Fundamentals of Big Data

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

What is Big Data?

- Big data is a term used to refer to the study and applications of data sets that are too complex for traditional data-processing software - **Wikipedia**

The 3 V's of Big Data

- Volume, Variety and Velocity
- **Volume**: Size of the data
- **Variety**: Different sources and formats
- **Velocity**: Speed of the data

Big Data concepts and Terminology

- **Clustered computing**: Collection of resources of multiple machines
- **Parallel computing**: Simultaneous computation
- **Distributed computing**: Collection of nodes (networked computers) 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

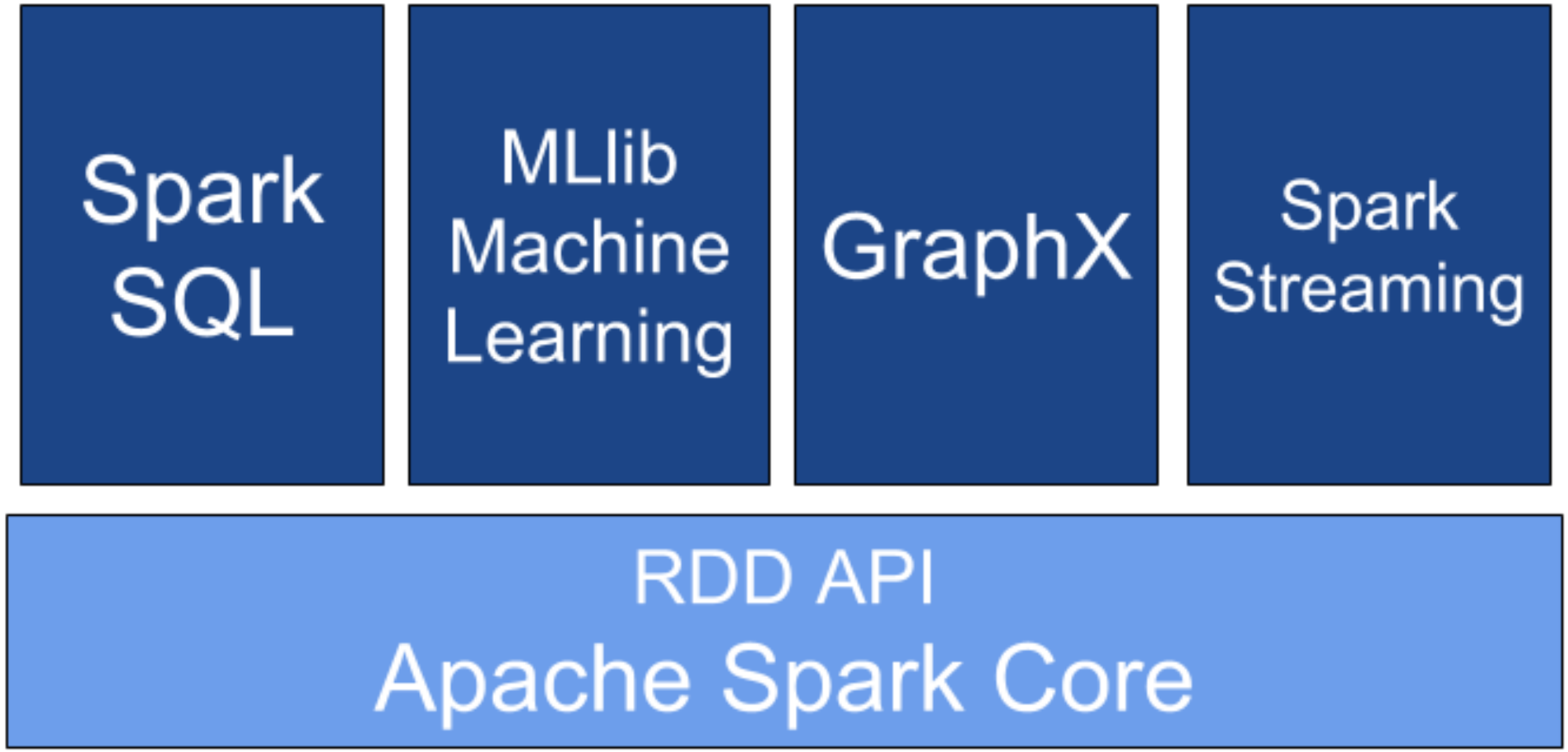
Big Data processing systems

- **Hadoop/MapReduce:** Scalable and fault tolerant framework written in Java
  - Open source
  - Batch processing
- **Apache Spark:** General purpose and lightning fast cluster computing system
  - Open source
  - Both batch and real-time data processing

Features of Apache Spark framework

- Distributed cluster computing framework
- Efficient in-memory computations for large data sets
- Lightning fast data processing framework
- Provides support for Java, Scala, Python, R and SQL

Apache Spark Components



Spark modes of deployment

- **Local mode:** Single machine such as your laptop
  - Local model convenient for testing, debugging and demonstration
- **Cluster mode:** Set of pre-defined machines
  - Good for production
- Workflow: Local -> clusters
- No code change necessary

PySpark: Spark with Python

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

Overview of PySpark

- Apache Spark is written in Scala
- To support Python with Spark, Apache Spark Community released PySpark
- Similar computation speed and power as Scala
- PySpark APIs are similar to Pandas and Scikit-learn

What is Spark shell?

- Interactive environment for running Spark jobs
- Helpful for fast interactive prototyping
- Spark's shells allow interacting with data on disk or in memory
- Three different Spark shells:
  - Spark-shell for Scala
  - PySpark-shell for Python
  - SparkR for R

PySpark shell

- PySpark shell is the Python-based command line tool
- PySpark shell allows data scientists interface with Spark data structures
- PySpark shell support connecting to a cluster

Understanding SparkContext

- SparkContext is an entry point into the world of Spark
- An entry point is a way of connecting to Spark cluster
- An entry point is like a key to the house
- PySpark has a default SparkContext called `sc`

Inspecting SparkContext

- **Version:** To retrieve SparkContext version

```
sc.version
```

```
2.3.1
```

- **Python Version:** To retrieve Python version of SparkContext

```
sc.pythonVer
```

```
3.6
```

- **Master:** URL of the cluster or “local” string to run in local mode of SparkContext

```
sc.master
```

```
local[*]
```

Loading data in PySpark

- SparkContext's `parallelize()` method

```
rdd = sc.parallelize([1,2,3,4,5])
```

- SparkContext's `textFile()` method

```
rdd2 = sc.textFile("test.txt")
```

Use of Lambda function in python - filter()

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

What are anonymous functions in Python?

