PYSPARK – INTRODUCTION

Spark keywords 1

Spark is a platform for cluster computing, for large datasets. Used so that both data processing and computation are performed in parallel over the nodes in the cluster.

Keyword	Description
cluster	A group of nodes.
node/ worker	Think of each node as a separate computer.
master	The computer that manages splitting up the data and the computations.
RDD	Resilient Distributed Dataset is a hard to work with low level object,
Spark DataFrame	Build on top of RDDs and similar to any data table.

.sql()

Attribute	Description
<Spark Session> .sql()	Using .sql() you can run spark sql queries on a TempView of a SparkSession.
<Data Frame> .show()	Prints the first 20 rows of the DataFrame. Argument truncate= false let you print longer string.

Example

```
query = "FROM flights SELECT * LIMIT 10" #Top 10 from flights.

flights10 = spark.sql(query)

flights10.show.(truncate= False)
# +------+------+-------+
  |field1|field2|field3|
  +------+------+-------+
   |val1 | val2  | val3   |
   |val4 | val5  | val6   |
```

Configure

The SparkContext class is used to make connection to a cluster. (old way) or SparkSession.builder can used (modern way).

For both methods the SparkConf() constructor can be used for configuration which takes a few optional arguments that allow you to specify the attributes of the cluster you're connecting to.

SparkContext()/ SparkSession.builder

Variable	Description
Conf/config()	The SparkConf object used to configure the SparkContext.
.getOrCreate()	Returns an existing SparkSession if there's already one in the environment or creates a new one if necessary.
.stop()	Stops the spark session.

SparkConf()

Variable	Description
appName	Sets the application name (used in logs and monitoring).
master	Sets the Spark master URL.(local, etc.).

Example

```
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession

conf = SparkConf()\
   .setAppName("My Spark App")\
   .setMaster("local[2]")  # Local mode with 2 cores.

# Use the SparkConf to create a SparkContext (old way).
sc = SparkContext(conf=conf)

# Or use SparkSession (recommended).
spark = SparkSession.builder\
   .config(conf=conf).getOrCreate()

....(code/pipeline etc.) spark.stop()
```

.catalog.listTables()

Description	Example
Returns a list of the names of the tables in the cluster.	import pyspark print(spark.catalog.listTables()) #spark = the SparkSession object. #[Table(name='flights', database=None, description=None, tableType='TEMPORARY', isTemporary=True)]

PYSPARK – INTRODUCTION

Spark →Pandas

As spark is quite heavy to run doing small queries is sometimes faster in a Pandas DataFrame.

Attribute	Description
.toPandas()	Transforms a spark DataFrame into a Pandas DataFrame.

Example

```
import pyspark
import pandas

query = "SELECT * FROM flights"
flight = spark.sql(query)

pd = flight.toPandas()
print(pd)
# origin dest   N
  0    SEA  RNO   8
  1    SEA  DTW  98
```

Pandas → Spark

Multiple steps are needed to Pandas DataFrame into a sparkSession (not available in the entire cluster only locally inside the session).

Attribute	Description
.createDataFrame()	Takes a Pandas DataFrame and returns a Spark DataFrame (only stored locally so can't use .sql() on the table).
.createOrReplace TempView()	Takes an alias name for the table, creates or overwrites the Spark DataFrame into the SparkSession as a temporary table (only accessible within the specific sparkSession).

Example

```
import pyspark
import pandas as pd
import numpy as np

pd_temp = pd.DataFrame(np.random.random(10))
spark_temp = spark.createDataFrame(pd_temp)
spark_temp.createOrReplaceTempView("temp")
```

File →Spark

With the .read attribute you read files into Spark DataFrames. (also read.json() etc.).

Attribute	Description	Example
.read.txt()	Used to read a txt() file into Spark DataFrame.	file_path = "/usr/local/share/datasets/airports.csv" airports = spark.read.csv(file_path, sep="," #Indicates the separator inside the file. ,header=True #Indicates that there are headers. ,inferSchema=True #Attemp to assign datatypes. ,nullValue='NA') #Null value in the source.
.read.csv()	Used to read a .csv() file into a Spark DataFrame.	

Overwrite/ Add a column to a Spark DataFrame

You can either use .sql with a query or use .withColumn() with a formula.

Attribute	Description	Example
.<Column name>	Select a singular column by doing <DataFrame>.<column name> .	import pyspark flights = spark.table("flights") flights = flights.withColumn("duration_hrs",\ flights.air_time / 60)
.withColumn()	Add or overwrites a column in the DataFrame, it requires two variables: Target field (new or existent). Values (can be a formula).	

PYSPARK – INTRODUCTION

Where clause

You can either use .sql with a query or use .filter() with a formula / bool/ sql query.

Attribute	Description
.filter()	Acts as a where clause and accepts both a formula or a Boolean variable or a string containing a sql where clause. it returns a DataFrame. Can use the "~" to get opposite results.

Example

```
import pyspark
# Filter by passing a string with a where clause.
long_flights1 = flights.filter(
   "distance > 1000")
# Filter by passing a  formula resulting in Boolean values.
long_flights2 = flights.filter(
   flights.distance > 1000)
```

Select 1

You can either use a .sql with a query or use .select() with a tuple containing .<column name> attributes or the column names in the form of strings.

Attribute	Description
.select()	Accepts a tuple of variables/ strings. Returns the specified columns as a DataFrame.

Example

```
import pyspark

# Select using columns names in the form of strings.
selected1 = flights.select("tailnum", "dest", "carrier")

# Select using the DataFrame  column attribute.
selected2 = flights.select(flights.tailnum, flights.dest, flights.carrier)
```

Select 2 AS

You can either use .sql() with a formula in the select and an AS or you can use a formula and the .alias() .

Attribute	Description
.alias()	Allows you to rename a column or name a column that results from a formula.

Example

```
import pyspark

#Select 3 fields and a calculated column using .alias().
avg_speed = (flights.distance/(
flights.air_time/60)).alias("avg_speed")

speed1 = flights.select("origin", "dest",
"tailnum", avg_speed)

#Select 3 fields and a calculated columns using strings and an AS.
speed2 = flights.selectExpr("origin", "dest",
"tailnum", "distance/(air_time/60) as avg_speed")
```

Joining tables

You can either use .sql() or a .join() attribute.

Attribute/ arguments	Description
.join()	An attribute of a DataFrame.
second DataFrame argument	First argument of .join() is reference too second DataFrame.
on argument	Is the name of the key column(s) as a string.
how argument	The kind of join (inner, outer, left_outer, right_outer, leftsemi, left, right).

Example

```
import pyspark

#Renames the column name.
airport =
airports.withColumnRenamed("faa", "dest")

#Example of a join.
flights_with_airports = flights.join(airports,
on="dest", how="left_outer")
```

PYSPARK – INTRODUCTION

GroupBy and aggregating

You can either use .sql() or .groupBy() to aggregate a DataFrame.

Attribute	Description
.groupBy()	Can be an attribute of .filter(). If left as .groupBy() it acts as a groupBy(*). If variables in the form of a tuple of column name(s) in the from of string(s) it acts as a normal sql group by.
.min()	An attribute of .groupBy() and only accepts a column name in the form of a string. Returns its minimum value.
.max()	An attribute of .groupBy() and only accepts a column name in the form of a string. Returns its maximum value.
.count()	An attribute of .groupBy() and only accepts a column name in the form of a string. Returns count of values found. If left empty .count() it counts all rows per group.
.avg()	An attribute of .groupBy() and only accepts a column name in the form of a string. Returns average of values found.
.sum()	An attribute of .groupBy() and only accepts a column name in the form of a string. Returns sum of values found.
.agg()	Can be an attribute of .filter(). Which lets you use any of the aggregate functions from the pyspark.sql.functions submodule e.g. standard deviations .stddev().

Example

```
import pyspark
import pyspark.sql.functions as F

# Example of min(), using filter.
flights.filter(flights.origin ==\  "PDX").groupBy().min("distance").show()

#Example of .sum(), using withColumn instead of filter.
flights.withColumn("duration_hrs", flights.air_time/60).groupBy().sum("duration_hrs").show()

#Example of an empty .count() and a groupBy() with a variable.
by_plane = flights.groupBy("tailnum"," dest") .count().show()

#Example of using agg().
by_month_dest.agg(F.stddev("dep_delay")).show()
```

Renaming a field

You can either use .sql with a query that uses an as or use .withColumnRenamed().

Attribute	Description	Example
. withColumn Renamed()	Accepts two arguments. One string indicating the current field and one string for the new field name.	import pyspark planes = planes.withColumnRenamed("year", "plane_year")

PYSPARK – BIG DATA FUNDAMENTALS

Big Data keywords

Big data is the study and application of complex data sets.

Keyword	Description
Volume	Size of data.
Variety	Different source and formats.
Velocity	Speed of the data.
Clustered computing	Collection of resources of multiple machines.
Parallel computing	Simultaneous computing on single computer.
Distributed computing	Collection of nodes that run in parallel.
Batch processing	Breaking the job into small pieces and running them on individual machines.
Real-time processing	Immediate processing of data.
Hadoop/ MapReduce	Scalable and fault tolerant framework written in java (open source) for batch processing.
Apache Spark	General purpose and fast cluster computing system (open source) for batch and real-time data processing.

Pyspark shell basic commands

Command	Description
sc.version	Returns the version of SparkContext.
sc.pythonVer	Returns the python version of SparkContext.
sc.master	Returns the url of the cluster or local string to run in local mode of SparkContext (local[*] means that is using all available thread on the computer where it is running.

Pyspark shell keywords 2

Keyword	Description
RDD	Default datatype in pyspark. Resilient Distributed Datasets. A collection of data distributed across the cluster.

