Emerging Paradigms for Big Data

Apache Spark

Resilient Distributed Datasets

MLlib

Fátima Leal

DCT DEPARTAMENTO CIÊNCIA E TECNOLOGIA

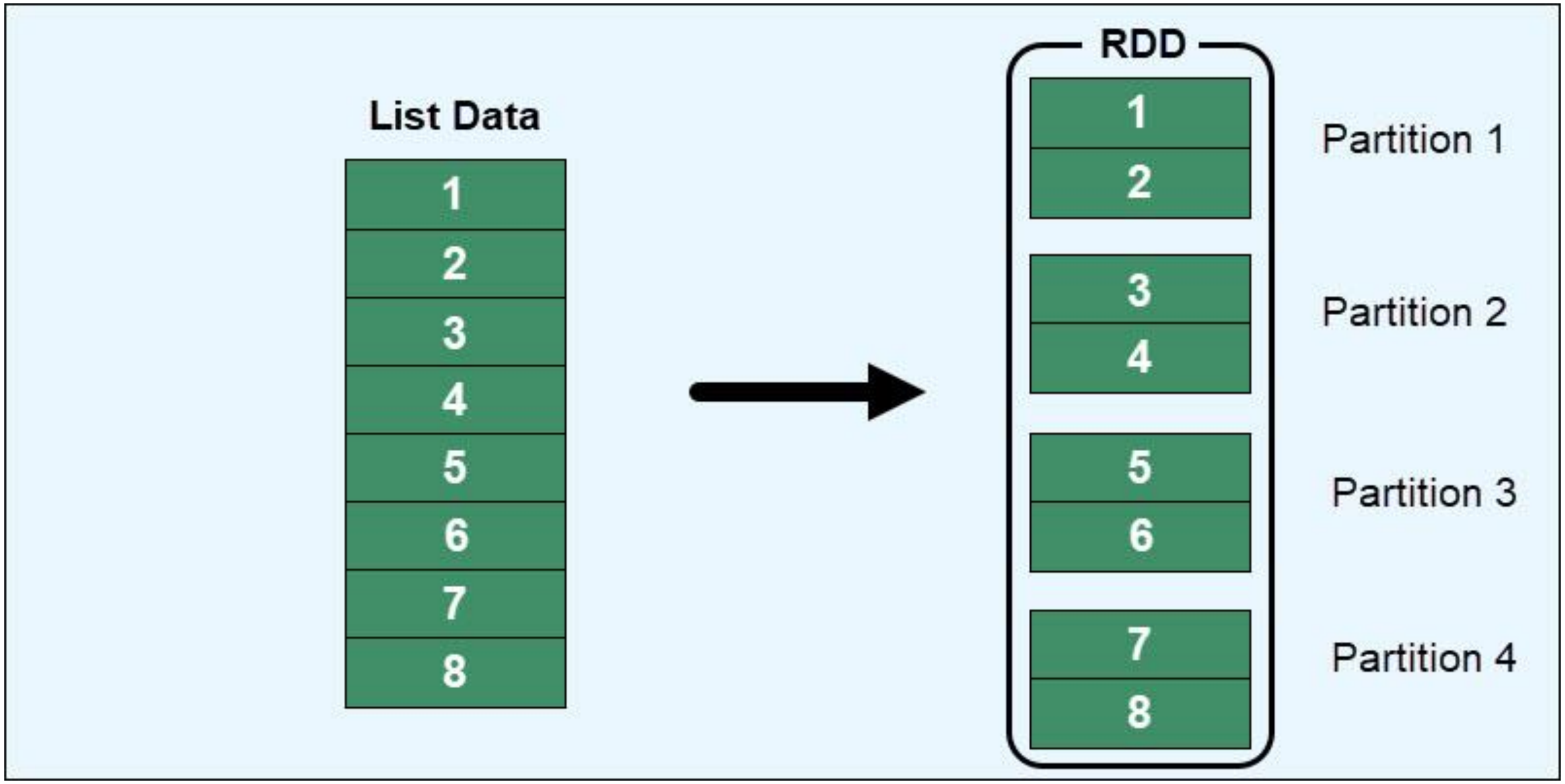
UPT UNIVERSIDADE PORTUCALENSE

Content

- Apache Spark
- Resilient Distribution datasets
- RDD Operations
- Kafka + Cassandra + Spark
- Example/Exercises

Overview

- The ideas behind RDDs are unique in the distributed data processing framework landscape.
- It were introduced in a timely manner to solve the pressing needs of dealing with the complexity and efficiency of iterative and interactive data processing use cases.
- RDDs represent both the idea of how a large dataset is represented in Spark and the abstraction for working with it.
- RDDs are immutable, fault-tolerant, parallel data structures that let users explicitly persist intermediate results in memory, control their partitioning to optimize data placement, and manipulate them using a rich set of operators.