- Lambda functions are anonymous functions in Python
- Very powerful and used in Python. Quite efficient with `map()` and `filter()`
- Lambda functions create functions to be called later similar to `def`
- It returns the functions without any name (i.e anonymous)
- Inline a function definition or to defer execution of a code

Lambda function syntax

- The general form of lambda functions is

```
lambda arguments: expression
```

- Example of lambda function

```
double = lambda x: x * 2
print(double(3))
```

```
6
```

Difference between def vs lambda functions

- Python code to illustrate cube of a number

```
def cube(x):
     return x ** 3
g = lambda x: x ** 3
print(g(10))
print(cube(10))
```

```
1000
1000
```

- No return statement for lambda
- Can put lambda function anywhere

Use of Lambda function in Python - map()

- map() function takes a function and a list and returns a new list which contains items returned by that function for each item
- General syntax of map()

```
map(function, list)
```

- Example of map()

```
items = [1, 2, 3, 4]
list(map(lambda x: x + 2 , items))
```

```
[3, 4, 5, 6]
```

Use of Lambda function in python - filter()

- filter() function takes a function and a list and returns a new list for which the function evaluates as true
- General syntax of filter()

```
filter(function, list)
```

- Example of filter()

```
items = [1, 2, 3, 4]
list(filter(lambda x: (x%2 != 0), items))
```

```
[1, 3]
```

Introduction to PySpark RDD

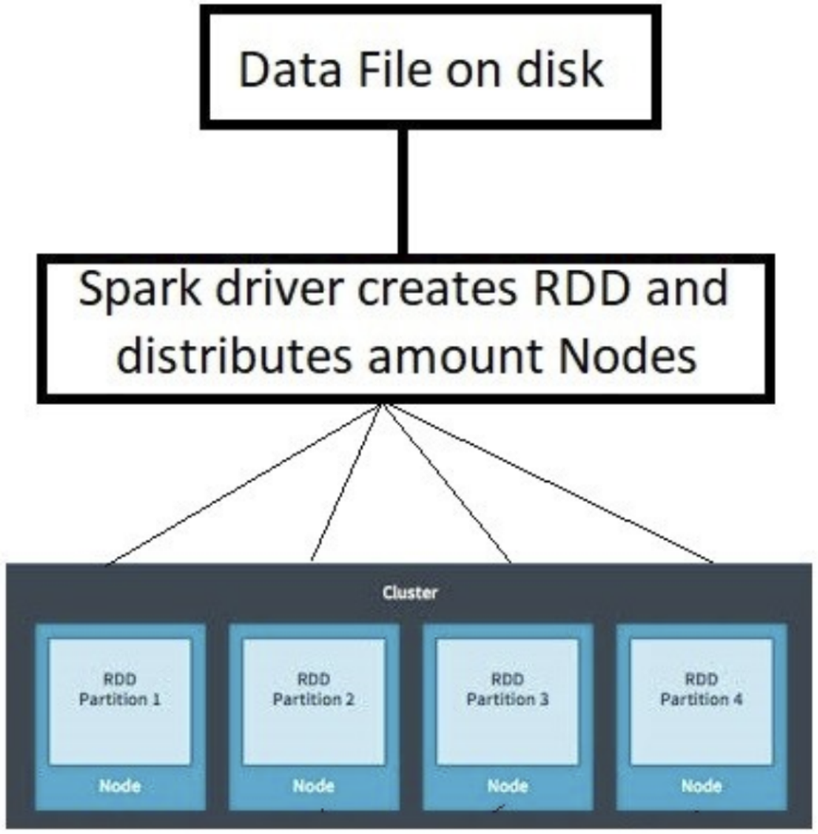
BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

What is RDD?

- RDD = **R**esilient **D**istributed **D**atasets



Decomposing RDDs

- Resilient Distributed Datasets
  - Resilient: Ability to withstand failures
  - Distributed: Spanning across multiple machines
  - Datasets: Collection of partitioned data e.g, Arrays, Tables, Tuples etc.,

Creating RDDs. How to do it?

- Parallelizing an existing collection of objects
- External datasets:
  - Files in HDFS
  - Objects in Amazon S3 bucket
  - lines in a text file
- From existing RDDs

Parallelized collection (parallelizing)

- `parallelize()` for creating RDDs from python lists

```
numRDD = sc.parallelize([1,2,3,4])
```

```
helloRDD = sc.parallelize("Hello world")
```

```
type(helloRDD)
```

```
<class 'pyspark.rdd.PipelinedRDD'>
```

From external datasets

- `textFile()` for creating RDDs from external datasets

```
fileRDD = sc.textFile("README.md")
```

```
type(fileRDD)
```

```
<class 'pyspark.rdd.PipelinedRDD'>
```

Understanding Partitioning in PySpark

- A partition is a logical division of a large distributed data set
- `parallelize()` method

```
numRDD = sc.parallelize(range(10), minPartitions = 6)
```

- `textFile()` method

```
fileRDD = sc.textFile("README.md", minPartitions = 6)
```

- The number of partitions in an RDD can be found by using `getNumPartitions()` method

RDD operations in PySpark

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

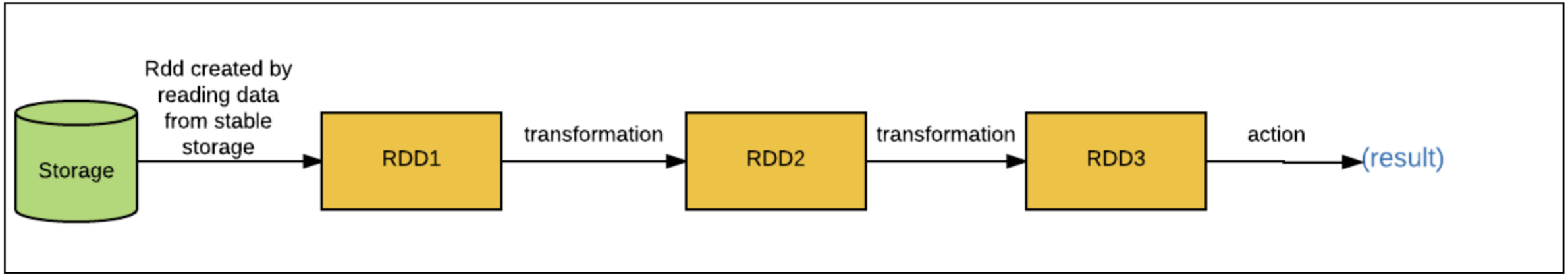
Overview of PySpark operations

Spark Operations =  TRANSFORMATIONS +  ACTIONS

- Transformations create new RDDS
- Actions perform computation on the RDDs

RDD Transformations

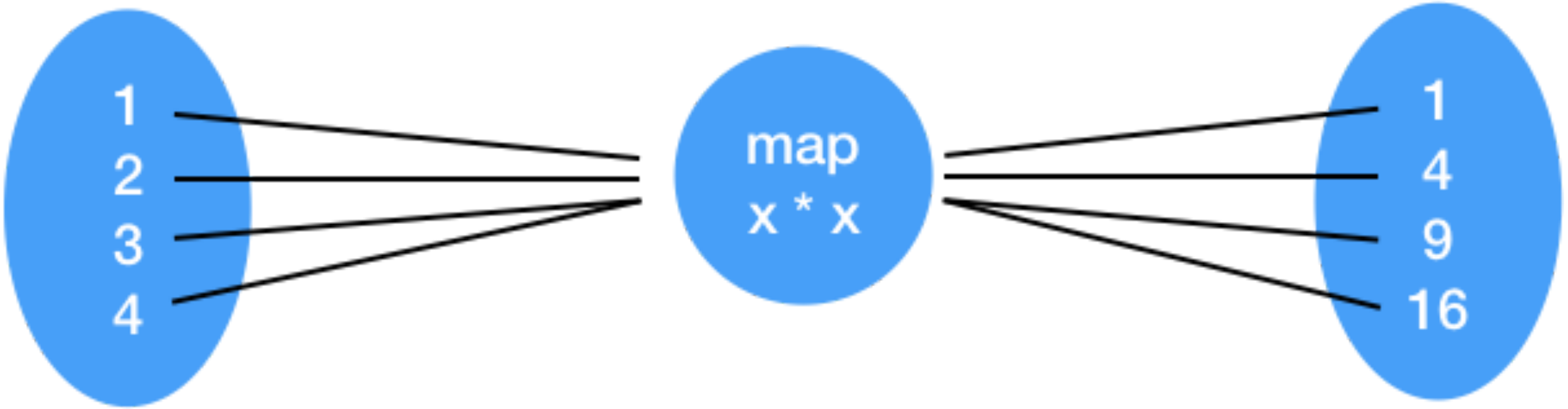
- Transformations follow Lazy evaluation



