Spark Overview for Scala Analytics

Learning Objectives

Bookmark this page

Learning Objectives

In this course you will learn about:

- how Spark and its ecosystem work
- how to use RDD and DataFrame APIs
- how to connect DataFrames with Spark Stream
- basic concepts of machine learning
- an example of using the Spark MLlib to analyze Twitter stream data

Lesson 1 - Introduction to Spark

- What is Spark
- Our first Spark Application
- The Spark Execution Model
- The Spark Console
- Running Spark in a Standalone Cluster

Lesson 2 - Introduction to RDDs

- Tuning RDDs
- The Spark Web Console: A Deep Dive
- Broadcast Variables and Accumulators
- More Transformations: the Inverted Index Algorithm
- Refining the Inverted Index

Lesson 3 - DataFrames for Large Scale Data Science

- Introduction to SparkML
- Exploring the DataFrame API
- DataFrame Data Sources I
- DataFrame Data Sources II
- Speeding Up with DataFrames

Lesson 4 - Advanced Spark Topics

- DataFrame Joins
- Other DataFrame Transformations
- Using Hive with Spark SQL
- Spark Streaming
- Data Frames and Spark Streaming: Hive ETL

Lesson 5 - Introduction to Spark MLlib

- Spark with Scala
- Overview of Spark ML Algorithms
- Introduction to GraphX
- Sentiment Analysis of Twitter Stream I
- Sentiment Analysis of Twitter stream II

1. Introduction to Spark

What is Spark

After completing this lesson, you should be able to:

- Describe what Apache Spark is, and how it can be used to derive valuable information from data
- Compare and contrast Spark to the Hadoop ecosystem

- Discuss the various components of the Spark ecosystem

Our first Spark Application

After completing this lesson, you should be able to:

- Describe an RDD and its properties
- Understand the basic workflow of a Word Count application with Spark

The Spark Execution Model

After completing this lesson, you should be able to:

- Describe the lineage of an RDD
- Explain how stages of Spark jobs work
- Discuss what shuffling of data means

The Spark Console

After completing this lesson, you should be able to:

- Describe what the Spark Console is, and how it can be useful for understanding the runtime details of your Spark application

Running Spark in a Standalone Cluster

After completing this lesson, you should be able to:

- Describe Spark's various deployment options
- Discuss the role Resource Negotiators play in deciding what work is performed on which physical resources

Lesson Objectives

After completing this lesson, you should be able to:

- Describe what Apache Spark is, and how it can be used to derive valuable information from data
- Compare and contrast Spark to the Hadoop ecosystem
- Discuss the various components of the Spark ecosystem

What is Spark?

Apache Spark™ is a fast and general engine for large-scale data processing, with built-in modules for streaming, SQL, machine learning and graph processing

How Did We Get Here?

Large organizations began to collect more and more data about their business, and used Hadoop to derive value from this data cheaply and with acceptable performance at drastically reduced costs

Map-Reduce became the standard for applying a calculation to a dataset and separating it into meaningful groupings and sets

Map-Reduce Shortcomings

Difficult programming model

Even more difficult API

Poor support for iterative, machine learning and graph algorithms

Poor performance for complex jobs

Does not scale down very well

Why Spark?

Flexible, composable programming model

Concise, powerful API

Excellent performance for complex jobs

Supports event-streaming applications

Efficient support for iterative, machine learning, and graph algorithms

Scales from a single laptop to a large cluster

Spark History

Started in 2009 as a Berkeley AMP Lab Project, as part of Matei Zaharia's PhD thesis project

Became part of the Berkeley Data Analytics Stack, along with Mesos/YARN, storage components like Tachyon and more

Now a top-level Apache project

Why Scala and Spark?

Spark is written in Scala and the experience is native

There is a vibrant, data-centric ecosystem, and most of the "Big Data" ecosystem is JVM-based

Static typing means more correctness in a large codebase

Scala versus Java

Scala has the REPL, which promotes exploration of data and algorithms

Type Inference allows you to know in advance what the types will be in your data

Tuples are a concise and easy way to aggregate data

Pattern Matching makes quick work of deconstructing record data

DSL support allows you to use common idioms to model your designs

Lesson Summary

Having completed this lesson, you should be able to:

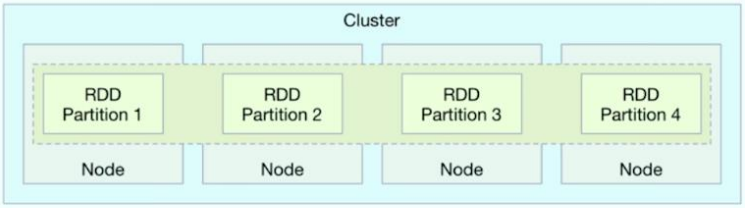
- Describe what Apache Spark is, and how it can be used to derive valuable information from data
- Compare and contrast Spark to the Hadoop ecosystem
- Discuss the various components of the Spark ecosystem

f transcript. Skip to the end.
welcome to coerce stewardess caliper data scientist curriculum in this course
we will be talking about the spark MapReduce engine which is an enabler for
data scientist to perform algorithms against their data this is not a deep
dive into Sparkman overview to explain basic capabilities of spark please see
the big deal university courses on spark fundamentals for a deeper look into its
capabilities after completing this first lesson what is spark you should be able
to describe what Apache spark is and how it can be used to derive valuable
information from data compare and contrast part to the Hadoop ecosystem
and discuss the various components of the spark ecosystem so what is spark
Apache spark is a fast and general engine for large-scale data processing
with built-in modules for streaming SQL machine learning and grab processing
spark is the engine behind it making transformations to data to derive value
from that data over the last twenty years
Internet giants like Amazon Google Yahoo eBay and Twitter inventing new tools for
working with data sets an unprecedented size far beyond the traditional tools
could handle they started the Big Data revolution characterized by the
inability to store and analyze these massive datasets with acceptable
performance at drastically reduce costs has been a spectacular success offering
cheap storage in the Hadoop distributed file system of data sets up to petabytes
in size with batch mode off-line analysis using MapReduce jobs other
tools in the Hadoop ecosystem include HBase NoSQL store and a customized SQL
base query engines like in power and drill many companies will continue to
write large Hadoop applications as before but these jobs can be replaced
for better performance and ease of programming using the spark programming
model MapReduce became a standard for applying calculations to a dataset and
separating into meaningful groupings and set
X however there were shortcomings this approach in a difficult programming
model in the Hadoop world and an even more difficult API for support for
iterative machine learning and graph algorithms as well as poor performance
for complex jobs it also does not scaled down very well requiring you to use more

resources than may be required for a single job these MapReduce shortcomings come largely from the fact that the API is not functional in nature in the previous course we saw that we could apply a function to a container of data those containers being a collection or future or an option if you will in those cases we took a function and we applied it to that data container in the case of spark we are going to do the exact same thing but the MapReduce engine of Hadoop did not support the application of functions to data and therefore it was more difficult to use so why spark it is a flexible composable programming model with a concise and powerful API is excellent performance for complex jobs because can do work in memory as opposed to all on disk it supports have been streaming applications and insufficient support for iterative machine learning and graph algorithms make it much simpler to use it also skills very well from a single laptop to a large cluster spark was started in 2009 as a berkeley project in the amp lab as part of Matei Zaharia his PhD thesis project it became part of the Berkeley data analytics stack also called PDAs along with me so soon yarn storage components like tachyon and more is now a top-level Apache project in the Apache foundation so the spark ecosystem contains several components include spark SQL an engine for performing SQL queries against Park data spark streaming for being able to handle data as it's coming in a very fast nature it also contains a male lib machine learning library written in Scala
allows you to perform analytics against your data and the graphics library also written in Scala for performing grafting operation so that you can figure out how data is connected together all built around the Apache spark MapReduce engine to spark is written in Scala and the experience is native when you use skeleton code against it
there's a vibrant data-centric ecosystem around spark and most of the big data ecosystem is JVM based already so you can leverage existing tools it also has static typing when you use Scala which means more correctness before you deploy an application particularly in a large code base which may need respect during school also has advantages over Joba Scala has a rebel to re-evaluate print loop that we discussed in the previous module which promotes exploration of data and algorithms against live data in-memory scott also a stipend printing allowing you to know in advance what the types will be in your data the two pools built into the Scala Collections library are concise and easy way to aggregate data and pattern matching makes quick work of deconstructing record data and DSL support allows you to use comments ATMs d'amato your designs
having completed this lesson you should be able to describe what apaches spark is and how it can be used to derive valuable information from data compare and contrast park to do because system and discuss the various components of the spark ecosystem
End of transcript. Skip to the start.

Scala APACHE Spark COGNITIVE CLASS.ai

What is an RDD?

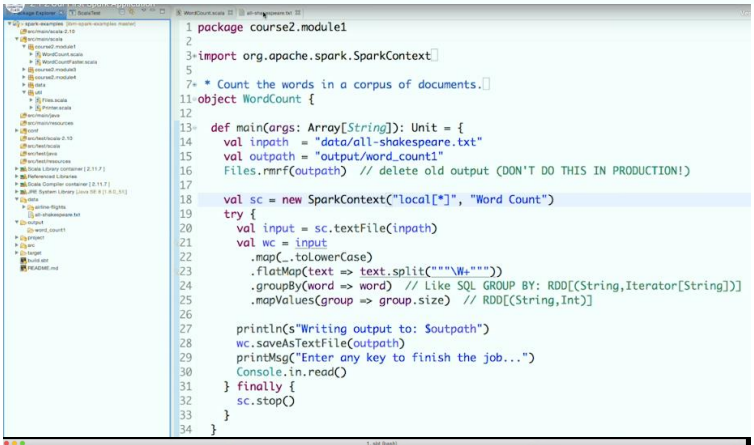


The RDD is Spark's core abstraction!

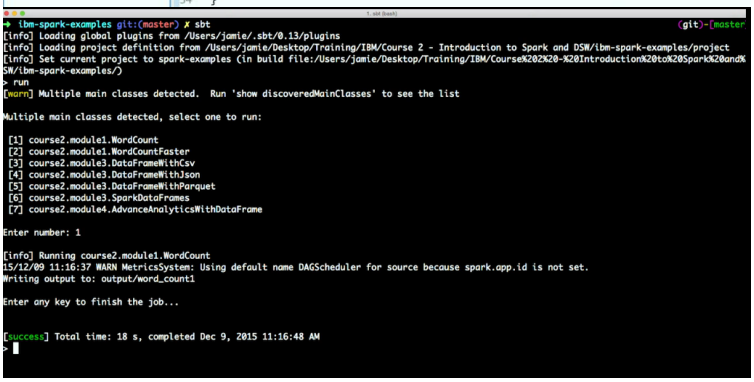
What is an RDD?

RDDs know their "parents", and transitively, all of their "ancestors" in the lineage of the data flow

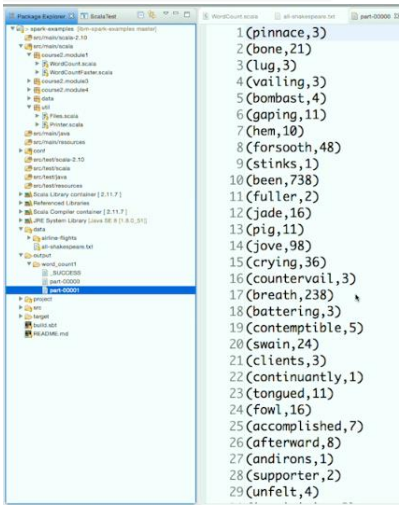
RDDs are resilient, and a lost partition can be reconstructed from its lineage

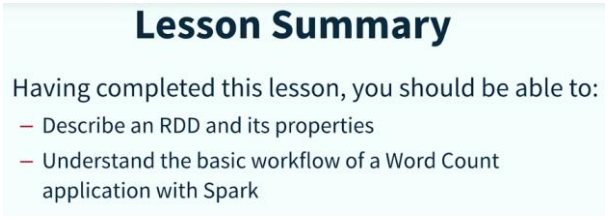


```
package course2.module1

import org.apache.spark.SparkContext

/ * Count the words in a corpus of documents.
object WordCount {

  def main(args: Array[String]): Unit = {
    val inpath  = "data/all-shakespeare.txt"
    val outpath = "output/word_count1"
    Files.rmrf(outpath)  // delete old output (DON'T DO THIS IN PRODUCTION!)

    val sc = new SparkContext("local[*]", "Word Count")
    try {
      val input = sc.textFile(inpath)
      val wc = input
        .map(_.toLowerCase)
        .flatMap(text => text.split("""\W+"""))
        .groupBy(word => word)  // Like SQL GROUP BY: RDD[(String,Iterator[String])]
        .mapValues(group => group.size)  // RDD[(String,Int)]

      println(s"Writing output to: $outpath")
      wc.saveAsTextFile(outpath)
      printMsg("Enter any key to finish the job...")
      Console.in.read()
    } finally {
      sc.stop()
    }
  }
```



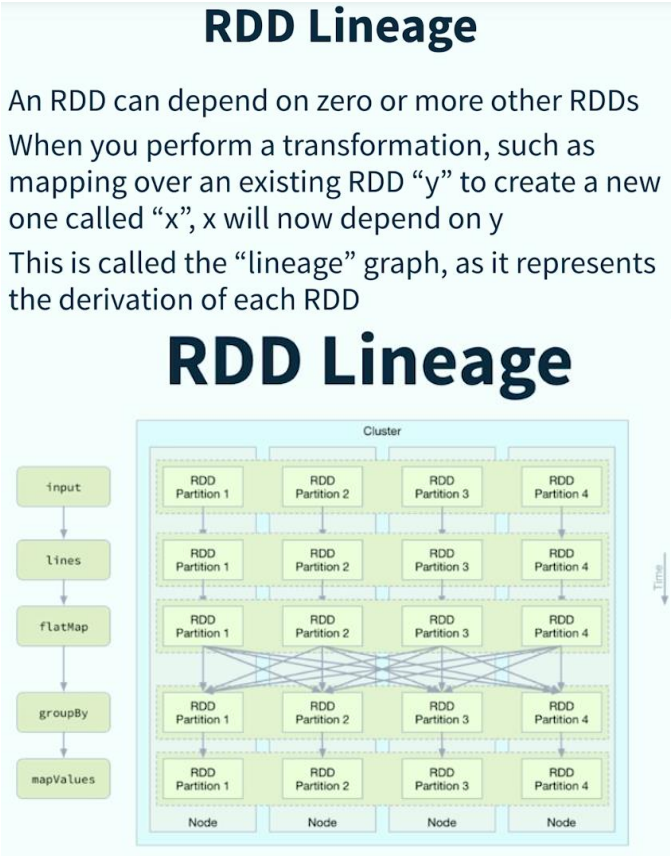
```
→ ibm-spark-examples git:(master) ✗ sbt                                    (git)-[master
[info] Loading global plugins from /Users/jamie/.sbt/0.13/plugins
[info] Loading project definition from /Users/jamie/Desktop/Training/IBM/Course 2 - Introduction to Spark and DSW/ibm-spark-examples/project
[info] Set current project to spark-examples (in build file:/Users/jamie/Desktop/Training/IBM/Course%202%20-%20Introduction%20to%20Spark%20and%
SW/ibm-spark-examples/)
> run
[warn] Multiple main classes detected.  Run 'show discoveredMainClasses' to see the list

Multiple main classes detected, select one to run:

 [1] course2.module1.WordCount
 [2] course2.module1.WordCountFaster
 [3] course2.module3.DataFrameWithCsv
 [4] course2.module3.DataFrameWithJson
 [5] course2.module3.DataFrameWithParquet
 [6] course2.module3.SparkDataFrames
 [7] course2.module4.AdvanceAnalyticsWithDataFrame

Enter number: 1

[info] Running course2.module1.WordCount
15/12/09 11:16:37 WARN MetricsSystem: Using default name DAGScheduler for source because spark.app.id is not set.
Writing output to: output/word_count1

Enter any key to finish the job...


[success] Total time: 18 s, completed Dec 9, 2015 11:16:48 AM
>
```





art of transcript. Skip to the end.
welcome to our first spark application after completing this lesson you should be able to describe an RDD and its properties and understand the basic workflow of a word count application with spark so what is an RDD Rd stands for a resilient distributed data set as the core abstraction of spark is simply just another collection in the Scala Collections library and therefore our disc is called it be alternate Scala collection is immutable at its core assuring thread safety and can be distributed across multiple machines here is an example of a cluster of machines working in a single spark context where an RDD is partitioned across four nodes these are TD's could be operated on with the single operation applied to each one of them each step of a data-flow that transforms an RDD results in a new Rd instance being created and our duties are lazy they represent a directed acyclic graph DAG of computation those constructed where looping is generally not possible with then it and it must always move forward through its transformations and the actual data is processed only when results are requested causing an action to be performed across that data are DD's have knowledge of their parents and transitive Lee all of their ancestors in the lineage of the data flow so as you apply multiple transformations the data in the yard eadie's there's always knowledge of where the data came from in the prior transformation and beyond and our DVDs are resilient if there is a las partition it can be reconstructed from its lineage by reapplying the function to the data the canonical example of how to do analytics on a set of data is due the word count of every word in all of Shakespeare's great works so if you have an input file that represents all of the texts from all of Shakespeare's plays
you could then transform the data to find out how many times each word and all of the place has been said here we have a simple object word count and inside of a we have a main method so that this is an application that can run on the GBM we specify that the input data should be all of Shakespeare in the text format and then we say where we want our output to be written in this case we create a spark context and we say it's going to be in local mode and will call this context word count we damn input the data from the text file

and then we tried to transform the data by first taking all the words and transforming them to be in lowercase format only
and then we split the words by a single space we can group them such that each word is representing an individual grouping of all the times it is shown
and then we map over the values to represent how many times it is shown we can then print out the results of this transformation and stop the job when we run this job we first create an SBT project and then I say to run the application it's going to ask me which one I want to run in my particular context nice a run the word count it then attempts to do so and whenever it's finished doing its work and asked me to press any key to finish the job
time to go through all of Shakespeare's works took eighteen seconds in that case with some latency for me to press the key however if I come down here inside of my project and I say show me the update to the output that was written I can see that several files were created including how many times each of these words were shown for example the word insurrection shows up six times in all of Shakespeare's works and I commend on shows up 100
having completed this lesson you should be able to describe an RDD and its properties and understand the basic workflow of a word count application with spark
End of transcript. Skip to the start.



Stages

Similar to an execution plan

Each stage can be executed as a pipeline in memory in a single node without any dependency on other nodes.

Operations in each stage is fused together like a pipeline so that output of one operation is piped as an input to next one in chain

Shuffling

When data has to be sent across the network to other partitions for proper grouping

By definition, a performance bottleneck

Stages

Some transformation steps do not need data from other partitions, such as `map`, `flatmap`, `filter`, etc.

Others, such as `groupBy` and `sort` require shuffling between partitions

Spark pipelines these steps into a single "stage," which uses a single JVM per partition

Why Are Stages Important?

They eliminate the overhead of running separate JVMs for each step

Makes management of intermediate data more efficient

Spark does not "materialize" the RDDs at each step - only the last RDD in each stage is actually computed, and shuffling data in intermediate steps is performed in memory and/or on disk

Lesson Summary

Having completed this lesson, you should be able to:

- Describe the lineage of an RDD
- Explain how stages of Spark jobs work
- Discuss what shuffling of data means

transcript. Skip to the end.
welcome to the spark execution model after completing this lesson you should be able to describe the lineage of an RDD explain how stages of spark jobs work and discuss what shuffling of data means an RDD can depend on SERO or more

other Rd DS depending itself on whether or not that RDD was created as a result of other transformations when you perform a transformation such as mapping over an existing RDD called why to create a new Rd called X RDX will now depend on rdy this is called the lineage graph as it represents a derivation of each RDD here's an example of how this might work if we had poor nodes in our cluster and an RDD partitioned across each one of those four nodes we then have the input transformation that we perform and then we have the lines transformation to count the number of lines inside of the dataset we then plat map the data and then perform a group I finally we perform a final transformation called map values to do the aggregation of the data very much like we saw in the previous lesson when we did our word count example we have stages inside of our execution of our jobs and this is similar to an execution plan if you will

each stage can be executed as a pipeline in memory in a single node without any dependency on any other node another way to think about this is operation fusion operations in each stage are fused together like a pipeline so that the output of one operation is piped as an input to the next one in the change this is one of the reasons why spark is faster it doesn't have to hit the disk or go over the network to do its work its all perform within a partition in isolation but the final output of each stage is usually a shuffled task in this case

imagine the group by task that we saw inside of our way word count where we had to figure out how we were going to group data across all of the partitions inside of our cluster we see that here in this example where we did the group by work and data had to be shuffled across each of the partition and our porno cluster some transformation steps do not need data from other partitions such as map flat map and filter but others such as group by or reduce by and sort requires shuffling between partitions and that means that these ones are more costly and performance than tasks such as map flag map and filter which could be performed and isolation spark pipelines he steps into a single stage which uses a single JBM per partition so why are stages important they eliminate the overhead of running separate GBM for each step and then make the management of intermediate data more efficient when you use the Hadoop MapReduce engine it's got a load the data from disk perform the tasks involved in the data transformation and then put it back on disk its then got a load from disk again

shuttle the data and then put it back on disk once more

all of this I O comes with a performance cost spark does not materialize the ending our DVDs at each step only the last resilient distributed data said in each stage is actually computed and shuffling data in intermediate steps is performed in memory and/or on disk depending on the size of that data having completing this lesson you should be able to describe the lineage of an RDD explain how stages of spark jobs work and discuss what shuffling of data means for performance

End of transcript. Skip to the start.

Spark Console

Every SparkContext launches a web UI, by default on port 4040

If multiple SparkContexts are running on the same host, they will bind to successive ports beginning with 4040 (4041, 4042, etc).

Provides insight into the runtime characteristics of your resource usage

Spark Console

A list of scheduler stages and tasks

A summary of RDD sizes and memory usage

Environmental information

Information about the running executors

Spark Console

Note that this information is only available for the duration of the application by default

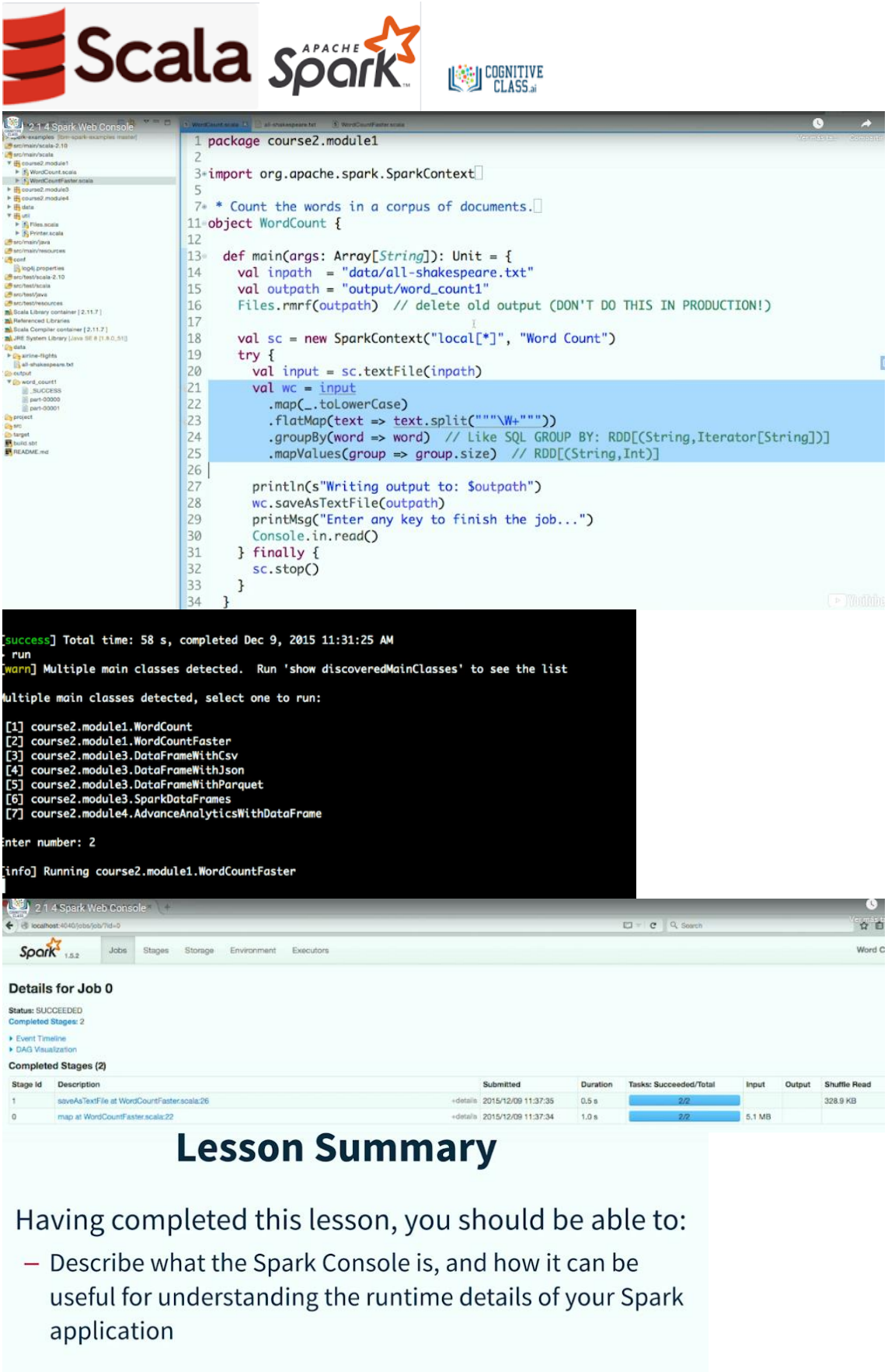
To view the web UI after the fact, set `spark.eventLog.enabled` to true before starting the application, and use the Spark history server

http://spark.apache.org/docs/latest/monitoring.html

Spark Console

Data scientists should use this tool when building Spark applications

It will help identify bottlenecks and inefficiencies in Spark jobs and your algorithms



welcome to the spark console after completing this lesson you should be able to describe what the spark console is and how it can be useful for understanding the runtime details of your spark application spark console is launched every time you create a spark context by default on port 4040 of whatever machine the spark context is being created on in the case where I'm going to run locally that would be localhost port 4048 multiple spark

context are running on the same post they will bind two successive ports beginning with forty 40 so the next bar context will be 40 41 the next 40 42 etcetera this provides insight into the runtime characteristics of your resource usage on that host the spark console gives you a list of scheduler stages in tasks a summary of RDD sizes and memory usage environmental information about the box on which it's running and information about the running executors and spark which are going to perform the work on your data note that this information is only available for the duration of the application by default to view the web UI after the fact that sparked have been logged unable to true before starting the application and then you want to host the summer box using something like sparks history server see the spark documentation on monitoring for more details such as the link as show below they decide to should use the spark console when building spark applications as it will help them identify bottlenecks and in efficiencies and sparked jobs and your algorithms let's have a look at how that would work in the previous example of how to do a word count will perform transformations on the data such that we did a group buy and then mapped the values together so that we were able to come up with the number of times a single word showed up inside of Shakespeare's great works however if we want to try slightly different approach that didn't do quite so much shuffling
we could instead map over the words and then reduce them by key which is going to do the exact same thing is our previous task but possibly more efficiently because of less shuffling being involved in the process so if we tried to execute the second job and we come into our SPT session and we run the task and we say we're going to run the word count faster example if we go to our browser and attempt to view the data as far as how well this job ran we can then see the data inside of the spark console telling us that the task succeeded
how long it took what resources it use and whether it did shuffles and what the cost was on the read and the right
having completed this lesson we should be able to describe what the spark console is and how it can be useful for understanding the runtime details of your spark application
End of transcript. Skip to the start.