Loading data in Pyspark shell

Command	Description	Example
parallelize()	Creating a parallelize collection. (creates an rdd) accepts argument minPartitions for specifying minimum number of partitions.	rdd = sc.parallelize ([1,2,3,4,5]) helloRDD = sc.*parallelize*("hello", minParitions = 4)
textFile()	Loads in lines of a txt file (creates an rdd). Accepts argument minPartitions.	rdd2 = sc.t*extFile* ("text.txt")
getNum Partitions()	Retrieves the numbers of partitions a rdd has.	rdd2.*getNum Paritions*()

Pyspark shell keywords

Pyspark shell is python-based command line tool.

Keyword	Description
Entry point	Way to connect too Spark Cluster.
SparkContext	Is the default entry point of spark.

Reduce()

An action used for aggregating element of a rdd. E.g. you can sum the entire rdd into a single value.

Example

```
x = [1,3,4,6]
rdd = sc.parallelize
rdd.reduce(lambda x , y: x +y)
#14
```

PYSPARK – BIG DATA FUNDAMENTALS

RDD Transformations

Operator	Description	Example
map()	A transformation which takes in a function and applies it to each element in the RDD.	rdd = sc.parallelize([1,2,3,4]) rdd_map = rdd.*map*(lambda x: x * x}
filter()	A transformation which takes in a function and returns an element with only the elements that pass the condition.	rdd = sc.parallelize([1,2,3,4]) rdd_filter = rrd.*filter*(lambda x: x >2)
flatMap()	A transformation returns multiple values for each element in the rdd. It accepts a function (often using split).	rdd = sc.parallelize(["hello world"," how are you"]) rdd_flatmap = rdd.f*latMap*(lambda x: x.split(" "))
union()	A transformation to combine multiple rdds into one rdd.	inputrdd = sc.textFile("logs.txt") errorrdd = inputrdd.filter(lambda x: "error" in x.split()) warningsrdd = inputrdd.filte*r*(lambda x: "warnings" in x.split()) combinedrdd = errorrdd.*union*(warningsrdd)

RDD Actions

Operator	Description	Example
collect()	An action that returns all elements of the rdd as an array.	rdd_map.*collect()* #[1,4,9,16]
take(N)	An action returns an array with the first N of elements of the rdd.	rdd_map.*take(2)* #[1,4]
first()	An action which returns the first element of an rdd.	redd_map.*first()* #[1]
count()	An action which returns the number of elements in the rdd.	redd_map.*count()* #4

Key/Value pairs RDDs

Pyspark provides a special data structure called pair rdds.

Create a pair rdd

Method	Example
from a list of key-value tuples	my_tuple = [("robin",27),("sabra", 31)] pairRRD_tuple = sc.parallelize(my_tuple)
from a rdd	my_list = ["robin 27", "sabra 31"] regularrdd = sc.parallelize(my_list) pairrdd_rdd = regularrdd.map(lambda s: (s.split(" ")[0], s.split(" ")[1]))

Transformations on pair rdds

Operator	Description	Example
reduceByKey(func)	Combine (e.g. .sums) values with the same key.	rdd_rk = rdd.*reduceByKey*(lambda x,y: x +y)
groupByKey()	Group values with the same key.	rdd_gb = rdd.*groupByKey*()
sortByKey()	Returns a rdd sorted by the key.	rdd.*sortByKey(ascending=False)*
join()	Join two pair rdds based on their key create a tuple of the values.	rdd1.*join*(rdd2)

PYSPARK – BIG DATA FUNDAMENTALS

saveAsTextFile()

Saves a rdd for each parathion as a separate file inside a directory. coalesce() can be used to save the rdd as a single text file.

Example

```
rdd.saveAsTextFile("tempFile") #Saves as many files.
rdd.coalesce(1).saveAsTextFile("tempFile")
#Saves as singular text file.
```

countByKey()

An action only available for key/value pair type rdds. Counts the number of elements for each key (uses allot of memory).

Example

```
rdd= sc.parallelize(["a",1), ("b",1),("a",1)])
for kee, val in rdd.countByKey().items():
  print(kee, val)
# ("a", 2) ("b",1)
```

collectAsMap()

An action only available for key/value pair type rdds. It returns the key value pairs as a dictionary. It is a memory heavy function.

Example

```
sc.parallelize([1,2),(3,4)]).collectAsMap()
#{1:2, 3:4}
```

RDD → Spark DataFrame

.createDataFrame() (see pandas → spark) is also used to convert a rdd to a DataFrame but takes a second argument being schema.

Example

```
people_rdd = sc.parallelize([
("robin",27, 1.80), ("sabra",31,1.60) ])
people= ["name", "age", "height"]
people_df = spark.createDataFrame(
people_rdd, schema=people)
```

Distinct

Can be done through .sql and a distinct or using dropDuplicates().

Attribute	Description
drop Duplicates()	Removes duplicate rows from a DataFrame. Accepts a column name [] to filter on that column.

Example

```
test_df_no_dup = test.select(
"name", "age").dropDuplicates()
```

Inspect table

Attribute	Description
print Schema()	Prints the types of columns in the DataFrame.
columns	Prints the column names of the DataFrame.
describe()	Compute summary statics of numerical columns of the DataFrame. Also accepts a single column.

Example

```
test_df.printSchema()
#|-- name: string(nullable = true)
  |--age: integer (nullable = true)

test_df.columns
#["name", "age"]

test_def.describe()
#summary |age
  count       | 2  ...(mean, stddev, min ,max)
```

Order BY

Can be done through .sql() and an order by or using orderby().

Attribute	Description
orderby()	Sorts the DataFrame based on one or more columns.

Example

```
test_df.orderBy("Age").show()
```

PYSPARK – BIG DATA FUNDAMENTALS

Data visualization		
Method	**Description**	
pyspark_dist_explore_library	Accept spark DataFrames and provides quick insight into DataFrames.	
Attribute	**Description**	**Example**
hist()	Creates a histogram.	test_df = spark.read.csv("test.csv", header = True, inferSchema=True) test_df_age = test_df.select("age" *hist*(test_df_age, bins=20, color= "red")
distplot()	Creates a distribution plot.	
pandas_histogram()	Creates a pandas like histogram.	
Method	**Description**	
toPandas()	Pyspark function (See Spark → Pandas) that converts a spark DataFrame to a Pandas . DataFrame which than with e.g. Matplotlib be visualised (see python cheat sheet) but is limited to single-server memory but is mutable.	
Method	**Description**	**Example**
HandySpark library	Using .toHandy() convert a spark DataFrame too handy DataFrame on which other handy attributes can be used like .hist(). Benefits is easy data fetching while retaining computations.	test_df = spark.read.csv("test.csv", header = True, inferSchema=True) hdf = tset_df.toHandy() hfd.cols["age"].hist()

PYSPARK – MACHINE LEARNING 1 Introduction

Spark keywords 2

The pyspark.ml can be used every stage of the machine learning pipeline. Spark only handles numeric data for modeling.

Keyword	Description
Transformer	Transformer classes have a .transform() method that takes a DataFrame and returns a new DataFrame.
Estimator	Estimator classes all implement a .fit() method. These methods also take a DataFrame, but instead of returning another DataFrame they return a model object.
double	Is a spark decimal value.

1

Cast

You can either use .sql() with a cast as or use .cast() on a column selected inside .withColumn().

Attribute	Description
.cast()	Accepts a singular argument indicating the type e.g. "integer" or "double".

Example

```
import pyspark

model_data = model_data.withColumn("arr_delay", model_data.arr_delay.cast("integer"))
```

2

Dealing with coded strings

As you can't use strings in a model you have to convert them into factors.

Constructors	Description
StringIndexer	An Estimator class that takes a DataFrame with a column of strings and map each unique string to a number which is returned as a transformer. Returning the entire DataFrame.
OneHotEncoder	An Estimator class that encodes the numeric column as a one-hot vector. Returning a transformer that returns a column that encodes your categorical feature as a vector that's suitable for machine learning routines. returns a column filled with tuples.

Example

```
from pyspark.ml.feature import StringIndexer, OneHotEncoder

carr_indexer = StringIndexer(inputCol="carrier", outputCol="carrier_index")

model= carr_index.fit(df)
indexed = mode.transform(df)

carr_encoder = OneHotEncoder(
   inputCol="carrier_index",
   outputCol="carrier_fact")

encoded_df = carr_encoder.transform(indexed)
```

3

Vector Assembler

The last step in the Pipeline is to combine all of the columns containing our features into a single column. Storing each of the values from a column as an entry in a vector.

Constructor	Description
Vector Assembler()	Transformer takes a list of the columns and combines them into a new vector column.

Example

```
from pyspark.ml.feature import VectorAssembler

vec_assembler = VectorAssembler(inputCols=["month", "air_time", "carrier_fact", "dest_fact", "plane_age"], outputCol="features")
```

PYSPARK – MACHINE LEARNING 1 Introduction

4

Modeling pipeline	
Constructor	**Description**
PipeLine()	A Constructor accepting a list of the assigned variables too the other constructor classes (StringIndexer, OneHotEncoder, VectorAssembler) making the whole modeling process easily reusable.
Attributes	**Description**
.fit()	Attribute of the pipeline. Accepts a DataFrame and returns a Model (by passing it through the PipleLine). ONLY FOR TRAINING DATA.
.transform()	Attribute of the pipeline. accept an DataFrame and returns an edited one. (by passing it through the PipeLine). FOR TESTING AND TRAINING DATA.
.random Split()	Attribute of PipeLine output. Accepts a list of values by which it splits the data into a test and training set.