- Basic RDD Transformations
  - `map()`, `filter()`, `flatMap()`, and `union()`

map() Transformation

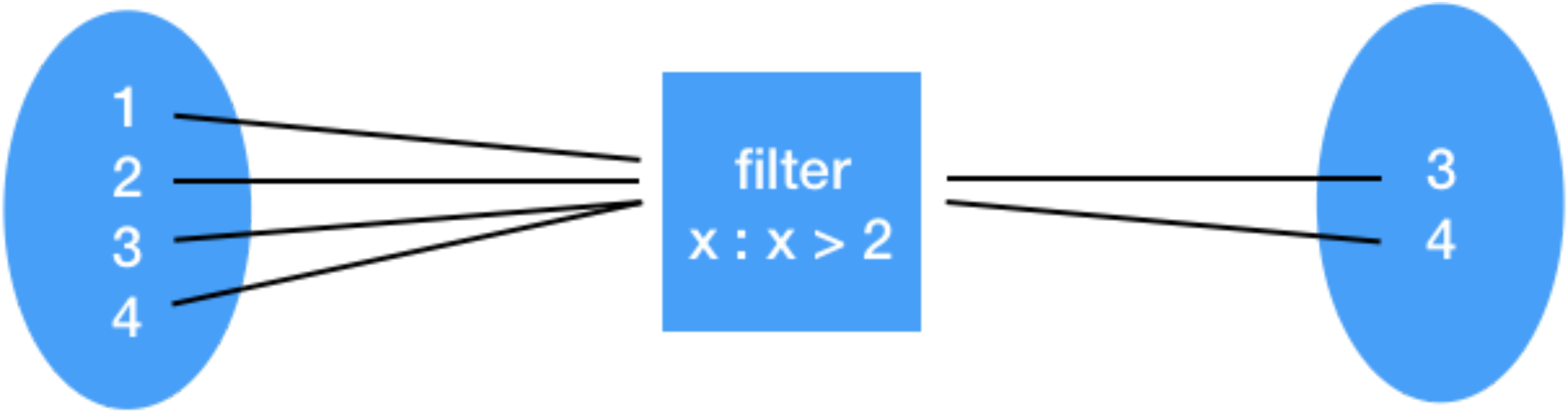
- map() transformation applies a function to all elements in the RDD




```
RDD = sc.parallelize([1,2,3,4])
RDD_map = RDD.map(lambda x: x * x)
```

filter() Transformation

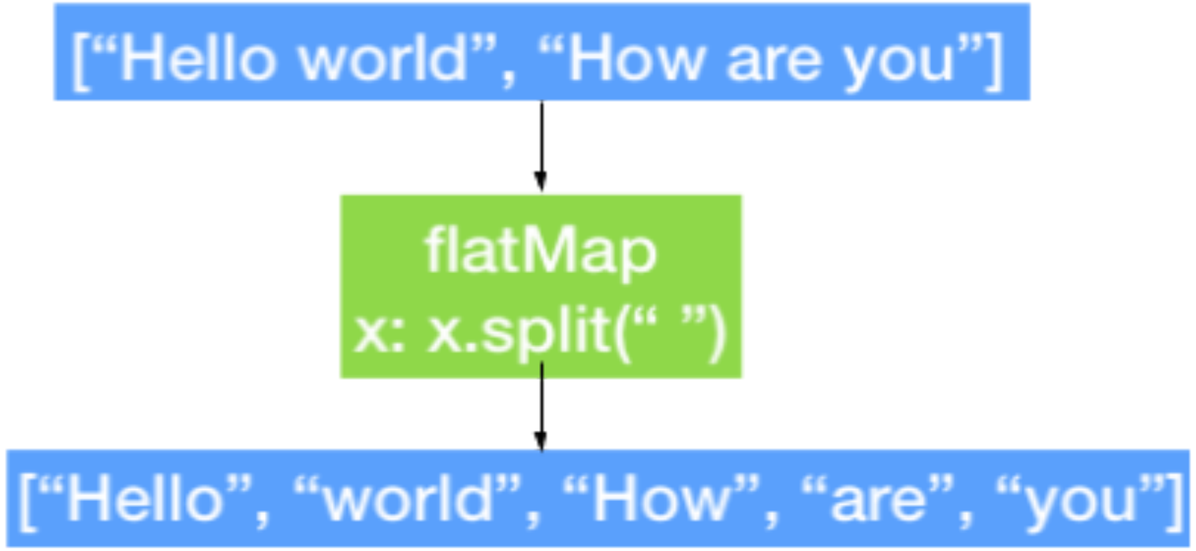
- Filter transformation returns a new RDD with only the elements that pass the condition




```
RDD = sc.parallelize([1,2,3,4])
RDD_filter = RDD.filter(lambda x: x > 2)
```

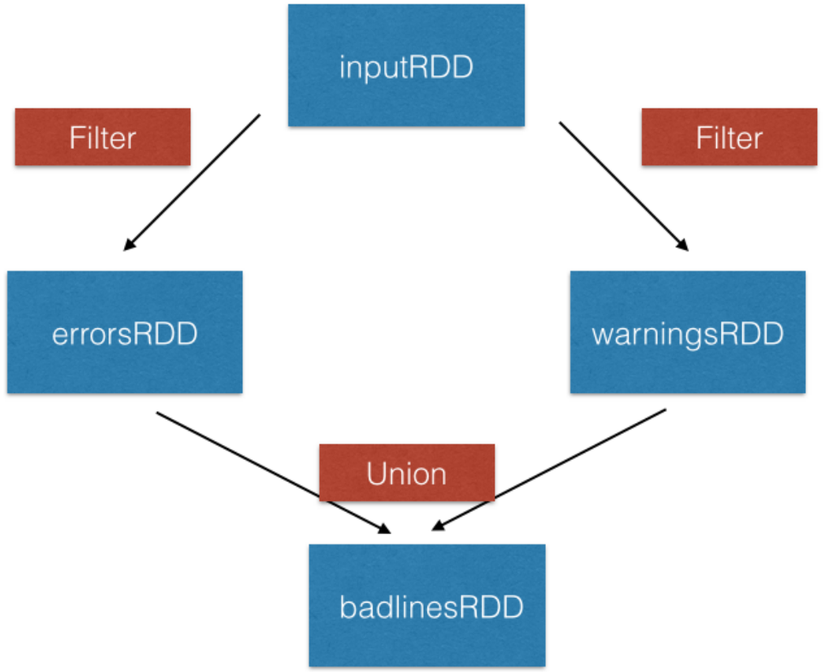
flatMap() Transformation

- flatMap() transformation returns multiple values for each element in the original RDD




```
RDD = sc.parallelize(["hello world", "how are you"])
RDD_flatmap = RDD.flatMap(lambda x: x.split(" "))
```

union() Transformation




```
inputRDD = sc.textFile("logs.txt")
errorRDD = inputRDD.filter(lambda x: "error" in x.split())
warningsRDD = inputRDD.filter(lambda x: "warnings" in x.split())
combinedRDD = errorRDD.union(warningsRDD)
```