List Data
1
2
3
4
5
6
7
8
RDD
1
2
3
4
5
6
7
8
Partition 1
Partition 2
Partition 3
Partition 4

Introduction to RDDs

- **Immutable** RDDs are designed to be immutable, which means you can’t specifically modify a particular row in the dataset represented by that RDD. You can call one of the available RDD operations to manipulate the rows in the RDD into the way you want, but that operation will return a new RDD.

- **Fault Tolerant** is the ability to process multiple datasets in parallel which usually requires a cluster of machines to host and execute the computational logic. The good news is that Spark automatically takes care of handling the failure on behalf of its users by rebuilding the failed portion using the lineage information

Introduction to RDDs

- **Parallel Data Structures -** the collection of rows is essentially the data structure that holds a set of rows and provides the ability to iterate through each row. Each chunk contains a collection of rows, and all the chunks are being processed in parallel. This is where the phrase parallel data structures comes from.

- **In-Memory Computing -** the idea of speeding up the computation of large datasets that reside on disks in a parallelized manner using a cluster of machines In the world of big data processing, once you are able to extract insights from large datasets in a reliable manner using a set of rudimentary techniques, then you want to use more sophisticated techniques as well to reduce the amount of time it takes to do that.

Introduction to RDDs

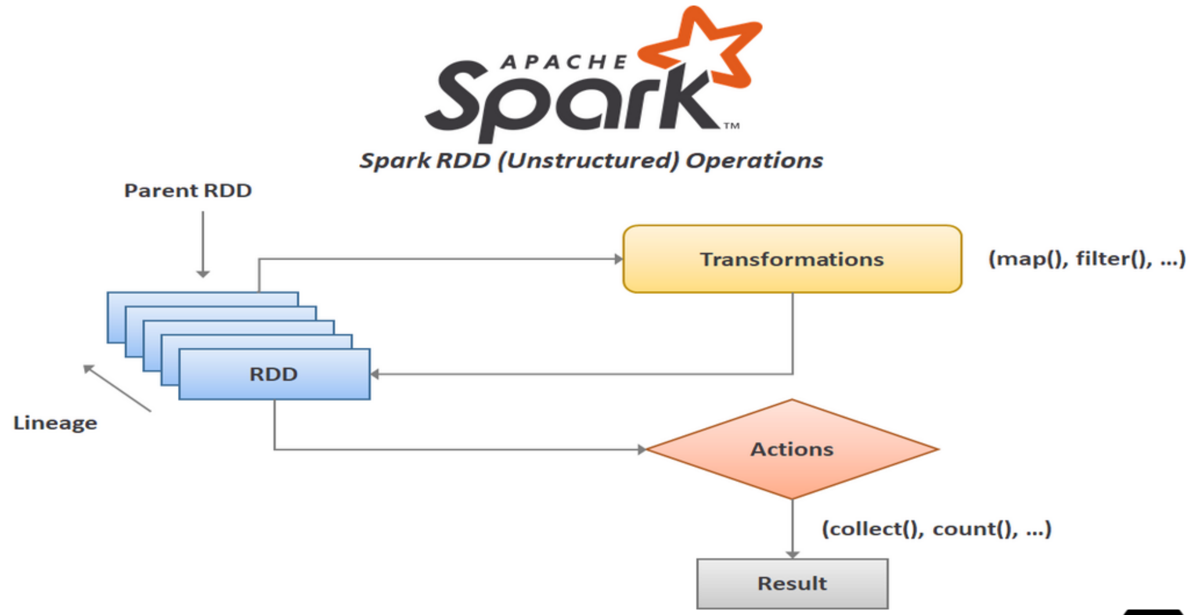
- **Data Partitioning and Placement -** the information about how the rows in a dataset are partitioned into chunks and about their physical location is considered to be the dataset metadata. Knowing where the chunks are located on a cluster, Spark can use those machines to host and execute the computational logic on those chunks, and therefore the time to read the rows from those chunks would be much less than reading them from a different node on the cluster.

- **Rich Set of Operations -** RDDs provide a rich set of commonly needed data processing operations. They include the ability to perform data transformation, filtering, grouping, joining, aggregation, sorting, and counting

RDD operations

- **Transformation** − These are the operations, which are applied on a RDD to create a new RDD. Filter, groupBy and map are the examples of transformations.

- **Action** − These are the operations that are applied on RDD, which instructs Spark to perform computation and send the result back to the driver.