Example

```
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline

model_data = model_data.withColumn("arr_delay", model_data.arr_delay.cast("integer"))

carr_indexer = StringIndexer(inputCol="carrier", outputCol="carrier_index")
carr_encoder = OneHotEncoder(inputCol="carrier_index", outputCol="carrier_fact")

dest_indexer = StringIndexer(inputCol="dest",outputCol="dest_index")
dest_encoder = OneHotEncoder(inputCol="dest_index",outputCol="dest_fact")

vec_assembler = VectorAssembler(inputCols=["month", "air_time", "carrier_fact", "dest_fact",
"plane_age"], outputCol="features")

flights_pipe = Pipeline(stages=[dest_indexer, dest_encoder, carr_indexer, carr_encoder,
vec_assembler])

piped_data = flights_pipe.fit(model_data).transform(model_data)

training, test = piped_data.randomSplit([.6, .4])
```

Spark keywords 3	
Keyword	**Description**
estimators	Are prediction models.
hyperpara meter	A value in the model that is provided instead estimated to improve performance.
k-fold cross validation	A method of estimating the model's performance on unseen data.

1

Logistic regression	
Estimator	**Description**
Logistic Regression()	Predicts the probability of something (between 0 and 1).

Example

```
from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression()
```

Machine Learning keywords	
Keyword	**Description**
Regression	Filling in missing value by proving part of the information.
Classification	Classifying by providing information.

PYSPARK – MACHINE LEARNING 1 Introduction

2 Evaluating models

Cross validation for model selection is a way to compare different models. pyspark.ml.evaluation submodule has classes for evaluating different kinds of models.

Class	Description
Binary Classification Evaluator()	Used to evaluate binary classification models.

Example

```
import pyspark.ml.evaluation as evals

evaluator = evals.BinaryClassificationEvaluator(
metricName="areaUnderROC")
```

3 Make a grid

You need a grid of values to search over when looking for the optimal hyperparameters.

Class	Description
ParamGrid Builder()	Class to create grids.
Attributes	**Description**
.addGrid()	Takes a model parameter (attribute of an Estimator) and a list of values you want to try.
.build()	Takes no arguments, returns the grid.

Example

```
import pyspark.ml.tuning as tune
import numpy as np

grid = tune.ParamGridBuilder()

grid = grid.addGrid(lr.regParam, np.arange(0, .1, .01))
grid = grid.addGrid(lr.elasticNetParam, [0,1])

grid = grid.build()
```

4 Make the validator

The validator performs the cross validation.

Class	Description
CrossValidator()	Estimator which takes the modeler, grid.
Arguments	**Description**
Estimator	The chosen Estimator variable (see 1).
estimator ParamMaps	The grid variable (see 3).
evaluator	The chosen evaluator variable (see 2).

Example

```
import pyspark.ml.tuning as tune

cv = tune.CrossValidator(
        estimator= lr,
        estimatorParamMaps=grid,
        evaluator= evaluator)
```

5 5 Find the best model(s)

Attributes	Description
.fit()	Attribute of the validator. Cross validates the models (see 4), accepts a dataset (see 4). This is very computationally intensive procedure.
.bestModel	Attribute of the validator.fit(). extracts the best model.

Example

```
import pyspark

models = cv.fit(training)

best_lr = models.bestModel
```

6 6 Evaluating models

Attribute	Description	Example
.transform()	Runs the model on a set e.g. the testset (see 4).	import pyspark test_results = best_lr.transform(test) print(evaluator.evaluate(test_results)) #0.7123313100891033
.evaluate()	Returns the metric, which can be used to measure how good a model is (see 5). (for a binary classification model, the closer to 1 the better).	

PYSPARK – CLEANING DATA

1

Schema

Keyword	Description
Data cleaning	Preparing raw data for use in data processing pipelines. E.g. formatting it into a specified format.
Spark Schema	Defines and validates the number and types of columns for a given DataFrame.

Example

```
import pyspark.sql.types
peopleSchema = StructType( [
  #Define name field and if its nullable.
  StructField("name", StringType(), True),
  #Define age field and if its nullable.
  StructField("age", IntegerType(), True),
  #Define city field and if its nullable.
  StructField("citye", StringType(), True)])
```

2

Load()

To load a file using a schema the load operator can be used using the schema variable.

Example

```
people_df =
spark.read.format('csv').load(name=
'rawdata.csv', schema = peopleSchema
```

.Drop()

Drops a (list of) column(s) from a DataFrame.

Example

```
aa_dfw_df = aa_dfw_df.drop(
aa_dfw_df['Destination Airport'])
df= df.drop(*list_of_cols)
```

Parquet → Spark

To load a .parquet file into a spark DataFrame two methods can be used the .load() method or the .parquet() method.

Example

```
#.format().load() method.
df = spark.read.format("parquet").load(
"filename.parquet")

#.parquet() method
df= spark.read.parquet("filename.parquet")
```

Spark → Parquet

To load a spark DataFrame to a .parquet. you can use the .save() or .parquet() method.

Example

```
#.format().save() method.
df.write.format("parquet").save(
"filename.parquet")

#.parquet() method
df.write.parquet("filename.parquet",
mode="overwrite")
```

distinct()

Operator	Description	Example
distinct()	Operator of a .select() and returns the distinct rows.	voter_df.select(voter_df['VOTER_NAME']).distinct().show()

String column transformations

Operator	Description	Example
.upper()	Returns uppercase version of the string. Opposite is lower().	import pyspark.sql.function as F voter_df.withColumn("upper", F.upper("name"))"
.split()	Returns a list from a string field by delimiter.	voter_df.withColumn("splits", F.split("name", " "))

PYSPARK – CLEANING DATA

ArrayType() column functions

Operator	Description	Example
.size(<column>)	Returns the length of the array.	voter_df = voter_df.withColumn('last_name', voter_df.splits.getItem(F.*size*('splits') - 1))
.getItem(<index>)	Returns the item of the array at the specified index.	voter_df = voter_df.withColumn("first_name", voter_df.splits.*getItem*(0)

Conditional clauses

.when()

Description

Argument within a .select method which create and if then statement. You can chain them together to create a case statement. Accepts two variables the condition and result.

.when(<if condition>, <then x>

Example

```
import pyspark.sql.function as F

#Singular when.
df.select(df.Name, df.Age, F.when(df.Age >= 18,
"Adult"))

#Chained When.
df.select(df.Name, df.Age,
  when(df.Age >= 18, "Adult")
  .when(df.Age < 18, "Minor"))

#Without select.
voter_df = voter_df.withColumn('random_val',
   when(voter_df.TITLE == 'Councilmember',
   F.rand()))
```

.otherwise()

Description

An Addition too the when clause and is used as the Else in a case statement.

.otherwise(<then x>)

Example

```
import pyspark.sql.function as F

df.select(df.Name, df.Age,
  .when(df.Age >=18, "Adult")
  .otherwise("Minor"))
```

.rand()

Creates a random value between 0.0 and 1.0.

Example

```
import pyspark.sql.function as F

voter_df =
voter_df.withColumn('random_val',
F.rand()
```

.monotonically_increasing_id()

Creates a unique id for each record. Like a row number but without order. Adding a .presist() keeps it the same over all future DataFrame operations.

Example

```
from pyspark.sql.functions import
monotonically_increasing_id

voter_df = voter_df.withColumn('ROW_ID',
F.monotonically_increasing_id()).presist()
```

PYSPARK – CLEANING DATA

UDF (user defined function) using a python DEF

Constructor	Description
pyspark.sql. function.udf	A udf is warped using pyspark.sql.function.udf constructor. It is used to make Python DEF functions usable in SPARK (you also use lambda instead of a def). it is stored as a variable and can be called as a Spark function. It takes 2 arguments the name of the DEF method, and the Spark data Type the DEF returns (can also be a schema object).

Example

```
from pyspark.sql.function import udf

#Create python function.
def reverseString(mystr):
  return myster[::-1}

#Create an udf.
udfReverseString = udf(reverseString, StringType())

#Use udf.
user_df = user_df.withColumn("reversename", udfReverseString(user_df.Name))
```

. rdd.getNumPartitions()

Returns the number of partitions (groups of data) of an rdd.

Example

voter_df.rdd.*getNumPartitions*()

. is_cached

Returns true if DataFrame is cached.

Example

print(voter_df.*is_cached*) #true

.presist(<storagelevel>)

Makes it possible to specify the storage level (MEMORY_ONLY, MEMORY_AND_DISK, DISK_ONLY).presist() is the same as cache().

Example

voter_df.cache().*presist*(MEMORY_ONLY)

Caching

Caching saves a DataFrame too local storage like SSD / NVMe while it should be used sparsely it can have performance benefits when doing multiple transformations on a not too large dataset.

Operator

.cache()

Example

voter_df = spark.read.csv("voter_data.txt.gz")
voter_df.*cache()*.count()

.unpresist()

Argument	Description
.unpresist()	Removes the DataFrame from cache.
catalog. clearCache()	Removes all cached tables.

Example

voter_df.*unpresist*()

spark.catalog.clearCache()

PYSPARK – CLEANING DATA

Importing large files

Importing large files can be intensive. Splitting a file prior improves performance by enabling separate parts to be loaded by separate parts of the cluster.

Splitting Example

Splitting can be done through a cmd line e.g. "*split -l 10000 -d largefile chunk*" splits a file in chunks of 10000 lines.

Import a spitted file Example

Importing a split file can be done using wild card letters e.g.
airport_df = spak.read.csv("airports-.txt.gz"* imports all files starting with airports.

Broadcast

Broadcast gives every worker its own copy of a DataFrame when joining this can speed up performance by providing each worker with the joining table (specifically if the broadcasted table is small).