Spark Resource Management

Local

Standalone

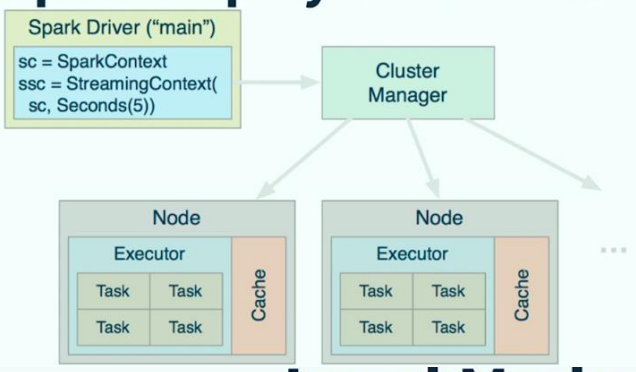
Mesos

Hadoop YARN

Standalone on EC2

Cassandra, Riak and other DBs soon

Spark Deployment Modes



Local Mode

So far, we have used local mode

A single JVM process to run:

- The driver
- The executor
- All tasks

Spark Standalone Cluster

The simplest way to start using a physical or logical cluster of nodes with Spark

Does not require an outside resource negotiator to determine what work should be run on which physical machine or virtual node

Spark Standalone Cluster

Manually configure the server instances

Spark Master instance is the Cluster Manager

Jobs run in a queue-like way (FIFO)

Includes stop scripts

Spark Standalone Cluster

Log contains master URL:

```
Master: Starting... spark://host:7077
```

ript. Skip to the end.
welcome to running spark in a stand-alone cluster after completing
this lesson you should be able to describe sparks various deployment
options and discuss the role resource negotiators play in deciding what work
is performed on which physical resources spark has several resource management
options and up until this point during this module we've been running
everything locally on the box on which I've been performing this work however
we also have stand-alone mode where spark itself can be responsible for
giving work across a cluster of machines spark also has the capability to
leverage external resource negotiators such as may Sosa Jorge dupes yarn
resource negotiator is also the standalone mode which works on Amazon's
ec2 and databases such as Cassandra react and other databases also have
their own modes for direct interaction with spark sparks main has a driver
which is responsible for creating the spark context and then dealing with the
data in some meaningful way in this example is streaming context that driver
then gives its information to the Cluster Manager and the Cluster Manager
dictates how work is distributed across the various nodes that make up the
partition data and where the work will take place in local mode we just have a
single JBM process to run and it's doing the work of the driver and the executor
and all the tasks but sparks stand-alone cluster is the simplest way to start
using more physical or logical nodes within a cluster in spark it does not
require an outside resource negotiator to determine what works should be run on
which physical machine or virtual node with sparks standalone mode we manually
configure our server instances and the spark master instance is the Cluster
Manager jobs run in a queue like way where you use first in first out
semantics and includes stops scripts for conveniens
to start the spark master we designate the spark home and then we use this
start master shell script we can then log the information is running on this
individual node using a master URL in the spark stand-alone cluster world we
have master and slave nodes and on each slave node we then start the slave using
the start slave shell script we designate the master by saying the
location of the master host and port on which you can be connected to having
completed this lesson you should be able to describe sparks various deployment
options and discuss the role resource negotiators play in deciding what work
is performed on which physical resources
End of transcript. Skip to the start.

2. Introduction to RDDs

Tuning RDDs

After completing this lesson, you should be able to:

- The different kinds of methods available for RDDs
- Partitioning, caching, and checkpointing, their importance and how to exploit them

The Spark Web Console: A Deep Dive

After completing this lesson, you should be able to:

- How to interpret the various pages of the Spark Web Console
- How to use the console to understand what your jobs are doing and how to improve their performance

Broadcast Variables and Accumulators

After completing this lesson, you should be able to:

- How to send support data to tasks using broadcast variables.
- How to accumulate information over all tasks for consumption back in the job's driver.

More Transformations: the Inverted Index Algorithm

After completing this lesson, you should be able to:

- More transformations available for building data applications
- How to deconstruct data problems into sequences of transformation steps

Refining the Inverted Index

After completing this lesson, you should be able to:

- Discuss how to sort data
- Understand why to remove "stop words" using a Broadcast Variable
- Explain how to identify which code runs in the driver and which code runs in the tasks across the cluster

Transformation, Action, and Control methods

- *Transformation* methods define the sequence of operations
- They are *lazy*
- They return new RDDs
- E.g., `map`, `flatMap`, `filter`, `groupBy`, `reduceByKey`, and `join`

Transformation, Action, and Control methods

- *Action* methods trigger execution of the "pipeline"
- They return results (or `Unit` when writing output)
- E.g., `count`, `collect`, `foreach`, and `saveAsTextFile`

Transformation, Action, and Control methods

- *Control* methods do *meta-operations* or return state information:
- E.g., `checkpoint`, `cache`, `persist`, `unpersist`, `coalesce`, and `repartition`
- Also `dependencies`, `getCheckpointFile`, `getStorageLevel`, and `isCheckpointed`

Cache/Persist

- *Cache* saves an RDD in memory. *Persist* let's you vary where the data is saved (e.g., to memory and disk)

```
import org.apache.spark.rdd.RDD
val ints1: RDD[Int] = sc.parallelize(0 until 5)
ints1.cache()     // Nothing happens until an action is called
                  // Same as ints2.persist(MEMORY)

val ints2: RDD[(Int, Int)] = ints1.map(i => (i, i*i))
ints2.persist(MEMORY_AND_DISK)
```

Cache/Persist

- *Cache / Persist* Prevents recomputing every ancestor of the RDD every time you use it. Storage options include:
- **MEMORY_ONLY** - (Default - cache calls `persist(MEMORY_ONLY)`)
- **MEMORY_AND_DISK** - Flush to disk if memory fills
- **DISK_ONLY** - Use on disk
- *_SER - Save serialized objects (byte arrays); more CPU expensive, more memory efficient
- **OFF_HEAP** - Experimental support for Tachyon

Checkpoint

- *Checkpoint* saves the RDD to the file system, it's *durable*
- The parent lineage is forgotten; no longer needed to reconstruct lost partitions
- Call checkpoint *before* evaluating the RDD

Checkpoint

Use checkpointing when:

- It would be too expensive to rely on caching to avoid recomputation of the RDD lineage
- It is impossible to go back to the data source, e.g., a socket in a streaming context

Checkpoint

```
import org.apache.spark.rdd.RDD
val ints1: RDD[Int] = sc.parallelize(0 until 5)
val ints2: RDD[(Int, Int)] = ints1.map(i => (i, i*i))

sc.setCheckpointDir("output/checkpoints")
ints2.checkpoint
ints2.isCheckpointed          // false - not computed yet!
ints2.getCheckpointFile       // None - same reason...
ints2.dependencies.head.rdd   // 1 dependency => ints1
ints2.count                   // == 5 - forces evaluation
ints2.isCheckpointed          // true
ints2.getCheckpointFile       // Some(".../checkpoints/...")
ints2.dependencies.head.rdd   // Now a checkpointed RDD
```

Checkpoint

Spark Streaming automatically sets up checkpointing and it cleans up old checkpoint files

For batch jobs, you have to set up checkpointing and clean up old checkpoint files yourself

Repartitioning

Sometimes the number of partitions is wrong:

- You just did a reduceByKey and now you have far fewer records, so you need fewer partitions
- You have too few partitions so each *task* takes a long time and cluster resources sit idle

Repartitioning

- `myRDD.repartition(n)` - converts to n partitions, where n can be < or > than original number
  - Might force a *shuffle* operation, which is expensive
- `myRDD.coalesce(n, shuffle = false)` - for n < original number. Shuffle is optional, so more efficient

Repartitioning

What should n be?

- It will depend on your application, data set, etc
- Use the Spark Web Console to see if partition sizes and task execution times are “reasonable”
  - Let’s explore that next…

Lesson Summary

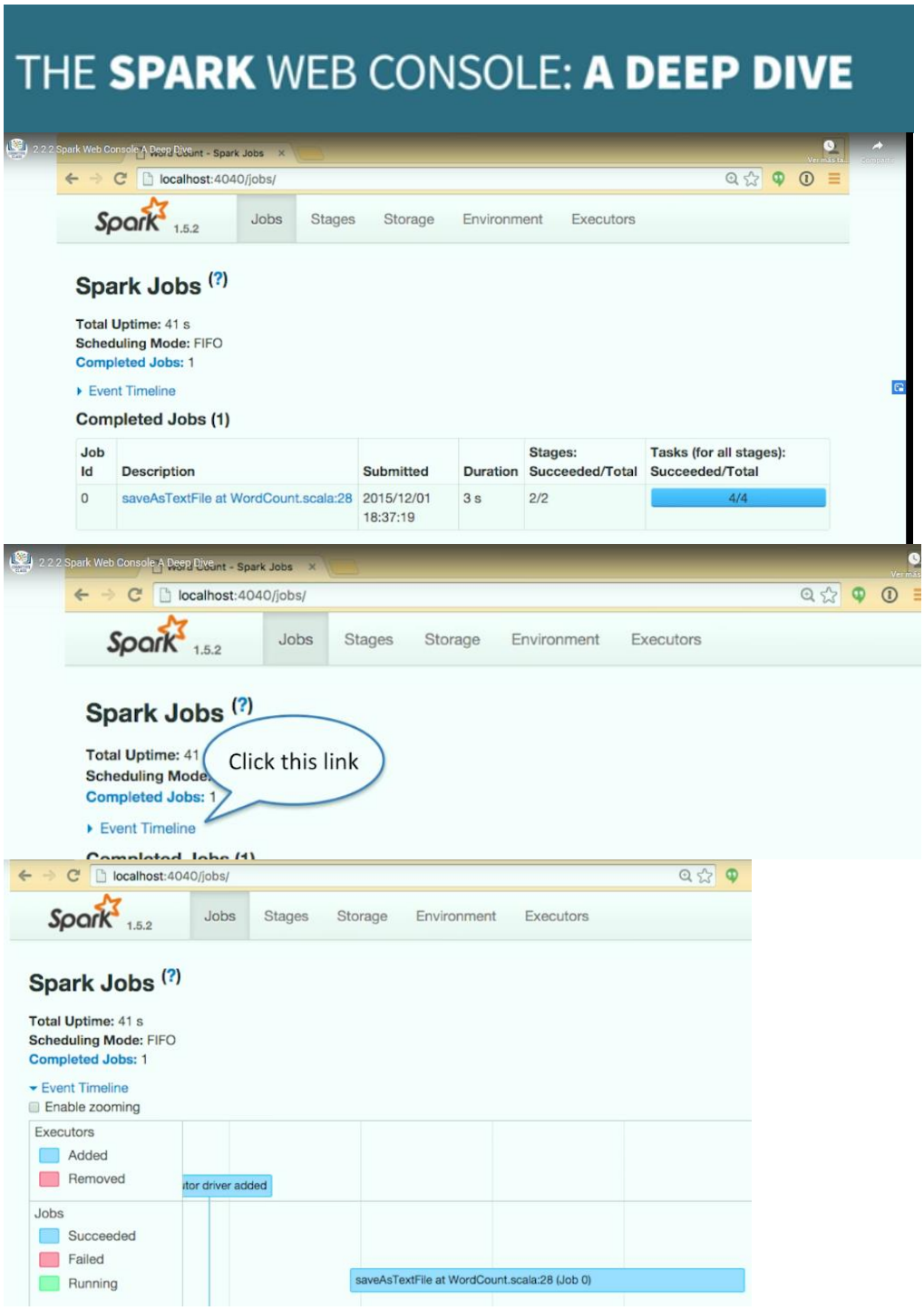
Having completed this lesson, you should be able to:

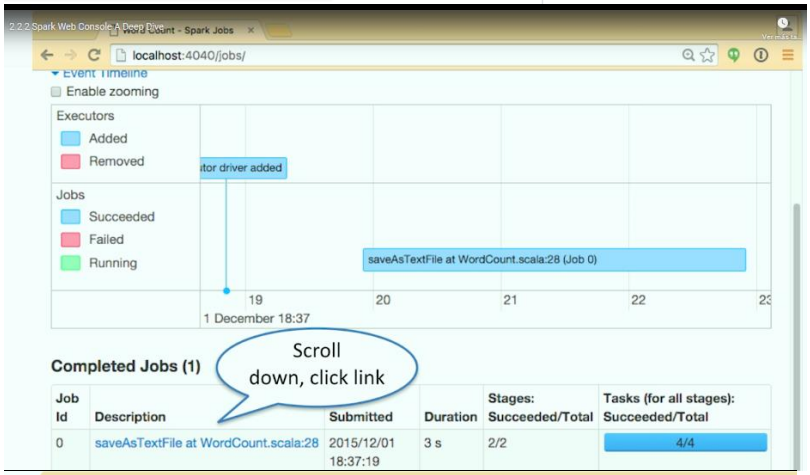
- Describe the different kinds of methods available for RDDs
- Explain partitioning, caching, and checkpointing, their importance and how to exploit them

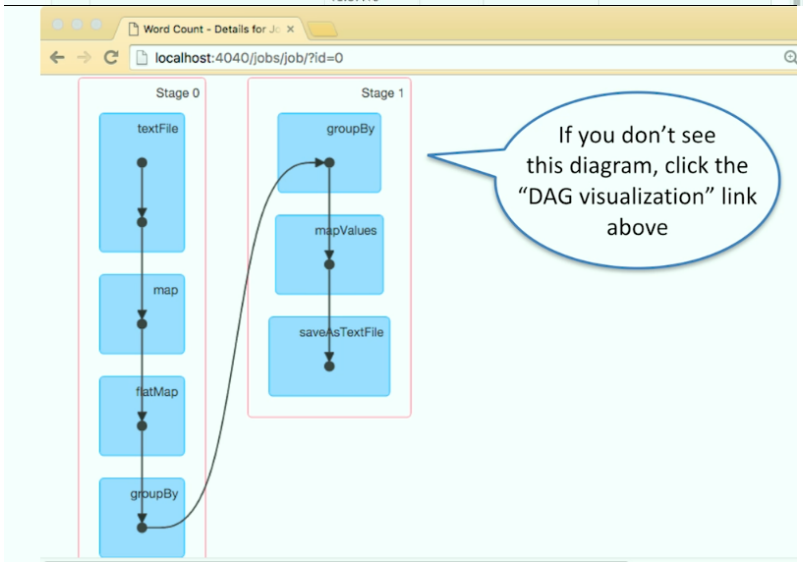
pt. Skip to the end.
welcome to tuning Rd deeds after completing this lesson should be able to describe the different kinds of methods available for our DVDs and explain partitioning caching and checkpointing their importance and how to exploit them
transformation methods defined the sequence of operations and they are lazy
they return new Rd DS examples of these include map flat map filter group by
reduced by key and join action methods trigger execution of the pipeline they
returned results or unit when writing output to understand unit consider Boyd in the sea or Joba examples of action methods include count collect for each and save as text file count returns along the number of records collect returns all the data and the RTD to the driver program as a scholar collections
such as an array or map but warning expectant how to memory error if the data is too big
control methods do med operations or return state information examples of this include checkpoint cash persist on persist coalesce and repartition also dependencies get checkpoint file get storage level and is check pointed our control methods for your operations cash saves an RDD in memory persisted lets
you very where the data is saved for example to memory or disk to our DD's where one is the parent of the other as shown here I'm showing the type signatures and had to import the RDD but they would be inferred if I let them in
this example we have to our DVDs and one is the parent of the other note that
paralyzed and map our transformation methods in this context cash and persist
are an optimization in a boy three computing the lineage of ancestor Rd DS but only if the RDD is still in the cash you want to call before invoking an action
it doesn't trigger the evaluation itself in the previous example you since to dot
on persist when you're done with the RDD I'm persist automatically cleans up the
RDD from the cash if it goes out of scope cash persist prevents computing every ancestor of the RDD every time you use it

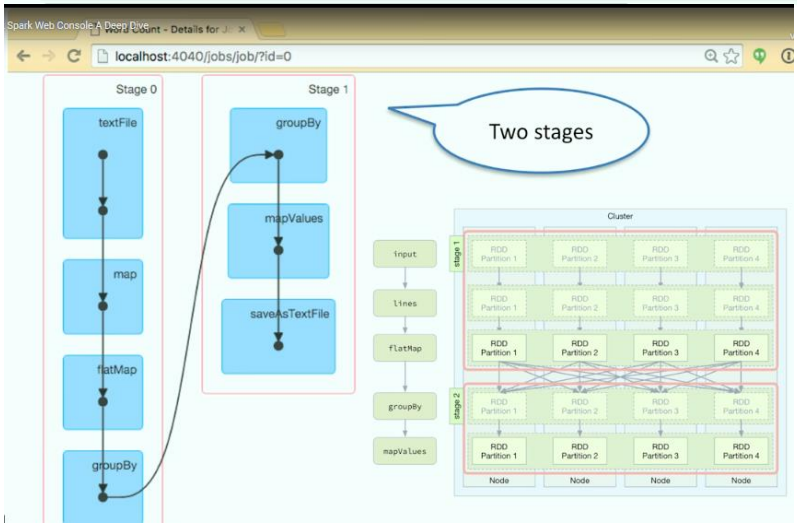
storage options include memory only memory and disk which blushes the disk
if memory pills disc only which means use only on disc and that starr underscore serwer you save serialized objects and by to raise its more CPU expensive but more memory efficient and has also offer deep which allows you to
use tachyon checkpoints saves the RGD to the file system and its durable the
parent lineage is forgotten it's no longer needed to reconstruct glass partitions because the data has been saved to disk will call checkpoint before evaluating the RDD we want to use checkpointing when it would be too
expensive to rely on cashing to avoid the computation of the RTD lineage and
it is impossible to go back to the data source for example a socket and streaming context where once the data has been processed you may no longer
have access to where it came from
here's an example of a checkpoint before we call count we've set up checkpointing
the spark context at checkpoint der is required first but nothing is actually check pointed yet after calling count we have a checkpoint file and a new parent
note that RDD dependencies returns a sequence of parents you have more than
one if this party is a joint group by etcetera
here we have just one so we get the head of the sequence the first element and
then ask that dependency for its corresponding RDD sparks streaming automatically sets up checkpointing and it cleans up the old checkpoint files for you
for batch jobs you have to set up the checkpointing and clean them up yourself
F we also have re- partitioning recalled it reduced by key and other group by variants bring together records with the same key so you might want to change the
number of records drastically but in a group by each record might be much bigger than the original records sometimes those partition numbers are wrong you just did a reduced by key and now you have far fewer records so you
need fewer partitions if you have too few partitions so each task takes a long time and cluster resources may sit idle to Usry partitioning in a more general method it requires a subtle operation if you want fewer partitions try using the
coalesce method so what should end be it would depend on your application in your
dataset used the spark web console to see if a partition sizes and task execution times are reasonable based upon your own non functional requirements let's explore that next

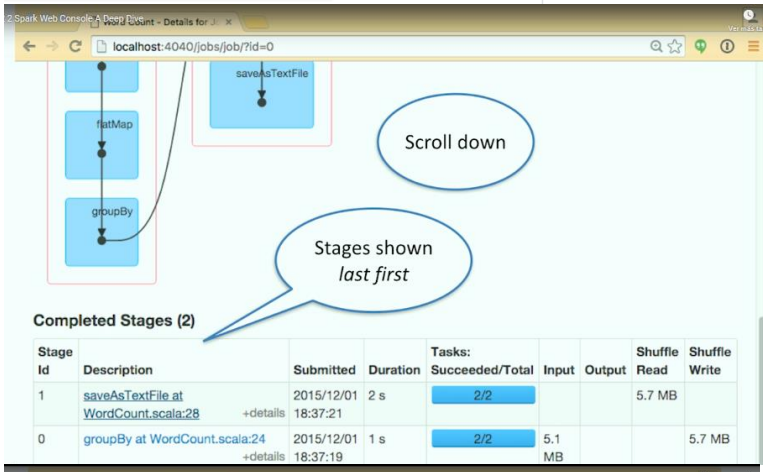
having completed this lesson you should be able to describe the different kind
of methods available for our DD's explain partitioning cashing in checkpointing as well as their importance in how to exploit them
End of transcript. Skip to the start.

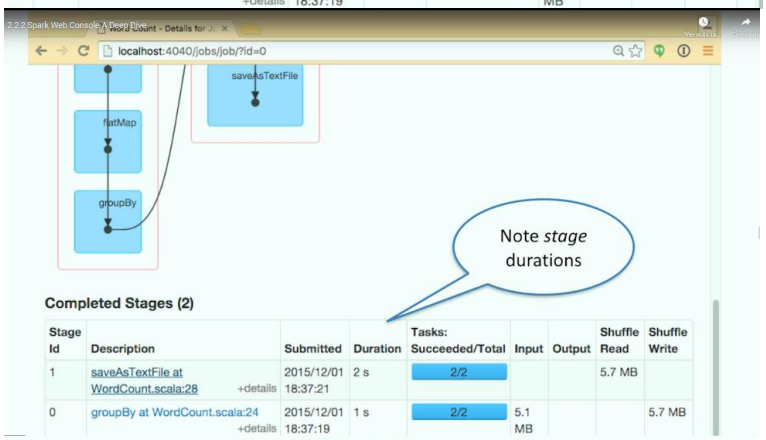



2.2.2 Spark Web Console A Deep Dive
Word Count - Spark Jobs
localhost:4040/jobs/
Enable zooming
Executors
Added
Removed
Jobs
Succeeded
Failed
Running
saveAsTextFile at WordCount.scala:28 (Job 0)
19
20
21
22
1 December 18:37
Scroll down, click link
Completed Jobs (1)
Job Id
Description
Submitted
Duration
Stages: Succeeded/Total
Tasks (for all stages): Succeeded/Total
0
saveAsTextFile at WordCount.scala:28
2015/12/01 18:37:19
3 s
2/2
4/4

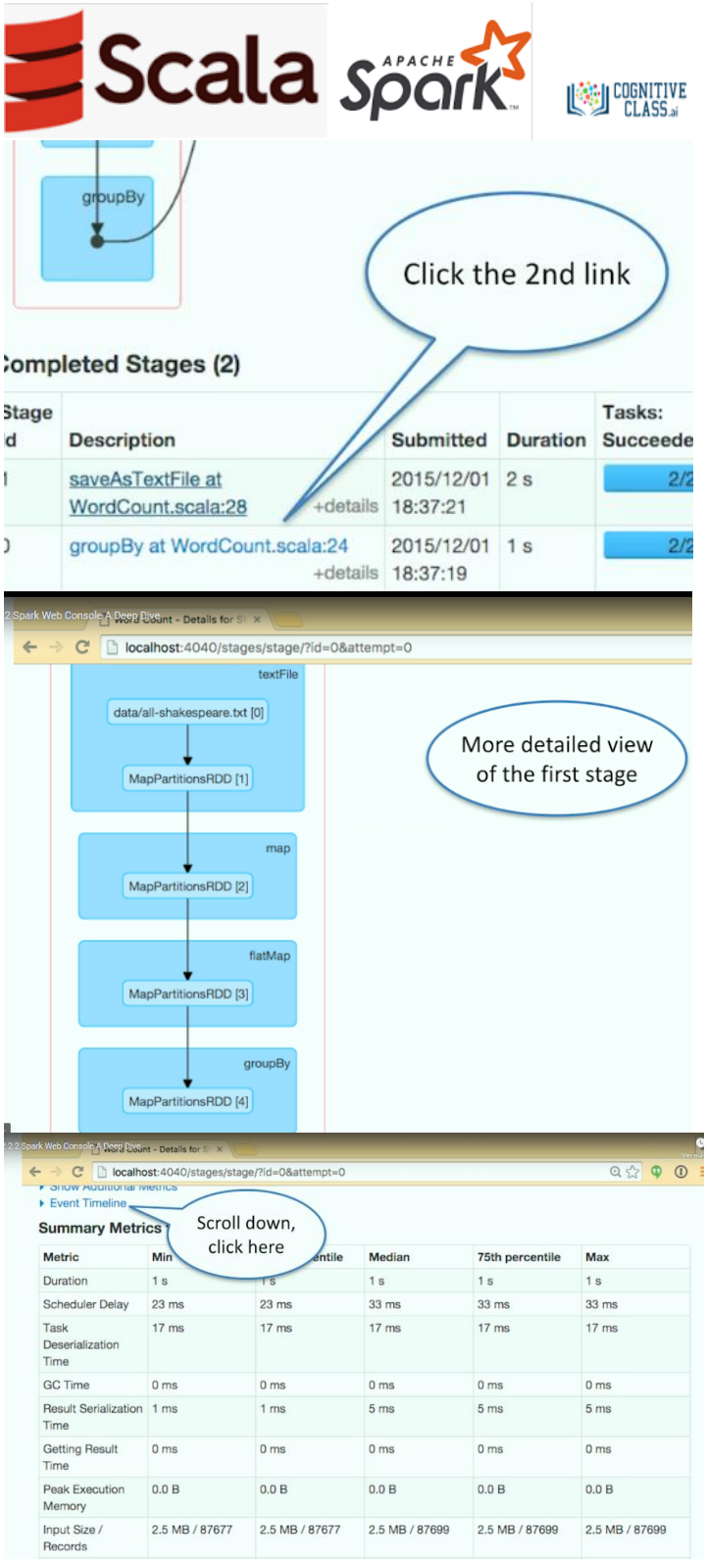

Word Count - Details for
localhost:4040/jobs/job/?id=0
Stage 0
Stage 1
textFile
map
flatMap
groupBy
groupBy
mapValues
saveAsTextFile
If you don't see this diagram, click the "DAG visualization" link above

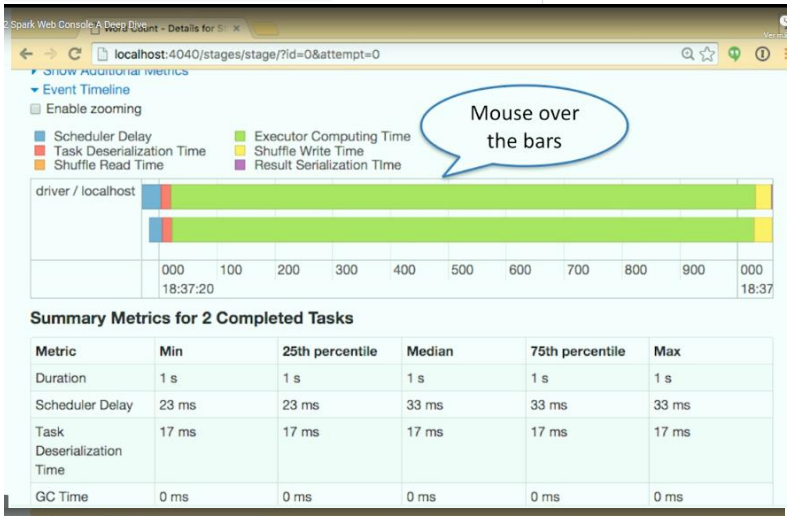

Spark Web Console A Deep Dive
Word Count - Details for
localhost:4040/jobs/job/?id=0
Stage 0
Stage 1
textFile
map
flatMap
groupBy
groupBy
mapValues
saveAsTextFile
Two stages
Cluster
input
lines
flatMap
groupBy
mapValues
stage 1
stage 2
RDD Partition 1
RDD Partition 2
RDD Partition 3
RDD Partition 4
Node


localhost:4040/jobs/job/?id=0
saveAsTextFile
flatMap
groupBy
Scroll down
Stages shown
last first
Completed Stages (2)
Stage Id
Description
Submitted
Duration
Tasks: Succeeded/Total
Input
Output
Shuffle Read
Shuffle Write
1
saveAsTextFile at WordCount.scala:28 +details
2015/12/01 18:37:21
2 s
2/2
5.7 MB
0
groupBy at WordCount.scala:24 +details
2015/12/01 18:37:19
1 s
2/2
5.1 MB
5.7 MB


localhost:4040/jobs/job/?id=0
saveAsTextFile
flatMap
groupBy
Note stage durations
Completed Stages (2)
Stage Id
Description
Submitted
Duration
Tasks: Succeeded/Total
Input
Output
Shuffle Read
Shuffle Write
1
saveAsTextFile at WordCount.scala:28 +details
2015/12/01 18:37:21
2 s
2/2
5.7 MB
0
groupBy at WordCount.scala:24 +details
2015/12/01 18:37:19
1 s
2/2
5.1 MB
5.7 MB