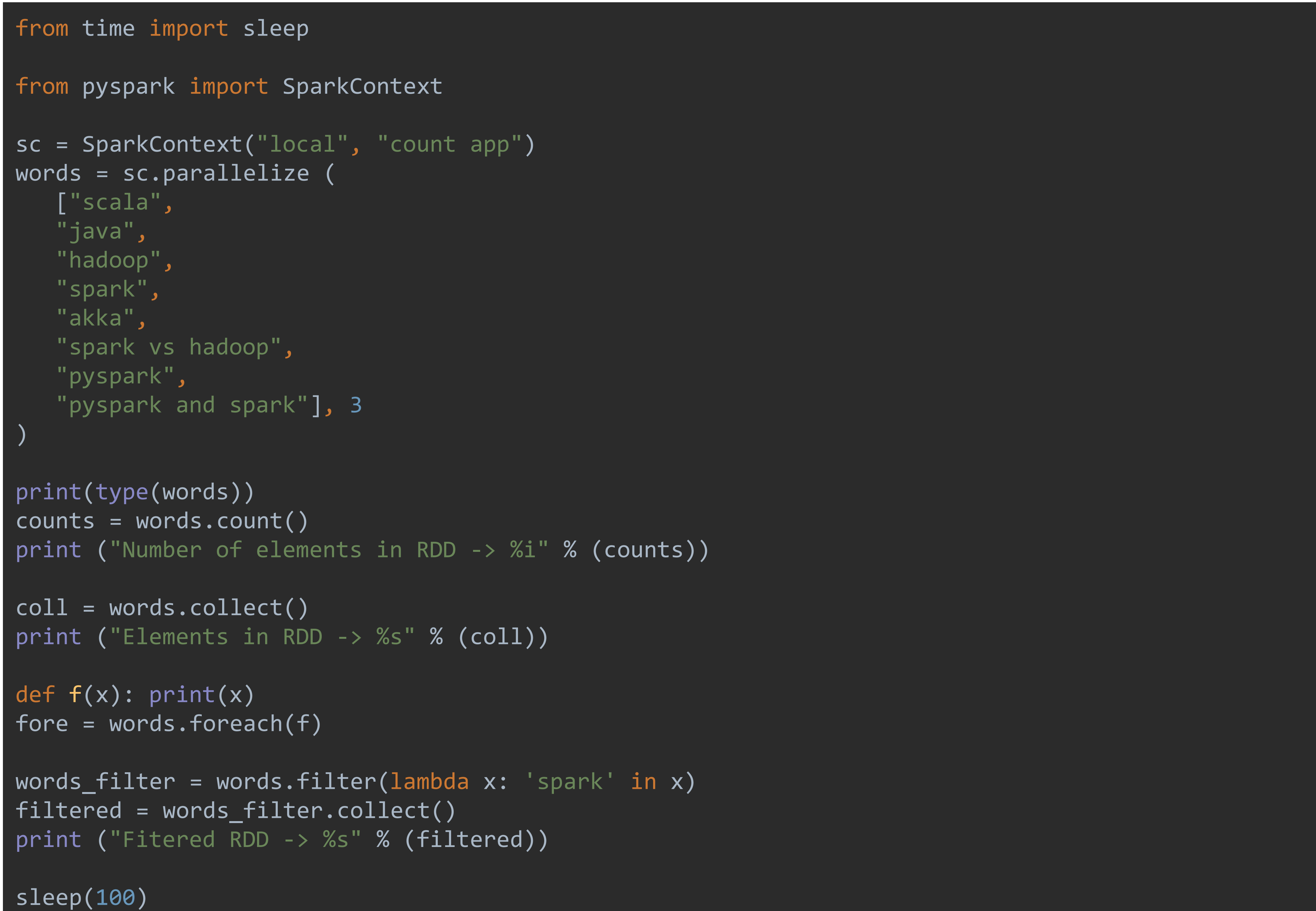
DEPARTAMENTO **CIÊNCIA E TECNOLOGIA**

RDD operations

- Each row in a dataset is represented as a Java object, and the structure of this Java object is opaque to Spark. The user of RDD has complete control over how to manipulate this Java object.
- Transformation operations are lazily evaluated, meaning Spark will delay the evaluations of the invoked operations until an action is taken.
- The lazy evaluation design makes sense in the world of big data. It is not desirable to immediately trigger an evaluation of every single filtering operation when a dataset is large in size.
- In short, RDDs are immutable, RDD transformations are lazily evaluated, and RDD actions are eagerly evaluated and trigger the computation of your data processing logic.