Operator

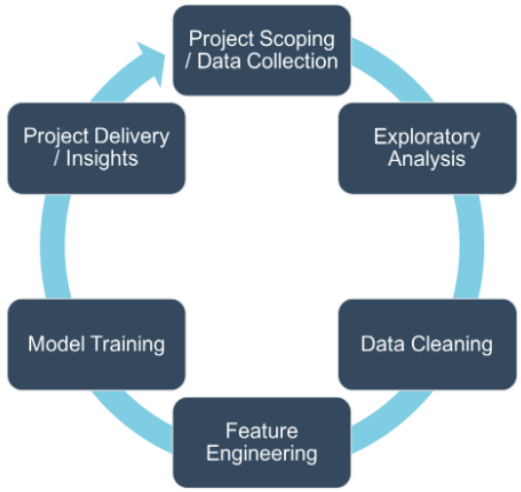
.broadcast(<DataFrame>)

Example

from pyspark.sql.functions import broadcast

combined_df = df_1.join(*broadcast*(df_2))

Data science processes



Cluster configuration

Operator	Description
spark.conf.get(<configuration name>)	Retrieves the configuration settings of the cluster.
spark.conf.set(<configuration name>))	Sets and writes the configuration of a cluster.

Spark execution plan

Spark has a build in execution plan which can be run on any DataFrame with an action.

This returns the actions and order of actions and where the source data comes from etc.

Operator

.explain() in dot n
in spark sql you can start the query with EXPLAIN

Example

#1
voter_df = df.select(df["VOTER NAME"].distinct()

voter_df.*explain*()

#2
spark.sql("*EXPLAIN* SELECT * FROM df")

Replace

Function	Description
regexp_replace("<column>", "<string1>", "<string2>")	Requires 3 arguments a column and two strings. It Replaces all occurrences of string1 with string2 within the column. Special characters have to be escaped.

Example

df = df1.select(*regexp_replace*("column", "mr\.", "mr")
#mr. holmes →mr holmes

PYSPARK – FEATURE ENGINEERING

.dtypes

Returns a list of tuples of columns and their data types.

Operator

<DataFrame>.dtypes

Example

df.*dtypes* #["no.", "integer"),("text", "string")}

.cov()

Covariance lets us see how two variables vary together. Takes two numeric columns and returns a value.

Operator

<DataFrame>.cov(<column1>,<column2>)

Example

df.*cov*("salesClosePrice", "yearBuilt"

Filtering using Where + Like

To filter using a Like SQL condition where() like() can be used.

Operator	Description
where()	Accepts an BooleanType condition or an string of SQL expression, filters set where True.
like()	Equivalent of the Like in SQL including %, returns a Boolean.
~	The NOT condition.
&	Equivalent of AND.

Example

df = df.*where((*~df["status"].*like*("%Not Disclosed")) *&* (df["price"] > 100))

Log Scaling

If data has to many outliers Log Scaling can be used to make the data look more like a normal distribution. Using the .log() function.

Example

from pyspark.sql.functions import log
df = df.withColumn("log_SalesClosePrice", *log(*df["SalesClosePrecie"]))

.mean()

Returns the average from the values in an aggregated column.

Operator

.mean()

Example

MSE = rates_and_preds.map(lambda r: (r[1][0] - r[1][1])**2).*mean()*

df.agg({"SalesClosePrice" : "*mean*"}).collect()

Sample()

creates a sample from a large DataFrame so it can be converted to a Pandas DataFrame that doesn't support large datasets. Which than can be used to make graphics using matplotlib or seaborn.

Arguments	Description
withReplacement	If repeats of the same values is allowed.
fraction	% of the DataFrame that should be taken from the dataset.
seed	Random seed for reproducibility.

Example

sample_df = *df.sample(False,0.5, 42)*
pandas_df = sample_df.toPandas()

.Dropna()

Drops any row where there is any null value.

Arguments	Description
how	"any"/"all" if any drop a row if it contains any nulls, if all drop record only if all values are null.
thresh	Int, if specified drop rows that has less than thresh amount of null values.
subset	Optional list of column names to consider.

Example

df= df.*dropna()*

PYSPARK – FEATURE ENGINEERING

MinMax scaling

To reduce errors in regression or KNN algorithms all variables should be on the same scale e.g. 0-1000 or 0-1. This can be done through MinMax scaling (the preferred scale is 0.0 – 1.0).

Example

```
#Define min and max values of a column.
max_days = df.agg({"daysOnMarket": "max"}).collect()[0][0]
min_days = df.agg({"daysOnMarket": "min"}).collect()[0][0]

#Create a new column based off the scale.
df = df.withColumn("scaled_days",(df["daysOnMarket"] – min_days) / (max_days – min_days))
```

Standardization

Standardization is transforming data to standard normal distribution (lower the peak and shift it too the middle) thus Having a Mean of 0 and a standard Deviation of 1. With anything beyond one being the outliers.

Example

```
#Define the mean and stddev of the column.
mean_days = df.agg({"daysOnMarket" : "mean"}).collect()[0][0]
stddev_days = df.agg({"daysOnMarket" : "stddev"}).collect()[0][0]

#Create a new column with the standardized scaled data.
df = dfwithClumn("ztrans_days", (df["daysOnMarket"] – mean_days / stddev_days)
```

Missing data

Function	Description	Example
isNull()	Returns true if a value is null.	df.where(df{"roof"].*isNull()*.count()
fillna(value, subset=<column name>)	Accepts two arguments, value, subset. With Value being with what the null replaced is and subset being a list of columns where this must be applied on.	*df.fillna(0, subset = ["roof"])*

Cast 2 (cast dates)

To cast a date instead of .cast() you use to_date()/ to_timestamp().

Attribute	Description	Attribute	Description
.to_date(<column>)	Cast a column to a date datatype.	.to_timpe_stampe(<column>)	Cast a column to a date_time datatype.
Example		**Example**	
from pyspark.sql.functions import to_date df = df.withColumn("listdate", *to_date*("listdate")		from pyspark.sql.functions import to_date df = df.withColumn("listdate", *to_timestamp(*"listdate")	

PYSPARK – FEATURE ENGINEERING

Day components

To retrieve ordinal features from a date you can use year(), month(), dayofmonth(), weekofyear(). ect

Functions	Description
year()	Returns the year from a date field.
month()	Returns the month from a date field.
dayofmonth()	Returns the day of the month from a date field.
weekofyear()	Returns the week number from a date field.
datediff(col1, col2)	Returns the difference between two date fields.

Example

```
from pyspark.sql.functions import year, month, dayofmonth(), weekofyear()

df = df.withColumn("year", year("date"))
df = df.withColumn("month", month("date"))
df = df.withColumn("day", dayofmonth("date"))
df = df.withColumn("weekNr", weekofyear("date")
df = df.withCOlumn("daysOnMarket", datediff("offerDate", "listDate"))
```

Lagging

To take into account how long it takes for one thing to affect another within a DataFrame that contains a datetype field, you can use and add a lagged field.

Functions	Description
Window()	Returns a record based off a group of records.
lag(col, count=1)	Returns the value that is offset by row before the current row.

Example

```
from pyspark.sql.functions import lag
from pyspark.sql.window import Window

w = Window().orderBy(m_df["date"])
m_df = m_df.withColumn( "mortgage_1wk", lag("mortgage", count=1).over(w)

#Output
date          | mortgage| mortgage-1wk|
2023-10-10| 4.23        | null            |
2023-10-17|  4.28       | 4.23            |
2023-10-24|  4.13       | 4.28            |
```

1

Explode()

To extract each value of a list into its own row you can use Explode().

Function	Description
explode([<column name>])	Accepts list type column and splits the list creating a row for each value.

Example

```
from pyspark.sql.functions import split, explode, lit, coalesce, first

#Split the column on commas into a list.
df= df.withColumn("roof_list", split(df["roof"], ', '))

#Explode list into new record for each value.
ex_df = df.withColumn("ex_roof_list", explode(df["roof_list"]))
```

PYSPARK – FEATURE ENGINEERING

2

Pivot()

To pivot an Exploded list giving each value of the list its own column you can use pivot().

Function	Description
pivot()	An aggregate function which creates columns from values inside column spread over multiple rows.

Example

```
#Create a dummy column of constant value which can be used to group by on during the pivot.
ex_df = ex_df.withColumn("constant_val", lit(1))

#Pivot the values into Boolean columns (grouping by the original row number before the explode) combined with coalesce to ignore nulls and first in order to take the first retrieved value.
piv_df = ex_df.groupBy("NO").pivot("ex_roof_list").agg(coalesce(first("constant_val")))
```

lit()

Lit creates a fixed value also known as literal value and accepts all sorts of data types.

Operator

lit(<value>)

Example

```
df = df.withColumn("fixed_val", lit("pizza"))
```

Bucketing

Bucketing also known as binning is a way to create ordinal variables or categorical variable (grouping of values into categories).

Constructor

Bucketizer(splits= <list of categories>, inputCol= <input column>, outputCol= <output column>).

Example

```
from pyspark.ml.feature import Bucketizer

#Define the categories.
splits = [0, 1, 2, 3, 5, float("inf")]

#Create bucketing transformer.
bukc= Bucketizer(splits=splits, inputCol= "bathstotal", outputCol = "baths")

#Apply the transformer
df = buck.transform(df)
```

Binarizing

Binarizing is the transformation of data into Boolean values. The binarizer constructor requires a double data type value as input.

Constructor

Binarizer(threshold= <any above this number is set to 1 everything below or = is set to 0> inputCol=<input column>, outputCol= <output column>.