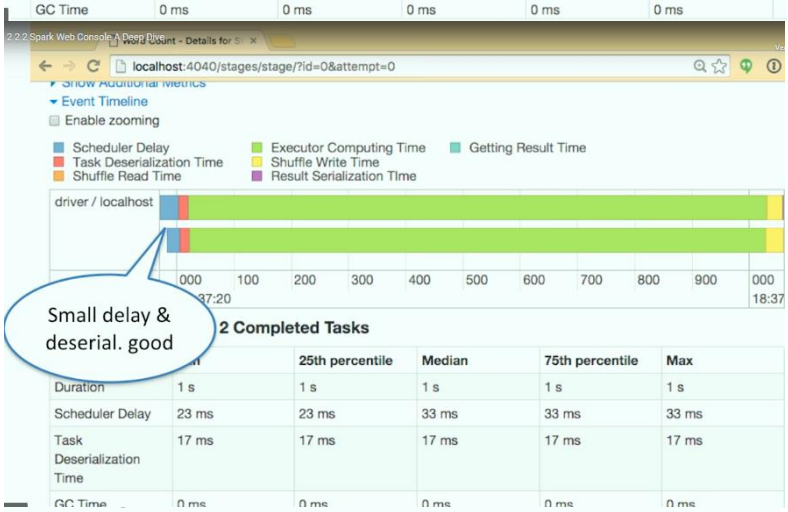

Data input to stage
Tasks: Succeeded/Total
Input
Output
Shuffle Read
Shuffle Write
2/2
5.7 MB
2/2
5.1 MB
5.7 MB


groupBy
Click the 2nd link
Completed Stages (2)
Stage Id
Description
Submitted
Duration
Tasks: Succeeded
saveAsTextFile at WordCount.scala:28 +details
2015/12/01 18:37:21
2 s
groupBy at WordCount.scala:24 +details
2015/12/01 18:37:19
1 s
localhost:4040/stages/stage/?id=0&attempt=0
textFile
data/all-shakespeare.txt [0]
MapPartitionsRDD [1]
map
MapPartitionsRDD [2]
flatMap
MapPartitionsRDD [3]
groupBy
MapPartitionsRDD [4]
More detailed view of the first stage
Event Timeline
Scroll down, click here
Summary Metrics
Metric
Min
Median
75th percentile
Max
Duration
1 s
1 s
1 s
1 s
Scheduler Delay
23 ms
23 ms
33 ms
33 ms
33 ms
Task Deserialization Time
17 ms
17 ms
17 ms
17 ms
17 ms
GC Time
0 ms
0 ms
0 ms
0 ms
0 ms
Result Serialization Time
1 ms
1 ms
5 ms
5 ms
5 ms
Getting Result Time
0 ms
0 ms
0 ms
0 ms
0 ms
Peak Execution Memory
0.0 B
0.0 B
0.0 B
0.0 B
0.0 B
Input Size / Records
2.5 MB / 87677
2.5 MB / 87677
2.5 MB / 87699
2.5 MB / 87699
2.5 MB / 87699



Summary Metrics for 2 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1 s	1 s	1 s	1 s	1 s
Scheduler Delay	23 ms	23 ms	33 ms	33 ms	33 ms
Task Deserialization Time	17 ms	17 ms	17 ms	17 ms	17 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms



2 Completed Tasks

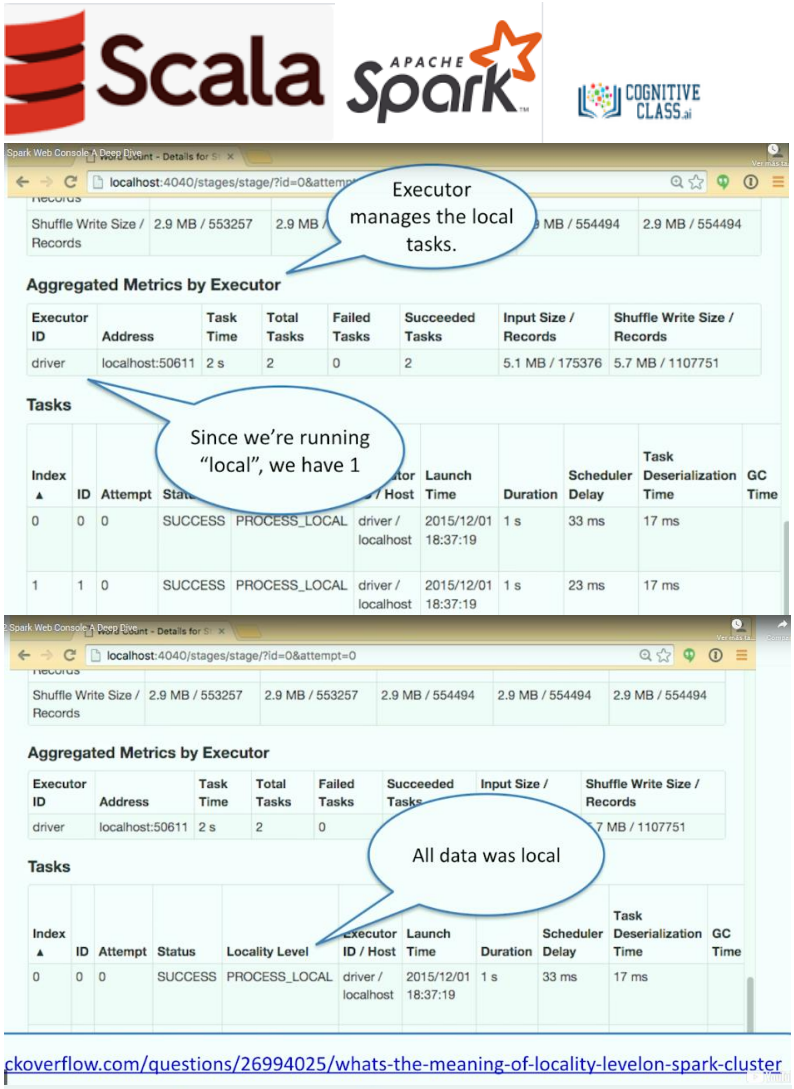
		25th percentile	Median	75th percentile	Max
Duration	1 s	1 s	1 s	1 s	1 s
Scheduler Delay	23 ms	23 ms	33 ms	33 ms	33 ms
Task Deserialization Time	17 ms	17 ms	17 ms	17 ms	17 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms

Summary Metrics for 2 Co

Metric	Min			75th percentile	Max
Duration	1 s	1 s	1 s	1 s	1 s

Summary Metrics for 2 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1 s	1 s	1 s	1 s	1 s
Scheduler Delay	23 ms	23 ms	33 ms	33 ms	33 ms
Task Deserialization Time	17 ms	17 ms	17 ms	17 ms	17 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Result Serialization Time	1 ms	1 ms	5 ms	5 ms	5 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Input Size / Records	2.5 MB / 87677	2.5 MB / 87677	2.5 MB / 87699	2.5 MB / 87699	2.5 MB / 87699
Shuffle Write Size / Records	2.9 MB / 553257	2.9 MB / 553257	2.9 MB / 554494	2.9 MB / 554494	2.9 MB / 554494



Lesson Summary

Having completed this lesson, you should be able to:

- Interpret the various pages of the Spark Web Console
- Use the console to understand what your jobs are doing and how to improve their performance

Start of transcript. Skip to the end.
welcome to the spark web console a deep died in the previous model we had less
than where we were introduced to the spark web console in this lesson you
will learn how to use the analytics provided by the web console to tune or
usage of our our DD's after completing this lesson you should be able to
interpret the various pages of the spark web console and use the console to
understand what your jobs are doing and how to improve their performance this is
the spark web console and is showing a job where we are doing our word count as
we showed in the previous module your notice that all tasks succeeded and it

shows you the duration and the stages of work if we click on the event timeline
link we get more information that zooms into our executors and jobs and how well
they worked we can mouse over these blue bars to get even more information about
how well they were performing if we scroll down for that we see the
completed jobs and now there's a link which allows us to see information about
the job for the word count work when we view that link we see that there were
two stages involved in this work the first involving the map in the flat map
and the group by where the group by resulted in a sharp of the data and a
new stage being created that group by led to the map values and then saving
the data is a text file again if you don't see this diagram click de dag
visualization link from above if you remember from the previous model we had
this diagram which showed our cluster for nodes and four partitions of our
data across those nodes where we did the import the lines and the flat map but
then the group by result in the shuffle that's where we see the breakdown of
stages here in this view
if we go down further will see that the complete in stages have a chart showing
more information
stages are shown last first and we can see that by the stage I D which is now D
sending note the stage durations the second stage took two seconds where the
first stage took one second that is largely due to the shuffle that took
place we can also see how many tasks took place one person I partition
there's the data input to the stage and the data output as well as shuttle data
read and shackled data written if we click on the second link to look at the
first stage of our operations we get a more detailed view of that per stage can
also scroll down and click on the event timeline just for that stage which shows
us information about the individual scheduler delay task serialization time
shuttle read time execution computing time shuttle right time and results
serialization time a small delay for Dec realizing is quite good we can also
compare the state into what we see in the table below we had to one second of
execution for this first stage so we're able to see fidelity of up to 100
milliseconds and we see the shuttle data for the next stage as the last part of
the work being done on two partitions for this first stage

the executor manages the local tasks and since world running in local mode we
only have one you can see that all data was local by viewing the process
underscore local value in the locality level for more information about
locality levels see this link explaining locality across the spark cluster
having completed this lesson you should be able to interpret the various pages
of the spark web console and use the console to understand what your jobs are
doing and how to improve their performance
End of transcript. Skip to the start.

Broadcast Variables

Use case: You need a large, read-only lookup table for all tasks

Keep a read-only variable local on each machine, rather than ship a copy of it with *each* task

It can be any serializable value

Broadcast Variables

```
val statesMap = Map("AL" -> "Alabama", "AK" -> "Alaska", ...)
val statesBV: Broadcast[Map[String, String]] =
  sc.broadcast(statesMap)
...

  .map {
    case (..., stateAbbrev, ...) =>
      val stateName = statesBV.value.getOrElse(stateAbbrev, ""
      (..., stateAbbrev, stateName, ...)
  }
```

Broadcast Variables

If the broadcast variable is mutable, it *should not* be modified after it is broadcast

- Any changes won't be propagated around the cluster!

Behind the scenes Spark uses a peer to peer (p2p, BitTorrent-like) protocol to distribute the value

Accumulators

Solve the opposite problem

"Accumulate" some results over the whole job, even when tasks are distributed around the cluster!

Accumulators

Use case: Count certain events that occur during job execution (e.g., number of bad records seen)

Analogous to MapReduce counters

Per-task value is modified through an *associative* operation

- Only tasks can *modify* the value

Support for numeric accumulators; You can add new types

Only the driver program can *read* the accumulator's value

Accumulators

```
val numBadLines = sc.accumulator(0)  // 0 is the seed value.
...
sc.textFile(inpath) map {
  case lineRE(name, text) => (name.trim, text.toLowerCase)
  case badLine =>
    numBadLines += 1    // Increment counter (in each task)
    ("", "")
}

...
// Back in the driver code:
println("Number of bad lines: " + badLines.value)
```

Accumulators: Accuracy

Spark can rerun failed or slow tasks, and run "speculative" copies

For accumulators used in *transformations*, Spark applies each task's update *every time the task is executed!*

– Hence, you can over count!

For accumulators used in *actions*, Spark applies each task's update to each accumulator only once

Need reliable counts? Use only inside actions like `foreach`

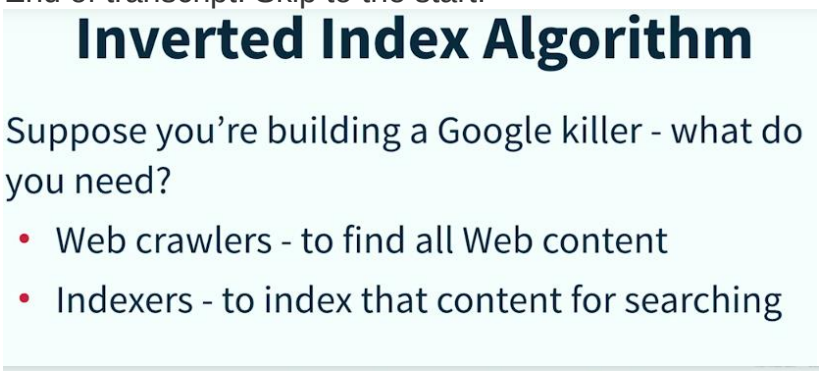
Lesson Summary

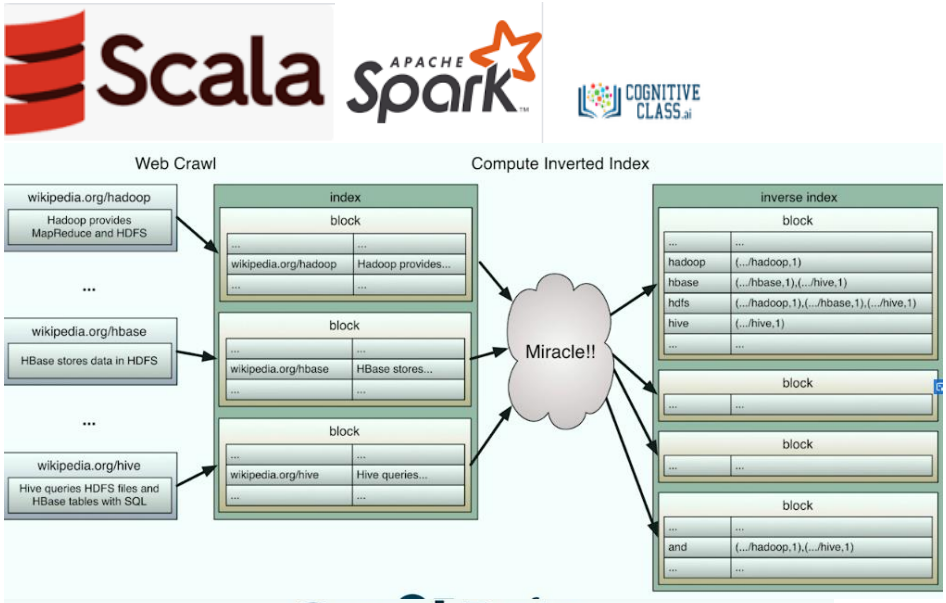
Having completed this lesson, you should be able to:

- Discuss how to send support data to tasks using broadcast variables
- Explain how to accumulate information over all tasks for consumption back in the job's driver

cript. Skip to the end.
welcome to broadcast variables and accumulators after completing this lesson you should be able to discuss how to send support data to tasks using
broadcast variables and explain how to accumulate information overall task for
consumption back in the jobs driver first let's discuss broadcast variables imagine a use case where you need a large readonly lookup table for all tasks you could keep a read only variable local on each machine rather than ship a copy of it with each task and it can be any serializable value here's an example we have our states map which is a map of keys to values including the abbreviation for state and the name of the USS state and then we
have the state's broadcast value which is a broadcast representation of that map if the broadcast variable is mutable it should not be modified after this broadcast because any changes won't be propagated around the cluster behind the
scenes spark is using a peer-to-peer protocol to distribute the value much like BitTorrent would accumulators solve the opposite problem they accumulate
some results over the whole job even when tasks are distributed around the cluster consider though that associativity is critical in this case
because you don't have control over how values are accumulating between tasks by
associativity I mean how you group them imagine arithmetic where you are adding
values together if you add 1+2 and then add three it doesn't matter if you did 2+3 and then added 12 grouping is irrelevant
note community is not required because community implies order consider this
use case we want to count certain events that occurred during job execution such
as the number of bad records we've seen this is analogous to MapReduce counters
paper task values modified through an associative operation where grouping does not matter and only tasks can modify this value we have support for numeric accumulators and we can add new types only the driver program can read

the result of the accumulators value so to create the accumulator we have our
spark context and we give it a seed value we then performer work and we had
to or increment the value inside are numb bad lines sparked can rerun failed or slow tasks and run speculative copies for accumulators using transformations
such as map or flat mapper filter spark applies each tasks update every time the
task is executed pants you could possibly over count for accumulators
used in actions spark applies each tasks update to each accumulator only once if
you need reliable counts only use them inside of actions like for each having completed this lesson you should be able to discuss how to send support data to
tasks using broadcast variables and explain how to accumulate information overall task for consumption back to the jobs driver
End of transcript. Skip to the start.





Crawl Data

Let's walk through the `Crawl.scala` program

(We'll sketch the implementation in the slides)

```
val separator = java.io.File.separator
val inpath  = "data/enron-spam-ham/*" // Note the *
val outpath = "output/crawl"
Files.rmrf(outpath)  // delete old (DON'T IN PRODUCTION!)
val sc = new SparkContext("local[*]", "Crawl")
...
```

Clean up old output, and setup the SparkContext

```
try {
  val files_contents = sc.wholeTextFiles(inpath).map {
    case (id, text) =>
      val lastSep = id.lastIndexOf(separator)
      val id2 = if (lastSep < 0) id.trim
                else id.substring(lastSep+1, id.length).trim
      val text2 = text.trim.replaceAll("""\s*\n\s*""", " ")
      (id2, text2)
  }
  println(s"Writing output to: $outpath")
  files_contents.saveAsTextFile(outpath)
} finally {
  sc.stop()
}
```

Map over the records, and transform them.


```
val inpath  = "output/crawl"
val outpath = "output/inverted-index"
Files.rmrf(outpath)  // delete old (DON'T IN PRODUCTION!)
val sc = new SparkContext("local[*]", "Inverted Index")
```

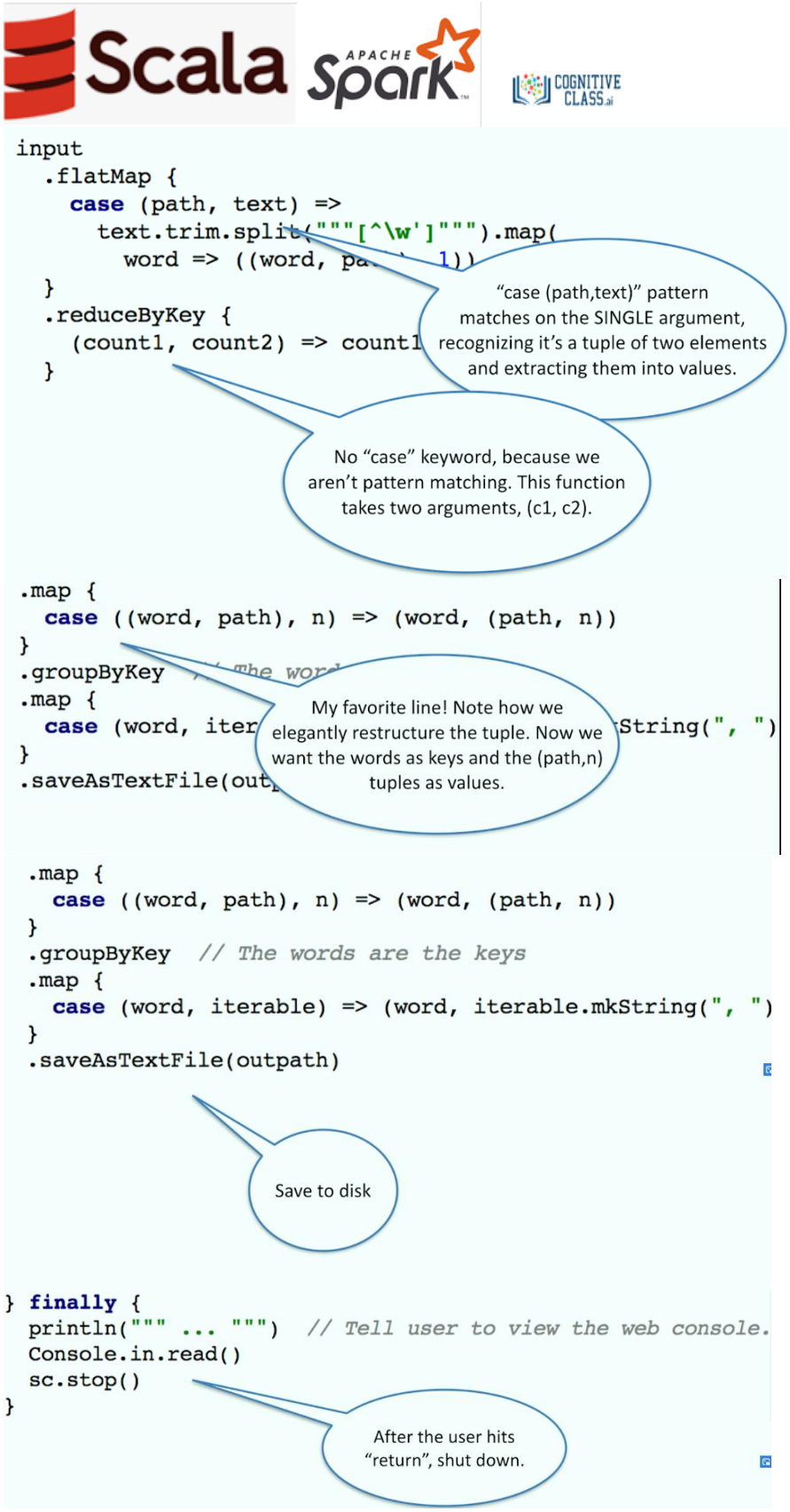
Use the crawl data as input

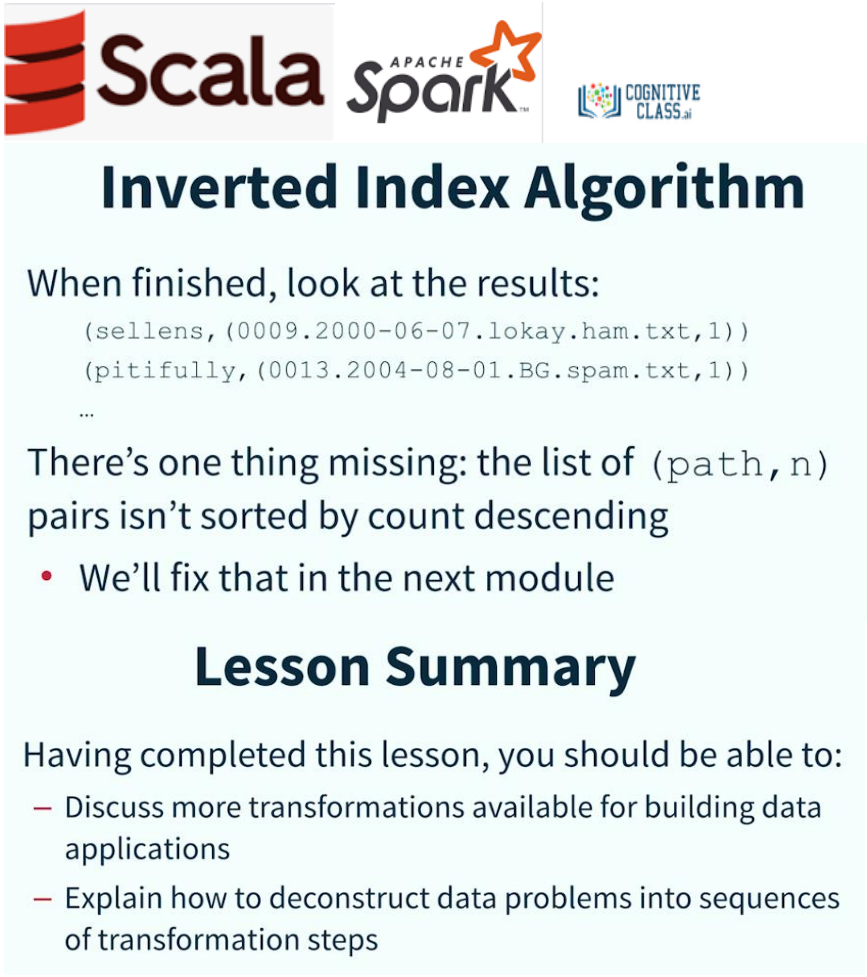
```
try {
  val lineRE = """^\s*\(([^,]+),(.*)\)\s*$""".r
  val input = sc.textFile(inpath).map {
    case lineRE(name, text) => (name.trim, text.toLowerCase)
    case badLine =>
      Console.err.println(s"Unexpected line: $badLine")
      ("", "")
  }
```

For other lines, write an error message, then return an empty tuple, which will be removed downstream

```
input
  .flatMap {
    case (path, text) =>
      text.trim.split("""[^\w']""").map(
        word => ((word, path), 1))
  }
  .reduceByKey {
    (count1, count2) =>
  }
```

Use flatMap to split the text in 1 record into a collection of 0-N words, then flatMap "flattens" (concatenates) these collections into one big collection of words.


input
.flatMap {
case (path, text) =>
text.trim.split("""[^\w']""").map(
word => ((word, pa
1))
}
.reduceByKey {
(count1, count2) => count1
}
"case (path,text)" pattern matches on the SINGLE argument, recognizing it's a tuple of two elements and extracting them into values.
No "case" keyword, because we aren't pattern matching. This function takes two arguments, (c1, c2).
.map {
case ((word, path), n) => (word, (path, n))
}
.groupByKey
.map {
case (word, iter
String(", ")
}
.saveAsTextFile(out
My favorite line! Note how we elegantly restructure the tuple. Now we want the words as keys and the (path,n) tuples as values.
.map {
case ((word, path), n) => (word, (path, n))
}
.groupByKey // The words are the keys
.map {
case (word, iterable) => (word, iterable.mkString(", ")
}
.saveAsTextFile(outpath)
Save to disk
} finally {
println(""" ... """) // Tell user to view the web console.
Console.in.read()
sc.stop()
}
After the user hits "return", shut down.



t of transcript. Skip to the end.
welcome to more transformations the inverted index algorithm after completing this lesson we should be able to discuss more transformations available for building data applications and explain how to deconstruct data problems into sequences of transformation steps the inverted index algorithm is a great way to be able to build views of data suppose you're building a Google killer what do you need
first you need a web crawler to find all web content and then you need indexers
to index that content for searching and our first iteration we want to build an index of words the keys will be all of the words we find and the values we lists of URLs and word counts for example food to food at org 1000 wikipedia.org food to 100 it's best to sort descending by words but why because
if we search for food in the Google killer we would probably find the pages that use food a lot to be the most interesting so imagine you have web crawlers running constantly finding pages on the internet and writing the initial data set with URLs or other ideas and the content of the pages found then the inverted index is computed over that data set to yield a new index were
the words are inverted moved from the initial value column to the key column
and for each word we have a list of URLs count pair's first let's generate fake role data you use a sampling of the email corpus from the now-defunct Enron
Corporation let's walk through a theoretical crawled out Scholar Program

and will sketch the implementation in the slides first of all we want to clean up the old output and set up the spark context note that you probably would not
want to delete the previous output in production this is only for how we would
do this
in this simple example first will read a directory and return records the filename and contents then we'll map over the records and transform them for
our dataset of emails we need to keep the file name which most Hadoop based I
O libraries ignore whole text files reads one or more directories and returns the data as a record per file with the file path as the key and the contents as the value we're going to do then remove the leading path part because it's not useful for us the paths will be Briggs ample path to exercises
data and ron's bam bam bam 100 file we don't need the leading directories most
of which are identical and handsome no value for distinguishing one mile from
the next chance we used the last / and if found we removed the text up to and
including it will remove embedded new lines in the text so it's all on one line we can then return the cleaned up data and write out the results and quit before we run the inverted index algorithm let's walk through the inverted index . Scholar Program we would use first we'll use the crowd data as input we use a regular expression to parse the input file name in text lines and we load the data and map over the lines we attempt to match with the Reg X
its successful we return a tuple with the name trimmed of white space and the
text in lower case for other lines we write an error message then return an empty tube or which will be removed downstream we used flat map to split the
text in one record into a collection of zero and words than flat map flattens or concatenate
these collections into one big collection of words actually we're going
to return to pools with word and path is the key and a 12 a seat count as the value and optimize group I could be performed using reduced by key where we
don't want the group's just the some of the elements and then we have a case of
the patent text pattern matching on a single argument recognizing its a tuple of two elements in extracting them into values there is no case keyword when we
do the reduced by key because we aren't pattern matching this function takes two
arguments about one in count to this is my favorite line know how we are

elegantly restructuring the tuple now we want the words keys and the path and two
poles as values are all SQL friend group buys then used it uses the first to Parliament the word as the key and the results end up being the word and then
some
iterable of the path to end count value another path to another and count value
and so on
at this point we're actually already done here we just reformat collection of paths and pairs that are the value part of the records and then we saved to disk
finally we allow the user to examine the web console before quitting by leaving
this up and running until somebody says yes they've hit return after the user hits return we shut down now let's generate the inverted index note that it pauses and waits for you to enter a return keystroke use this opportunity to browse the web console and understand this three-stage job when finished we
can look at the results there's only one thing missing the list of paths to end pairs isn't sorted by count descending will fix that in the next module
having completed this lesson
you should be able to discuss more transformations available for building data applications and explain how to deconstruct data problems into sequences
of transformation steps
End of transcript. Skip to the start.

Lesson Objectives

After completing this lesson, you will be able to:

- Discuss how to sort data
- Understand why to remove “stop words” using a Broadcast Variable
- Explain how to identify which code runs in the driver and which code runs in the tasks across the cluster

Inverted Index Algorithm

Recall our inverted index output:

```
(sellens,(0009.2000-06-07.lokay.ham.txt,1))
(pitifully,(0013.2004-08-01.BG.spam.txt,1))
…
```

Records that have more than one `(path,n)` pair are not sorted by n (count) descending.