Creating RDD

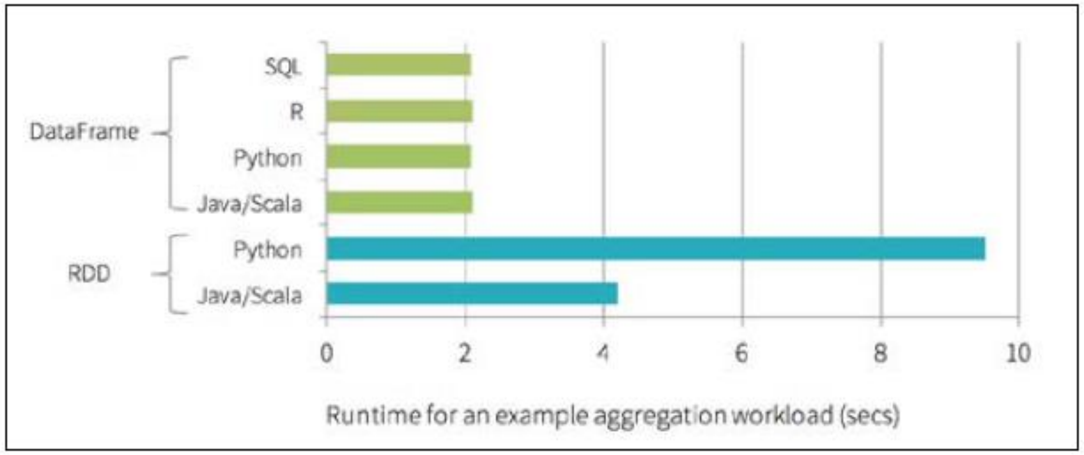
- The first way to create an RDD is to parallelize an object collection, meaning converting it to a distributed dataset that can be operated in parallel.
- Parallelized collections are created by calling SparkContext’s parallelize method on an existing iterable or collection in your driver program
- A cluster size can be as small as a few machines or as large as thousands of machines.
- The largest publicly announced Spark cluster in the world has more than 8000 machines



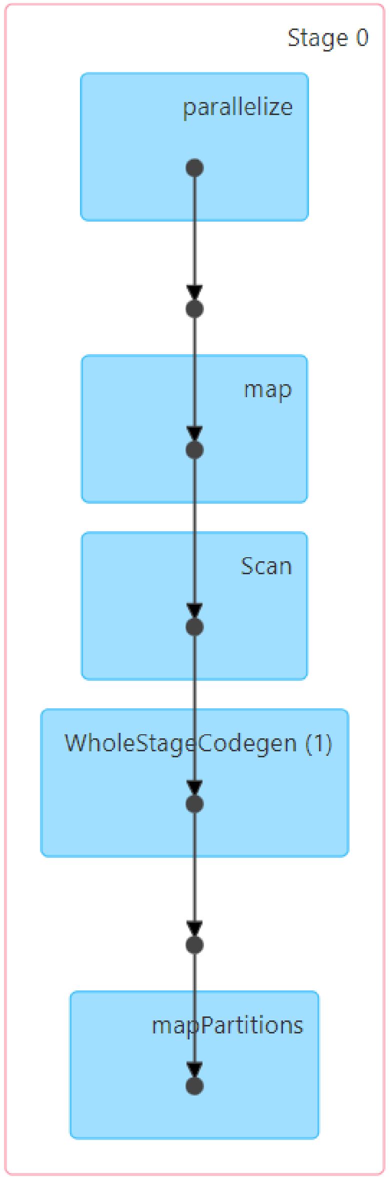
```python
from time import sleep

from pyspark import SparkContext

sc = SparkContext("local", "count app")
words = sc.parallelize (
   ["scala",
   "java",
   "hadoop",
   "spark",
   "akka",
   "spark vs hadoop",
   "pyspark",
   "pyspark and spark"], 3
)

print(type(words))
counts = words.count()
print ("Number of elements in RDD -> %i" % (counts))

coll = words.collect()
print ("Elements in RDD -> %s" % (coll))

def f(x): print(x)
fore = words.foreach(f)

words_filter = words.filter(lambda x: 'spark' in x)
filtered = words_filter.collect()
print ("Fitered RDD -> %s" % (filtered))

sleep(100)
```