Example

```
from pyspark.ml feature import Binarizer

#Cast the int data type to double.
df = df.withColumn("fireplaces", df["fireplaces"].cast("double"))

#Create binarizing constructor.
bin = Binarizer(threshold= 0.0, inputCol = "fireplaces", outputCol = "fireplace_bool")

#Apply the transformer.
df = bin.transform(df)
```

PYSPARK – FEATURE ENGINEERING

SPARK UI keywords

The spark UI is a web interface to inspect spark execution. It shows cache, settings and stored SQL queries.
The spark UI generally runs on htt://<driver_host>:4040 with 4040 being the port when its available else 4041 etc..

Keyword	Description
spark task	Is a uni of execution that runs on a single cpu.
spark stage	A group of tasks that perform same task in parallel.
spark job	A computation triggered by an action.

Has attribute()

Function	Description
hasattr(object, value)	Build in Python functions. Is a reliable way to determine that an object is a spare vector. By checking if an object has a specific attribute.

Example

hasattr(vector, "toArray ")

CountVectorizer

Constructor	Description
Countver ctorizer(inputCol, outputCol	Feature of Extractor its input is an array of strings and returns a sparse vector. In the form of (total number of unique words found, [list of the word_ids],[list how frequent the words were found within total dataset].

Example

```
from pyspark.ml.feature import CountVectorizer

cv = CountVectorizer(inputCol="words", outputCol="features")

model = cv.fit(df)
result = model.transform(df)
```

Repartitioning

Repartitioning is splitting data into groups that can be send too different workers.

Argument	Description
.repartition(<int>, "<column name>")	Partitions a DataFrame in multiple parts indicated by the first argument. The second argument column name is used to decided how partition it (thus creating groups where the value of the column is the same).

Example

df2 = df.*repartition*(4, "column")

.dropTempView

Argument	Description
.dropTempView("<table>")	Drops a temp table from memory.

Example

spark.catalog.*dropTempView*("table1")

length

Function	Description
length	Returns the length of a character string (including trailing spaces).

Example

```
from pyspark.sql.functions import length

df.where(length("sentence") ==0)
```

numNonzeros()

Function	Description
numNonzeros()	Build in Python function. Returns the number of non-zero values in a vector.

Example

vector.*numNonzeros*()

PYSPARK – SPARK SQL

Inspecting table schema

To inspect a table schema using Spark SQL you can add "SHOW COLUMNS FROM" in a spark sql query. Other queries are also possible.

Example

#1
columns = spark.sql(*"SHOW COLUMNS FROM tablename")*
#2
columns = spark.sql("*SELECT * FROM tablename LIMIT 0"*
#3
columns = spark.sql("*DESCRIBE tablename")*

SQL top ...

The equivalent of an t-sql top int in spark SQL is LIMIT int, LIMIT is placed at the end rather than the beginning.

Example

spark.sql("SELECT train_id as train, station FROM schedule *LIMIT 5*"

#Is the same as.
schedule.show(5)

unix_timestamp

Reformats a datetime/ date/ time data field into a unix_timestamp format. It has both an don't n and spark sql equivalent from the unix_timestamp t-sql function.

Clause	Description
UNIX_TIMEPSTAMP(column, "format") in spark sql, .unix_TimeStamp("column", "format") in dot n	Formats the column argument to the unix format argument.

Example

query = """*UNIX_TIMESTAMP*(time, 'H:m') from table """

Window function

A window function allows when calculating a value each row can use the information of any other row to calculate its value.

Clause	Description
LEAD() in spark sql lead("column", int) in dot n	Let's you query more than one row at a time. Accepts a column and how many need to be selected.
OVER() in spark sql .over() in dot n	Designated the query as a window function query. Must contain an ORDER BY CLAUSE. (optional PARTIION BY clause).
PARTITION BY() in spark sql .partitionBy() in dot n	How rows should be grouped before ordering (optional).
ORDER BY in spark sql, .orderBy() in dot n	How data should be ordered.

Examples

#Time_next is time of the row below the current row achieved through LEAD, OVER, ORDER BY.

#1 SPARK SQL.
QUERY = """SELECT
train_id,
station,
time,
LEAD(time,1) *OVER (PARTITION BY* train_ID *ORDER BY* time) as time_next
FROM sched
"""

#2 DOT NOTATION.
from pyspark.sql import Window
from pyspark.sql.functions import row_number

dot_df = df.withColumn('time_next', *lead(*'time', 1)
.over(*Window.partitionBy(*'train_id')
.*orderBy*('time')))

PYSPARK – MACHINE LEARNING 2 Recursive/ Logistic

Machine Learning keywords 2

Keyword	Description
Decision tree	A way to answer a classification question. By splitting it up in multiple yes or no questions.
Recursive Partitioning	Machine learning technique closely related to decision trees.
Logistic regression	Another method to answer classification question but instead answering true or false. (1/0). It does this by creating a logistic curve and placing the given data on that curse and then deciding if it is closer to 1 or 0.

MulticlassClassificationEvaluator

Function	Description
Multi Classification Evaluator	Classifier object which accepts the test results and an evaluator.metricName including weightedRecall, weightedPrecision, accuracy and f1 (combination).

Example

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator()
evaluator.evaluate(predicition, {evaluator.metricName: "wieghtedPrecision"})

Building a decision tree model

Keyword	Description
Decision Tree Classifier	Classifier object. Which creates ad Decision tree model.
Confusion matrix	By grouping the evaluation on label, prediction and including a count. You create a confusion matrix showing true positives, false positives, false negatives and true negatives. accuracy =(TN + TP) / (TN + TP +FN +FP).

Example

from pyspark.ml.classification import DecisionTreeClassifier

.... #(Loading and preparing data, creating test and training set).

#Create the classifier object.
tree= *DeicisionTreeClassifier()*
tree_model = tree.fit(cars_train)

#Evaluating (comparing the label and prediction column).
prediction = tree_model.transform(cars_test)

#Confusion matrix.
prediction.groupBy("label","predicition").count()
.show()

Hashing

Function	Description
HashingTF	A function which converts words into numbers. Accepts 3 arguments inputCol, outputCol, and numFeatures (which indicates the number of unique numbers the hash can create and if smaller than actual max of different words it will group words. The hashed column will contain a tuple with first the numFeatures amount, then a list of the hashed values and then a list of how frequent the word appeared in the pre hashed list.

Example

from pyspark.ml.feature import HashingTF

hasher = *HashingTF*(inputCol="words", outputCol="hash", numFeatues=32)
books = hasher.transform(books)

PYSPARK – MACHINE LEARNING 2 Recursive/ Logistic

Turning text into tables

Keyword	Description
term-document matrix	A table containing a column for each word found into a text or list of texts, with the values representing the count of how many times the word appeared in the text. With one row per text.
regexp_replace()	A regex function which takes 3 arguments, column, list of what needs to be replaced and what it has to be replaced with.
Tokenizer()	A function which splits an columns value into a list of values. Accepts two arguments inputCol and outputCol.
StopWordsRemover()	A function which removes a specific list of words that frequent occur in texts but don't add any value.
.gestStopWords()	An operator of StopWordsRemover which returns all the stop words that will be removed when applying stopWordsRemover. This list can be customized.
IDF	Function used to account of words that occur frequently across multiple rows. As those words should be valued less for building a classifier model than words that only occur rarely. It accepts 2 arguments inputCol (a hashed column) and outputCol and returns a column containing a tuple first the numFeatures of the hashed column and then a list of the hashed values and then a list of the values of each of the words. Which can be used for building a logistic regression model.

Example

```
from pyspark.sql.functions import regexp_replace
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, IDF
from pyspark.ml.classification import LogisticRegression

books = spark.read.csv(booktitles.csv)

# Removing punction and replacing with a space.
Regex = "[,\\-]"
books.withColumn("text", regexp_replace(books.text, REGEX, " ")

# Tokenize the column.
books = Tokenizer(inputCol="text", outputCol="tokens").transform(books)

# Remove stop words.
stopwords = StopWordsRemover()
stopwords.getStopWords()
stopwords = stopwords.setInputCol("tokens").setOutputCol("words")
books = stopwords.transform(books)

# Has the left-over words.
hasher = HashingTF(inputCol="words", outputCol="hash", numFeatues=32)
books = hasher.transform(books)

# Give value to each of the hashed words.
books = IDF(inputCol = "hash", outputCol= "features").fit(books).transform(books)

# Train a model using the output.
books_train, books_test = books.randomSplit([0.8, 0.2], seed=13)
logistic = LogisticRegression(regParam=0.2).fit(books_train)
```

PYSPARK – MACHINE LEARNING 2 Recursive/ Logistic

1

Building a LinearRegression model

Keyword	Description
Linear Regresion	Used for predicting continue outcomes based on one or more predictor variables. E.g. predicting stock prices. It accepts a singular column that should be predicted.
Regression Evaluator	Evaluate linear regression model. Accepts the predictable column and has as an orgistrator containing the test.

Example

```
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator

.... #Creating feature and one hot encode the feature.and creating a test and train set.

# Create model.
regression = LinearRegression(labelCol="consumption")

# Train and test data.
regression = regression.fit(cars_train)
predicitions = regression.transform(cars_test)

RegressionEvaluator(labelCol="consumption")
.evaluate(predictions)
```

Bucketing

Keyword	Description
Bucketing	Is making groups of data according to ranges and labelling each range with its own value in a separate column.
Class	**Description**
Bucketizer	Creates a bucketed column. Accepts 3 arguments. Splits which is a list of bin boundaries, InputCol and outputCol.