Let’s fix that…

Exercises

I'll suggest how to enhance the implementation

You should pause the video and try it yourself

Then resume the video to see the solution

Sorting (name,count) Pairs

Exercise: Before the final formatting step, we had records like this:

- `(word1,Iterable((p11,n11), (p12,n12), …))`
- **`Iterable`** is a Scala type, not a Spark type:
  - scala-lang.org/api/2.10.6/#scala.collection.Iterable
- From this ***Scaladoc*** page, how could we sort it?

Sorting (name,count) Pairs

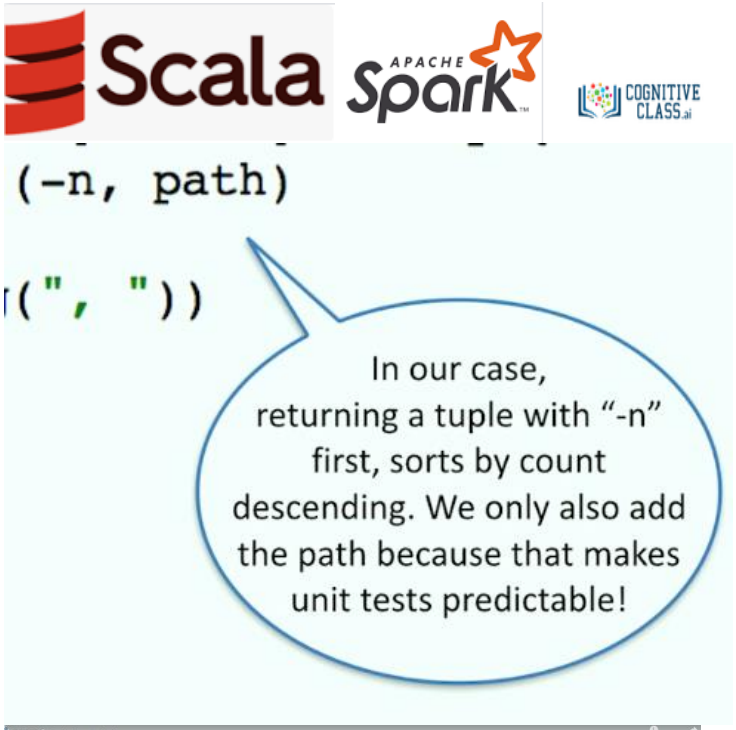
Actually, we can't - **`Iterable`** is a trait and we need to convert it to a collection with a sort function:

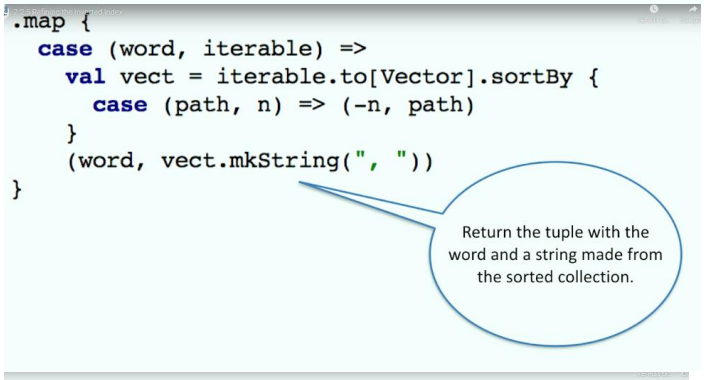
- In the last map step, before calling `mkString`, make a **`Vector`** using `iterable.to[Vector]`
- Then try using Vector's `sortBy` to sort by count descending, and name ascending.

```
.map {
  case (word, iterable) =>
    val vect = iterable.to[Vector].sortBy {
      case (path, n) => (-n, path)
    }
    (word, vect.mkString(", "))
}
```

Convert the iterable to a Vector, then call sortBy

The function passed to sortBy takes each record and returns something new that when sorted "naturally", results in the ordering you want.





Stop Words

Do we really want records for "the", "he", "she", "a", …? Probably not

These "stop words" can be removed with a filter step, but how do we get the list of stop words to each task

- We already know how: use a broadcast variable

Stop Words

- We have a util class called **`StopWords`**:

```
object StopWords {
  // We filter for "", "'", and "_", which can 'leak'
  // through our simple inverted index tokenization.
  val words = Set(
    "", "'", "_",
    "a's", "able", "about", "above", "according",
    ...)
}
```

Stop Words

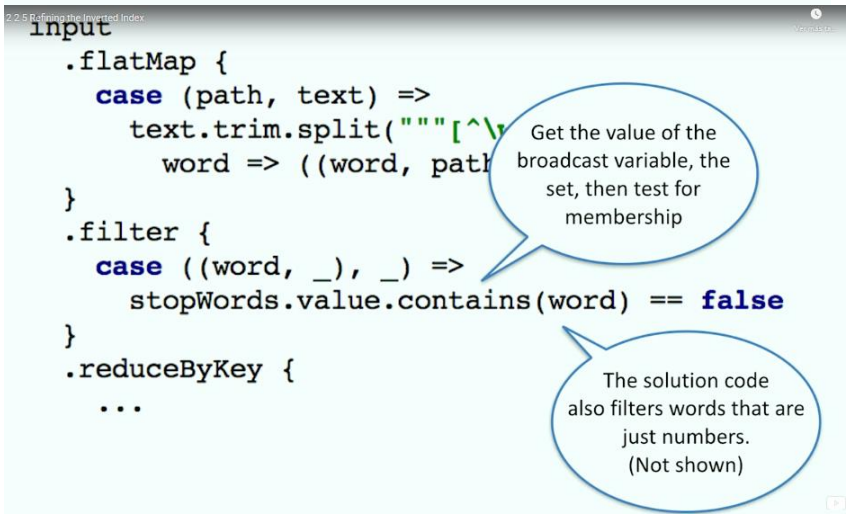
Exercise: Use the broadcast variable technique we discussed in module 3 to share this data with all tasks.

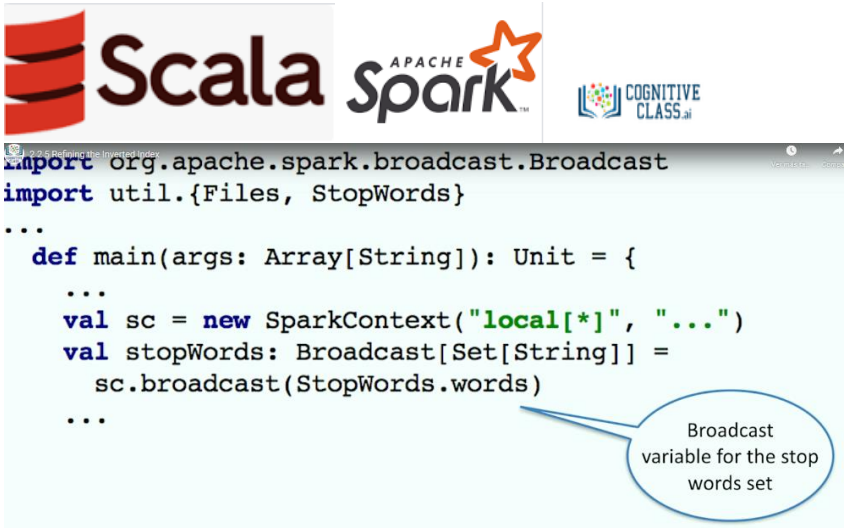
Use the `RDD.filter` method to remove words that are in the stop words set.

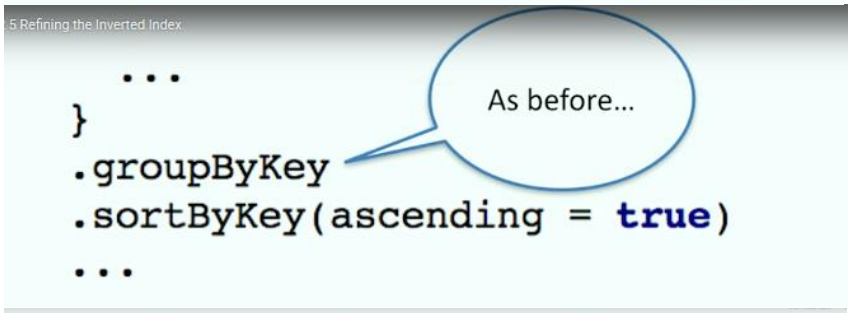
Stop Words

Pause the video here and try it yourself

Continue the video to see my solution







What Code Runs Where?

What code runs where?

- In all of these examples, some code blocks run in the driver and some run in the tasks remotely (when running in a real cluster)

```
input
  .flatMap {
    case (path, text) =>
      text.trim.split("""[^\w']""") map (
        word => ((word, path), 1))
  }
  .filter {
    case ((word, _), _) =>
      stopWords.value.contains(word) == false
  }
  .reduceByKey {
    ...
```

These functions serialized, sent to cluster nodes to run in tasks.

Lesson Summary

Having completed this lesson, you should be able to:

- Discuss how to sort data
- Understand how to remove “stop words” using a Broadcast Variable
- Explain how to identify which code runs in the driver and which code runs in the tasks across the cluster

of transcript. Skip to the end.
welcome to refining the inverted index after completing this lesson you will be
able to discuss how to sort data understand why to remove stop words using a broadcast variable and explain how to identify which code runs in the
driver and which code runs in the tasks across the cluster to recall are inverted index output from the previous lesson records that have more than one
path and pair are not sorted by end count descending let's fix that we're going to suggest how to enhance the implementation we should pause the video
and try it yourself then resumed the video to see the solution in this exercise before the final formatting step we had records that look like this the word and then an iterable the path in the number
another path and another number iterable is is collotype it is not a spark type from the sky ladakh page how could we sort it actually we can't enter a boy is
a trait and we need to converted into a collection with a sort function in the last map step before calling make string make a vector using an iterable . to vector call then try using vectors sort by to sort by count descending and name
ascending positive video here and try it yourself
continue the video to see my solution we will show you the modified map step at
the end
first we convert the iterable to of actor and then call sort by the function passed to sort by takes each record in return something new that when sorted
naturally results in the ordering you want note that this called a sort by is purely Escala collection thing is nothing to do with spark we are
operating on the data in a single record at a time now in our keys returning it to bowl with and first sort by count descending
we only also add the path because that makes unit test predictable if we do not
add the path when we have path one to name one and past 12 another name where
that is named to where the name one is equal to name two sometimes the first

two boys will get sorted first and sometimes the second to make this predictable especially when running unit tests and comparing with expected results it's best to sort by both will then return the two bowl with the word in the strength made from the sorted collection and we simplify the code using map values so we don't have to pattern match on the input to pool and construct a new output tuple do we really want for the EEC Hayward's show up probably not these stop words can be removed with the filter step but how do
we get the list of stop words to each task we already know how we use a broadcast variable we have a util class we create called stop words we put in
the words we do not care about is a sad because we only care about membership of
these words sets by definition do not include duplicates so let's do another exercise used to broadcast variable technique we discuss the module three to
share this data with all tasks use the RDD . filter method to remove words that
are in the stop words yet pause the video here and try it yourself
continue the video to see my solution here's the solution with only what's changed
first of all we have new imports and then we have a broadcast variable that we've created for the stop words in our set first of all we had what we had before and are flat map and then we have a new filter step what should we keep
though for this function we only need the word soyuz the undersea
soar to ignore the rest we get the value of the broadcast variable in the set and
then we test for membership using the contains function passing the word we're
checking the solution code also filters words that are just numbers but that's not shown and then we reduced by key as before the group by key as before with a
bonus sort by key we sort the RDD globally by the words which is potentially expensive so what code runs where in all of these examples some code
blocks run the driver and some run in the tasks remotely when running in a real cluster all of this code shown runs in the driver its setting up work for all of the tasks and then these functions are serialize sent to the
cluster nodes and run in tasks so we had code that ran in the driver to set up the broadcast variable than the code circled in this slide in the middle here runs in one or more tasks to use that broadcast variable these transformation
method calls run in the driver they define the pipeline that will run in tasks
having completed this lesson you should be able to discuss how to sort data understand how to remove stop words using a broadcast variable and explain

how to identify which code runs in the driver and which code runs in the tasks
across the cluster
End of transcript. Skip to the start.

3. DataFrames for Large Scale Data Science

Introduction to SparkML
After completing this lesson, you should be able to:
- Describe what a DataFrame is
- Outline the difference between RDDs and DataFrames, and why you might choose either one

Exploring the DataFrame API
After completing this lesson, you should be able to:
- Leverage the DataFrame API to perform basic transformations
- Describe the various ways DataFrames can be used

DataFrame Data Sources I
After completing this lesson, you should be able to:
- Describe the various data sources supported by DataFrames
- Discuss how to read and write JSON data with DataFrames

DataFrame Data Sources II
After completing this lesson, you should be able to:
- Discuss how to add support for a new data source

Speeding Up with DataFrames
After completing this lesson, you should be able to:
- Discuss how to create optimized data transformations using DataFrames
- Describe how to limit the amount of data that is read in
- Discuss basic optimization strategies

What is a DataFrame?

Created with the goal of enabling a wider audience to leverage the power of distributed processing

Inspired by data frames in R and Python (Pandas), but designed from the ground-up to support modern big data and data science applications

DataFrame Features

Ability to scale from kilobytes of data on a single laptop to petabytes on a large cluster

Support for a wide array of data formats and storage systems

State-of-the-art optimization and code generation through the Spark SQL Catalyst optimizer (the API is part of the SparkSQL package)

Seamless integration with all big data tooling and infrastructure via Spark

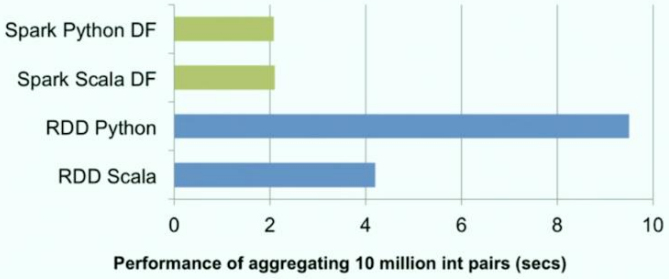
APIs for Python, Java, Scala, and R (in development via SparkR)

DataFrame Benefits

Spark DataFrames should make new users who are familiar with data frames in other programming languages feel at home

For existing Spark users, this extended API should make Spark easier to program, and improve performance through intelligent optimizations and code-generation

More DataFrame Benefits



Why Scala?

Modularity and type safety, particularly in larger code bases

Native interaction with Spark, and first implementations of new features arrive in Scala

Upcoming datasets feature, based on Scala records

RDDs versus DataFrames

- The RDD API is very flexible, but difficult to optimize
- The DataFrame API is much easier to optimize, but lacks the features of RDDs
  - Harder to use user-defined functions (UDFs)
  - Lack of strong types, but more type-safe than SQL queries embedded in Strings
- Datasets will be both, and interoperable with DataFrames

Catalysts

Optimization engine for Spark SQL

Agnostic to the back end

Used for manipulation of trees of relational operators and expressions

Execution (core)

- Query planner that translates Catalyst's logical queries into DataFrame operations
- Is defined by the `SQLContext` we use in the Spark core project

```
val sc = new SparkContext(master, name)
val sqlContext = new SQLContext(sc)
```

Example

- We could interoperate with Hadoop Hive tables using `HiveContext`, which extends `SQLContext`
- Access the Hive metastore
- Create, read and delete Hive tables
- Use Hive serializers/deserializers as well as UDFs
- SparkSQL's SQL dialect is a subset of HiveQL, and the goal is to make it a superset

Lesson Summary

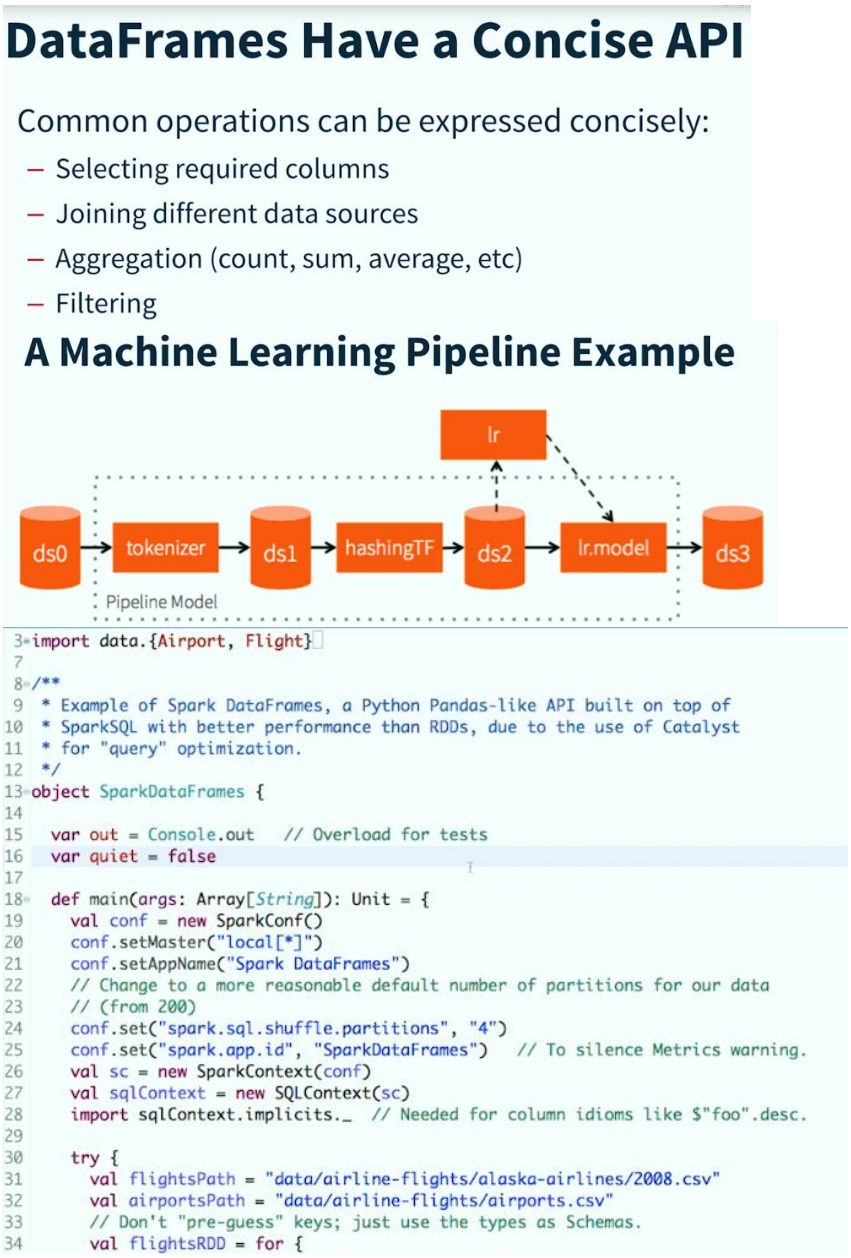
Having completed this lesson, you should be able to:

- Describe what a DataFrame is
- Outline the difference between RDDs and DataFrames, and why you might choose either one

welcome to data frames for large-scale data signs after completing this lesson
you should be able to describe what a data frame is an outline the difference

between our TD's and data frames and why you might choose either one data frames
were created with the goal of enabling a wider audience to leverage the power of
distributed processing they were inspired by data frames in our and Python where they're called pandas but designed from the ground up to support
modern big data and data science applications data frame features include the ability to scale from kilobytes of data on a single laptop to petabytes of data on a large cluster machines their support for a wide array of data formats
and storage systems as well as a state of the art optimization and code generation tool through the spark SQL catalyst optimizer of which the API is part of the spark SQL package their seamless integration with all Big Data tooling and infrastructure bias park and their API for Python Java Scala and our
spark data frame should make new users who are familiar with data frames and
other programming languages feel at home for existing spark users this extended
API should make spark easier to program and improve performance through intelligent optimizations and code generation in this diagram you can see the performance benefits of the data frame implementation there used to be wide variance between the performance of Scala RDD implementations vs Python Rd
implementations however using data frames the performance of both Python and Scala has been normalized so why would you use Collins that a python scholar provides modularity and type safety which is particularly useful in larger code bases and when we factoring your implementations also Scala's native
and its interaction with spark and first implementations of new features arrived
in Scala the upcoming data sets feature will also be based on scholar records
so when comparing our DVDs and data frames you must consider the following
Rd DAPI is very flexible but difficult to optimize the data frame API is much easier to optimize but lacks the features of our DD's it can be harder to use user-defined functions also called UCF's and it lacks strong types but can
be more type saved in SQL queries which are typically embedded in strings datasets will be both easy to optimize and have the fully flexible API as well as being fully interoperable with data frames catalyst is the optimization engine for spark SQL is agnostic to the back end implementation of the storage
layer it is used for manipulation of trees of relational operators and expressions for execution of the queries a query planner that translates catalysts logical queries into data frame operations is used is defined by the SQL context type we use in the spark or project as an example we could

interoperate with Hadoop I've tables using high of context which extends sequel context
allow us to access the hype medicine tour and create read and delete hype tables could use hype serialize and deserialize as well as user-defined functions and spark SQL SQL dialect is a subset itself of hype you well and the
goal is to make it a superset having completed this lesson we should be able to describe what it data frame is an outline the difference between our DVDs and data frames and why you might choose either one
End of transcript. Skip to the start.




```
3 import data.{Airport, Flight}
7
8 /**
9  * Example of Spark DataFrames, a Python Pandas-like API built on top of
10 * SparkSQL with better performance than RDDs, due to the use of Catalyst
11 * for "query" optimization.
12 */
13 object SparkDataFrames {
14
15   var out = Console.out   // Overload for tests
16   var quiet = false
17
18   def main(args: Array[String]): Unit = {
19     val conf = new SparkConf()
20     conf.setMaster("local[*]")
21     conf.setAppName("Spark DataFrames")
22     // Change to a more reasonable default number of partitions for our data
23     // (from 200)
24     conf.set("spark.sql.shuffle.partitions", "4")
25     conf.set("spark.app.id", "SparkDataFrames")   // To silence Metrics warning.
26     val sc = new SparkContext(conf)
27     val sqlContext = new SQLContext(sc)
28     import sqlContext.implicits._  // Needed for column idioms like $"foo".desc.
29
30     try {
31       val flightsPath = "data/airline-flights/alaska-airlines/2008.csv"
32       val airportsPath = "data/airline-flights/airports.csv"
33       // Don't "pre-guess" keys; just use the types as Schemas.
34       val flightsRDD = for {
```


```
2008,1,1,2,2057,2052,2312,2258,AS,324,N306AS,135,126,112,14,5,SEA,SJC,697,7,16,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,703,715,958,951,AS,572,N302AS,175,156,144,7,-12,SEA,PSP,987,6,25,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,2011,1846,2248,2145,AS,511,N564AS,157,179,136,63,85,SAN,SEA,1050,7,14,0,,0,0,0,0,0,63
2008,1,1,2,2301,2300,2354,2359,AS,376,N309AS,53,59,35,-5,1,SEA,GEG,224,5,13,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1221,1221,1422,1438,AS,729,N317AS,181,197,164,-16,0,TUS,SEA,1216,6,11,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1843,1840,2110,2125,AS,283,N318AS,147,165,124,-15,3,LAX,SEA,954,7,16,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,2045,2045,2314,2330,AS,211,N305AS,149,165,126,-16,0,LAX,SEA,954,6,17,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,49,50,547,523,AS,100,N315AS,238,213,222,24,-1,ANC,PDX,1542,2,14,0,,0,0,0,24,0,0
2008,1,1,2,1719,1715,1939,1956,AS,665,N302AS,140,161,118,-17,4,LAS,SEA,866,8,14,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,613,630,815,844,AS,531,N755AS,122,134,96,-29,-17,SJC,SEA,697,7,19,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,753,720,1125,1103,AS,571,N320AS,152,163,124,22,33,SEA,DEN,1024,5,23,0,,0,22,0,0,0,0
2008,1,1,2,NA,205,NA,620,AS,154,N309AS,NA,195,NA,NA,NA,ANC,SEA,1449,NA,NA,1,A,0,NA,NA,NA,NA,NA
2008,1,1,2,741,740,1128,1139,AS,728,N317AS,167,179,150,-11,1,SEA,TUS,1216,4,13,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1702,1530,1941,1804,AS,518,N564AS,159,154,142,97,92,SEA,SAN,1050,2,15,0,,0,92,0,5,0,0
2008,1,1,2,1306,1245,1551,1525,AS,580,N307AS,165,160,142,26,21,SEA,SAN,1050,2,21,0,,0,21,0,5,0,0
2008,1,1,2,2043,2040,2308,2323,AS,85,N302AS,205,223,181,-15,3,SEA,ANC,1449,5,19,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1405,1410,1637,1633,AS,640,N302AS,152,143,123,4,-5,SEA,LAS,866,13,16,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1050,1031,1329,1301,AS,292,N319AS,159,150,131,28,19,SEA,LAX,954,10,18,0,,0,0,0,9,0,19
2008,1,1,2,1610,1555,1850,1830,AS,478,N566AS,160,155,142,20,15,SEA,PSP,987,5,13,0,,0,15,0,5,0,0
2008,1,1,2,1639,1640,1903,1925,AS,485,N577AS,144,165,124,-22,-1,LAX,SEA,954,7,13,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1554,1555,1827,1837,AS,277,N323AS,153,162,123,-10,-1,LAX,SEA,954,10,20,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,26,30,508,444,AS,198,N320AS,222,194,199,24,-4,ANC,SEA,1449,6,17,0,,0,0,0,24,0,0
2008,1,1,2,717,700,1003,930,AS,811,N319AS,286,270,245,33,17,DFW,SEA,1660,7,34,0,,0,0,0,16,17,0
2008,1,1,2,629,630,856,903,AS,210,N318AS,147,153,131,-7,-1,SEA,LAX,954,6,10,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1211,1153,1408,1403,AS,539,N320AS,177,190,151,5,18,DEN,SEA,1024,4,22,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1935,1922,2130,2128,AS,295,N315AS,115,126,90,2,13,SFO,SEA,679,8,17,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,657,700,808,800,AS,254,N309AS,71,60,41,8,-3,GEG,SEA,224,8,22,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,929,941,1024,1040,AS,81,N799AS,55,59,40,-16,-12,ANC,FAI,261,3,12,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1816,1753,1956,1942,AS,413,N779AS,100,109,80,14,23,SMF,SEA,605,10,10,0,,0,NA,NA,NA,NA,NA
```

```
package data

import scala.math.Ordered

/** * Represent flight data from RITA (http://www.rita.dot.gov/), hosted at
case class Flight(
                  date:              Flight.Date,  // fields 1-4.
                  times:             Flight.Times, // fields 5-8, 12-16, 20-21
                  uniqueCarrier:     String,       //  9:  unique carrier code
                  flightNum:         Int,          // 10:  flight number
                  tailNum:           String,       // 11:  plane tail number
                  origin:            String,       // 17:  origin IATA airport code
                  dest:              String,       // 18:  destination IATA airport code
                  distance:          Int,          // 19:  in miles
                  canceled:          Int,          // 22:  was the flight canceled?
                  cancellationCode:  String,       // 23:  reason for cancellation (A = carrier, B = weather, C = NAS,
                  diverted:          Int,          // 24:  1 = yes, 0 = no
                  carrierDelay:      Int,          // 25:  in minutes
                  weatherDelay:      Int,          // 26:  in minutes
                  nasDelay:          Int,          // 27:  in minutes
                  securityDelay:     Int,          // 28:  in minutes
                  lateAircraftDelay: Int           // 29:  in minutes
                  )  // No class body needed. What methods are added automatically by "case"?

object Flight {
```

```
object SparkDataFrames {

  var out = Console.out   // Overload for tests
  val quiet = false

  def main(args: Array[String]): Unit = {
    val conf = new SparkConf()
    conf.setMaster("local[*]")
    conf.setAppName("Spark DataFrames")
    // Change to a more reasonable default number of partitions for our data
    // (from 200)
    conf.set("spark.sql.shuffle.partitions", "4")
    conf.set("spark.app.id", "SparkDataFrames")   // To silence Metrics warning.
    val sc = new SparkContext(conf)
    val sqlContext = new SQLContext(sc)
    import sqlContext.implicits._  // Needed for column idioms like $"foo".desc.

    try {
      val flightsPath = "data/airline-flights/alaska-airlines/2008.csv"
      val airportsPath = "data/airline-flights/airports.csv"
      // Don't "pre-guess" keys; just use the types as Schemas.
      val flightsRDD = for {
        line <- sc.textFile(flightsPath)
        flight <- Flight.parse(line)
      } yield flight

      val airportsRDD = for {
        line <- sc.textFile(airportsPath)
        airport <- Airport.parse(line)
```