Data Frames

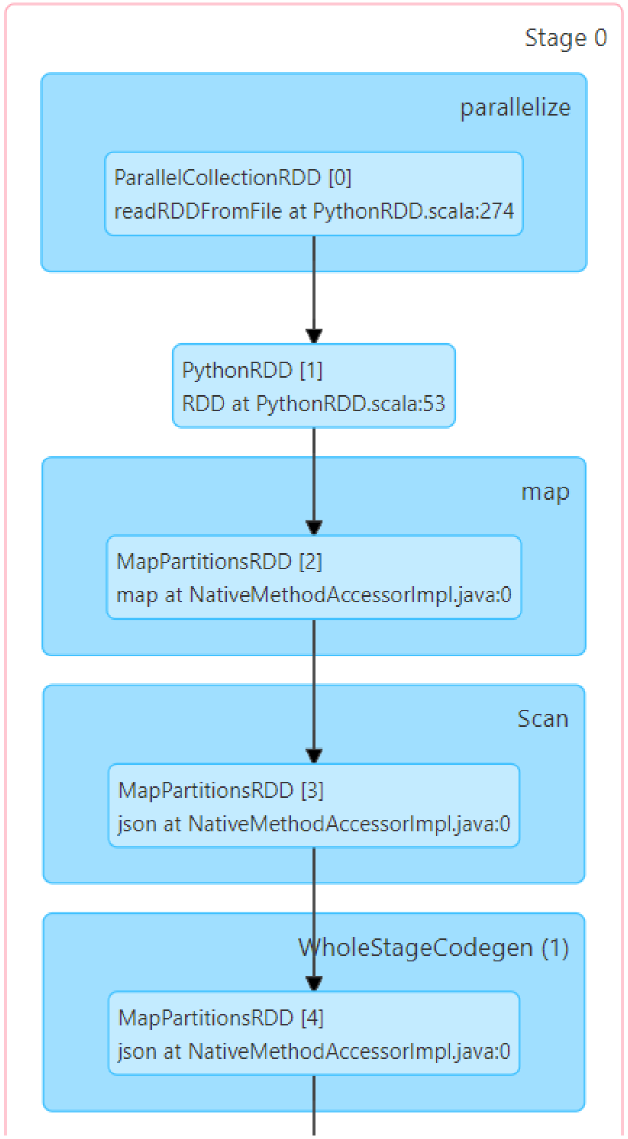
- A DataFrame is an immutable distributed collection of data that is organized into named columns analogous to a table in a relational database.
- Similar to Python pandas DataFrame
- The introduction of DataFrames, Python query speeds up the performance



```
from time import sleep

from pyspark.shell import spark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
sc=spark.sparkContext

stringJSONRDD = sc.parallelize(("""
{ "id": "123",
"name": "Katie",
"age": 19,
"eyeColor": "brown"
}""",
"""{
"id": "234",
"name": "Michael",
"age": 22,
"eyeColor": "green"
}""",
"""{
"id": "345",
"name": "Simone",
"age": 23,
"eyeColor": "blue"
}""")
)

swimmersJSON = spark.read.json(stringJSONRDD)

sleep(10000)
```

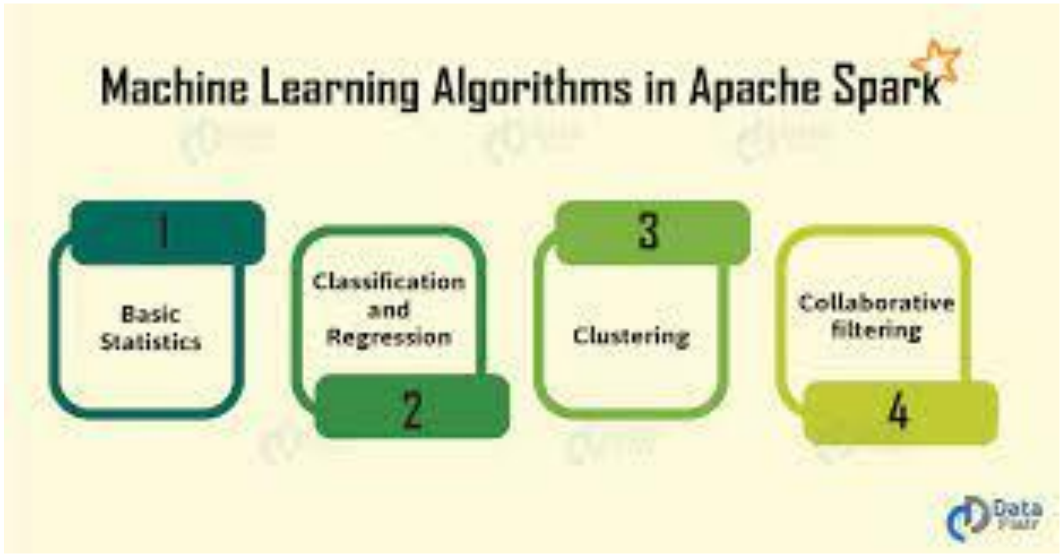


DEPARTAMENTO **CIÊNCIA E TECNOLOGIA**


Stage 0
parallelize
ParallelCollectionRDD [0]
readRDDFromFile at PythonRDD.scala:274
PythonRDD [1]
RDD at PythonRDD.scala:53
map
MapPartitionsRDD [2]
map at NativeMethodAccessorImpl.java:0
Scan
MapPartitionsRDD [3]
json at NativeMethodAccessorImpl.java:0
WholeStageCodegen (1)
MapPartitionsRDD [4]
json at NativeMethodAccessorImpl.java:0

Data Frames

- Queries
- Data Modelling
  - Drop
  - Missing observations
  - Descriptive statistics
  - Correlations
- MlLib


Machine Learning Algorithms in Apache Spark
1
Basic Statistics
Classification and Regression
2
3
Clustering
Collaborative filtering
4
Data Flair

Spark ML

- ML is a vast and fascinating field of study, which combines parts of other fields of studies such as mathematics, statistics, and computer science.