Example

```
from pyspark.ml.feature import Bucketizer

#Create 3 buckets.
bucketizer = Bucketizer(
splits=[3500, 4500, 6000, 6500],
inputCol="rpm",
outputCol="rpm_bin"

bucketed = bucketizer.transform(cars)

... #(hot encode before using in model)
```

Cross validation with folds

Used to validate a model by putting multiple test sets (folds) though it and evaluating the results.

Function	Description
Cross Validator	Object accepts the estimator (model), estimatorParamMaps (grid), evaluator, numFolds, seed.
.avg Metrics	Operator of the CrossValidator which returns the average Root means squared error for a more trustable validation.

Example

```
from pyspark.ml.tuning import CrossValidator
....# Creation of a pipeline, a  LinearRegression model  and a RegressionEvaluator.

params = ParamGridBuilder().build()

cv = CrossValidator(estimator = regression, estimatorParamMaps = params, evaluator = evaluator, numFolds=10, seed=13

cv=cv.fit(cars_training)
cv.avgMetrics #[0.8000663722151572]
```

PYSPARK – MACHINE LEARNING 2 Recursive/ Logistic

2

Regularization LinearRegression

Overfitting occurs when there are too many predictors supplied to a model resulting in the model thinking things are important while they are not or just making the model slower.

Keyword	Description
Lasso	Absolute value of the coefficients.
Ridge	Square of the coefficients.
Arguments	**Description**
elesticNetParam	Argument of LinearRegression model creator. When >0 you get a Lasso Regression. Identifying the most import predictors and setting the other coefficients to 0.
regParam	Argument of LinearRegression model creator.. When >0 you get a Rdige Regression.
Operator	**Description**
.coefficients	It's an Operator of the trained mode. Returns a dense vector of all the predictors that are used for predicting any that are not 0 are contributing to the prediction. The further away from 0 the more important they are.

Example

```
...#Building a linear regression model example.
regression.coefficients
#[-0.012, 0.147, -0.897, -1.445, -0.985, -1.071, -1.335, 0.189, -.078, 1.160])

ridge = LinearRegression(labelCol = "consumption", elasticNetParam=1, regParam=0.1)
ridge fit(cars_train)
ridge.coefficients

# Combination of the two Regression methods resulted in that only two predictors are now used to make the prediction.
#([0.0, 0.0, 0.0, -0.056, 0.0, 0.0, 0.0, 0.026, 0.0, 0.0])
```

Ensemble 1 Random Forest

Is a collection of models which combined the results from multiple mode to create better predictions.

Model/ Attributes	Description
RandomForest Classifier	An ensemble of Decision tree models each trained on random subset of data, which all operator parallel. Has an argument numtrees which by default is 20.
.trees	An operator to offers a way to access the individual trees within a forest.
.transform()	Using a .transform() on a random forest model will return a consensus between all the trees and provide a probability and a prediction column.

Example

```
from pyspark.ml.classifiction import RandomForestClassifier
... # Prepare training and test data like you would for a decision tree model.

forest = RandomForestClassifier(numTrees= 5)
forest = forest.fit(cars_train)

forest.trees
```

PYSPARK – MACHINE LEARNING 2 Recursive/ Logistic

Ensemble 2 Gradient-Boosted Trees

Instead of multiple trees that work in parallel like in the randomforest model gradient-boosted trees work in a series. How it works is, it builds a decision tree than trains it, it tests it and then create a new decision tree model emphasizing where it originally had it wrong and do this over and over again improving the model.

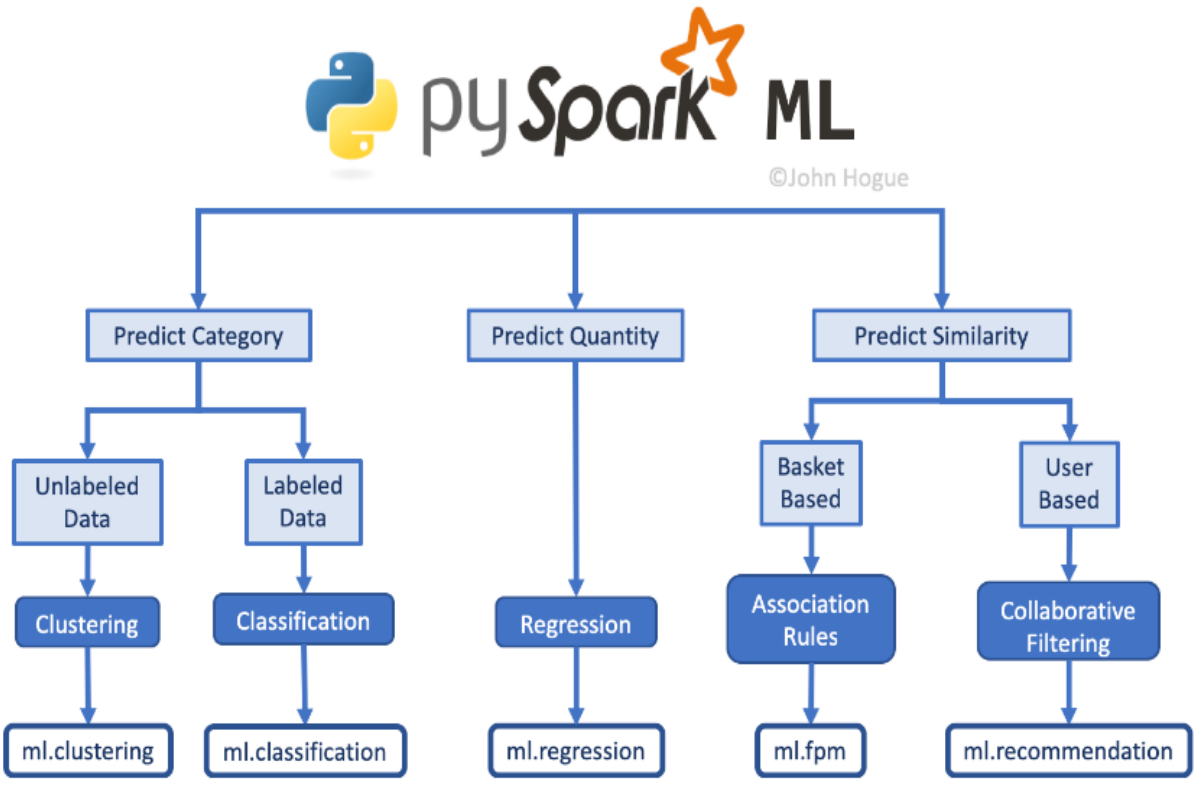
Model/ Attributes	Description
GBTClassifier	The Gradient-Boosted trees model accepts the maxIter argument of how many iterations it needs to run.

Example

```
from pyspark.ml.classification import GBTClassifier
... # Prepare training and test data like you would for a decision tree model.

gbt = GBTClassifier(maxIter=10)
gbt = gbt.fit(cars_train)
```

Choosing an Pyspark Machine Learning module

Different type of predictions and data require different module to be used.



PYSPARK – MACHINE LEARNING 3 Rec engines

Machine Learning keywords 3	
Keyword	**Description**
Recommendation engine	Think recommended section in a web shop based on what you have previously looked at and bought. There are 2 basic types of content based- and collaborative filtering. But can also be used for item grouping, dimensionality reduction and image compression.
Content-based filtering	Based on features of the items. (columns) e.g. Genre, animation, language actors etc. for movies.
Collaborative filtering	Based on user similarity. What others with similar ratings and watches have also rated high or watched that you haven't yet.
Explicit rating	For collaborative filtering. Means like actual ratings of movies by users.
Implicit Ratings	For Collaborative filtering. Means like how many of which genre/ actor/ etc. you have watched. With low score for genre, you have watched least and high score for genre you watched allot of.
latent feature	Are customers that are grouped on their behaviour, likes to create distinct groups. For collaborative filtering.

1

Data preparation for rec model (explicit)

Like for most models data should be in row based format meaning preferably non unique ids with feature column/ columns and rating of this feature /feature combination.
Also, there are no null values and all columns are only integers no strings.

Example

```
....# Piece of code that pivots columns to create just an users, movies ,ratings column and filters out the nulls instead of a column for each movie.

#Extract just the users out of the table.
user = long_ratings.select("userid").distinct()

#Create one single partition to prevent double ids.
users = users.coalesce(1)

#Create a new column containg a row number
users = users.withColumn( "userIntId", monotonically_increasing_id()).persist()

...# Same thing but with movie ids.

# Join the tables.
ratings_w_int_ids = long_ratings.join(users, "userId", "left").join(movies, "title", "left")

# Select just the int fields.
ratings_dat = rating_w_int_ids.select(
col("userIntId").alias("userid"),
col("titleId").alias("title"),
col("rating"))
```

1

Data preparation for rec model (implicit)

Implicit readings aren't provided by users directly but are taken from their habits (e.g. count of listening to a certain song). ALS needs to know which songs are played and not played by the user.

Example

```
users = ratings.select("userId").distinct()
songs = ratings.select("songId").distinct()

#Extract unique user and song ids combo through.
#Join the original numplays back.
# Fill the un played songs with 0.
cross_join = users.crossJoin(songs) .join(ratings, ["userId", "songId", "left").fillna(0)
```

PYSPARK – MACHINE LEARNING 3 Rec engines

2 2

Recommendation ALS model	
Argument	**Description**
userCol	Name of column that contains user ids.
itemCol	Name o column that contains item ids.
ratingCol	Name of column that contains the ratings.
rank	The amount of groups users can be grouped into (groups of similarities) (The most optimal you can find using ParamGridBuilder() and CrossValidator(setting the estimator to "als")).
maxIter	Number of iterations adjusting the values to reduce RMSE (the more the longer it will take to complete but the lower the less reliable the model is. (The most optimal you can find using ParamGridBuilder() and CrossValidator(setting the estimator to "als")).
regPara	If 0 all features are used for the prediction if > 0 only the important features are used preventing overfitting. (The most optimal you can find using ParamGridBuilder() and CrossValidator(setting the estimator to "als")).
alpha	Only used for implicit ratings, and tells how much each view of a genre/song should add to the models confidence if the user likes that genre/song. (The most optimal you can find using ParamGridBuilder() and CrossValidator(setting the estimator to "als")).
nonnegative	True indicating there are no negative ratings possible
coldStartStrategy	If set to "drop" it will learn from those users who have rows in both training and test set.
implicitPrefs	Indicating with true or false if the ratings are implicit or explicit ratings.