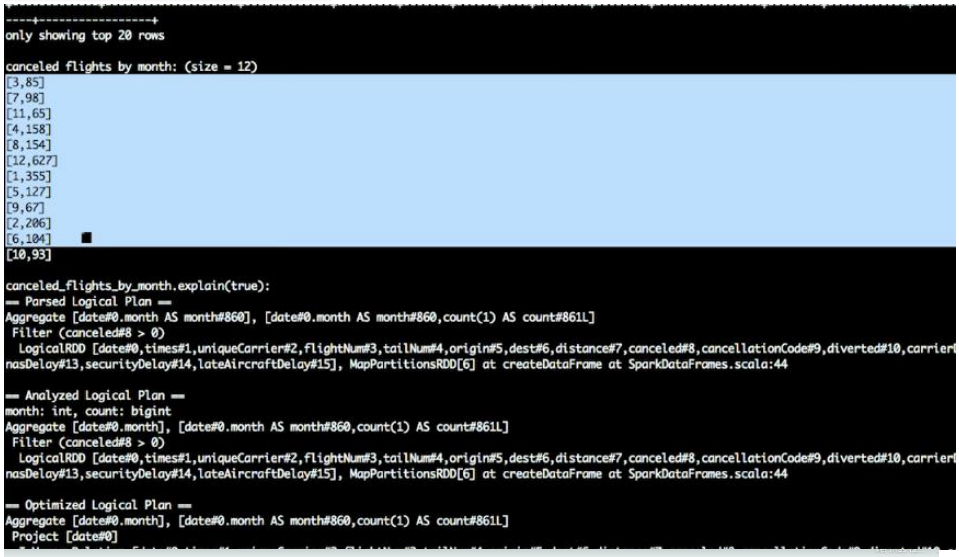

```
} yield flight

val airportsRDD = for {
  line <- sc.textFile(airportsPath)
  airport <- Airport.parse(line)
} yield airport

val flights  = sqlContext.createDataFrame(flightsRDD)
val airports = sqlContext.createDataFrame(airportsRDD)
// Cache just the flights and airports.
flights.cache
airports.cache

// SELECT COUNT(*) FROM flights f WHERE f.canceled > 0;
val canceled_flights = flights.filter(flights("canceled") > 0)
Printer(out, "canceled flights", canceled_flights)
```

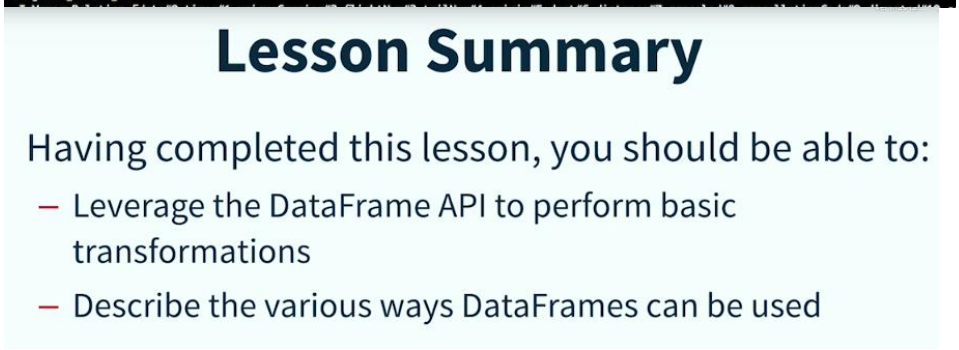
```
// SELECT COUNT(*) FROM flights f WHERE f.canceled > 0;
val canceled_flights = flights.filter(flights("canceled") > 0)
Printer(out, "canceled flights", canceled_flights)
if (!quiet) {
  out.println("\ncanceled_flights.explain(extended = false):")
  canceled_flights.explain(extended = false)
  out.println("\ncanceled_flights.explain(extended = true):")
  canceled_flights.explain(extended = true)
}
canceled_flights.cache
```

```
// Note how we can reference the columns several ways:
if (!quiet) {
  flights.orderBy(flights("origin")).show
  flights.orderBy("origin").show
  flights.orderBy($"origin").show
  flights.orderBy($"origin".desc).show
}
```

```
// SELECT cf.date.month AS month, COUNT(*)
//   FROM canceled_flights cf
//   GROUP BY cf.date.month
//   ORDER BY month;
val canceled_flights_by_month = canceled_flights.
  groupBy("date.month").count()
Printer(out, "canceled flights by month", canceled_flights_by_mor
if (!quiet) {
  out.println("\ncanceled_flights_by_month.explain(true):")
  canceled_flights_by_month.explain(true)
}
canceled_flights.unpersist
} finally {
  sc.stop()
}
```

```
Enter number: 10

[info] Running course2.module3.SparkDataFrames
[Stage 0:>                                                  (0 + 2) / 2]
```

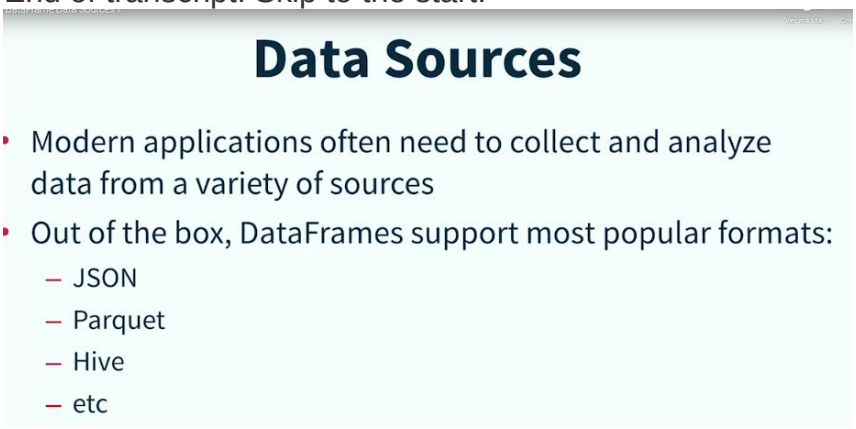


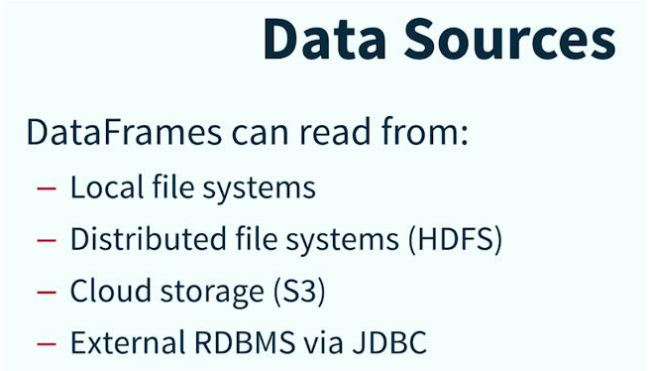


e end.
welcome to exploring the data frame API after completing this lesson you should
be able to leverage the data frame API to perform basic transformations and describe the various ways data frames can be used data frames have been very
concise API and common operations can be expressed concisely you can select
required columns join different data sources aggregated data using tools like counter some or average and you can filter Davis well here's an example of a
machine learning pipeline and you'll notice that we start out with some data said 0 we're going to token eyes that data resulting in a new dataset and then
we're gonna perform hashing transformer which is going to take a set of terms
and convert those set into a fixed length feature vector so this is an example coming from the spark documentation that results in a new dataset dataset to upon which we can perform some sort of linear regression which is used in machine learning to predict output for a new data based on previous data sets this in turn results in a final dataset called dataset three

let's look at some code for how we might use data frames to perform some data
analysis in this example we would like to show the number of canceled flights
in airline might have during a given month based on historical data we're going to use several inputs first of all Alaska Airlines has published information about their flights from 2008 in a comma-separated values format
we also have information about airports where they are
what they're called and their latitude and longitude we need to take that data from those comma separated value files and ingested into a data representation
for flights and airlines
and airports and this data has already been created in types that are defined in our airline . Scala class were first going to say that we have a quiet mode so that we can print out information when we want to debug but not printed out whenever we're just running this particular transformation in order to perform the data transformation we want to show the cancelled flights for this
airline during all months of 2008 next we need to set up our spark configuration and we're going to make this my local mode we're gonna call this
application spark data frames were going to use for partitions and then we're going to create a spark context using this configuration and create a sequel context from the spark context from which we will have access to the data frames API next we're going to say that we have our data in these files first there's the flight's path and secondly there's the airports path and then we're going to ingest the data where we get an RDD from having read and parse that file
data for flights and airports once we have the RDD representations of the data
and they've been ingested from the files we can then create the data frames for
boat and once we've done that we cash the data because we are not specifying
any parameter we know that we are placing this data in memory for quick access
next we want to filter out only the flights were canceled so that we don't try to process information that doesn't represent data we care about if we are
in quiet mode will ignore these out statements that are going to print out information for us by default I've set this to true so that we don't overload the amount of information that were printing out to the console and once we have our canceled flights with cash that data in memory as well
finally we go through the transformation to figure out the cancelled flights by month the first thing we do is we group that data by its date and month nation and we get a count once we done that we can explain that data which is

an API call from the data frame to show information about the way this data is
being transformed its information about the logical and physical plan that spark
is generating to perform this work and once we finished we on persist the data
and finally we stop the spark context if we run this application you'll notice that we're going to start up the application is going to start printing
out information about the data that is being loaded it's going to show the physical and logical plan information and then finally it's going to give us the information that we wanted which were what flights were canceled in which
month where the first number represents the month of the year and a one base
number representing January 2 12 being December and then the number of flights
were canceled for that month in 2008 on Alaska Airlines you'll notice from having done this that we see that the month of December has the highest number
of cancellations probably reflecting the number of flights that were being made
around holiday travel
having completed this lesson you should be able to leverage the data frame API
to perform basic transformations and describe the various ways data frames can be used
End of transcript. Skip to the start.





Scala APACHE Spark COGNITIVE CLASS.ai

Data Sources

- DataFrames use SparkSQL's external data sources API
- DataFrames can be extended to support any third-party data formats or sources
  - Avro
  - CSV
  - ElasticSearch
  - Cassandra
  - etc.

Data Sources

Formats and Sources supported by DataFrames

```
1  package course2.module3
2
3  import org.apache.spark.rdd.RDD
7
8  object DataFrameWithJson {
9
10   var out = Console.out
11   def main(args: Array[String]): Unit = {
12     val conf = new SparkConf()
13     conf.setMaster("local[*]")
14     conf.setAppName("Spark DataFrames")
15     // Change to a more reasonable default number of partitions for our data
16     // (from 200)
17     conf.set("spark.sql.shuffle.partitions", "4")
18     conf.set("spark.app.id", "DataFrameWithJson")   // To silence Metrics warning.
19     val sc = new SparkContext(conf)
20     val sqlContext = new SQLContext(sc)
21
22     try {
23
24       //each line should have a complete json record
25       val json: DataFrame =
26         sqlContext.read.json("data/airline-flights/carriers.json")
27
28       //spark infers the schema as it reads the json document. Since there is a invalid
29       //json record the schema will have an additional column called _corrupt_record
30       //for invalid json record.
31       // It doesn't stop the processing when it finds an invalid records which is great for
32       //large jobs as you don't want to stop for each invalid data record
```

```
    //spark infers the schema as it reads the json document. Since there is a
    //json record the schema will have an additional column called _corrupt_r
    //for invalid json record.
    // It doesn't stop the processing when it finds an invalid records which
    //large jobs as you don't want to stop for each invalid data record
    json.printSchema()
    Printer(out, "Loaded carrier information", json)


    //Printing out the records that failed to parse
    json.where(json("_corrupt_record").isNotNull).collect().foreach(println)
  } finally {
    sc.stop()
```

```
Enter number: 8

[info] Running course2.module3.DataFrameWithJson
```

Lesson Summary

Having completed this lesson, you should be able to:

- Describe the various data sources supported by DataFrames
- Discuss how to read and write JSON data with DataFrames

welcome to data frame data sources one after completing this lesson we should
be able to describe the various data sources supported by data frames and discuss how to read and write JSON data with data frames modern applications
often need to collect and analyze data from a variety of sources out of the box
spark data frame sport most popular formats such as JSON Park a hive and more spark data frames can read from local filesystems distributed file system such as the Hadoop distributed file system commonly called HDFS cloud
storage such as Amazon's s3 or external relational database formats by a JDBC
spark data frames use Park SQL as external data sources API data frames can be extended to support any third party data formats or sources such as a
ver o for comma-separated values or elasticsearch or Cassandra and more and here's an example of all the data sources that could be supported and more
you could use park a hive MySQL Postgres JSON s3 OpenStack HDFS the shows the
flexibility of the data frame data sources API let's reform an example where we're going to load JSON data in the form of carrier information for airlines in the spark application we create to load this JSON data we're going to do the exact same thing that we did in the previous lesson we create a
spark configuration and we make it run in local mode called again sparked data
friends and we make it run on four different partitions we then load the configuration into a spark context and create an SQL context from that sparked
context we just created once we've done mad we can define that we have a JSON
data frame and read in that data from the file by expressing exactly where the
fire was located now sparking hampered the scheme up the JSON
as it is reading the JSON document and remember the first record was invalid

because it did not have a closing of the record it doesn't stop processing whenever it finds invalid records which is great for large jobs however you do
need to make sure that you know that the invalid data was not processed as a
result at the end we can print out information for records that were not processed before we stop the spark context and shut down and when we run
the application we're going to load the JSON data as we didn't side that source
file in your notice that does print out the one record which did not meet the required JSON format because it was invalid
you also notice that the beginning of each record there is a delimiter that shows whether or not this was a corrupt record it also shows the carrier code
and the description of the carrier and we print out about fifty of the different carriers that were loaded out of the almost 1,500 having completed this lesson you should be able to describe the various data sources supported by data frames and discuss how to read and write JSON dado using data
frames

How to Add a New Data Source

- We mentioned in the previous lesson that you can add data sources
- We are going to add the Spark CSV package from Databricks, which can be found here:

https://github.com/databricks/spark-csv

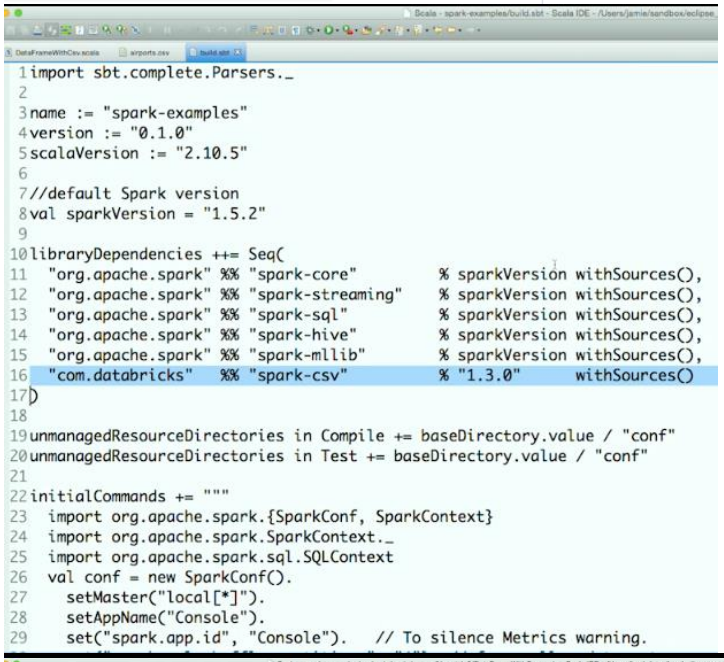
spark-packages.org

Adding Packages

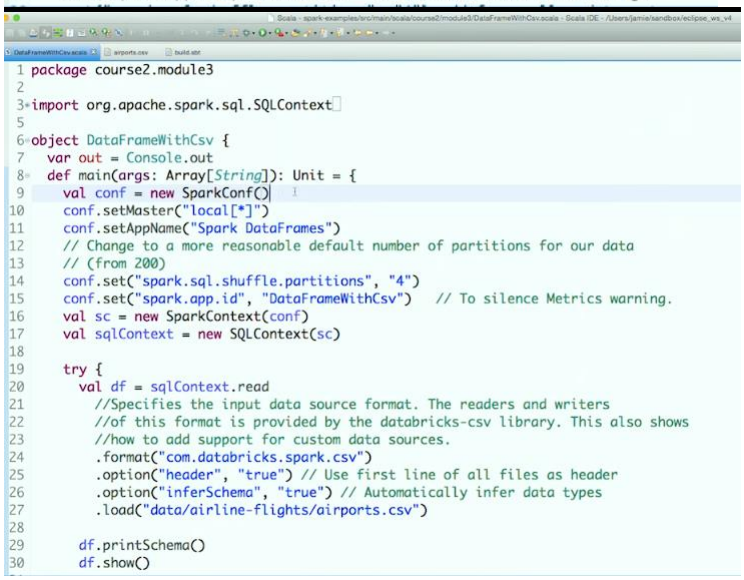
We can add a dependency to the project build file

If we are trying to add the dependency in a Spark REPL session, we can do so using the `packages` flag:

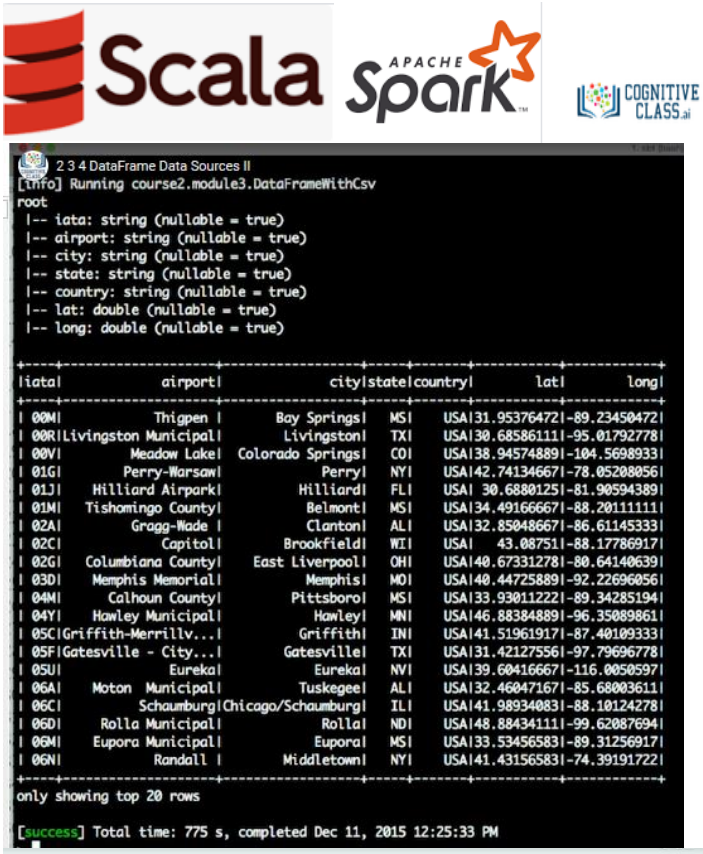
```
--packages com.databricks:spark-csv_2.11:1.3.0
```



```
import sbt.complete.Parsers._

name := "spark-examples"
version := "0.1.0"
scalaVersion := "2.10.5"

//default Spark version
val sparkVersion = "1.5.2"

libraryDependencies ++= Seq(
  "org.apache.spark" %% "spark-core"         % sparkVersion withSources(),
  "org.apache.spark" %% "spark-streaming"    % sparkVersion withSources(),
  "org.apache.spark" %% "spark-sql"          % sparkVersion withSources(),
  "org.apache.spark" %% "spark-hive"         % sparkVersion withSources(),
  "org.apache.spark" %% "spark-mllib"        % sparkVersion withSources(),
  "com.databricks"   %% "spark-csv"          % "1.3.0"      withSources()
)

unmanagedResourceDirectories in Compile += baseDirectory.value / "conf"
unmanagedResourceDirectories in Test += baseDirectory.value / "conf"

initialCommands += """
  import org.apache.spark.{SparkConf, SparkContext}
  import org.apache.spark.SparkContext._
  import org.apache.spark.sql.SQLContext
  val conf = new SparkConf().
    setMaster("local[*]").
    setAppName("Console").
    set("spark.app.id", "Console").   // To silence Metrics warning.
```



```
package course2.module3

import org.apache.spark.sql.SQLContext

object DataFrameWithCsv {
  var out = Console.out
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf()
    conf.setMaster("local[*]")
    conf.setAppName("Spark DataFrames")
    // Change to a more reasonable default number of partitions for our data
    // (from 200)
    conf.set("spark.sql.shuffle.partitions", "4")
    conf.set("spark.app.id", "DataFrameWithCsv")   // To silence Metrics warning.
    val sc = new SparkContext(conf)
    val sqlContext = new SQLContext(sc)

    try {
      val df = sqlContext.read
        //Specifies the input data source format. The readers and writers
        //of this format is provided by the databricks-csv library. This also shows
        //how to add support for custom data sources.
        .format("com.databricks.spark.csv")
        .option("header", "true") // Use first line of all files as header
        .option("inferSchema", "true") // Automatically infer data types
        .load("data/airline-flights/airports.csv")

      df.printSchema()
      df.show()
```



Lesson Summary

- Having completed this lesson, you should be able to:
  - Discuss how to add support for a new data source

nscript. Skip to the end.
welcome to data frame data sources to after completing this lesson you should
be able to discuss how to add support for a new data source we mentioned in
the previous lesson that you can add other data sources than the ones that come to you from spark out of the box we're going to add the spark comma-separated values package that is created by data bricks which can be
found here you go to get have dot com and go to the data bricks namespace you'll find the repository for spark dash csd you could also find spark CSV at spark dash packages dot org
which is a community index of packages for spark within a dependency to the
project build file for applications if we are trying to add the dependency in a spark rubble session we can do so using the packages flag when we create the
rebel in this example we are going to take an existing spark project and add the spark CSV dependency to it you have to work inside of a build file and if you are using SBT you add the group name comdata breaks the artifactId spark CSV
and then the version that you want
one-dot 320 in this case and you can specify optionally that you want the sources versions of you can see the source code if need be we are going to read in a group of airport information in this case you'll see the top line

includes a header header describes each one of the columns inside of the CSV
file
the first column is the IATA acronym for the airport and then the airport named
the city the state the country and the latitude and longitude of that airport
when we look at the source code for this application where we're pulling in the
new data source will once again create our spark configuration and we run in
local mode calling this application spark data frames were going to run with four partitions once again and then we're going to create our spark context using the configuration and create the sequel context from that spark on text when we read in the data from the comma separated values file we specify
the name of the file that we wanted the end and give it any options that we want
from beforehand for example the format is listed as CSV and we say whether or
not there's a header at the beginning of the file and whether or not to infer the
schema we then print out the scheme I we show it and we stopped the spark context
when we run this application we see that the data frame with CSV file is going to
be run and then you see the schema of the comma separated values being shown
when the data is displayed it shows it in a meaningful way that is pretty printed so you can see the IATA acronym the airport the city the state the country in the latitude and longitude clearly the show will only show the top 20 rows unless you specify otherwise
having completed this lesson you should be able to discuss how to add support
for a new data source in spark
End of transcript. Skip to the start.

Optimized DataFrame Usage

The main goal of the DataFrame API is to create and run Spark programs faster

- Write less code
- Read less data
- Let the optimizer do the hard work

Reading Less Data

The fastest way to process a lot of data is to never read it

SparkSQL helps you do this in several ways

Using Columnar Formats

Data we ingest can frequently contain information that we do not need for our analysis

We can use columnar formats such as Parquet to easily skip columns we do not care about

Using Partitioning

For example: `/year=2014/month=02/...`

This is a great technique for reading less data

If you store data in folders (like `year` and `month` above), the DataFrame API can easily skip folders that contain data you do not need

Using Statistics

For some data types, you can use statistics, such as `min` and `max`, to help you skip data

You can provide a predicate (`x => x > minval`) that allows the DataFrame to automatically skip files entirely

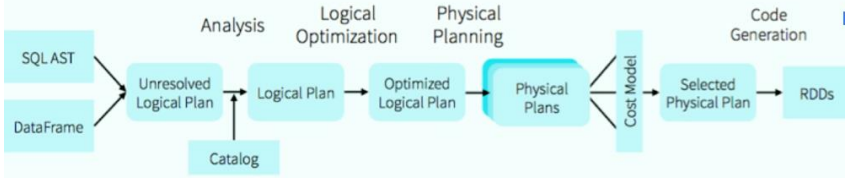
Pushing Predicates to Storage

Depending on the capabilities of the data source, DataFrames can push a predicate (i.e. - `x => x > minVal`) to the data source so that not all data is ingested

If you are trying to filter with the equivalent of an SQL where clause, you can use this feature to only return data that meets this condition

Plan Optimization

- Happens as late as possible, so Spark can optimize even across functions



Performance Tuning Options

Caching table data or the DataFrame in memory

```
sqlContext.cacheTable("tableName")
dataFrame.cache()
```

SparkSQL will scan only the required columns and automatically tune compression to minimize memory usage and garbage collection pressure

spark.sql.autoBroadcastJoinThreshold

- Configures the maximum size in bytes for a table that will be broadcast to all worker nodes when performing a join
- Not applicable to all operations, so check performance with the Spark Console

Lesson Summary

Having completed this lesson, you should be able to:

- Discuss how to create optimized data transformations using DataFrames
- Describe how to limit the amount of data that is read in
- Discuss basic optimization strategies

he end.
welcome to speeding up with data frames after completing this lesson you should
be able to discuss how to create optimized data transformations using data frames describe how to limit the amount of data that is read in and discuss basic optimization strategies with data frames the main goal of the

data frame API is to create and run sparked programs faster we do this by writing less code reading less data and allowing our optimizer to do the hard work for us the fastest way to process a lot of data is to never read it in spark
SQL helps you do this in several ways we can convert to more efficient formats
for example we can ingest clickstream data as JSON and converted to a binary
format such as averell the averell format is more compressed and easier to ship to other data processing components we can use columnar formats data we
ingest can frequently contain information that we do not need for our analysis when we use columnar formats such as parquet we can easily skip columns we do not care about you can partition our data for example year being 2014 month equal the second month of the year and this is a great technique for reading less data if you store data in folders such as year or month above the data frame API can easily skip folders that contain data you do not need and we can use statistics for some data types you can use stats such as men or max to help you skip data you can provide a predicate
which is a statement that results in a boolean value such as this
X where the X is greater than the minimum value and that allows the data frame to automatically skip files entirely we can also push these predicates to the storage depending on the capabilities of the data source data
frames can push that predicate like we saw before to the data source so that not all data is returned if you're trying to filter with the equivalent of SQL where
cause you can use this feature to only return dated that meets this condition plan optimization and spark happens as late as possible so spark an optimized
even across multiple functions here you see we have our SQL AST the abstract
syntax tree and the data frame and we get an unresolved logical plan which is
then catalog for analysis we get a logical plan as a result of that upon which we can perform optimizations of the logical plan we then create a physical plant for execution and cost model analysis is performed to select the appropriate that's cool plan resulting in the RDD that comes back
other performance tuning options come from cashing table data or data frames
in memory you can use this using the cash table method call giving the table name or calling cash on the data frame itself spark a skewer will scan only the
required columns and automatically tune compression to minimize memory usage and
garbage collection pressure we can also use the audio broadcast joined threshold
which configures the maximum size in bytes for a table that will be broadcast

to all worker nodes when performing a join its not applicable on all operations so check performance with these Park console to see if it helps you can also use the shuffling of partitions this configures the number of partitions to use when shuttling data for joins our aggregations based on the size of the cluster the number of partitions can improve the performance of the job
having completed this lesson we should be able to discuss how to create optimized data transformations using data frames describe how to limit the amount of data that is read in and discuss basic optimization strategies
End of transcript. Skip to the start.