- Unlike traditional, hard-coded software, ML gives you only probabilistic outputs based on the imperfect data you provide. The more data you can provide to ML algorithms, the more accurate the output will be.

ML Terminologies

- To ideally make it easier to understand these terms, the explanations are provided in the context of the canonical ML example called the spam email classification:
- Observation – An observation is an instance of the entity that is used for learning.
- Label – A value used to label an observation.
- Features – These are important attributes about observations that most likely have the strongest influence in the output of the prediction.

ML Terminologies

- Training data – This is a portion of the observations used to train a chosen ML algorithm to produce a model.
- Validation data – This is a portion of the observations used to evaluate the performance of the ML model during the model tuning process.
- Test data – This is a portion of the observations used to evaluate the performance of the ML model after the tuning process is finalized.
- ML algorithm – The main goal of an ML algorithm is to learn a mapping from inputs to outputs.
- Model – After an ML algorithm learns from the given input data, it produces a model, which is used to perform predictions or make decisions on the new data.

ML Types

- ML is about teaching machines to learn patterns from previous data for the purpose of making decisions or predictions.

- These tasks are widely applicable to many different types of problem, and each problem type requires a different way of learning.

- Supervised Learning – Among the different learning types, this one is widely used and more popular because it can help solve a large class of problems in the area of classification and regression.
  - Classification is about classifying the observations into one of the discrete or categorical classes of labels. When the result of the classification has only two discrete values, that is called binary classification, and when it has more than two discrete values, that is called multiclass classification.
  - Regression is about predicting real values from observations. Unlike classification, the predicted value is not discrete, but rather it is continuous.

ML Types

- Unsupervised Learning – The name of this learning method implies there is no supervision; in other words, the data used to train the ML algorithm wouldn’t contain the labels, and it is up to the learning algorithm to come up with its own findings

- This learning type is designed to solve a different class of problem, that is, to discover the hidden structure or patterns inside the data, and it is up to us, the humans, to interpret the meaning behind those insights.

- A certain type of hidden structure called clustering, which is an exploratory analysis technique in data analytics, is a good method for structuring information to derive meaningful relationships or find similarities of the observations within the clusters.

ML Types

- Reinforcement Learning – it learns from interacting with an environment through a series of actions, and the feedback loop provides the information it uses to make adjustments with the goal of maximizing some reward.

- This type of learning hasn't gotten as much attention as the first two because it has not yet had significant practical success.

ML Processes

- The first step in this process is to clearly understand the business objective or challenge that you think ML can help you with. After the problem is clearly understood, the next part is to establish a set of success metrics that all stakeholders can agree on.

- The next step is to identify and collect the necessary types of and an appropriate amount of data to support the problem at hand. The quality and quantity of the collected data will have a direct impact on the performance of the trained ML model.

- Feature engineering is one of the most important and time-consuming steps in this process. This step is mainly about data cleaning and using domain knowledge to identify key attributes or features about observations that will be useful to the ML algorithms to learn the direct relationship between the training data and provided labels.

ML Processes

- The data cleaning task is usually done using the exploratory data analysis framework to gain a better understand of the data in terms of data distribution, correlations, outliers, and so on.

- The next step after feature engineering is selecting an appropriate ML model or algorithm and training it. The goal of all the previous steps is to produce a model that is generalized, meaning that it performs well on data it has never seen before.

- Another important step in the ML development process is the model evaluation task. It is both necessary and challenging. The goal of this step is to not only answer the question of how well a model performs but also to know when to stop tuning the model because its performance has reached the established success metrics in the first step.

Spark ML Library

- In the era of big data, there are two reasons to pick Spark MLlib over the other options.
- The first one is the ease of use. MLlib library provides a means to build, manage, and persist complex ML pipelines.
- The second reason is performing ML at scale. The combination of the Spark unified data analytic engine and the MLlib library can support training machine learning models with billions of observations and thousands of features.

ML Pipelines

- ML process is essentially a pipeline that consists of a series of steps that run in a sequential manner and that usually need to be repeated several times to arrive at an optimal model.
- Spark MLlib provides a set of abstractions to help simplify the steps of data cleaning, featuring engineering, model training, model tuning, and evaluation as well as organizing them into a pipeline to make it easy to understand, maintain, and repeat
- There are four main abstractions to form an end-to-end ML pipeline: transformers, estimators, evaluators, and pipelines.
- The one thing in common across these abstractions is that their inputs and output are mostly DataFrames, which means it is necessary to convert the input data into a DataFrame to work with these abstractions.