2

Example recommendation ALS model (explicit)

```
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
...#Formatting the data and creating a training, test set.

als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating", rank=25, nonnegative= True,
coldStartStrategy = "drop", implicitPrefs= False)


paramgrid = ParamGridBuilder().addGrid(als.rank[5, 40,80,120]).addGrid(als.maxIter, [5, 100,
250, 500]).addGrid(als.regParam, [0.05, .1, 1.5]). Build()

evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
predictionCol="prediction")
cv =CrossValidator(estimator = als, estimatorParamMaps = param_grid, evaluator = evaluator,
numFolds=5)

model = cv.fit(training)
best_model = model.bestModel


prediction = best_model.transform(test)
rmse = evaluator.evaluate(prediction)
# Use print best_model.rank to get the rank, best_model.java_obj.parent().getMaxIter() to get the
maxIter, best_model.java_obj.parent().getRegParam() to get the regPara.
```

PYSPARK – MACHINE LEARNING 3 Rec engines

2

Example recommendation ALS model (implicit)

```
from pyspark.ml.recommendation import ALS
...#Formatting the data and creating a training, test set.

ranks = [10, 20 ,30, 40]
maxIters = [10, 20, 30, 40]
regParams = [.05, .1, .15]
alphas = [20, 40, 60, 80]
model_list = []

#As there is no crossvalidator and bestModel for implicit ALS models it has to be done by hand.
for r in ranks
  for mi in maxIters:
    for rp in regParams:
      for a in alphas:
        model_list.append(ALS(userCol= "userId", itemCol= "songId", ratingCol= "num_plays", rank = r,
        maxIter = mi, regParam = rp, alpha= a, coldStartStrategy="drop", nonnegative=True,
        implicitPrefs= True))

#Find the best model by fitting the training data too each.
for model in model_list:
  trained_model = model.fit(train)
  predicitions = trained_model.transform(test)

  #ROEM is not a function code you can find on github. This can then be checked manually find the best.
  ROEM(predictions)
```

3

recommendForAllUsers()

Generates the top int recommendations for all users. It returns 2 columns the id and recommendation with the recommendation being a long list which needs to be exploded. This list also contains already rated records so these need to be filtered out.

Example

```
# Generate top 100 recommendation for all users.
ALS_recommendation = recommendForAllusers(100)

#Add the table to a temp table so an .sql can be used on it.
Als_recommendation.registerTempTable("ALS_recs_temp")

#Explode the table and retrieve the non hexed (strings) back.
clean_recs = spark.sql("SELECT userId, movieIds_and_ratings.movieId AS movieId,
movieIds_and_ratings.rating AS prediction
FROM ALS_recs_temp
LATERAL VIEW EXPLODE(recommendations) exploded_table AS movieIds_and_ratings")

clean_recs.join(movie_info, ["movieId"], "left")

# To remove the movies already watched by the users join the rated table to see which remain null.
clean_recs.join(movie_ratigns, ["userId", "movieId"], "left").filter(movie_ratigns.rating.isNull()).show()
```

PYSPARK – MACHINE LEARNING 4 Examples

Machine Learning cs keywords		
Keyword	**Description**	**Imports**
Collaborative filtering	Produce recommendations.	from pyspark.mllib.recommendation import ALS #Import ALS Alternating least squares.
Classification	Identifying to which of a set of categories a new observation belongs.	from pyspark.mllib.classification import LogisticRegressionWithLBFGS #Imports binary classification function.
Clustering	Groups data based on similar characteristics.	from pyspark.mllib.clustering import KMeans #Imports pyspark-dot-mllib-dot-clustering submodule.

Collaborative filtering	
Collaborative filtering is a method of making automatic predictions about the interest of a user by finding users that share common interest and is used for e.g. recommend section of a webshop.	
Approach	**Description**
User-User Collaborative filtering	Finds users that are similar to the target user.
Item-Item Collaborative filtering	Finds and recommends items that are similar to items with the target user.
Building a recommendation system	
Steps	**Description**
Rating class	A constructor that is useful for parsing the rdd and creating a tuple of user, product, rating.
Alternating least Squares using ALS.train()	Is An Algorithm training method that helps to find products that customers might like based on their previous purchases on ratings. ALS-dot-train method requires rating objects represented as (userid, itemid, rating) tuples along training parameters rank and iterations. With rank representing number of features and iteration the number of iterations to run the computation.
Predicting ratings using predictAll()	A method returns a list of predict ratings for input user and product pair.
Model evaluation using MSE	The Mean Squared Error measures the average of the squares of the errors what is estimated and existing data.

Example

```
from pysarpk.mllib.recommendation import Rating
#Rating class.
r = Rating(user = 1, product = 2, rating = 5.0) ... # also create (r1 r2 r3)
ratings = sc.parallelize([r, r1, r2, r3])
#Alternating lest squares.
model = ALS.train(training, rank=10, iterations=10)
#Predicitng ratings.
unrated_rdd = sc.parallelize([(1, 2)]) # can be a list tuples
predictions = model.predictAll(unrated_rdd) # [rating(user=1, product=1, rating = 1.00)]
#Model evaluation.
rates = ratings.map(lambda x: ((x[0], x[1]), x[2]))
preds = predictions.map(lambda x: ((x[0], x[1]), x[2]))
rates_preds = rates.join(preds)
MSE = rates_preds.map(lambda r: (r[1][0] – r[1][1])**2).mean()
```

Collaborative filtering Example

An example of a simple movie recommendation system. Using the collaborative filtering model.

Load the MovieLens data (ratings.csv) into RDD.

```
# Load the data into RDD.
data = sc.textFile(file_path)

# Split the RDD.
ratings = data.map(lambda l: l.split(','))

# Transform the ratings RDD.
ratings_final = ratings.map(lambda line: Rating(int(line[0]), int(line[1]), float(line[2])))

# Split the data into training and test.
training_data, test_data = ratings_final.randomSplit([0.8, 0.2])
```

Model training and predictions.

```
# Create the ALS model on the training data.
model = ALS.train(training_data, rank=10, iterations=10)

# Drop the ratings column.
testdata_no_rating = test_data.map(lambda p: (p[0], p[1]))

# Predict the model.
predictions = model.predictAll(testdata_no_rating)
```

Model evaluation using MSE.

```
# Prepare ratings data.
rates = ratings_final.map(lambda r: ((r[0], r[1]), r[2]))

# Prepare predictions data.
preds = predictions.map(lambda r: ((r[0], r[1]), r[2]))

# Join the ratings data with predictions data.
rates_and_preds = rates.join(preds)

# Calculate and print MSE.
MSE = rates_and_preds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
print("Mean Squared Error of the model for the test data = {:.2f}".format(MSE))

#Output:
Mean Squared Error of the model for the test data = 1.35.
```

PYSPARK – MACHINE LEARNING 4 Examples

Classification

Is a machine learning algorithm that identifies which category an item belongs to. E.g. whether an email is spam or non-spam based on labelled examples.

Approach	Description
binary classification	Classify entities into two distinct categories.
multi-cass classification	Classify entities into more than two entities.

Building a logistic regression machine learning method

Logistic regression predicts a binary response. It measures the relationship between the label and features.

Vectors

A vector is what is provided to the ai to classify, often in the form of a list of values e.g. (send address, keywords, time of receiving , etc of an email) but in an array of floats form (thus encoded).
A dense vector stores the entire list no matter the size.
A sparse vector stores only positive values and in which position they are.

Method	Example
Vectors.dense()	dense = *Vectors.dense*([1.0, 2.0] #[1.0, 2.0]
Vectors.sparse()	sparse = *Vectors.sparse*(4, {1: 1.0, 3: 5.5}) #(4, {1: 1.0, 3: 5.5})

LabeledPoint

Is what you want the ai the predict e.g. (is it spam) needs to be provided in the training data. For binary classification a label is either 0(negative or 1 positive).

Example

positive = *LabeledPoint*(1.0, [1.0, 0.0, 3.0]) #1.0 = positive, the [] is the list of vectors.
negative = *LabeledPoint*(0.0 [2.0, 1.0, 1.0) #0.0 = negative, the [] is the list of vectors.

Method	Description
HashingTF()	Is a labelling machine it accepts an integer indicating how long the vector list can get (thus in how many groups it can group it) and groups the words provided too it using the .transform() function.

Example

```
from pyspark.mllib.feature import HashingTF
sentence = "hello hello world"
words = sentence.split()
tf = HashingTF(10000)
tf.transform(words) #Output is sparseVector(10000, {3065: 1.0, 6861:2.0}) indicating that world is in group 3065 and the two hellos are in group 6861.
```

LogisticRegressionWithLBFGS

The minimum requirements for LogisticRegressionWithLBFGS is a rdd of labeledPoints.
This can then be loaded into the module using .train().
The module can then be used using .predict() supplying a vector (don't forget to encode).