4. Avdaced Spark Topics

DataFrame Joins
After completing this lesson, you should be able to:
- Describe the different kinds of joins supported by DataFrames
- Explain how to write SQL queries
- Discuss performance considerations

Other Transformations
After completing this lesson, you should be able to:
- Describe other available transformations, including:
  - agg
  - sample
  - randomSplit

Using Hive with Spark SQL
After completing this lesson, you should be able to:
- Describe how to create and drop Hive tables with Spark SQL
- Discuss how to run Hive queries
- Explain how to use the DataFrame API with query results

Spark Streaming
After completing this lesson, you should be able to:
- Describe how Spark Streaming works
- Explain the concept of DStreams, the core abstraction in Spark Streaming

Data Frames and Spark Streaming: Hive ETL
After completing this lesson, you should be able to:
- Describe how to combine Spark Streaming and Hive to implement an ETL workflow

Scala APACHE Spark COGNITIVE CLASS.ai

Lesson Objectives

After completing this lesson, you should be able to:

- Describe the different kinds of joins supported by DataFrames
- Explain how to write SQL queries
- Discuss performance considerations

Joins

In the Spark DataFrames example in previous lessons, we loaded two data sets for commercial flights and airports

Let's join them (and skip some now-familiar boilerplate for creating config, SparkContext, etc)

```
val af = "data/airline-flights"
val flightsPath = s"$af/alaska-airlines/2008.csv"
val airportsPath = s"$af/airports.csv"
val flightsRDD = for {
  line <- sc.textFile(flightsPath)
  flight <- Flight.parse(line)
} yield flight

val airportsRDD = for {
  line <- sc.textFile(airportsPath)
  airport <- Airport.parse(line)
} yield airport
```

Load the data, parse it, and construct the RDDs, as before

```
val flights  = sqlContext.createDataFrame(flightsRDD)
val airports = sqlContext.createDataFrame(airportsRDD)
flights.cache
airports.cache

println("Flights schema:")
flights.printSchema
println("Flights, first 10 records:")
flights.show(10)    // Omit the argument: default is 20

println("Airports schema:")
airports.printSchema
println("Airports, first 10 records:")
airports.show(10)
```

"Wrap" in DataFrames and cache them.

```
root
 |-- date: struct (nullable = true)
 |     |-- year: integer (nullable = false)
 |     |-- month: integer (nullable = false)
 |     |-- dayOfMonth: integer (nullable = false)
 |     |-- dayOfWeek: integer (nullable = false)
 |-- times: struct (nullable = true)
 |     |-- depTime: integer (nullable = false)
 |     |-- crsDepTime: integer (nullable = false)
 |     |-- arrTime: integer (nullable = false)
 |     |-- crsArrTime: integer (nullable = false)
 |     |-- actualElapsedTime: integer (nullable = false)
 |     |-- crsElapsedTime: integer (nullable = false)
 |     |-- airTime: integer (nullable = false)
 |     |-- arrDelay: integer (nullable = false)
 |     |-- depDelay: integer (nullable = false)
```

Flights schema

```
root
 |-- iata: string (nullable = true)
 |-- airport: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- country: string (nullable = true)
 |-- latitude: float (nullable = false)
 |-- longitude: float (nullable = false)
```

Airports schema

```
val flights_between_airports =
  flights.select($"origin", $"dest").
    groupBy($"origin", $"dest").count().
    orderBy($"count".desc, $"origin", $"dest")
```

Query from before.

```
+------+----+-----+
|origin|dest|count|
+------+----+-----+
|   ANC| SEA| 5536|
|   SEA| ANC| 5534|
|   LAX| SEA| 4692|
|   SEA| LAX| 4681|
|   ANC| FAI| 3217|
|   SEA| SFO| 2869|
+------+----+-----+
...
```

```
flights_between_airports_sql.cache
flights_between_airports_sql.registerTempTable("fbasql")

val fba = flights_between_airports
val air = airports
```

Prep for the next query...

Convenient aliases (shorter!)

Let's join "air" twice against "fba" to get the "origin" and "dest" airport names.

```
val fba2 = fba.
  join(air, fba("origin") === air("iata")).
    select("origin", "airport", "dest", "count").
    toDF("origin", "origin_airport", "dest", "count").
  join(air, $"dest" === air("iata")).
    select("origin", "origin_airport",
      "dest", "airport", "count").
    toDF("origin", "origin_airport",
      "dest", "dest_airport", "count")
```

The "ON" clause. Note the ===

```
t", "count").
  "dest", "count").
.
",

")
```

Effectively aliases all the columns

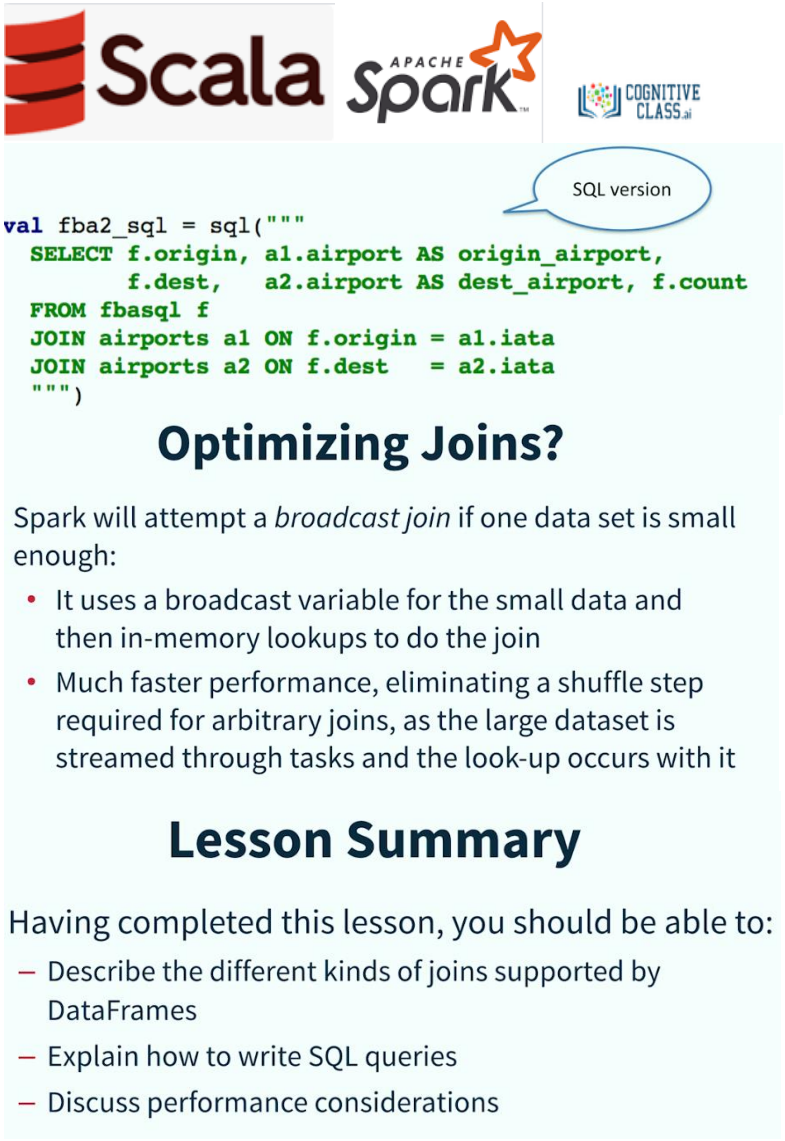
Due to a current bug, you MUST use toDF for intermediate joins.

fba2's schema

```
root
 |-- origin: string (nullable = true)
 |-- origin_airport: string (nullable = true)
 |-- dest: string (nullable = true)
 |-- dest_airport: string (nullable = true)
 |-- count: long (nullable = false)
```

```
+------+--------------------+----+--------------------+-----+
|origin|      origin_airport|dest|        dest_airport|count|
+------+--------------------+----+--------------------+-----+
|   ANC|Ted Stevens Ancho...| SEA| Seattle-Tacoma Intl| 5536|
|   SEA| Seattle-Tacoma Intl| ANC|Ted Stevens Ancho...| 5534|
|   LAX|Los Angeles Inter...| SEA| Seattle-Tacoma Intl| 4692|
|   SEA| Seattle-Tacoma Intl| LAX|Los Angeles Inter...| 4681|
|   ANC|Ted Stevens Ancho...| FAI|Fairbanks Interna...| 3217|
|   SEA| Seattle-Tacoma Intl| SFO|San Francisco Int...| 2869|
|   SFO|San Francisco Int...| SEA| Seattle-Tacoma Intl| 2866|
|   SNA|John Wayne /Orang...| SEA| Seattle-Tacoma Intl| 2862|
|   SEA| Seattle-Tacoma Intl| SNA|John Wayne /Orang...| 2860|
|   FAI|Fairbanks Interna...| ANC|Ted Stevens Ancho...| 2853|
+------+--------------------+----+--------------------+-----+
only showing top 10 rows
```

fba2's first 10 rows



transcript. Skip to the end.
welcome to data frame joins after completing this lesson you should be able to describe the different kind of joint supported by data frames explain how to write SQL queries and discuss the performance considerations involved in
the spark data frames example in previous lessons we loaded two datasets for commercial flights and airports this time let's join them and skip over some
now-familiar boilerplate for creating the configuration creating the spark context and more
first we're going to load the data are set and construct the rd's as we did before we get there comma-separated values from the various files representing the flights and the airports and then we load them into our DD's we then wrap the rd's as data frames and cash them and we can print the schemas and the first 10 records of each this is the flight schema it's quite long so in this case on the slide where only showing the first few items inside of it including the date and the times but this is the data that resulted from ingestion where you see the date and then the time information first as well as unique carrier identifier the flight number tail number
origin destination distance etc
this is the airport schema with the IATA code the airport named the city the state the country and the latitude and longitude and when we ingest the data

the first 10 rows look like this from before we had a query where we selected on the origin and destination and then grouped together account and then ordered where the count was to sending the sequel version of the statement would look like this in this case we're going to get the flights between airports and prep this query where we're going to use aliases to make our code a
little easier to read on these small slides will call flights between airports FBA and will call the airport's air now let's join the airports twice against the flights
between airports to get the origin and destination airport names we can join our flights between airports as a second value where we joined on the original
flights between airports on the origin where the IATA is equal to that airport we then join to get the origin origin airport the destination to destination airport and the count note that the triple equals is the ON clause and we project the columns that we want this effectively aliases all of the columns but due to a current bug in spark you must use the two data frame for intermediate joins as you see at the end of each joint clause here the second
joint for destination followed by a projection and a leasing the schema for the second flights between airports representation shows that we have the origin the origin airport the destination and the destination airport
as well as the count and the resulting data looks like this and its first 10 rows the sequel version of the statement looks like this where you see us selecting from two different tables the origin the airport the destination and that airport and then the count and then we use a left outer join to return all their boards even if they don't have a flight spark war tempt a broadcast join if one dataset is small know it uses a broadcast variable for the small data and then in memory lookups to do the join this gives you much faster performance eliminating a shuffle step required for arbitrary joins as the large datasets stream through tasks and the lookup occurs with it we can set a
threshold using the property audio broadcast joined threshold where the default is 10 megabytes but please also see the spark documentation for other
configuration options
having completed this lesson we should be able to describe the different kinds of joint supported by data frames explain how to write SQL queries and
discuss the performance considerations
End of transcript. Skip to the start.

Lesson Objectives

After completing this lesson, you should be able to:

- Describe other available transformations, including:
  - agg
  - sample
  - randomSplit

agg

agg is the general method for computing aggregations

df.agg(…) is shorthand for df.groupBy(…).agg(…)

```
flights.registerTempTable("flights")

flights.agg(
  min($"times.actualElapsedTime"),
  max($"times.actualElapsedTime"),
  avg($"times.actualElapsedTime"),
  sum($"times.actualElapsedTime")).show()

sql("""
  SELECT
    MIN(times.actualElapsedTime) AS min,
    MAX(times.actualElapsedTime) AS max,
    AVG(times.actualElapsedTime) AS avg,
    SUM(times.actualElapsedTime) AS sum
  FROM flights
  """).show()
```

Work with just the Flights data. Register as a temporary table, as before.

Same results for both queries.

```
+------+------+------------------+--------+
|min(…)|max(…)|           avg(…)|  sum(…)|
+------+------+------------------+--------+
|    -1|   535|150.49712114995168|22740416|
+------+------+------------------+--------+
```

```
flights.agg(count($"*")).show
flights.agg(count($"tailNum")).show
flights.agg(countDistinct($"tailNum")).show()
flights.agg(approxCountDistinct($"tailNum")).show()

sql("SELECT COUNT(*) AS count FROM flights").show()
sql("SELECT COUNT(tailNum) AS count FROM flights").show()
sql("SELECT COUNT(DISTINCT tailNum) AS cd FROM flights").show
```

Same, because no tailNum values NULL.

SQL equivalents. (No count approximate supported.)

```
flights.agg(count($"*")).show
flights.agg(count($"tailNum")).show
```

```
+---------------+
|count(tailNum)|
+---------------+
|         151102|
+---------------+
```

Same as COUNT(*)

```
flights.agg(count($"*")).show
flights.agg(count($"tailNum")).show
flights.agg(countDistinct($"tailNum")).show()
flights.agg(approxCountDistinct($"tailNum")).show()
```

```
+----------------------------------+
|APPROXIMATE COUNT(DISTINCT tailNum)|
+----------------------------------+
|                               122|
+----------------------------------+
```

Exact count was 126

```
flights.agg(count($"*")).show
flights.agg(count($"tailNum")).show
flights.agg(countDistinct($"tailNum")).show()
flights.agg(approxCountDistinct($"tailNum")).show()

sql("SELECT COUNT(*) AS count FROM flights").show()
sql("SELECT COUNT(tailNum) AS count FROM flights").show()
sql("SELECT COUNT(DISTINCT tailNum) AS cd FROM flights").show
```

```
+------+
|countd|
+------+
|   126|
```

Cubes and Rollups

In data warehouses, it's common to construct multidimensional cubes and rollups for analytics, such as aggregations, in addition to using `groupBy`

The available functions are provided by GroupedData (spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.GroupedData)


```
flights.cube("origin", "dest").avg().show()
```

Compute averages for all numeric fields, for each origin, dest pair.

```
+------+----+------------------+------------------+--------------
|origin|dest|    avg(flightNum)|     avg(distance)|       avg(can
+------+----+------------------+------------------+--------------
|   PDX| BOS|276.78688524590166|            2537.0| 0.00819672131
|   SEA| OGG|             857.0|            2640.0|0.005952380952
|   CDV| YAK|              66.0|             213.0|0.019337016574
|  null| ORD|52.504455106237145|2003.2138450993832| 0.01028101439
...
```

Why null??

```
sql("""
  SELECT origin, dest,
  AVG(flightNum),
  AVG(distance),
  AVG(canceled),
  AVG(diverted),
  AVG(carrierDelay),
  AVG(weatherDelay),
  AVG(nasDelay),
  AVG(securityDelay),
  AVG(lateAircraftDelay)
  FROM flights
  GROUP BY origin, dest
  """).show()
```

Similar SQL (note the GROUP BY), but you have to name the columns in AVG explicitly.

```
flights.cube($"origin", $"dest").agg(Map(
  "*" -> "count",
  "times.actualElapsedTime" -> "avg",
  "distance" -> "avg"
)).orderBy($"avg(distance)".desc).show()
```

Use agg(Map()) to specify columns and functions to call. Convenient, but not flexible.

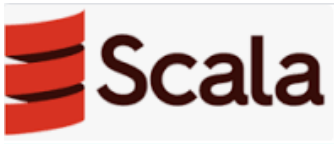
```
sql("""
  SELECT origin, dest,
    COUNT(*) AS count,
    AVG(times.actualElapsedTime) AS avg_time,
    AVG(distance) AS avg_dist
  FROM flights
  GROUP BY origin, dest
  ORDER BY avg_dist DESC
  """).show()
```

```
val dist_time = flights.cube($"origin", $"dest").agg(
  avg("distance").as("avg_dist"),
  min("times.actualElapsedTime").as("min_time"),
  max("times.actualElapsedTime").as("max_time"),
  avg("times.actualElapsedTime").as("avg_time"),
  (avg("distance")/avg("times.actualElapsedTime")).as("t_d")
).orderBy($"avg_dist".desc).cache
```

More flexible format.

```
val dist_time_sql = sql("""
  SELECT origin, dest,
    AVG(distance) AS avg_dist,
    MIN(times.actualElapsedTime) AS min_time,
    MAX(times.actualElapsedTime) AS max_time,
    AVG(times.actualElapsedTime) AS avg_time,
    (AVG(distance) / AVG(times.actualElapsedTime)) AS t_d
  FROM flights
  GROUP BY origin, dest
  ORDER BY avg_dist DESC
  """).cache
```

Corresponding SQL query




```
val dist_time_sql = sql("""
  SELECT origin, dest,
    AVG(distance) AS avg_dist,
    MIN(times.actualElapsedTime) AS min_time,
    MAX(times.actualElapsedTime) AS max_time,
    AVG(times.actualElapsedTime) AS avg_time,
    (AVG(distance) / AVG(times.actualElapsedTime)) AS t_d
  FROM flights
  GROUP BY origin, dest
  ORDER BY avg_dist DESC
  """).cache
```

Corresponding SQL query

```
val delays = flights.cube($"origin", $"dest").avg(
  "canceled",
  "weatherDelay"
).toDF("origin", "dest", "cancel
delays.orderBy($"canceled".desc).
```

ADK (Adak, Alaska) and ANC (Anchorage) in Alaska. You might expect frequent cancellations.

```
+------+----+--------------------+--------------------+
|origin|dest|            canceled|        weatherDelay|
+------+----+--------------------+--------------------+
|   ADK| ANC| 0.09803921568627451| -0.7745098039215687|
|  null| ADK| 0.09803921568627451|0.05882352941176470
|   ADK|null| 0.09803921568627451| -0.774509803
|   ANC| ADK| 0.09803921568627451|0.0588235294
|   PDX| BUR| 0.06666666666666667| -0.98333333
|   BUR| PDX| 0.06666666666666667| -0.91666666
...
```

PDX is Portland, OR and BUR is Burbank, CA.

```
sql("""
  SELECT origin, dest,
    AVG(canceled) AS canceled,
    AVG(weatherDelay) AS weatherDelay
  FROM flights
  GROUP BY origin, dest
  ORDER BY canceled DESC
  """).show
```

Corresponding SQL

Others

A few other important operations. (See the DataFrames Scaladocs):

- `sample` - Take a random sample of the data
- `randomSplit` - Split the data into 2 or more subsets

6:48 / 6:48

Press UP to enter the speed menu then use the UP and DOWN arrow keys to navigate the different speeds, then press ENTER to change to the selected speed.

Click on this button to mute or unmute this video or press UP or DOWN buttons to increase or decrease volume level.
Maximum Volume.
Video transcript

Start of transcript. Skip to the end.
welcome to other data frame transformations after completing this lesson you should be able to describe other available transformations on data frames including aggregation sample and random split aggregation is the general method for computing aggregations of data you can do an aggregation call use in the AGG method on the data friend and this is shorthand for calling group buy on the data frame and then aggregating that in this case we will work with just the flight's data and again we're going to register a temporary table we're going to aggregate all rose compute the minimum maximum average in some of the flights elapsed times this would be the equivalent SQL query and we get the same results for both we could count all roads we can count all tail numbers in our rose but since there's a tail number for every single one of the flights were going to get the exact same number in our count we could ask four distinct him numbers to tell us how many distinct planes were used for all of the routes and then we could ask a pretty approximation of how many distinct planes there might be an SQL we get very equivalent syntax however there is no count approximate supported and therefore we cannot show something

similar to the last line when we count all of the flights will get a hundred and fifty one thousand different flights and we count all the tail numbers we get the exact same number when we asked for the distinct number of town numbers we get only the number of planes involved in all of the different flights in this case a hundred and twenty-six but when we do an approximate count distinct we get one hundred and twenty-two which is closed useful with you don't want to go through the cost of counting every single one distinctly and you just want an approximation of what that value might be when we do the same thing using SQL we end up with the 151,000 distinct flights and we end up with the exact same number of towns because we have not yet said that we want to stink townhomes if we ask for the distinct helms once again we get a hundred and twenty six different planes used for all of the plates in data warehouses it's common to construct multi-dimensional cubes and roll ups R analytics such as aggregations in addition to using group by the available functions are provided by the group Data API which you see in the spark documentation under the spark SQL group data page in this case I'm going to use cube to compute averages for all numeric fields for each origin and destination pair and when we see the results of this we see the IATA code for the origin and destination airports as well as the average flight number the average distance the average canceled flights you may say why do we have an 04 origin IATA code and this case it may be from the dataset that we're using I could also do the same thing where I averaged over all destinations where the origin is lax or Los Angeles also note that in this case there is an old estimation which results in some invalid data similarly the SQL you use the GROUP BY clause as shown at the bottom of the SELECT statement but you have to name the columns to average explicitly when you use the data frames AP ionCube you could aggregate to a map we say a value to some function however this is convenient but not very flexible in this case I say for everything give me account for the times the actual apps time give me the average and for

the distance give me the average and then I want to order by the average distance in a descending order
it's more flexible for me to say that I went to cuba my data and result in an aggregation of the average distance which is aliased is average test the men
the actual elapsed time
alias Tasman time the max of the actual elapsed time aliased as Max time an
average of the actual elapsed time also aliased is average underscored time we
can then get the average distance in 25 by the average actual elapsed time and
call that te underscore d where the aliases or what show up when we show our
data at the end I don't order by the average distance in a descending order
and I cash the data which results in this I can also see the lowest average distance this would be the corresponding SQL query for what we just did we can
look at our origin and destination pairs with the most flight delays we can see
that going from addict alaska to Anchorage Alaska might result in
frequent cancellations because the weather in alaska can be very harsh will
also note that we see a flight going from Portland Oregon on the west coast
to the USS and burbank california also on the West Coast of the USA Mike
frequently be delayed weather delays are also common in places such as Minneapolis and Seattle also very northern in their location the corresponding SQL to do this work would look like this
a few other important operations can be found in the data frame scholar docs for
example there's the sample method we take a random sampling of the data or
random split where we split the data into two or more subsets having completed this lesson you should be able to describe other available transformations including aggregation sample and random split
End of transcript. Skip to the start.

Lesson Objectives

After completing this lesson, you should be able to:

- Describe how to create and drop Hive tables with Spark SQL
- Discuss how to run Hive queries
- Explain how to use the DataFrame API with query results

HiveContext

SparkSQL supports two SQL dialects:

- It's own, a subset of HiveQL
- HiveQL - integrated with Hive *metastores*

The HiveQL SQL is richer and is preferred when doing more extensive data processing with SQL from Spark

HiveContext

To use Hive, instantiate a `HiveContext` instead of a `SQLContext`

```
import org.apache.spark.sql.hive.HiveContext

val conf = new SparkConf()
...
val sc = new SparkContext(conf)
val hiveContext = new HiveContext(sc)
import hiveContext.implicits._
import hiveContext.sql
import org.apache.spark.sql.functions._  // for min, max, etc
```

Import the HiveContext

Create the HiveContext (like before)

```
val dir = "data/airline-flights/hive/airports"
val data = new java.io.File(dir)
val path = data.getCanonicalPath

sql(s"""
  CREATE EXTERNAL TABLE                orts (
    iata      STRING,
    airport   STRING,
    city      STRING,
    state     STRING,
    country   STRING,
    lat       DOUBLE,
    long      DOUBLE)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
  LOCATION '$path'
  """).show
  long      DOUBLE)
ROW FORMAT DELIMITED FIELDS TERMINATED B
LOCATION '$path'
""").show
```

Special version of airports data.

Hive wants the full path.

"External" means Hive doesn't own the data. It's in the location we specify.

```
sql(s"""
  CREATE EXTERNAL TABLE IF NOT EXISTS airports (
    iata      STRING,
    airport   STRING,
    city      STRING,
    state     STRING,
    country   STRING,
    lat       DOUBLE,
    long      DOUBLE)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
  LOCATION '$path'
  """).show
```

The columns

Comma separated text

Location

```
sql("SHOW TABLES").show
sql("DESCRIBE airports").show(100)
```

```
+--------+---------+-------+
|col_name|data_type|comment|
+--------+---------+-------+
|    iata|   string|   null|
| airport|   string|   null|
|    city|   string|   null|
|   state|   string|   null|
| country|   string|   null|
|     lat|   double|   null|
|    long|   double|   null|
+--------+---------+-------+
```

```
sql("SHOW TABLES").show
sql("DESCRIBE airports").show(100)
sql("DESCRIBE EXTENDED airports").show(100)
```

```
+--------------------+--------------------+-------+
|            col_name|           data_type|comment|
+--------------------+--------------------+-------+
|                iata|              string|   null|
|             airport|              string|   null|
|                city|              string|   null|
|               state|              string|   null|
|             country|              string|   null|
|                 lat|              double|   null|
|                long|              double|   null|
|Detailed Table In...|Table(tableName:a...|       |
+--------------------+--------------------+-------+
```

Use show(100,false) to see this detail

```
sql("SHOW TABLES").show
sql("DESCRIBE airports").show(100)
sql("DESCRIBE EXTENDED airports").show(100)
sql("DESCRIBE FORMATTED airports").show(100)
sql("DESCRIBE FORMATTED airports").show(100, truncate = false
sql("DESCRIBE FORMATTED airports").foreach(println)
```

the second show argument, truncate, causes wide output,but it's hard to read on a slide. Similarly, using DataFrame.foreach(println) might be more useful.

```
sql("""
  SELECT country, COUNT(*) FROM airports
  GROUP BY country
  """).show(100)
```

Airports per country.

```
+--------------------+----+
|             country| _c1|
+--------------------+----+
|            Thailand|   1|
|   N Mariana Islands|   1|
|Federated States ...|   1|
|                 USA|3372|
|               Palau|   1|
+--------------------+----+
```

Odd that a few non-US airports are in this data set!

```
sql("""
  SELECT iata, airport, country
  FROM airports WHERE country <> 'USA'
  """).show(100)
```

More data about the non-US airports

```
+----+--------------------+--------------------+
|iata|             airport|             country|
+----+--------------------+--------------------+
| ROP|         Prachinburi|            Thailand|
| ROR|    Babelthoup/Koror|               Palau|
| SPN|Tinian Internatio...|   N Mariana Islands|
| YAP|   Yap International|Federated States ...|
+----+--------------------+--------------------+
```

```
sql("""
  SELECT state, COUNT(*) AS count FROM airports
  WHERE country = 'USA'
  GROUP BY state
  ORDER BY count DESC
  """).show(100)
```

Count of US airports by state

```
+-----+-----+
|state|count|
+-----+-----+
|   AK|  263|
|   TX|  209|
|   CA|  205|
|   OK|  102|
|   FL|  100|
|   OH|  100|
+-----+-----+
...
```

```
val latlong = sql("SELECT state, lat, long FROM airports")
latlong.cube("state").agg(
  min("lat"),
  max("lat"),
  avg("lat"),
  min("long"),
  max("long"),
  avg("long")
```

Results are DataFrames, so let's use that API!

```
+-----+----------+----------+-----------------+-----------+-----------+------------------+
|state|  min(lat)|  max(lat)|         avg(lat)| min(long)| max(long)|         avg(long)|
+-----+----------+----------+-----------------+-----------+-----------+------------------+
|   OH|38.41924861|41.77797528|40.396679633400005| -84.7841425|-80.64140639| -82.88624561979998|
|   MN|43.59534389|48.94138889|       45.6859403|-96.94300306|-90.38313889| -94.22007025595508|
|   IN|38.01769694|41.71935833|40.163398602769234|-87.53062667|   -84.84282| -86.19105122276919|
+-----+----------+----------+-----------------+-----------+-----------+------------------+
...
```

DataFrame.rdd returns the RDD, so those methods can be called, too.