Transformers

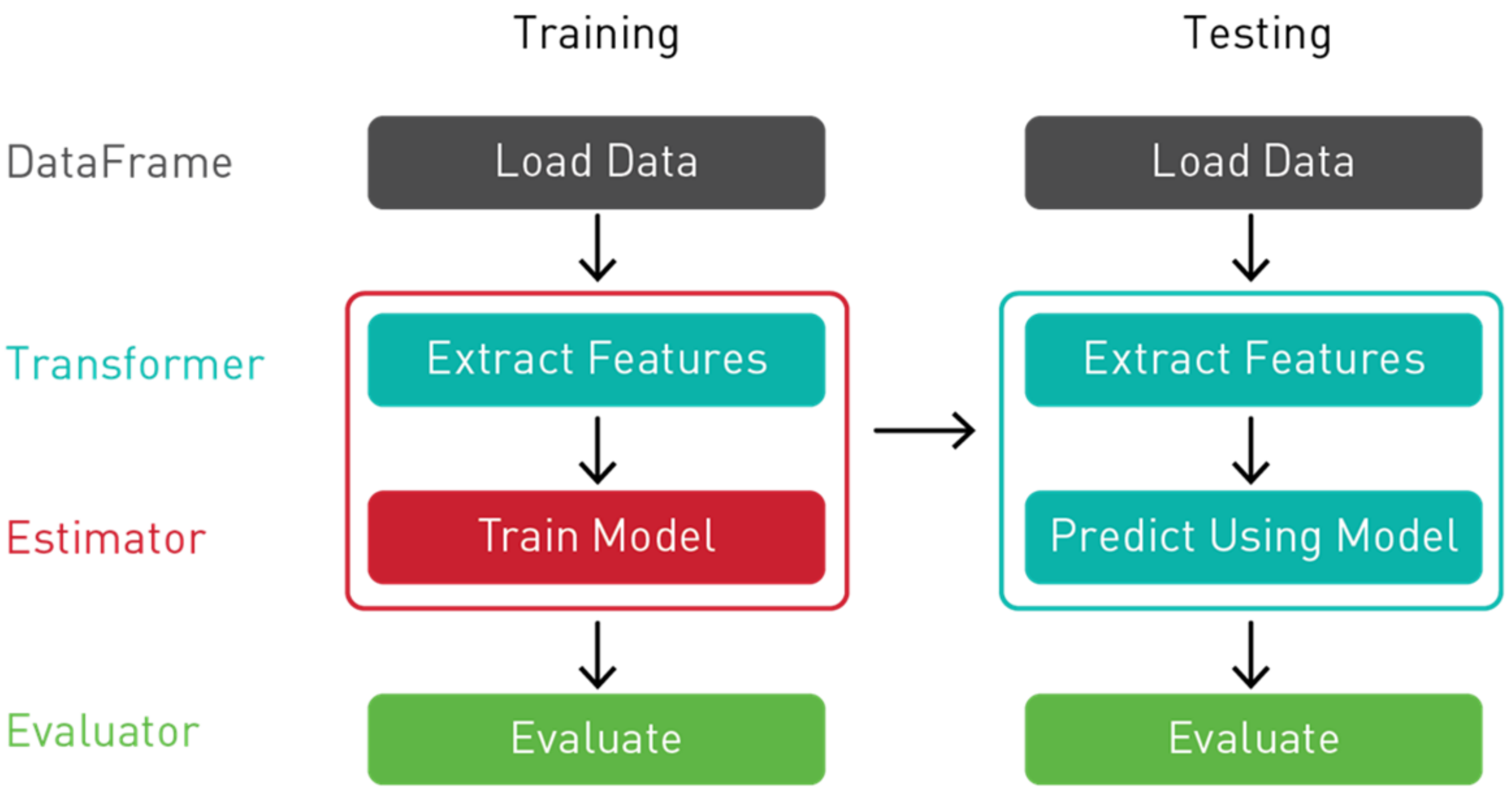
- Transformers are designed to transform data in the DataFrame by manipulating one or more columns during the feature engineering step and the model evaluation step.
- This process usually involves adding or removing columns (features), converting the column values from text to numerical value, or normalizing the values of a particular column.
- From a technical perspective, a transformer has a function called transform that performs transformations on the input column, and the result is stored in the output column

Estimators

- Estimators are an abstraction for either an ML learning algorithm that trains on data or any other algorithm that operates on data. It is rather confusing that an estimator can be one of two kinds of algorithm.

