 1|
# +----+-----+
```

Running SQL Queries Programmatically

```
# Register the DataFrame as a SQL temporary view
```



```
df.createOrReplaceTempView("people")

sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
# +----+-------+
# | age| name|
# +----+-------+
# |null|Michael|
# | 30| Andy|
# | 19| Justin|
# +----+-------+
```

How to read CSV file and its other operations follow the link:
https://spark.apache.org/docs/latest/sql-data-sources-csv.html

Machine Learning Library (MLlib) Guide

MLlib is Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. At a high level, it provides tools such as:

- ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
- Featurization: feature extraction, transformation, dimensionality reduction, and selection
- Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
- Persistence: saving and load algorithms, models, and Pipelines

- Utilities: linear algebra, statistics, data handling, etc.

Link for all the machine learning approaches in Pyspark

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

The given below is the example of Machine Learning algorithm in Pyspark

- Finding correlation

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.stat import Correlation

data = [(Vectors.sparse(4, [(0, 1.0), (3, -2.0)]),),
 (Vectors.dense([4.0, 5.0, 0.0, 3.0]),),
 (Vectors.dense([6.0, 7.0, 0.0, 8.0]),),
 (Vectors.sparse(4, [(0, 9.0), (3, 1.0)]),)]
df = spark.createDataFrame(data, ["features"])

r1 = Correlation.corr(df, "features").head()


print("Pearson correlation matrix:\n" + str(r1[0]))

r2 = Correlation.corr(df, "features", "spearman").head()


print("Spearman correlation matrix:\n" + str(r2[0]))
```

It is list of all python apis: https://spark.apache.org/docs/latest/api/python/reference/index.html

This Link gives you insights how machine learning algorithm has been implemented using machine learning libraries from pyspark.

Lab Tasks

You are provided dataset “Movies.csv” that contains information about different aspects of movies explore the dataset and do lab task 1 to 5 using pyspark data frames.

1. Inner Join: Perform an inner join on emp_id between rdd1 and rdd2 to get an RDD containing the employee ID, employee name, employee department, and employee salary for employees who exist inboth RDDs.
2. Left Outer Join: Perform a left outer join on emp_id between rdd1 and rdd2 to get an RDD containingthe employee ID, employee name, employee department, and employee salary for all employees in rdd1 and matching employees in rdd2. For employees in rdd1 that do not have a matching employee ID in rdd2, the employee salary should be set to None.
3. Calculate the total salary of employees in RDD1 by joining it with RDD2 on "emp_id" andmultiplying "emp_salary" with "emp_experience".
4. Write a PySpark code to perform a broadcast join between two pair RDDs. The first RDD, contains key value pair: "id" and "name". The second RDD contains pair of "id" and "age". Perform a broadcastjoin on the "id" column and return a new RDD, result, containing "id", "name", and "age".

5. Write a PySpark code to calculate the sum of all values in an RDD using an accumulator. Create an RDD of integers from 1 to 10. Define an accumulator with an initial value of 0. Use the foreach() action on the RDD to update the accumulator with each element. Finally, print the final value of the accumulator after processing all elements in the RDD.
6. Find the title, year, and director of action films that won an award.
7. For each award-winning actor, find the movies he acted it. Print the names of the movies and the director of the movie.
8. Find the top 10 most popular movies that did not win an award.
9. Find the 10 least popular movies that were released before 1980.
10. Sort the movie's release before 1990 by the title.
11. Explore and preprocess the "wine" data set which was given you in previous lab, use spark dataframes and analyse it. Find the outliers or noise in the data and find the correlation between different features.
12. Use PySpark built-in K-means and bisecting K-means clustering algorithms for clustering.