Lesson Summary

Having completed this lesson, you should be able to:

- Describe how to create and drop Hive tables with Spark SQL
- Discuss how to run Hive queries
- Explain how to use the DataFrame API with query results

t. Skip to the end.
welcome to using high with spark SQL after completing this lesson you should
be able to describe how to create and drop hype tables with spark SQL discuss
how to run high queries and explain how to use the data frame API with query
results sparked SQL supports two different SQL dialects there's its own SQL dialect which is a subset of high Q well but also hype QL integrated with
hype medicine stores the hype UL SQL is richer and his preferred when doing more
extensive data processing with SQL from spark to use hive instantiate a high of
context instead of an SQL context and here's an example of how we do that first of all we import the hype context and then as before we create a spark in
Fig
we create a spark context from that config and then instead of creating an SQL context we create a new high context from that spark context we can then
import some extra helpers which allow us to write code without having to import
them or mention them explicitly as far as their name space in this case we have
our airports data and we prepared a version of the comma separated values file with double quotes
extra commas and extra whitespace removed for a more complete comma-separated values parser use the one that we mentioned in an earlier lesson called spark csd from the data bricks packages project in this case we
have that special version of the airport's data which we've cleaned and high once the full path to that data when we call SQL in order to generate a prepared statement we declare a
then three quotes around the SQL statement that three quotes do know that this is going to be a multiline string and the ass in front of it means that we're using a feature of Scala called string interpolation to insert values

directly into the strength this means we don't have to use any percent as or format commands and we know that we're always putting the right value into our
string in a type Safeway we mentioned the columns that we want to leverage inside of our data and that our columns are separated by a comma we also provide
the location of where that data is located mostly this looks like a typical DDL data definition language statement from SQL accept higher ads expressions
for specifying file format information the row format delimited is a verbose way to say taxed with fields separated by comma but this is not a full-fledged CSV parser in this case the culture show returns an empty data frame we can do
more with our SQL statements for example we can call show tables and this will
show the table name and whether or not that table was temporary
can describe the information in the airports in this case we have the IATA the airport the city in the state in the country as well as the Latin long this is very much like describing a table inside of a relational database we can describe the extended airports where we see more information about what exists
inside of this data and we can also described the formatted airports however this might be hard to read in my second formatted airports I say show and also
only show one hundred possible items but then I say truncated equal to false this
could cause wide output but it can be very hard to read on the slides
as we're looking at right now
similarly we could use the higher-order function for each on data frame passing
the print lind function to it which might be more useful for showing us the data that we want in this statement we select account from the airports showing
us the number of airports there are inside of our dataset in this case we get a number of three thousand three hundred and seventy-six if I want to show airports per country I say select the country and the count from the airports grouping by the country and in this case it's odd that my data only shows a few other countries and very few airports outside of the USA I can get
more information about non-us airports with what I select where I say give me
specific information such as the IATA code the airport in the country from airports where the country is not equal to the USA this data may reflect an airline that doesn't have many flights that go outside of the us- we can get a count of all USA airports by state where we say select the state and the count as
count from airports whereas countered saying what am I going to call the column name of the data that results and I say where my country is equal to USA

but a group it by state and order by count descending the most airports actually show up in a state called alaska which is a very remote state so this is quite surprising from our dataset but given that this is Alaska Airlines it shouldn't be surprising because that's their home state from my data said if I would like to find out aggregations of men max an average latitude and longitude data I can do that by using cube and aggregated since the results are data frames we can use that API I can call the RDD method on a
day to frame which returns an RDD representation of the data so those methods and Matty P I could be called as well
all and if I say SQL DROP TABLE airports and empty data frame has returned which
in this case means that the operation was successful
will see a few other ways to use hybrid spark in the next few lessons
having completed this lesson you should be able to describe how to create drop
hype tables with spark SQL discuss how to run hide queries and explain how to
use the data frame API with query results
End of transcript. Skip to the start.

Why Spark Streaming?

What if you want to process data as soon as it arrives, within seconds or minutes, rather than wait for the next batch job?

Spark Streaming provides this capability

Why Spark Streaming?

Spark Streaming is a large topic with significant operational considerations

Here, we introduce the Spark Streaming model and how it can be useful

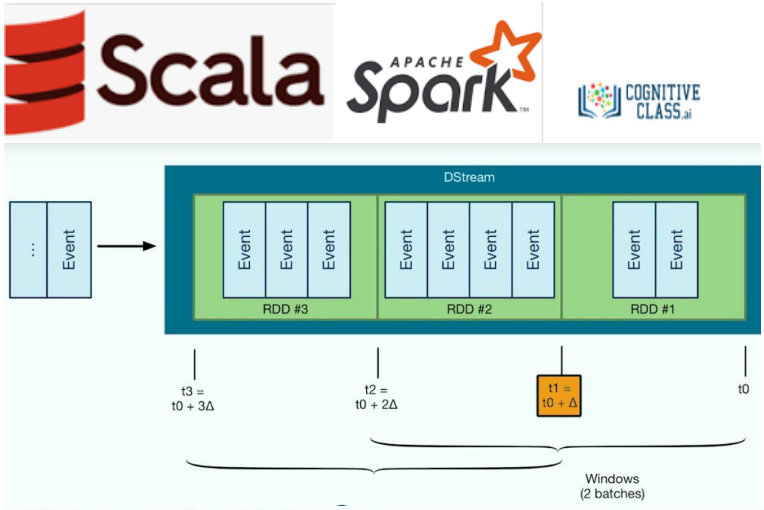
The next lesson illustrates how to use Spark Streaming and SparkSQL's Hive support together

Spark Streaming

Each *minibatch* is stored in an RDD

The sequence of RDDs is held in a DStream

DStream provides window functions




```
package course2.module4

import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.dstream.DStream
import org.apache.spark.streaming.Seconds
```

Package declaration and imports, including new streaming code

```
object Streams {

  val port = 9000
  val interval = Seconds(2)
  val pause = 10  // milliseconds
  val server = "127.0.0.1"
  val checkpointDir = "output/checkpoint_dir"
  val runtime = 30 * 1000   // run for N*1000 millisecs
  val numIterations = 100000
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf()
    conf.setMaster("local[*]")
    conf.setAppName("Spark DataFrames")
    conf.set("spark.sql.shuffle.partitions", "4")
    conf.set("spark.app.id", "Aggs")
    val sc = new SparkContext(conf)
```

constants we'll use below.

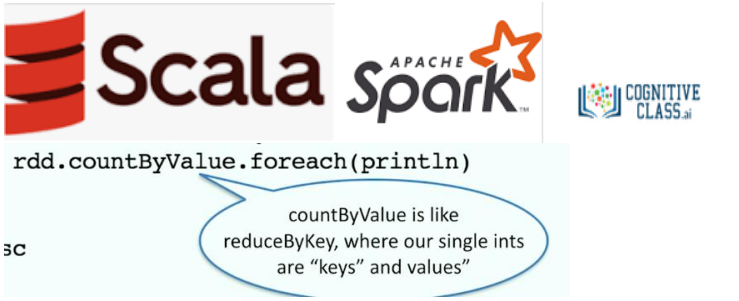
Initialize as usual...

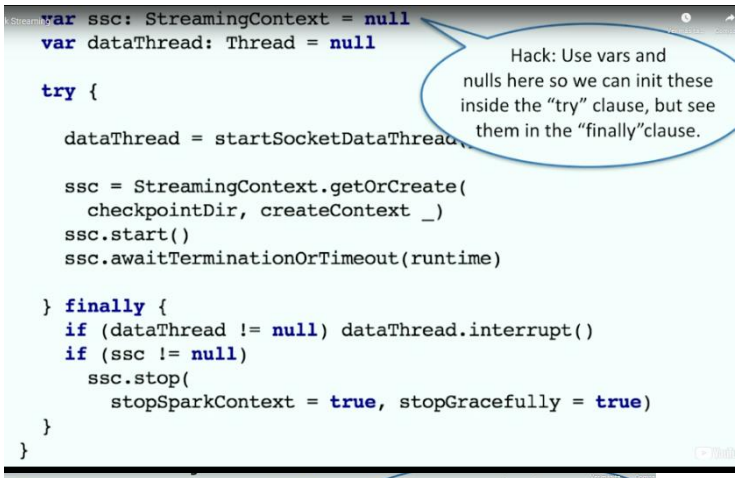
```
  def createContext(): StreamingContext = {
    val ssc = new StreamingContext(sc, interval)
    ssc.checkpoint(checkpointDir)

    val dstream = ssc.socketTextStream(
    val numbers = for {
      line <- dstream
      number <- line.trim.split("\\s+")
    } yield number.toInt
    numbers.foreachRDD { rdd =>
      rdd.countByValue.foreach(println)
    }

    ssc
  }
```

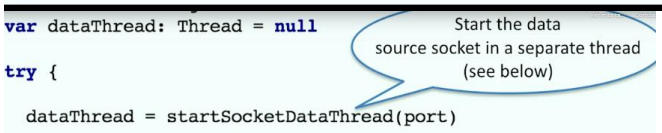
Minibatch interval (2 seconds here)

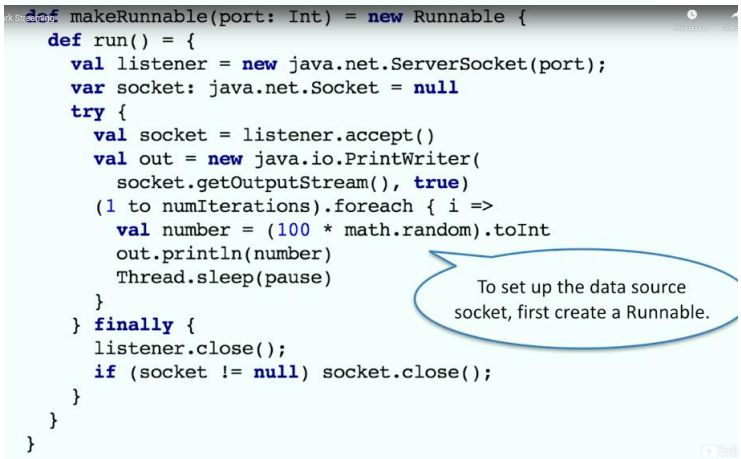
```
("\\s+")

println)
```

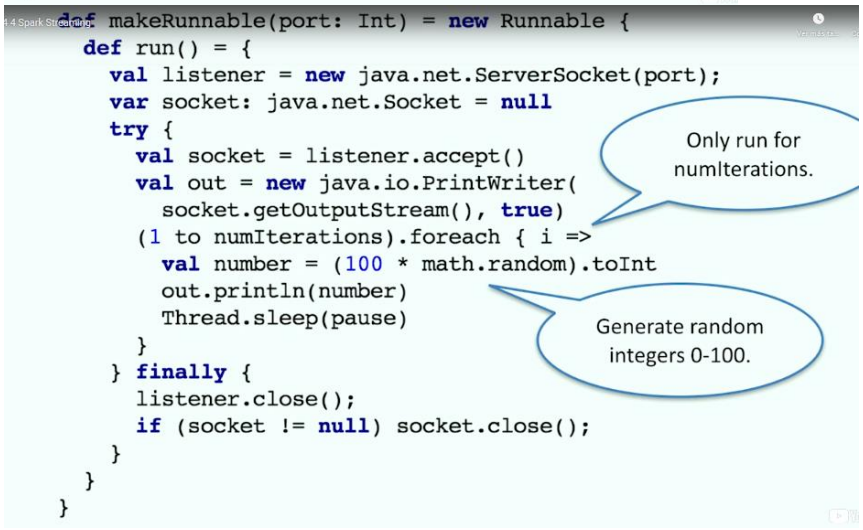
For each line, split on whitespace, convert to an Int

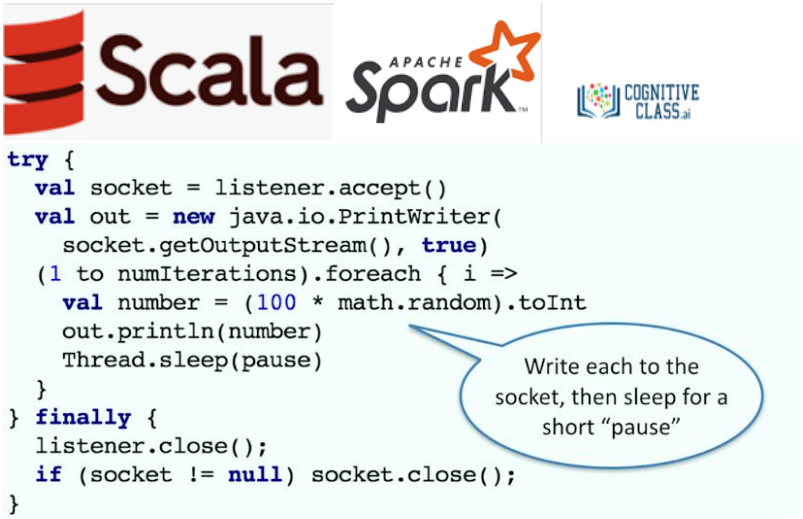

rdd.countByValue.foreach(println)
countByValue is like reduceByKey, where our single ints are "keys" and values"


var ssc: StreamingContext = null
var dataThread: Thread = null
try {
dataThread = startSocketDataThread(
ssc = StreamingContext.getOrCreate(
checkpointDir, createContext _)
ssc.start()
ssc.awaitTerminationOrTimeout(runtime)
} finally {
if (dataThread != null) dataThread.interrupt()
if (ssc != null)
ssc.stop(
stopSparkContext = true, stopGracefully = true)
}
}
Hack: Use vars and nulls here so we can init these inside the "try" clause, but see them in the "finally"clause.


var dataThread: Thread = null
try {
dataThread = startSocketDataThread(port)
Start the data source socket in a separate thread (see below)


def makeRunnable(port: Int) = new Runnable {
def run() = {
val listener = new java.net.ServerSocket(port);
var socket: java.net.Socket = null
try {
val socket = listener.accept()
val out = new java.io.PrintWriter(
socket.getOutputStream(), true)
(1 to numIterations).foreach { i =>
val number = (100 * math.random).toInt
out.println(number)
Thread.sleep(pause)
}
} finally {
listener.close();
if (socket != null) socket.close();
}
}
}
To set up the data source socket, first create a Runnable.


def makeRunnable(port: Int) = new Runnable {
def run() = {
val listener = new java.net.ServerSocket(port);
var socket: java.net.Socket = null
try {
val socket = listener.accept()
val out = new java.io.PrintWriter(
socket.getOutputStream(), true)
(1 to numIterations).foreach { i =>
val number = (100 * math.random).toInt
out.println(number)
Thread.sleep(pause)
}
} finally {
listener.close();
if (socket != null) socket.close();
}
}
}
Only run for numIterations.
Generate random integers 0-100.



Lesson Summary

Having completed this lesson, you should be able to:

- Describe how Spark Streaming works
- Explain the concept of DStreams, the core abstraction in Spark Streaming

transcript. Skip to the end.
welcome to spark streaming after completing this lesson you should be able to describe how spark streaming works and explain the concept of D streams the core abstraction and spark streaming so what if you wanted to process data as soon as it arrives within seconds or minutes rather than wait for that data to be added to your data said i'm for the next batch job to run sparked streaming provides this capability spark streaming is a large topic with significant operational considerations here we introduce the spark streaming model and how it can be useful
the next lesson will strain how to use Park streaming and spark SQL is hype support together to spark streaming supports a mini badge model the data is captured in fixed time intervals and you define what those animals will be whether it's half of a second or a minute in with each many badge is stored in an RDD the sequence of our DD's is held in a D stream and the D stream provides windowing functions of the you can group the data together here is a
visual example of what it looks like when data is coming into a D stream on the far right hand side we see that we have T zero time zero when we started
receiving data and then from time 0 two-time 1 represents a time interval that you provided for how long should wait for data before you say that is a single many batch in this case we got two events in an RDD we call RDD number
one in the first interval in the second interval we received four events we call that RDD number two and then in the third batch of data we have three events
included in RD number three D stream gives us the ability to group those many
batches together such that we can treat Rd number one and RTD number two as a
single window of data regardless of the

many batch interval that we provided it is important to note that the time intervals are of a fixed size for your many batches however the number of events captured in each one could vary depending on how much data is coming in
through the socket in order to use Park streaming we will declare are packaged
in imports such that we don't have to do that in line in our code these are the imports that we will require here is how we would work with sparks streaming
first of all we would set up a bunch of constants for our operations we would set the port that we were listening on for our data coming in we're set the interval we will use for her many batches in this case two seconds
note that I have a pause constant representing 10 milliseconds which I'm going to use for data that I'm pushing into spark streaming a producer of data
so that we can test our implementation I define the server to be my localhost I
said a checkpoint directory and then I have a runtime which i'm saying is going
to be thirty times 1000 milliseconds in other words thirty seconds and I'm also
gonna have a number asians here of a hundred thousand just like before I set
up my spark in Fig to be in local mode I set up the number of partitions in the application name I one and then I create my spark context from their config it's
a convention in Sparks streaming to create a function that represents what to do when handling an interval of data in this case I set up a new streaming context representing my interval of two seconds and then I set up a checkpoint
at from my checkpoint directory as I said before next I'm going to listen to data on the server and socket that I do find my constants in this case the local
host and the port of 9,000 and then for each line I'm going to split on whitespace converting to an integer in each
case once I've done that for each many batches RDD after construction I'm going
to count by value and then call for each in print line the result can buy value is like reduce bikie were single ends our keys and values but count by value is like doing a group buy over the integers then counting the size of the group's word count which we saw earlier lessons could also abused something like
this there is a drawback it returns a scholar collections map to the driver not a new Rd instance and hence you should be careful to avoid out of memory
errors on your JVM next I'm going to set up the actual reading of data from the
socket in order to do this cleanly I'm going to set up to bars 14 a streaming context and the other for thread and set them to know this means that I've

declared them but I have not provided them with definitions of what they actually are yet I'm doing this for scoping reasons i den inside my triblock set the data thread equal to some start socket data threat on the port and then
I'd buying the SSC spark streaming context from the values that I provided earlier including the function that I want to provide to do work for each many batch I start the spark streaming context and then I away determination or time out of the data coming in but then also in my finally block I wanna do all of the cleanup that would be required and had a defined by SSC and my data to
read only within the context of the try block they would not be scoped to be visible inside of the finally block if I want to be able to check whether or not those were actually set up with real values and are not still no I had to have to find them before I created the try block
luck next I'm going to define a function that does the work to push out data which my spark streaming context can now read and handle this is going to set up
a datasource socket by first creating a runnable which is a representation of some function that can be run in a thread on the GBM I create a server socket on a port I accept connections and I use a PrintWriter to print the data out I only run for num iterations in this case 100,000 times and I generate random numbers from the integer 0 to 100 I write each one to the socket
and then sleeper short pause in this case 10 milliseconds as we defined earlier and then I clean up my sock it when I'm done in the next lesson will combine what we've learned about hi I'm in streaming to examine a real-world use
case such as extracting transforming in loading of data into high
having completed this lesson you should be able to describe how spark streaming
works explain the concept of D streams the core abstraction in Sparks streaming
End of transcript. Skip to the start.

ETL with Spark Streaming

In Hadoop ecosystems, a common scenario is to use an ETL pipeline (extract, transform, and load) to populate Hive tables with incoming data

Let's sketch an implementation using Spark Streaming and SparkSQL's support for Hive

ETL with Spark Streaming

We'll ingest the flights data into Hive table partitions using the same streaming socket technique we used before

Our streaming job will use the HiveContext to create the table, if necessary, and for each year, month, and day in the flight data, construct a corresponding partition

```
package course2.module4

import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.dstream.DStream
import org.apache.spark.streaming.Seconds
import org.apache.spark.streaming.scheduler.{
  StreamingListener, StreamingListenerReceiverError,
  StreamingListenerReceiverStopped}
import org.apache.spark.sql.hive.HiveCont
import data.Flight
import util.Files
import scala.util.control.NonFatal
import java.net.{Socket, ServerSocket}
import java.io.{File, PrintWriter}
```

Listen for stream state transitions

```
def main(args: Array[String]): Unit = {
  val port =
    if (args.size > 0) args(0).toInt else defaultPort
  val conf = new SparkConf()
  conf.setMaster("local[*]")
  conf.setAppName("ETL with Spark Streaming and Hive")
  conf.set("spark.sql.shuffle.partitions", "4")
  conf.set("spark.app.id", "HiveETL")
  val sc = new SparkContext(conf)
```

```
object HiveETL {

  val defaultPort = 9000
  val interval = Seconds(5)
  val pause = 10   // milliseconds
  val server = "127.0.0.1"
  val hiveETLDir = "output/hive-etl"
  val checkpointDir = "output/checkpoint_dir"
  val runtime = 10 * 1000    // quit after N seconds
  val numRecordsToWritePerBlock = 10000
```

A forced delay during data generation.

Where the Hive table data will go

Write this many records, then sleep.


Files.rmrf("derby.log")
Files.rmrf("metastore_db")
Files.rmrf(checkpointDir)
Files.rmrf(hiveETLDir)
Delete the Hive metadata and table data, and the streaming checkpoint directory from previous runs.
def createContext(): StreamingContext = {
val ssc = new StreamingContext(sc, interval)
ssc.checkpoint(checkpointDir)
val dstream = readSocket(ssc, server, port)
processDStream(ssc, dstream)
}
Create the StreamingContext as before. The processing is defined by processDStream
var ssc: StreamingContext = null
var dataThread: Thread = null
try {
dataThread = startSocketDataThread(port)
ssc = StreamingContext.getOrCreate(
checkpointDir, createContext _)
ssc.addStreamingListener(
new EndOfStreamListener(ssc, dataThread))
ssc.start()
ssc.awaitTerminationOrTimeout(runtime)
} finally {
shutdown(ssc, dataThread)
}
}
Start the thread that writes the data.
Create & process the stream as before, but also add an event listener
def readSocket(
ssc: StreamingContext,
server: String, port: Int): DStream[String] =
try {
banner(s"Connecting to $server:$port...")
ssc.socketTextStream(server, port)
} catch {
case th: Throwable =>
ssc.stop()
throw new RuntimeException(
s"Failed to initialize server:port socket with $se
th)
}
As before

```
    } finally {
      listener.close();
      if (socket != null) socket.close();
    }
  }
}

def startSocketDataThread(port: Int): Thread = {
  val dataThread = new Thread(makeRunnable(port))
  dataThread.start()
  dataThread
}
```

Shutdown

Start the source data thread.

```
def processDStream(
    ssc: StreamingContext,
    dstream: DStream[String]): StreamingContext = {
  val hiveContext = new HiveContext(ssc.sparkContext)
  import hiveContext.implicits._
  import hiveContext.sql
  import org.apache.spark.sql.functions._

  val hiveETLFile = new java.io.File(hiveETLDir)
  val hiveETLPath = hiveETLFile.getCanonicalPath
```

Create the Hive context and obtain the full path for the HTL dir.

```
sql(s"""
  CREATE EXTERNAL TABLE IF NOT EXISTS flights2 (
    depTime         INT,
    arrTime         INT,
    uniqueCarrier   STRING,
    flightNum       INT,
    origin          STRING,
    dest            STRING)
  PARTITIONED BY (
    depYear         STRING,
    depMonth        STRING,
    depDay          STRING)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
  LOCATION '$hiveETLPath'
  """).show
println("Tables:")
sql("SHOW TABLES").show
```

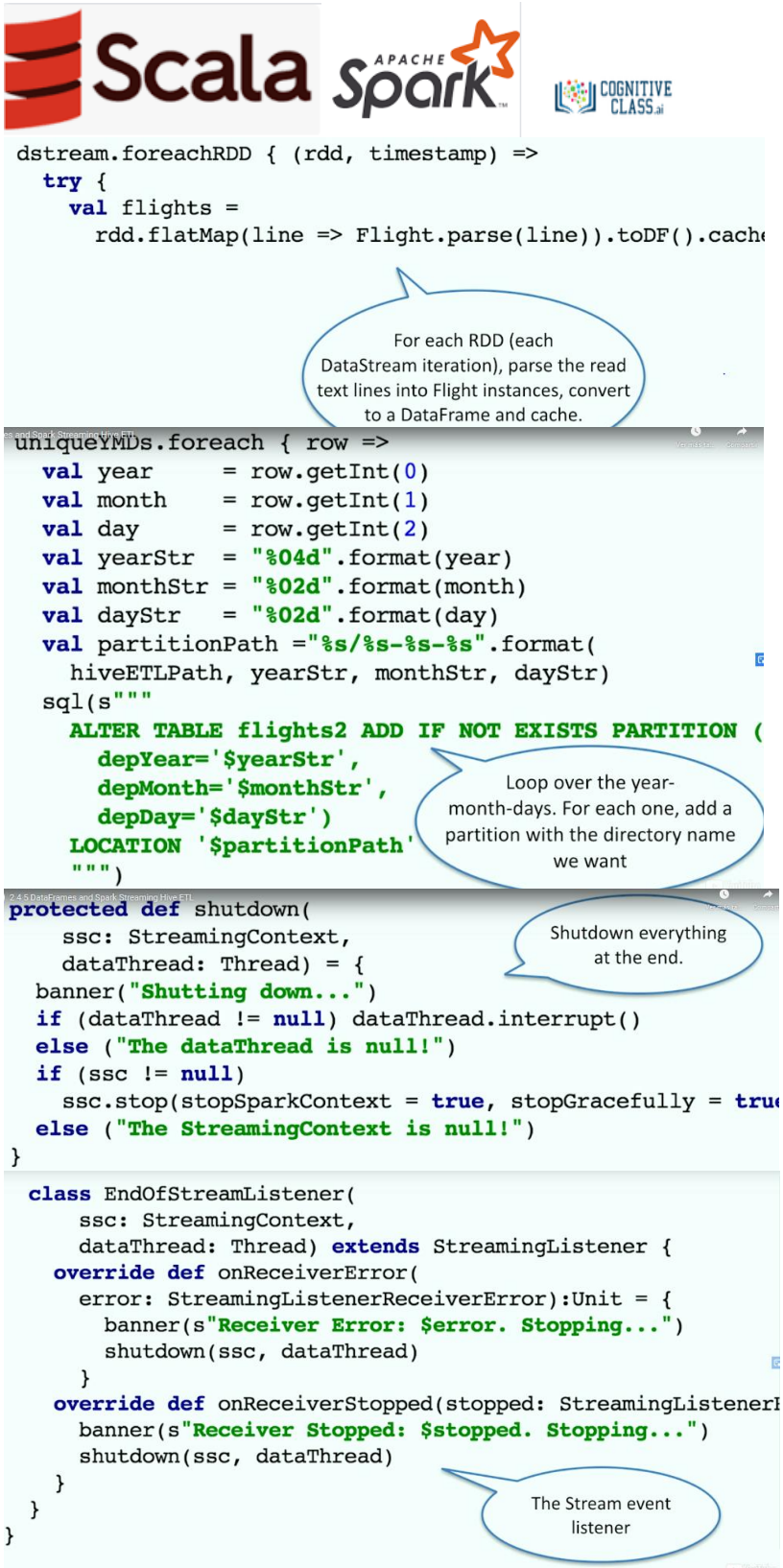
Create an external table (not managed by Hive itself).

Use only some Flights columns. Use 3 of them for partitioning

"|" is the column delimiter in the text data.

```
S TERMINATED BY '|'
```

Should see just our table.



Lesson Summary

Having completed this lesson, you should be able to:

- Describe how to combine Spark Streaming and Hive to implement an ETL workflow

rt of transcript. Skip to the end.
welcome to data frames and sparks streaming with high et al
after completing this lesson you should be able to describe how to combine spark
streaming in high to implement an ETL workflow in Hadoop ecosystem to common
scenario is to use an ETL pipeline to extract transform and load data to populate hype tables with incoming data let sketching implementation using spark
streaming and spark SQL support for high in this case were going to ingest the
flight stated that we've been using before into high people partitions using the same streaming soccer technique we used before as well
are streaming job will use the hype context to create the table if necessary and for each year month and day in the flight data construct a corresponding partition of the imports we've been using but we've added a few these will listen for stream state transitions as before we're setting up our default port
are interval are pause that will be used for the writing of the data the server that we're using the output directory the checkpoint directory as well as to run for 10 seconds we're also going to record her right block 10,000 records we
will once again set up our spark configuration as needed and create our spark context but next we're going to delete the hyde medidata and table data
and the streaming checkpoint directory from previous runs normally you would
not do this in production but this is simply for a rerun of an application
that we're using locally we then create the streaming context as before the processing is defined by the process D stream method that we will discuss shortly
we start the thread that writes the data and we create and process the stream as
before but we also add an event listener note that we have an end of stream listener which attempts to recover from errors more gracefully this is as before
where we are reading data from its socket and this is also very similar to before where we have a runnable thread which is going to write data to the socket it reads from the airline data in this CSV file and writes blocks to the socket with sleep intervals we shut down the socket and the final method start
socket data thread starts the source data to read in the process D stream we create the hype context and obtain the full path for the html directory
this is what does the real work it creates the hype table ingest the socket data sources into flights splits the flights into year month day and rights
to new high table partitions next we create an external table not to be managed by hype itself and we used only some flight columns for this case we

used three of them for partitioning we're going to use that blights data and we're going to do a little more work than necessary
hype could partition creation forest for example if we use insert into with values for new partition columns etc before processing the stream create the table and note that hype DDL statements require absolute pads
pipe is the column delimiter in the text data and you should see just our table when the show is printed out
for each RDD each data stream iteration we're going to parse the read text lines
into flight instances and convert to a data frame and cash that data next we're
going to find the unique year month and day combinations in the data it appears
to be safer to collect into a collection in the driver and then go through it seriously to create corresponding table partitions well loop over the year month
days and for each one had a partition with the directory name we want finally will filter the data for that date
form the new records and write it to the partition directory we write pipe separated strings but consider replacing with parquet in production for performance reasons we show the partitions we've created and we handle failures by catching any exception that might be non-fatal and exiting with the
one and finally as part of our shutdown process we make sure that all threads
and spark context have been closed gracefully and finally we define an end of stream listener that was used or defined earlier in this application and this make sure that we have behavior defined for what to do when our receiver
is stopped having completed this lesson we should be able to describe how to
combine spark streaming and high to implement and ETL workflow.