RDD Actions

- Operation return a value after running a computation on the RDD
- Basic RDD Actions
  - collect()
  - take(N)
  - first()
  - count()

collect() and take() Actions

- collect() return all the elements of the dataset as an array
- take(N) returns an array with the first N elements of the dataset

```
RDD_map.collect()
```

```
[1, 4, 9, 16]
```

```
RDD_map.take(2)
```

```
[1, 4]
```

first() and count() Actions

- first() prints the first element of the RDD

```
RDD_map.first()
```

```
[1]
```

- count() return the number of elements in the RDD

```
RDD_flatmap.count()
```

```
5
```

Working with Pair RDDs in PySpark

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

Introduction to pair RDDs in PySpark

- Real life datasets are usually key/value pairs
- Each row is a key and maps to one or more values
- Pair RDD is a special data structure to work with this kind of datasets
- Pair RDD: Key is the identifier and value is data

Creating pair RDDs

- Two common ways to create pair RDDs
  - From a list of key-value tuple
  - From a regular RDD
- Get the data into key/value form for paired RDD

```
my_tuple = [('Sam', 23), ('Mary', 34), ('Peter', 25)]
pairRDD_tuple = sc.parallelize(my_tuple)
```

```
my_list = ['Sam 23', 'Mary 34', 'Peter 25']
regularRDD = sc.parallelize(my_list)
pairRDD_RDD = regularRDD.map(lambda s: (s.split(' ')[0], s.split(' ')[1]))
```

Transformations on pair RDDs

- All regular transformations work on pair RDD
- Have to pass functions that operate on key value pairs rather than on individual elements
- Examples of paired RDD Transformations
  - reduceByKey(func): Combine values with the same key
  - groupByKey(): Group values with the same key
  - sortByKey(): Return an RDD sorted by the key
  - join(): Join two pair RDDs based on their key

reduceByKey() transformation

- `reduceByKey()` transformation combines values with the same key
- It runs parallel operations for each key in the dataset
- It is a transformation and not action

```
regularRDD = sc.parallelize([("Messi", 23), ("Ronaldo", 34),
                             ("Neymar", 22), ("Messi", 24)])
pairRDD_reducebykey = regularRDD.reduceByKey(lambda x,y : x + y)
pairRDD_reducebykey.collect()
[('Neymar', 22), ('Ronaldo', 34), ('Messi', 47)]
```

sortByKey() transformation

- `sortByKey()` operation orders pair RDD by key
- It returns an RDD sorted by key in ascending or descending order

```
pairRDD_reducebykey_rev = pairRDD_reducebykey.map(lambda x: (x[1], x[0]))
pairRDD_reducebykey_rev.sortByKey(ascending=False).collect()
[(47, 'Messi'), (34, 'Ronaldo'), (22, 'Neymar')]
```

groupByKey() transformation

- `groupByKey()` groups all the values with the same key in the pair RDD

```
airports = [("US", "JFK"),("UK", "LHR"),("FR", "CDG"),("US", "SFO")]
regularRDD = sc.parallelize(airports)
pairRDD_group = regularRDD.groupByKey().collect()
for cont, air in pairRDD_group:
  print(cont, list(air))
FR ['CDG']
US ['JFK', 'SFO']
UK ['LHR']
```

join() transformation

- `join()` transformation joins the two pair RDDs based on their key

```
RDD1 = sc.parallelize([("Messi", 34),("Ronaldo", 32),("Neymar", 24)])
RDD2 = sc.parallelize([("Ronaldo", 80),("Neymar", 120),("Messi", 100)])
```

```
RDD1.join(RDD2).collect()
[('Neymar', (24, 120)), ('Ronaldo', (32, 80)), ('Messi', (34, 100))]
```

reduce() action

- reduce(func) action is used for aggregating the elements of a regular RDD
- The function should be commutative (changing the order of the operands does not change the result) and associative
- An example of `reduce()` action in PySpark

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

```
14
```

saveAsTextFile() action

- `saveAsTextFile()` action saves RDD into a text file inside a directory with each partition as a separate file

```
RDD.saveAsTextFile("tempFile")
```

- `coalesce()` method can be used to save RDD as a single text file

```
RDD.coalesce(1).saveAsTextFile("tempFile")
```

Action Operations on pair RDDs

- RDD actions available for PySpark pair RDDs
- Pair RDD actions leverage the key-value data
- Few examples of pair RDD actions include
  - `countByKey()`
  - `collectAsMap()`

countByKey() action

- `countByKey()` only available for type (K, V)
- `countByKey()` action counts the number of elements for each key
- Example of `countByKey()` on a simple list

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

```
('a', 2)
('b', 1)
```

collectAsMap() action

- `collectAsMap()` return the key-value pairs in the RDD as a dictionary
- Example of `collectAsMap()` on a simple tuple

```
sc.parallelize([(1, 2), (3, 4)]).collectAsMap()
```

```
{1: 2, 3: 4}
```

Introduction to PySpark DataFrames

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

What are PySpark DataFrames?

- PySpark SQL is a Spark library for structured data. It provides more information about the structure of data and computation
- PySpark DataFrame is an immutable distributed collection of data with named columns
- Designed for processing both structured (e.g relational database) and semi-structured data (e.g JSON)
- Dataframe API is available in Python, R, Scala, and Java
- DataFrames in PySpark support both SQL queries (`SELECT * from table`) or expression methods (`df.select()`)

SparkSession - Entry point for DataFrame API

- SparkContext is the main entry point for creating RDDs
- SparkSession provides a single point of entry to interact with Spark DataFrames
- SparkSession is used to create DataFrame, register DataFrames, execute SQL queries
- SparkSession is available in PySpark shell as `spark`

Creating DataFrames in PySpark

- Two different methods of creating DataFrames in PySpark
  - From existing RDDs using SparkSession's createDataFrame() method
  - From various data sources (CSV, JSON, TXT) using SparkSession's read method
- Schema controls the data and helps DataFrames to optimize queries
- Schema provides information about column name, type of data in the column, empty values etc.,

Create a DataFrame from RDD

```
iphones_RDD = sc.parallelize([
    ("XS", 2018, 5.65, 2.79, 6.24),
    ("XR", 2018, 5.94, 2.98, 6.84),
    ("X10", 2017, 5.65, 2.79, 6.13),
    ("8Plus", 2017, 6.23, 3.07, 7.12)
])
```

```
names = ['Model', 'Year', 'Height', 'Width', 'Weight']
```

```
iphones_df = spark.createDataFrame(iphones_RDD, schema=names)
type(iphones_df)
```

```
pyspark.sql.dataframe.DataFrame
```

Create a DataFrame from reading a CSV/JSON/TXT

```
df_csv = spark.read.csv("people.csv", header=True, inferSchema=True)
```

```
df_json = spark.read.json("people.json", header=True, inferSchema=True)
```

```
df_txt = spark.read.txt("people.txt", header=True, inferSchema=True)
```

- Path to the file and two optional parameters
- Two optional parameters
  - `header=True` , `inferSchema=True`

Interacting with PySpark DataFrames

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

DataFrame operators in PySpark

- DataFrame operations: Transformations and Actions
- DataFrame Transformations:
  - select(), filter(), groupby(), orderby(), dropDuplicates() and withColumnRenamed()
- DataFrame Actions :
  - printSchema(), head(), show(), count(), columns and describe()

  Correction: printSchema() is a method for any Spark dataset/dataframe and not an action

select() and show() operations

- `select()` transformation subsets the columns in the DataFrame

```
df_id_age = test.select('Age')
```