- From a technical perspective, an estimator has a function called fit that applies an algorithm on the input column, and the result is encapsulated in an object type called Model, which is a Transformer type.

- There are many more estimators available in MLlib to perform numerous data transformations and mappings, and they all follow a standard abstraction that fits the input data and produces an instance of a model


Training
Testing
DataFrame
Load Data
Load Data
Transformer
Extract Features
Extract Features
Estimator
Train Model
Predict Using Model
Evaluator
Evaluate
Evaluate

Pipeline

- In machine learning, it is common to run a sequence of steps to clean and transform data, then train one or more ML algorithms to learn from the data, and finally tune the model to achieve the best possible performance

- The pipeline abstraction in MLlib is designed to make this workflow easier to develop and maintain

- MLlib has a class called Pipeline, which is designed to manage a series of stages, and each one is represented by PipelineStage. A PipelineStage can be either a transformer or an estimator.

Pipeline

- The first step in setting up a pipeline is to create a series of stages and then create an instance class Pipeline and configure it with an array of stages.

- MLlib provides a feature called ML persistence that makes it easy to persist a pipeline or a model to disk and load it later for use.

- Real-life production pipelines consist of many stages. When the number of stages gets large, it is difficult to understand the flow as well as challenging to maintain. MLlib pipeline abstraction can really help with these areas.

Model Tuning

- The goal of the model tuning step is to train a model with the right set of parameters to achieve the best performance to meet the object defined in the first step of the ML development process.
- This step is usually tedious, repetitive, and time-consuming because it may involve trying different ML algorithms or a few sets of parameters.
- Before going into the details of the two tools that MLlib provides, let's first have a clear understanding of the following terminologies, where one of them is an input to the model tuning process (hyperparameters and parameters)
- The two commonly used classes in MLlib to help with model tuning are CrossValidator and TrainValidationSplit, and both them are of type Estimator. These classes are also known as validators.

Spark MLlib Pipeline

```
# create a sample dataframe with 4 features and 1 label column
sample_data_train = spark.createDataFrame([
    (2.0, 'A', 'S10', 40, 1.0),
    (1.0, 'X', 'E10', 25, 1.0)
], ['feature_1', 'feature_2', 'feature_3', 'feature_4', 'label'])

# view the data
sample_data_train.show()

# define stage 1: transform the column feature_2 to numeric
stage_1 = StringIndexer(inputCol= 'feature_2', outputCol= 'feature_2_index')
# define stage 2: transform the column feature_3 to numeric
stage_2 = StringIndexer(inputCol= 'feature_3', outputCol= 'feature_3_index')

# define stage 3: create a vector of all the features required to train the logistic
regression model
stage_3 = VectorAssembler(inputCols=['feature_1', 'feature_2_index', 'feature_3_index',
'feature_4'],
                          outputCol='features')

# define stage 5: logistic regression model
stage_4 = LogisticRegression(featuresCol='features',labelCol='label')

# setup the pipeline
regression_pipeline = Pipeline(stages= [stage_1, stage_2, stage_3, stage_4])

# fit the pipeline for the trainind data
model = regression_pipeline.fit(sample_data_train)
# transform the data
sample_data_train = model.transform(sample_data_train)
```

Spark MLlib Tuning

```
paramGrid = ParamGridBuilder().addGrid (stage_4.regParam, [0.1 ,
0.01]).build()

tvs = TrainValidationSplit ( estimator = stage_4 ,
estimatorParamMaps = paramGrid ,
evaluator = BinaryClassificationEvaluator() ,
trainRatio =0.8)
```

Spark MLlib Recommendation

```
ratings = spark.createDataFrame(ratingsRDD)
(training, test) = ratings.randomSplit([0.8, 0.2])
als = ALS(maxIter=5, regParam=0.01, userCol="userId",
itemCol="movieId", ratingCol="rating", coldStartStrategy="drop")

model = als.fit(training)
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))

userRecs = model.recommendForAllUsers(10)
movieRecs = model.recommendForAllItems(10)
users = ratings.select(als.getUserCol()).distinct().limit(3)

userSubsetRecs = model.recommendForUserSubset(users, 10)
movies = ratings.select(als.getItemCol()).distinct().limit(3)
movieSubSetRecs = model.recommendForItemSubset(movies, 10)
userSubsetRecs.show()
```

Work to be done to the next class

- ML model for predictive maintenance

- In the paper:
  - Description of ML model
  - Experiments and result analysis

References

- Estrada, R., & Ruiz, I. (2016). Big data smack. Apress, Berkeley, CA.
- Kumar, M., & Singh, C. (2017). Building Data Streaming Applications with Apache Kafka. Packt Publishing Ltd

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.