Example

```
data = [LabeledPoint(0.0,[0.0, 1.0]), LabeledPoint(1.0, [1.0, 0.0]),]
rdd = sc.parallelize(data)

#Train the module.
lrm = LogisticRegressionWithLBFGS.train(RDD)

#Supply a vector to make a prediction.
#lrm.predict([1.0, 0.0])
```

PYSPARK – MACHINE LEARNING 4 Examples

Classification Example

An example of a model deciding if an email is spam or not spam.

Loading spam and non-spam data

```
# Load the datasets into RDDs.
spam_rdd = sc.textFile(file_path_spam)
non_spam_rdd = sc.textFile(file_path_non_spam)

# Split the email messages into words.
spam_words = spam_rdd.flatMap(lambda email: email.split(' '))
non_spam_words = non_spam_rdd.flatMap(lambda email: email.split(' '))
```

Feature hashing and LabelPoint

```
# Create a HashingTF instance with 200 features.
tf = HashingTF(numFeatures=200)

# Map each word to one feature (which will be used to decided whether a message is spam or non spam).
spam_features = tf.transform(spam_words)
non_spam_features = tf.transform(non_spam_words)

# Label the features: 1 for spam, 0 for non-spam.
spam_samples = spam_features.map(lambda features:LabeledPoint(1, features))
non_spam_samples = non_spam_features.map(lambda features:LabeledPoint(0, features))

# Combine the two datasets.
samples = spam_samples.union(non_spam_samples)
```

Logistic Regression model training

```
# Split the data into training and testing.
train_samples,test_samples = samples.randomSplit([0.8, 0.2])

# Train the model.
model = LogisticRegressionWithLBFGS.train(train_samples)

# Create a prediction label from the test data.
predictions = model.predict(test_samples.map(lambda x: x.features))

# Combine original labels with the predicted labels
labels_and_preds = test_samples.map(lambda x: x.label).zip(predictions)

# Check the accuracy of the model on the test data.
accuracy = labels_and_preds.filter(lambda x: x[0] == x[1]).count() / float(test_samples.count())
print("Model accuracy : {:.2f}".format(accuracy))

#Output Model Accuracy 0.76.
```

PYSPARK – MACHINE LEARNING 4 Examples

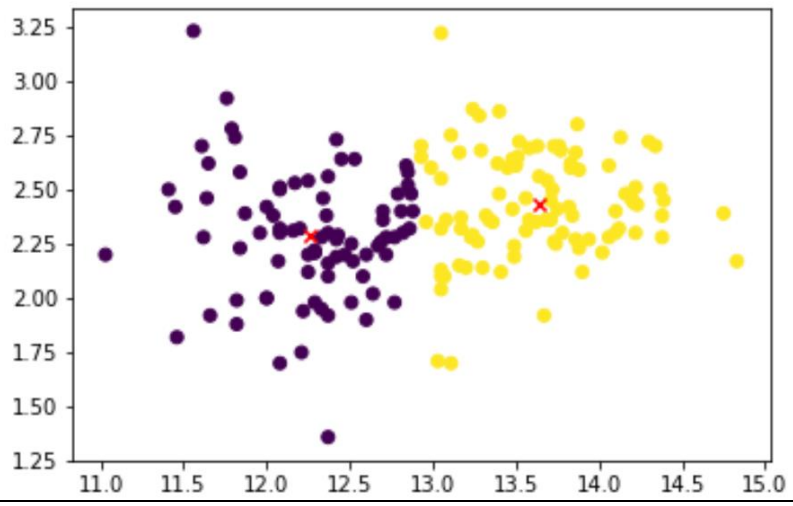
Clustering

An unsupervised learning method to group unlabelled data together.

Approach	Description
K-means	Algorithm that through a series of iterations creates clusters of the provided data. Requires that the data is a set of numerical features and a variable indicating the number of groups the algorithm should make.
Steps	**Example**
Load data into an rdd.	rdd= sc.textFile("WineData.csv").map(lambda x: x.split(",")).map(lambda x: [float(x[0]), float(x[1])])
Train the k-means. Using KMeans.train() accepting the rdd , k (meaning the number of clusters) and maxIterations.	from pyspark.mllib.clustering import KMeans model = KMeans.train(RDD, k = 2, maxIterations = 10) model.clusterCenteres

Evaluating the model.	from math import sqrt
	def error(point):
	  center = model.centers[model.predict(point)]
	  return sqrt(sum([x**2 for x in (point – center)]))
	
	WSSE = rdd.map(lambda point: error(point)).reduce(lambda x, y : x + y)
	prints("within set sum of squared error =" + str(WSSE))

(Optional) visualizing k-means clusters.	#Create a pandas DataFrame.
	wine_data_df = spark.createDataFrame(rdd, schema=["col1", "col2"])
	wine_data_df_pandas = wine_data_df.toPandas()
	
	#Create the centers.
	cluster_centers_pandas = pd.DataFrame(model.clusterCenters, columns=["col1", "col2"])
	
	#Create the scatter plot.
	plt.scatter(wine_data_df_pandas["col1"], wine_data_df_pandas["col2"]);
	plts.scatter(cluster_centers_pandas["col1"],
	cluster_centers_pandas["col2"], color="red", marker="x")



PYSPARK – MACHINE LEARNING 4 Examples

Cluster Example

Unlike the supervised tasks, where data is labelled, clustering can be used to make sense of unlabelled data. This is an example of finding out how many clusters are there in a dataset containing 5000 rows and 2 columns.

Load the data into an RDD.

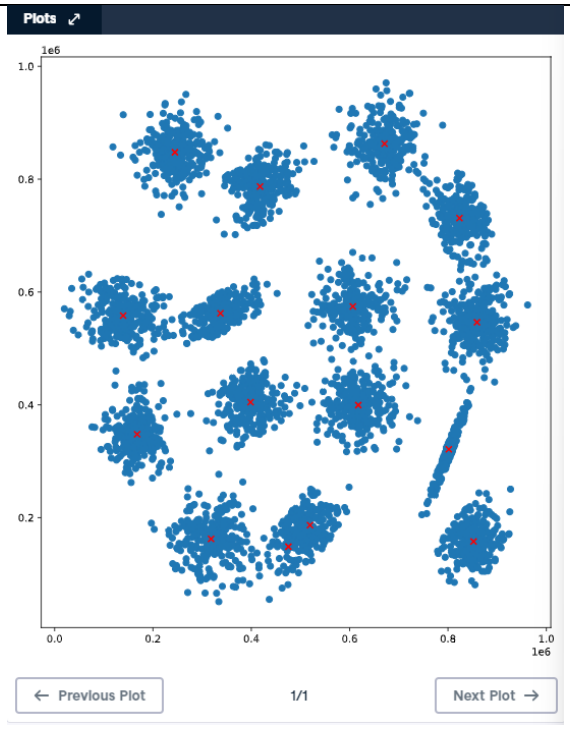
```
# Load the dataset into an RDD.
clusterRDD = sc.textFile(file_path)

# Split the RDD based on tab.
rdd_split = clusterRDD.map(lambda x: x.split("\t"))

# Transform the split RDD by creating a list of integers.
rdd_split_int = rdd_split.map(lambda x: [int(x[0]), int(x[1])])
```

K-means training.

```
# Train the model with clusters from 13 to 16 and compute WSSSE.
for clst in range(13, 17):
    model = KMeans.train(rdd_split_int, clst, seed=1)
    WSSSE = rdd_split_int.map(lambda point: error(point)).reduce(lambda x, y: x + y)
    print("The cluster {} has Within Set Sum of Squared Error {}".format(clst, WSSSE))

# Train the model again with the best k.
model = KMeans.train(rdd_split_int, k=16, seed=1)

# Get cluster center.
cluster_centers = model.clusterCenters
```

Visualizing clusters.

```
# Convert rdd_split_int RDD into Spark DataFrame and then to Pandas DataFrame.
rdd_split_int_df_pandas =
spark.createDataFrame(rdd_split_int, schema=["col1", "col2"]).toPandas()

# Convert cluster_centers to a pandas DataFrame.
cluster_centers_pandas =
pd.DataFrame(cluster_centers, columns=["col1", "col2"])

# Create an overlaid scatter plot of clusters and centroids.
plt.scatter(rdd_split_int_df_pandas["col1"],
rdd_split_int_df_pandas["col2"])
plt.scatter(cluster_centers_pandas["col1"],
cluster_centers_pandas["col2"], color="red",
marker="x")
plt.show()
```



PYSPARK – MACHINE LEARNING 4 Examples

Interpreting a model

To interpret what columns and data is most important and affects the prediction the most a simple query can be used using .featureImportances() attribute.

Attribute	Descriptions
.featureImportances()	Returns a string containing the features and their importance according to the model.

Example

```
import pandas as pd

#Convert feature importances to a pandas column.
fi_df = pd.DataFrame(model.featureImportances.toArray(), columns=["importance"])

#Convert list of feature names to pandas column.
fi_df["feature"] = pd.Series(feature_cols)

#Sort the data based on feature importance.
fi_df.sort_values(by=["importance"], ascending=False, inplace=True)

#Show top 9.
model_df.head(9)
```

Saving a model

To save a model the attribute .save() can be used.

Attribute	Descriptions
.save()	Saves a model accepts one argument being the path to save it as, including the final directory.

Example

```
model.save("save_as_name")
```

Loading a model

To load a model the attribute .load() can be used, combined with the type of model your importing.

Attribute	Descriptions
.load()	Imports and loads a model into pyspark. Accepts as only argument the directory of the model your trying to import.

Example

```
from pyspark.ml.regression import RandomForestRegressionModel

model = RandomForestRegressionModel.load("rfr_real_estate_model"
```