- `show()` action prints first 20 rows in the DataFrame

```
df_id_age.show(3)
```

```
+---+
|Age|
+---+
| 17|
| 17|
| 17|
+---+
only showing top 3 rows
```

filter() and show() operations

- `filter()` transformation filters out the rows based on a condition

```
new_df_age21 = new_df.filter(new_df.Age > 21)
new_df_age21.show(3)
```

```
+-------+------+---+
|User_ID|Gender|Age|
+-------+------+---+
|1000002|     M| 55|
|1000003|     M| 26|
|1000004|     M| 46|
+-------+------+---+
only showing top 3 rows
```

groupby() and count() operations

- `groupby()` operation can be used to group a variable

```
test_df_age_group = test_df.groupby('Age')
test_df_age_group.count().show(3)
```

```
+---+------+
|Age| count|
+---+------+
| 26|219587|
| 17|     4|
| 55| 21504|
+---+------+
only showing top 3 rows
```

orderby() Transformations

- `orderby()` operation sorts the DataFrame based on one or more columns

```
test_df_age_group.count().orderBy('Age').show(3)
```

```
+---+-----+
|Age|count|
+---+-----+
|  0|15098|
| 17|    4|
| 18|99660|
+---+-----+
only showing top 3 rows
```

dropDuplicates()

- `dropDuplicates()` removes the duplicate rows of a DataFrame

```
test_df_no_dup = test_df.select('User_ID','Gender', 'Age').dropDuplicates()
test_df_no_dup.count()
```

```
5892
```

withColumnRenamed Transformations

- `withColumnRenamed()` renames a column in the DataFrame

```
test_df_sex = test_df.withColumnRenamed('Gender', 'Sex')
test_df_sex.show(3)
```

```
+-------+---+---+
|User_ID|Sex|Age|
+-------+---+---+
|1000001|  F| 17|
|1000001|  F| 17|
|1000001|  F| 17|
+-------+---+---+
```

printSchema()

- `printSchema()` operation prints the types of columns in the DataFrame

```
test_df.printSchema()
```

```
 |-- User_ID: integer (nullable = true)
 |-- Product_ID: string (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Age: string (nullable = true)
 |-- Occupation: integer (nullable = true)
 |-- Purchase: integer (nullable = true)
```

columns actions

- `columns` operator prints the columns of a DataFrame

```
test_df.columns
```

```
['User_ID', 'Gender', 'Age']
```

describe() actions

- `describe()` operation compute summary statistics of numerical columns in the DataFrame

```
test_df.describe().show()
```

```
+-------+------------------+------+------------------+
|summary|           User_ID|Gender|               Age|
+-------+------------------+------+------------------+
|  count|            550068|550068|            550068|
|   mean|1003028.8424013031|  null|30.382052764385495|
| stddev|1727.5915855307312|  null|11.866105189533554|
|    min|           1000001|     F|                 0|
|    max|           1006040|     M|                55|
+-------+------------------+------+------------------+
```

Interacting with DataFrames using PySpark SQL

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

DataFrame API vs SQL queries

- In PySpark You can interact with SparkSQL through DataFrame API and SQL queries
- The DataFrame API provides a programmatic domain-specific language (DSL) for data
- DataFrame transformations and actions are easier to construct programmatically
- SQL queries can be concise and easier to understand and portable
- The operations on DataFrames can also be done using SQL queries

Executing SQL Queries

- The SparkSession `sql()` method executes SQL query
- `sql()` method takes a SQL statement as an argument and returns the result as DataFrame

```
df.createOrReplaceTempView("table1")
```

```
df2 = spark.sql("SELECT field1, field2 FROM table1")
df2.collect()
```

```
[Row(f1=1, f2='row1'), Row(f1=2, f2='row2'), Row(f1=3, f2='row3')]
```

SQL query to extract data

```
test_df.createOrReplaceTempView("test_table")
```

```
query = '''SELECT Product_ID FROM test_table'''
```

```
test_product_df = spark.sql(query)
test_product_df.show(5)
```

```
+----------+
|Product_ID|
+----------+
| P00069042|
| P00248942|
| P00087842|
| P00085442|
| P00285442|
+----------+
```

Summarizing and grouping data using SQL queries

```
test_df.createOrReplaceTempView("test_table")
```

```
query = '''SELECT Age, max(Purchase) FROM test_table GROUP BY Age'''
```

```
spark.sql(query).show(5)
```

```
+-----+-------------+
|  Age|max(Purchase)|
+-----+-------------+
|18-25|        23958|
|26-35|        23961|
| 0-17|        23955|
|46-50|        23960|
|51-55|        23960|
+-----+-------------+
only showing top 5 rows
```

Filtering columns using SQL queries

```
test_df.createOrReplaceTempView("test_table")
```

```
query = '''SELECT Age, Purchase, Gender FROM test_table WHERE Purchase > 20000 AND Gender == "F"'''
```

```
spark.sql(query).show(5)
```

```
+-----+--------+------+
|  Age|Purchase|Gender|
+-----+--------+------+
|36-45|   23792|     F|
|26-35|   21002|     F|
|26-35|   23595|     F|
|26-35|   23341|     F|
|46-50|   20771|     F|
+-----+--------+------+
only showing top 5 rows
```

Data Visualization in PySpark using DataFrames

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

What is Data visualization?

- Data visualization is a way of representing your data in graphs or charts
- Open source plotting tools to aid visualization in Python:
  - Matplotlib, Seaborn, Bokeh etc.,
- Plotting graphs using PySpark DataFrames is done using three methods
  - pyspark_dist_explore library
  - toPandas()
  - HandySpark library

Data Visualization using Pyspark_dist_explore

- `Pyspark_dist_explore` library provides quick insights into DataFrames
- Currently three functions available – `hist()`, `distplot()` and `pandas_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")
```

Using Pandas for plotting DataFrames

- It's easy to create charts from pandas DataFrames

```
test_df = spark.read.csv("test.csv", header=True, inferSchema=True)
```

```
test_df_sample_pandas = test_df.toPandas()
```

```
test_df_sample_pandas.hist('Age')
```

Pandas DataFrame vs PySpark DataFrame

- Pandas DataFrames are in-memory, single-server based structures and operations on PySpark run in parallel
- The result is generated as we apply any operation in Pandas whereas operations in PySpark DataFrame are lazy evaluation
- Pandas DataFrame as mutable and PySpark DataFrames are immutable
- Pandas API support more operations than PySpark Dataframe API

HandySpark method of visualization

- HandySpark is a package designed to improve PySpark user experience

```
test_df = spark.read.csv('test.csv', header=True, inferSchema=True)
```

```
hdf = test_df.toHandy()
```

```
hdf.cols["Age"].hist()
```

Overview of PySpark MLlib

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

What is PySpark MLlib?

- MLlib is a component of Apache Spark for machine learning
- Various tools provided by MLlib include:
  - ML Algorithms: collaborative filtering, classification, and clustering
  - Featurization: feature extraction, transformation, dimensionality reduction, and selection
  - Pipelines: tools for constructing, evaluating, and tuning ML Pipelines

Why PySpark MLlib?

- Scikit-learn is a popular Python library for data mining and machine learning
- Scikit-learn algorithms only work for small datasets on a single machine
- Spark's MLlib algorithms are designed for parallel processing on a cluster
- Supports languages such as Scala, Java, and R
- Provides a high-level API to build machine learning pipelines

PySpark MLlib Algorithms

- **Classification (Binary and Multiclass) and Regression**: Linear SVMs, logistic regression, decision trees, random forests, gradient-boosted trees, naive Bayes, linear least squares, Lasso, ridge regression, isotonic regression
- **Collaborative filtering**: Alternating least squares (ALS)
- **Clustering**: K-means, Gaussian mixture, Bisecting K-means and Streaming K-Means

The three C's of machine learning in PySpark MLlib

- Collaborative filtering (recommender engines): Produce recommendations
- Classification: Identifying to which of a set of categories a new observation
- Clustering: Groups data based on similar characteristics

PySpark MLlib imports

- `pyspark.mllib.recommendation`

```
from pyspark.mllib.recommendation import ALS
```

- `pyspark.mllib.classification`

```
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
```

- `pyspark.mllib.clustering`

```
from pyspark.mllib.clustering import KMeans
```

Introduction to Collaborative filtering

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

What is Collaborative filtering?

- Collaborative filtering is finding users that share common interests
- Collaborative filtering is commonly used for recommender systems
- Collaborative filtering approaches
  - **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

Rating class in pyspark.mllib.recommendation submodule

- The Rating class is a wrapper around tuple (user, product and rating)
- Useful for parsing the RDD and creating a tuple of user, product and rating

```
from pyspark.mllib.recommendation import Rating
r = Rating(user = 1, product = 2, rating = 5.0)
(r[0], r[1], r[2])
```

```
(1, 2, 5.0)
```

Splitting the data using randomSplit()

- Splitting data into training and testing sets is important for evaluating predictive modeling
- Typically a large portion of data is assigned to training compared to testing data
- PySpark's `randomSplit()` method randomly splits with the provided weights and returns multiple RDDs

```
data = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
training, test=data.randomSplit([0.6, 0.4])
training.collect()
test.collect()
```

```
[1, 2, 5, 6, 9, 10]
[3, 4, 7, 8]
```

Alternating Least Squares (ALS)

- Alternating Least Squares (ALS) algorithm in `spark.mllib` provides collaborative filtering
- `ALS.train(ratings, rank, iterations)`

```
r1 = Rating(1, 1, 1.0)
r2 = Rating(1, 2, 2.0)
r3 = Rating(2, 1, 2.0)
ratings = sc.parallelize([r1, r2, r3])
ratings.collect()
```

```
[Rating(user=1, product=1, rating=1.0),
 Rating(user=1, product=2, rating=2.0),
 Rating(user=2, product=1, rating=2.0)]
```

```
model = ALS.train(ratings, rank=10, iterations=10)
```

predictAll() – Returns RDD of Rating Objects

- The predictAll() method returns a list of predicted ratings for input user and product pair
- The method takes in an RDD without ratings to generate the ratings

```
unrated_RDD = sc.parallelize([(1, 2), (1, 1)])
```

```
predictions = model.predictAll(unrated_RDD)
predictions.collect()
```

```
[Rating(user=1, product=1, rating=1.0000278574351853),
 Rating(user=1, product=2, rating=1.989035570377812 2)]
```

Model evaluation using MSE

- The MSE is the average value of the square of `(actual rating - predicted rating)`

```
rates = ratings.map(lambda x: ((x[0], x[1]), x[2]))
rates.collect()
```

```
[((1, 1), 1.0), ((1, 2), 2.0), ((2, 1), 2.0)]
```

```
preds = predictions.map(lambda x: ((x[0], x[1]), x[2]))
preds.collect()
```

```
[((1, 1), 1.0002785743518 53), ((1, 2), 1.9890355703778122)]
```

```
rates_preds = rates.join(preds)
rates_preds.collect()
```

```
[((1, 2), (2.0, 1.9890355703778122)), ((1, 1), (1.0, 1.0002785743518 53))]
```

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

Classification

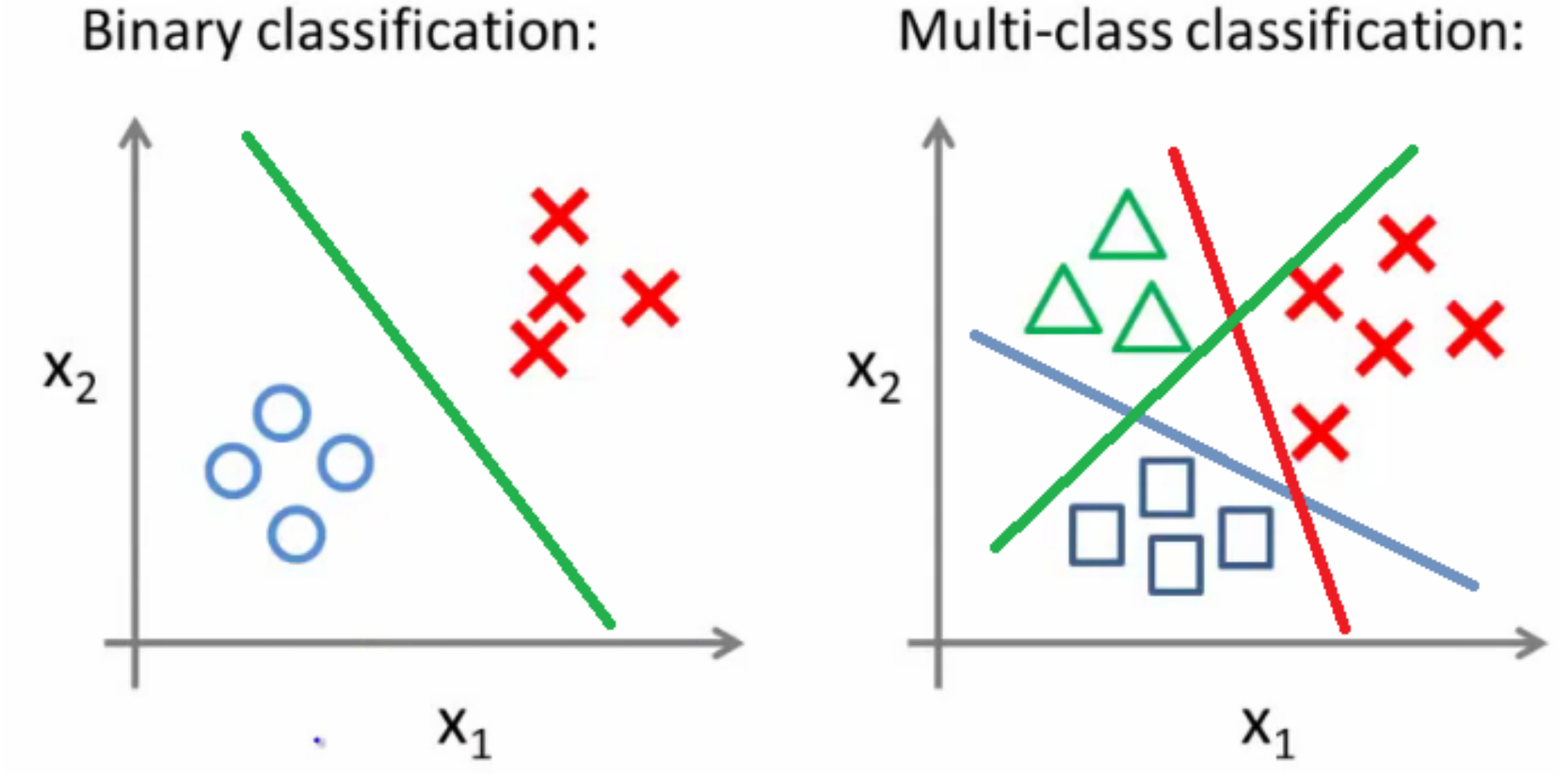
BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

datacamp

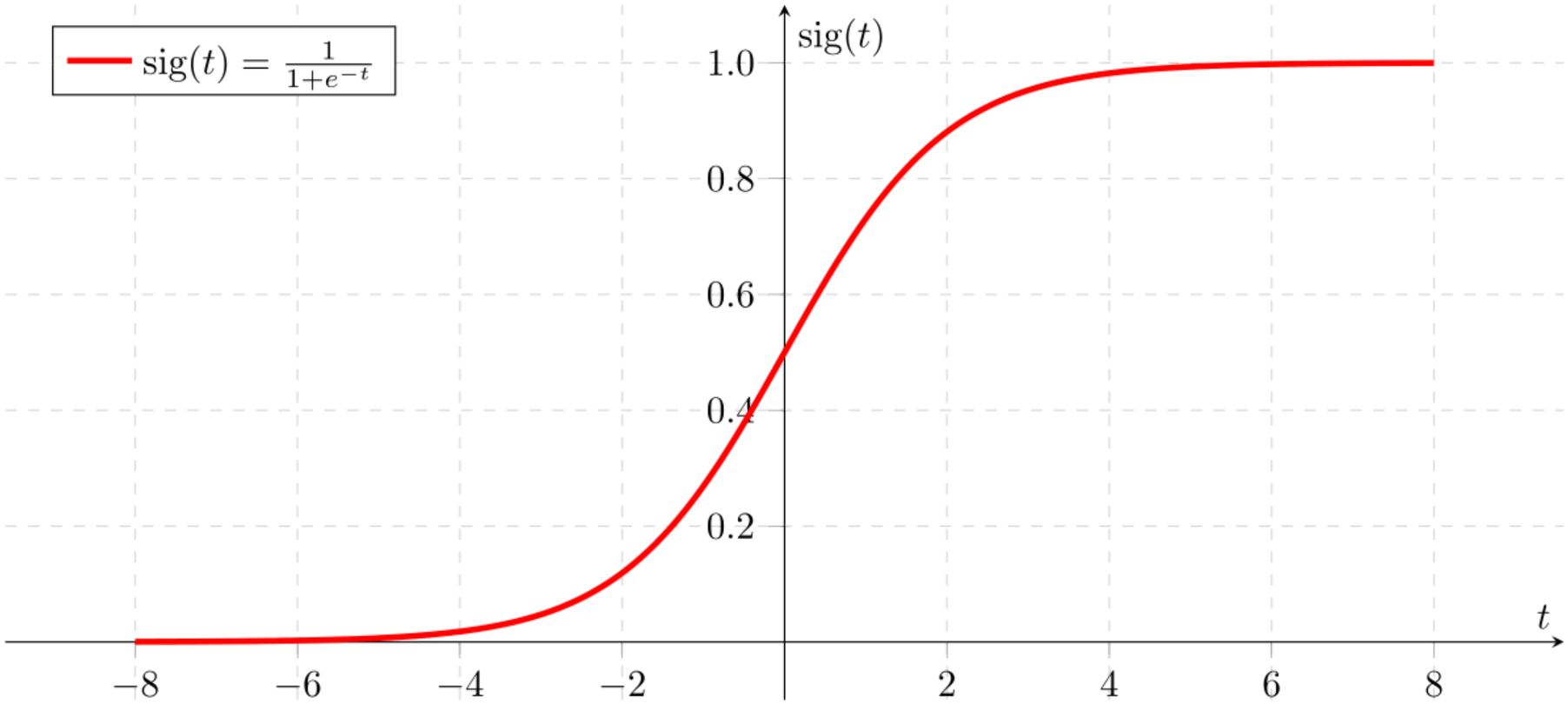
Classification using PySpark MLlib

- Classification is a supervised machine learning algorithm for sorting the input data into different categories



Introduction to Logistic Regression

- Logistic Regression predicts a binary response based on some variables



Working with Vectors

- PySpark MLlib contains specific data types Vectors and LabelledPoint
- Two types of Vectors
  - Dense Vector: store all their entries in an array of floating point numbers
  - Sparse Vector: store only the nonzero values and their indices

```
denseVec = Vectors.dense([1.0, 2.0, 3.0])
```

```
DenseVector([1.0, 2.0, 3.0])
```

```
sparseVec = Vectors.sparse(4, {1: 1.0, 3: 5.5})
```

```
SparseVector(4, {1: 1.0, 3: 5.5})
```

LabelledPoint() in PySpark MLlib

- A LabeledPoint is a wrapper for input features and predicted value
- For binary classification of Logistic Regression, a label is either 0 (negative) or 1 (positive)

```
positive = LabeledPoint(1.0, [1.0, 0.0, 3.0])
negative = LabeledPoint(0.0, [2.0, 1.0, 1.0])
print(positive)
print(negative)
```

```
LabeledPoint(1.0, [1.0,0.0,3.0])
LabeledPoint(0.0, [2.0,1.0,1.0])
```

HashingTF() in PySpark MLlib

- `HashingTF()` algorithm is used to map feature value to indices in the feature vector

```
from pyspark.mllib.feature import HashingTF
sentence = "hello hello world"
words = sentence.split()
tf = HashingTF(10000)
tf.transform(words)
```

```
SparseVector(10000, {3065: 1.0, 6861: 2.0})
```

Logistic Regression using LogisticRegressionWithLBFGS

- Logistic Regression using Pyspark MLlib is achieved using LogisticRegressionWithLBFGS class

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

```
lrm = LogisticRegressionWithLBFGS.train(RDD)
```

```
lrm.predict([1.0, 0.0])
lrm.predict([0.0, 1.0])
```

```
1
0
```

Introduction to Clustering

BIG DATA FUNDAMENTALS WITH PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

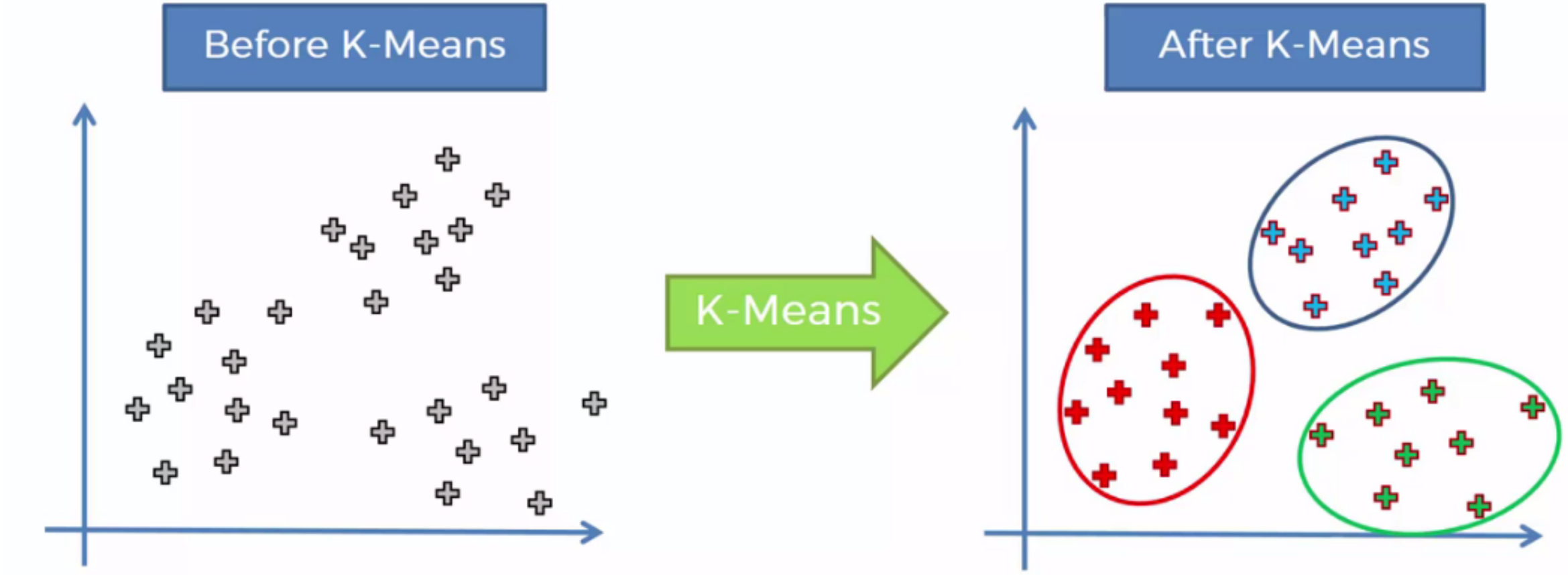
datacamp

What is Clustering?

- Clustering is the unsupervised learning task to organize a collection of data into groups
- PySpark MLlib library currently supports the following clustering models
  - K-means
  - Gaussian mixture
  - Power iteration clustering (PIC)
  - Bisecting k-means
  - Streaming k-means

K-means Clustering

- K-means is the most popular clustering method



K-means with Spark MLLib

```
RDD = sc.textFile("WineData.csv"). \
        map(lambda x: x.split(",")).\
        map(lambda x: [float(x[0]), float(x[1])])
RDD.take(5)
```

```
[[14.23, 2.43], [13.2, 2.14], [13.16, 2.67], [14.37, 2.5], [13.24, 2.87]]
```

Train a K-means clustering model

- Training K-means model is done using `KMeans.train()` method

```
from pyspark.mllib.clustering import KMeans
model = KMeans.train(RDD, k = 2, maxIterations = 10)
model.clusterCenters
```

```
[array([12.25573171,  2.28939024]), array([13.636875  ,  2.43239583])]
```

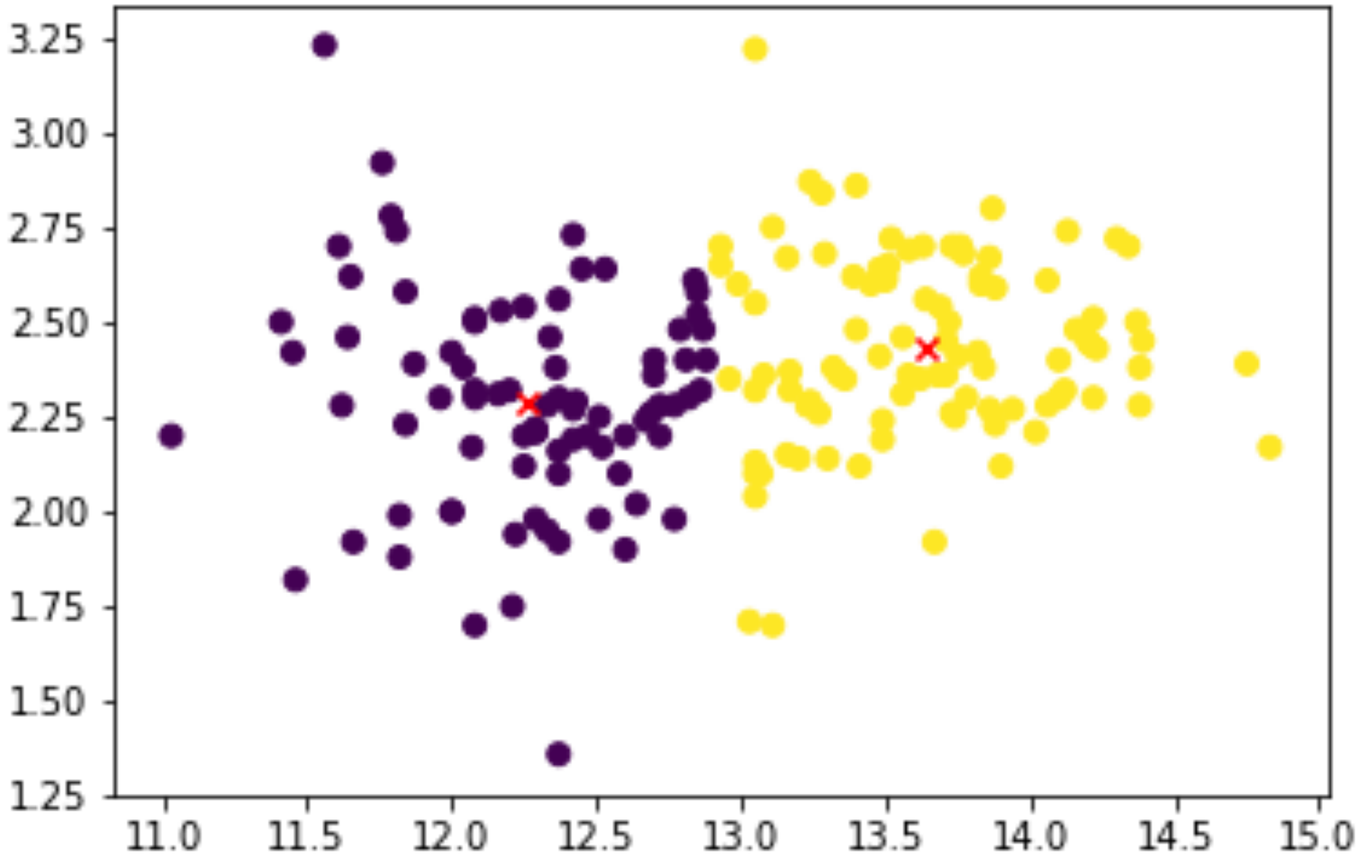
Evaluating the K-means Model

```
from math import sqrt
def error(point):
    center = model.centers[model.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))
```

```
WSSSE = RDD.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))
```

```
Within Set Sum of Squared Error = 77.96236420499056
```

Visualizing K-means clusters



Visualizing clusters

```
wine_data_df = spark.createDataFrame(RDD, schema=["col1", "col2"])
wine_data_df_pandas = wine_data_df.toPandas()
```

```
cluster_centers_pandas = pd.DataFrame(model.clusterCenters, columns=["col1", "col2"])
cluster_centers_pandas.head()
```

```
plt.scatter(wine_data_df_pandas["col1"], wine_data_df_pandas["col2"]);
plt.scatter(cluster_centers_pandas["col1"], cluster_centers_pandas["col2"], color="red", marker="x"
```

Fundamentals of BigData and Apache Spark

- **Chapter 1:** Fundamentals of BigData and introduction to Spark as a distributed computing framework
  - Main components: Spark Core and Spark built-in libraries - Spark SQL, Spark MLlib, Graphx, and Spark Streaming
  - PySpark: Apache Spark's Python API to execute Spark jobs
  - PySpark shell: For developing the interactive applications in python
  - Spark modes: Local and cluster mode

Spark components

- **Chapter 2**: Introduction to RDDs, different features of RDDs, methods of creating RDDs and RDD operations (Transformations and Actions)
- **Chapter 3**: Introduction to Spark SQL, DataFrame abstraction, creating DataFrames, DataFrame operations and visualizing Big Data through DataFrames
- **Chapter 4**: Introduction to Spark MLlib, the three C's of Machine Learning (Collaborative filtering, Classification and Clustering)