5. Introduction to Spark MLib

Spark with Scala
After completing this lesson, you should be able to:
- Describe what machine learning is and how machines learn
- Explain how Machine learning jobs are different than map-reduce jobs
- Discuss the key components of Spark MLlib library

Overview of Spark ML Algorithms
After completing this lesson, you should be able to:
- Describe some of the algorithms that come with Spark MLlib
- Explain the difference between supervised and unsupervised learning
- Implement Alternating Least Squares and Kmeans algorithms

Introduction to GraphX
After completing this lesson, you should be able to:

- Describe the importance of graph processing
- Discuss the GraphX library and its building blocks

Sentiment Analysis of Twitter Stream I

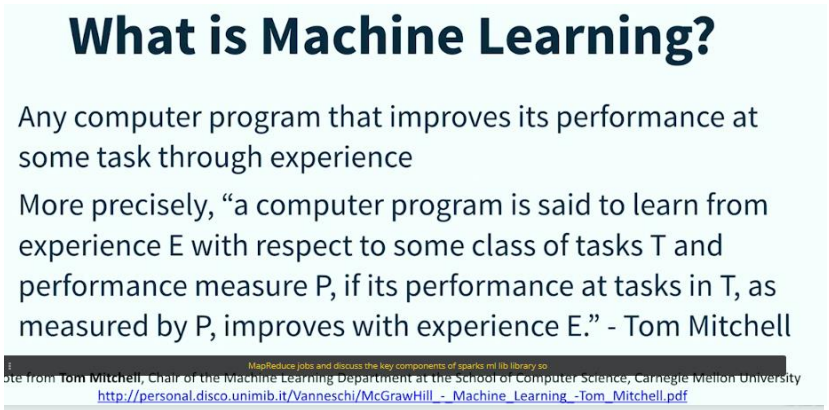
After completing this lesson, you should be able to:

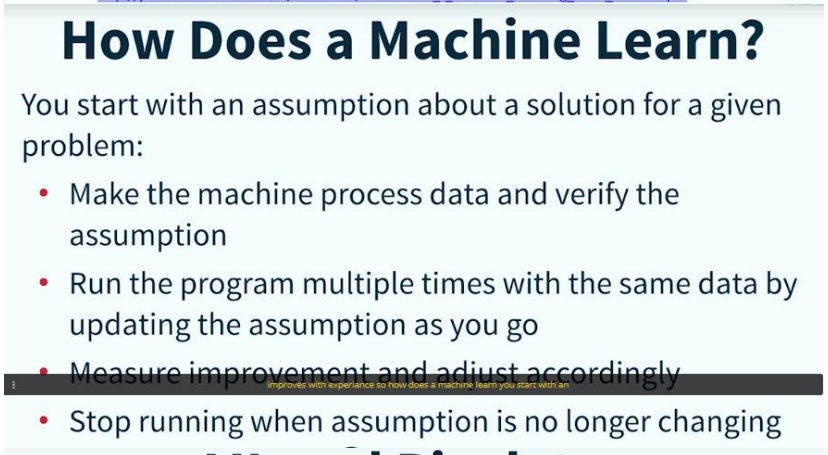
- Describe Naïve Bayes algorithm
- Discuss how to build and train a machine learning algorithm from scratch

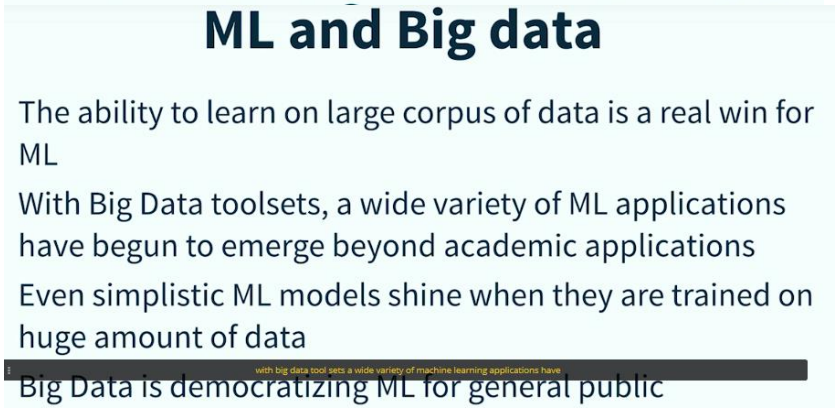
Sentiment Analysis of Twitter stream II

After completing this lesson, you should be able to:

- Describe how to connect Spark streaming to Twitter
- Explain steps required to integrate an ML algorithm to streaming







Machine Learning Jobs

Tend to differ from traditional map-reduce jobs in following ways:

- They are iterative computations
- They are CPU intensive
- They maintain state between iterations

Spark MLlib

MLlib is Spark's machine learning library

A scalable and easy library for machine learning algorithms

Supports popular machine learning algorithms like Linear regression, Logistic regression, decision trees

Spark MLlib

The Spark MLlib is divided into two broad packages:

- **spark.mllib** - contains algorithms that work on top of RDDs
- **spark.ml** - provides a higher-level API built on top of DataFrames for constructing ML pipelines

Why Spark MLlib?

There are many machine learning libraries, so why use Spark MLlib?

- Built on Spark's lightweight yet powerful API
- Spark's open source community is very large
  - Lots of contributions and improvements from the community
  - Spark MLlib is one of the most active Spark component projects
- Scalability and performance

sparking my lib is one of the most active spark component projects it also

Lesson Summary

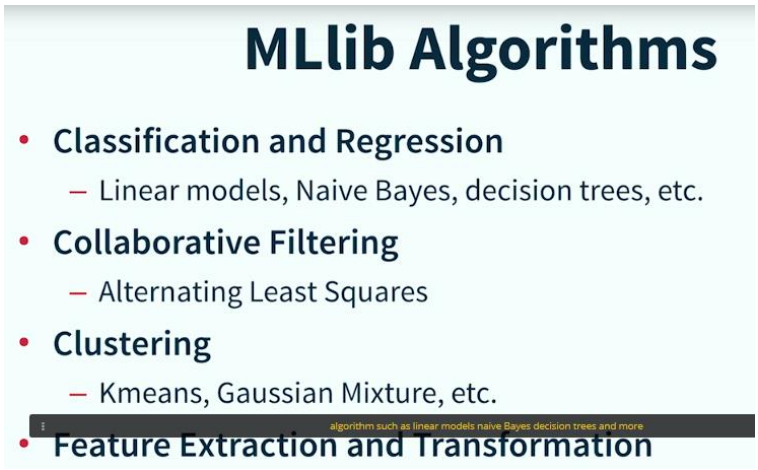
Having completed this lesson, you should be able to:

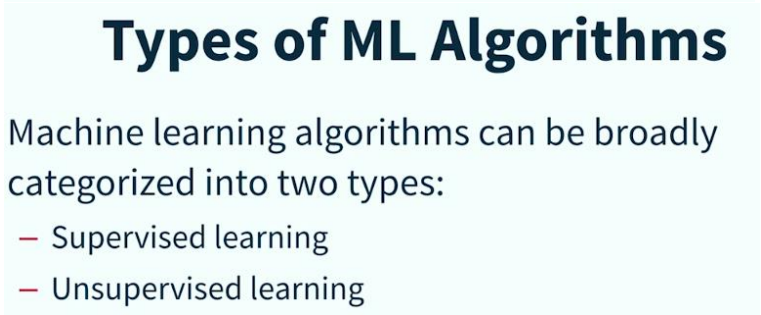
- Describe what machine learning is and how machines learn
- Explain how Machine learning jobs are different than map-reduce jobs
- Discuss the key components of Spark MLlib library

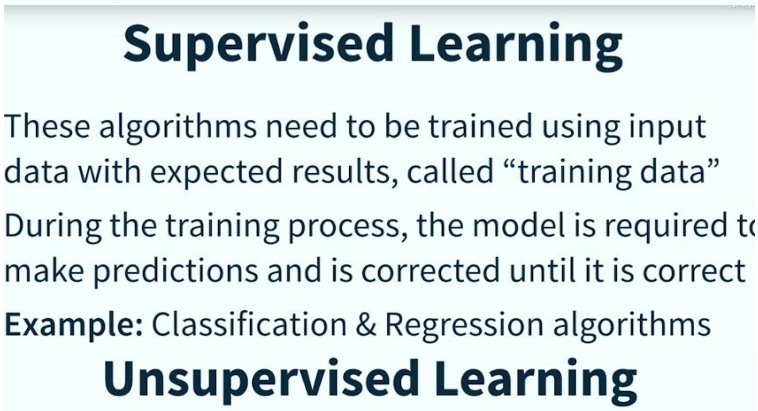
end.

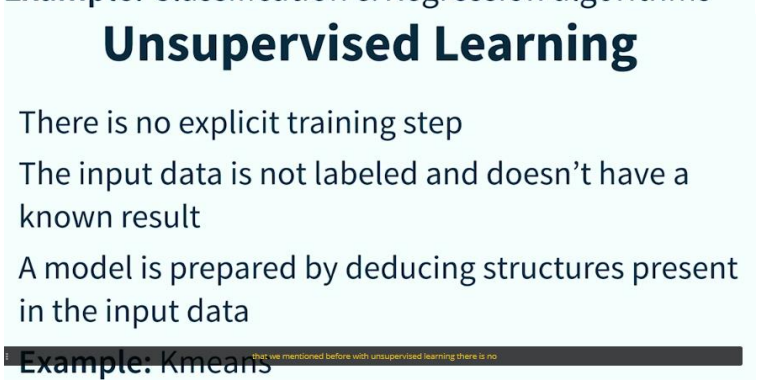
welcome to introduction to spark ml lib after completing this lesson you should
be able to describe what machine learning is and how machines learn you should be able to explain how machine learning jobs are different than MapReduce jobs and discuss the key components of sparks ml lib library so what is machine learning machine learning is any computer program that improves its performance at some task through experience more precisely according to Tom Mitchell the chair machine learning department at the school of computer science at Carnegie Mellon University a computer program is
said to learn from experience II with respect to some class of task T and performance measure P if its performance at St as measured by P improves with experiance so how does a machine learn you start with an assumption about a solution for a given problem you make the machine process
data and verify the assumption you then run the program multiple times with the
same data by updating the assumption as you go you measure improvement and
adjust accordingly you stop running when the assumption is no longer changing the
ability to learn on a large corpus of data is a real win for machine learning with big data tool sets a wide variety of machine learning applications have begun to emerge beyond academic applications and even simplistic a male model shine when they are trained on a huge amount of data so big data is democratizing machine learning for the general public
machine learning jobs tend to differ from traditional MapReduce jobs in the following ways their interactive computations they are working through
the data repeatedly they are CPU intensive and not bound by socket or 50 I O and they maintain state between iterations spark emily is sparks machine
learning library
a scalable and easy library for machine learning algorithms it supports popular
machine learning algorithm such as linear regression logistic regression decision trees and more out of the box the spark ml live is divided into two broad packages sparked ml lib contains algorithms that work on top of our DVDs
but sparked ml provides a higher level API built on top of data frames for constructing ml pipelines there are many machine learning libraries out there so
why you Sparky may live it's built on sparks lightweight did powerful API and sparks open source community is very large and active there are lots of contributions in improvements from the community on a regular basis and sparking my lip is one of the most active spark component projects it also has excellent scalability and performance characteristics
having completed this lesson we should be able to describe what machine learning is and how machines learn explain how machine learning jobs are

different from MapReduce jobs and discuss the key components of the spark
ml live library









Code Example: Supervised Learning

- Collaborative filtering is commonly used for recommender systems
- We will use movie rating data from MovieLens and use the ALS (Alternating Least Squares) algorithm to build a simple movie recommender application

```
package course2.module5


import org.apache.log4j.{Level, Logger}
import org.apache.spark.ml.recommendation.{ALSModel, ALS}
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.{DataFrame, SQLContext}
import org.apache.spark.{SparkConf, SparkContext}

case class Rating(userId: Int, movieId: Int, rating: Double)

object Rating {
  // Format is: userId, movieId, rating, timestamp
  def parseRating(str: String): Rating = {
    val fields = str.split("::")

    // We don't care about the timestamp
    Rating(fields(0).toInt, fields(1).toInt, fields(2).toDouble)
  }
}

case class Movie(movieId: Int, title: String, genres: Seq[String])
```

this data into an application first we have to define what rating da

```
object Movie {
  // Format is: movieId, title, genre1|genre2
  def parseMovie(str: String): Movie = {
    val fields = str.split("::")
    assert(fields.size == 3)
    Movie(fields(0).toInt, fields(1), fields(2).split("|"))
```

```
object SupervisedLearningExample {

  val numIterations = 10
  val rank = 10 //number of features to consider when training the model
  //this file is in UserID::MovieID::Rating::Timestamp format
  val ratingsFile = "data/als/sample_movielens_ratings.txt"
  val moviesFile = "data/als/sample_movielens_movies.txt"
  val testFile = "data/als/test.data"

  def main(args: Array[String]): Unit = run()

  def run(): Unit = {
    val conf = new SparkConf()
      .setAppName("SupervisedLearning")
      .setMaster("local[*]")
      .set("spark.app.id", "ALS")   // To silence Metrics warning
```

next we're going to start a collaborative filtering example to rate

```
    val sc = new SparkContext(conf)
    val sqlContext = new SQLContext(sc)
    import sqlContext.implicits._

    Logger.getRootLogger.setLevel(Level.ERROR)

  def main(args: Array[String]): Unit = run()

  def run(): Unit = {
```

and the movies fire that we're going to pull in as well as a test data file when

```
    val conf = new SparkConf()
      .setAppName("SupervisedLearning")
      .setMaster("local[*]")
      .set("spark.app.id", "ALS")   // To silence Metrics warning.
```

```
val als = new ALS()
              .setUserCol("userId")
              .setItemCol("movieId")
              .setRank(rank)
              .setMaxIter(numIterations)

// Training the model, converting rdd to dataframe for spark.ml
val model: ALSModel = als.fit(ratingsData.toDF)

// Now trying the model on our testdata
val predictions: DataFrame = model.transform(testData.toDF).cache

// Metadata about the movies
val movies = sc.textFile(moviesFile).map(Movie.parseMovie).toDF()

// Try to find out if our model has any falsePositives
//   And joining with movies so that we can print the movie names
val falsePositives = predictions.join(movies)
// Metadata about the movies
val movies = sc.textFile(moviesFile).map(Movie.parseMovie).toDF()

// Try to find out if our model has any falsePositives
//   And joining with movies so that we can print the movie names
val falsePositives = predictions.join(movies)
  .where((predictions("movieId") === movies("movieId"))
  && ($"rating" <= 1) && ($"prediction" >= 4))
  .select($"userId", predictions("movieId"), $"title", $"rating", $"prediction
val numFalsePositives = falsePositives.count()
println(s"Found $numFalsePositives false positives")
if (numFalsePositives > 0) {
  println(s"Example false positives:")
  falsePositives.limit(100).collect().foreach(println)
}

// Show first 20 predictions
predictions.show()
```

model we create a new als instance with the user column a user I D and the item

movies so that we can print the movie named in our output and finally we run

```
Enter number: 19

[info] Running course2.module5.SupervisedLearningExample
Got 1501 ratings from 30 users on 100 movies.
```

2.5.2 Overview of Spark ML Algorithms

```
|userId|movieId|rating|prediction|
+------+-------+------+----------+
|     1|      1|   5.0| 1.1709752|
|     2|      1|   5.0| 1.6331534|
|     3|      1|   1.0|0.99562675|
|     4|      1|   1.0|0.88097143|
|     1|      2|   1.0| 1.9992656|
|     2|      2|   1.0|  2.616164|
|     3|      2|   5.0| 1.0928926|
|     4|      2|   5.0| 3.2745652|
|     1|      3|   5.0| 1.0621433|
|     2|      3|   5.0| 0.9467654|
|     3|      3|   1.0| 1.4908013|
|     4|      3|   1.0| 1.2000617|
|     1|      4|   1.0| 1.8375804|
|     2|      4|   1.0| 2.7834222|
|     3|      4|   5.0| 2.1620073|
|     4|      4|   5.0| 2.0354443|
+------+-------+------+----------+

>>> Find out predictions where user 26 likes movies 10, 15, 20 & 25
+------+-------+----------+
|userId|movieId|prediction|
+------+-------+----------+
|    26|     10| 1.3337111|
|    26|     15| 0.9124926|
|    26|     20| 1.2709454|
|    26|     25| 1.2574424|
+------+-------+----------+

[success] Total time: 26 s, completed Dec 14, 2015 12:50:11 PM
```

comes up with zero false positives on its predictions for what people

Code Example: Unsupervised Learning

- Kmeans, a commonly used clustering algorithm that groups the data points into a predefined number of clusters

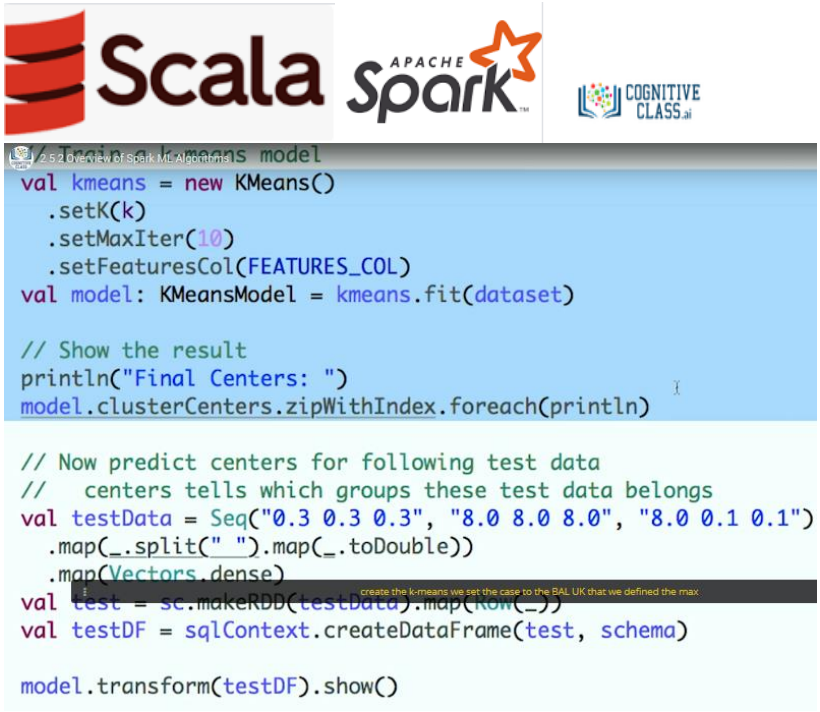
```
0.0 0.0 0.0
0.1 0.1 0.1
0.2 0.2 0.2
9.0 9.0 9.0
9.1 9.1 9.1
9.2 9.2 9.2
```

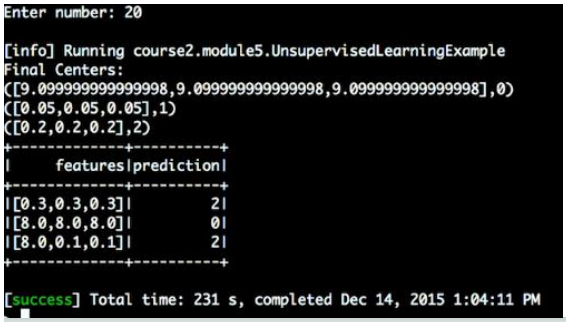
```
// In this case we are using Kmeans algorithm.
// Clustering is the best-known type of unsupervised learning.
// Clustering algorithms try to find natural groupings in data.
// These are data points that are like one another, but unlike others,
//   are likely to represent a meaningful grouping, and so clustering
//   algorithms try to put such data into the same cluster.
// This example is also taken from Spark distribution and simplified.
object UnsupervisedLearningExample {

  val FEATURES_COL = "features"

  def main(args: Array[String]): Unit = run("data/kmeans_data.txt", 3)

  def run(input: String, k: Int): Unit = {
    val conf = new SparkConf()
      .setAppName("UnsupervisedLearning")
      .setMaster("local[*]")
      .set("spark.app.id", "Kmeans") // To silence Metrics warning.

    val sc = new SparkContext(conf)
    val sqlContext = new SQLContext(sc)

    Logger.getRootLogger.setLevel(Level.ERROR)

    // Loads data
    val rowRDD = sc.textFile(input)
      .filter(_.nonEmpty)
      .map(_.split(" ").map(_.toDouble))
      // Vectors is the input type KMeans algo expects. It represents a
```

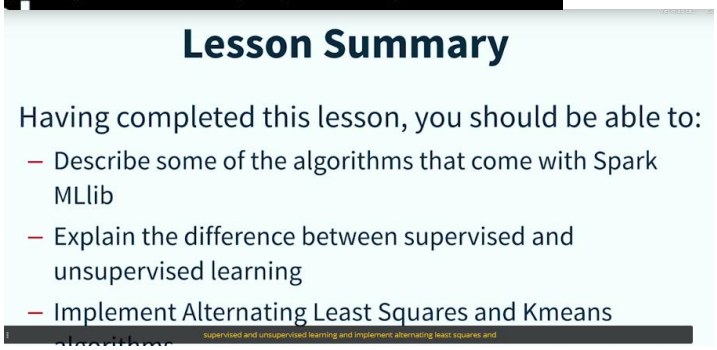
we're using the k-means algorithm

```
Logger.getRootLogger.setLevel(Level.ERROR)

// Loads data
val rowRDD = sc.textFile(input)
  .filter(_.nonEmpty)
  .map(_.split(" ").map(_.toDouble))
  // Vectors is the input type KMeans algo expects. It represents a
  //   numeric vector of features that algo will use to create cluster.
  .map(Vectors.dense)
  .map(Row(_)) // Each element in DataFrame is represented as Row, think of it as
  .cache()
val schema = StructType(Array(StructField(FEATURES_COL, new VectorUDT, false)))
val dataset = sqlContext.createDataFrame(rowRDD, schema)

// Train a k-means model
val kmeans = new KMeans()
  .setK(k)
  .setMaxIter(10)
  .setFeaturesCol(FEATURES_COL)
val model: KMeansModel = kmeans.fit(dataset)
```

that the algorithm we used to create the cluster and then each element in the





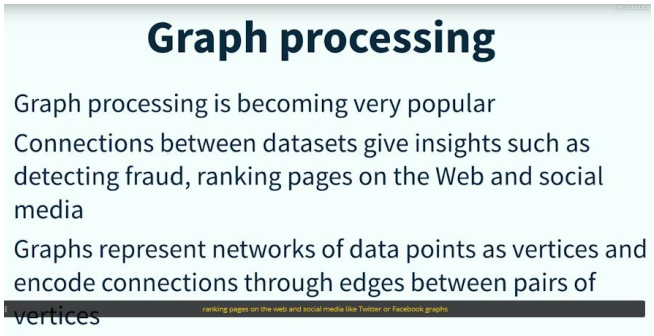


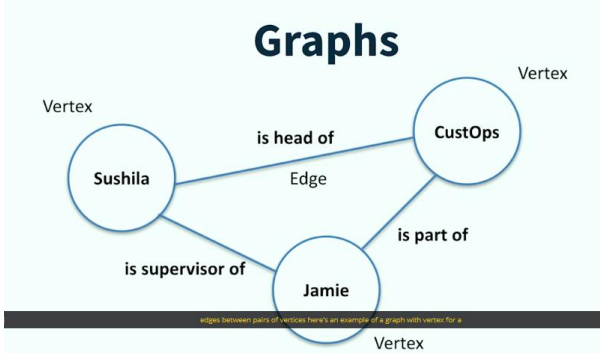
pt. Skip to the end.
welcome to overview of spark an L algorithms after completing this lesson he should be able to describe some of the algorithm that come with spark ml live explain the difference between supervised and unsupervised learning and
implement alternating least squares and k-means algorithms ml live algorithms
can be broken down into several groups there are classification and regression
algorithm such as linear models naive Bayes decision trees and more classification algorithms are used to separate data sets into classes for example how to organize a large corpus of data into categories regression is much more like the predicted price of a consumer good based on some variables

another group is collaborative filtering such as the alternating least squares algorithm these are used for building recommendation engines clustering algorithms may include k-means are Gaussian mixture others and they are used to organizing group a dataset finally we have beecher extraction and transformation which are used for text mining machine learning algorithms can
be broadly categorized into two types they're supervised learning and unsupervised learning there's another kind called semi-supervised learning which is sort of a half of both supervised learning algorithm to need to
be trained using input data with the expected results which we call training data during the training process the model is required to make predictions and is corrected until it is correct
examples of these include the classification and regression algorithms that we mentioned before with unsupervised learning there is no
explicit training step the input data is not labeled and doesn't have known result a model is prepared by deducing structures present in the input data an
example of unsupervised learning my became means unsupervised learning can
be used to perform tasks such as detecting network intrusion you really don't know what the
jack is going to look like unless you see one happen and you can't train the model in advance you can use k-means to group similar things together and try to
find which one is the odd one out let's look at a code example for supervised learning collaborative filtering is commonly used for recommender systems we
will use the movie rating data from movie lands and used the ALS also called
alternating least squares algorithm to build a simple movie recommender application in the supervised learning example we are going to take data about
movies and their ratings that were given by individual users we're going to load
this data into an application first we have to define what rating data looks like and how we're going to parse that data in note that we do not care about the final field in the ratings as we don't care about the timestamp of the rating in our analysis we also have to define what movie looks like and how we're going to parse that data as well
next we're going to start a collaborative filtering example to rate
these movies the data comes from movie lands which is a non commercial movie
recommendation website and the example is inspired from the movie lends a less
dats Col example from the spark distribution first we're going to define
a number of iterations it we are going to perform when training the model and
also the number of features were going to consider we define the ratings file and the movies fire that we're going to pull in as well as a test data file when

we execute this model we set up the spark config as before
create a spark context and create a sequel context of that we can use our data frames next we get our training dataset and then we get our test dataset
to verify the model
we then determine the number of ratings the number of users and the number of
movies and print that out so that we can see it as we're executing to start the model we create a new als instance with the user column abuser I D and the item
column of the movie I D we set the rank and the max number of iterations we then
have our model when we performed a fit method on the ALS this allows for training the model converting an RTD to a data frame as well now we try to model
on our test data and then we capture metadata about the movies in the next section we try to find out if our model has any false positives and we join with
movies so that we can print the movie named in our output and finally we run
predictions for user number twenty six as an example to find out whether the user like some movies finally we stop the spark context we execute and a supervised learning example begins it finds fifteen hundred and one rating from thirty users on a hundred movies it then starts performing its analysis it comes up with zero false positives on its predictions for what people would like it then uses user 26 and specific movies to test and verify that it did what we expect as MX code example let's look at unsupervised learning we were
used k-means a commonly used clustering algorithm the group's the data points
into predefined number of clusters in this unsupervised learning example we're
going to perform a k-means analysis on some numeric data this may look rather
random accept most of the time when you're performing this kind of analysis you are looking at numeric representation of some text in this case
we're using the k-means algorithm
and clustering is the best known type of unsupervised learning these algorithms
try to find natural groupings and Ada these data points are like one another but unlike others are likely to represent a meaningful grouping and so clustering algorithms try to put such data into the same cluster this example is also taken from the spark distribution and simplified somewhat
first we define an alias for the features column and then we define how we're going to get the data set up our spark context set up our secret context next we load the data and we filter it based on non empty data we mapped over
to split it by space and we converted to double values vectors of the input type

k-means algorithm expects in ML it represents a numeric vector of pitchers that the algorithm we used to create the cluster and then each element in the
data frame is represented as a row to think of it just like a database role once we finish with this week asked the data next we define the schema and set
up the data set from the data frame using the row Rd and the schema next we
create the k-means we set the case to the BAL UK that we defined the max iterations and the teachers column from the alias we defined earlier as well we
then do a fit on the algorithm with this data set and print out the results that we get finally we predict centers for following test data and those centers tell us which groups of these test data belong then we transform and show that
result and stop when we run this model we see our final centers and the groupings of our data as well as the prediction that we expected
having completed this lesson you should be able to describe some of the algorithm succumb with SPARQL live explain the difference between supervised and unsupervised learning and implement alternating least squares and
k-means algorithms
End of transcript. Skip to the start.





GraphX

GraphX is the graph processing layer on top of Spark that brings the power of big data processing to graphs – graphs that would be too large to fit on a single machine

GraphX comes with the standard Spark distribution

GraphX

GraphX represents a graph using two RDDs, vertices and edges

This allows GraphX to partition large graphs

GraphX is not a database, its a graph processing library

GraphX building blocks

VertexRDD - contains Vertex information

EdgesRDD - Edge information

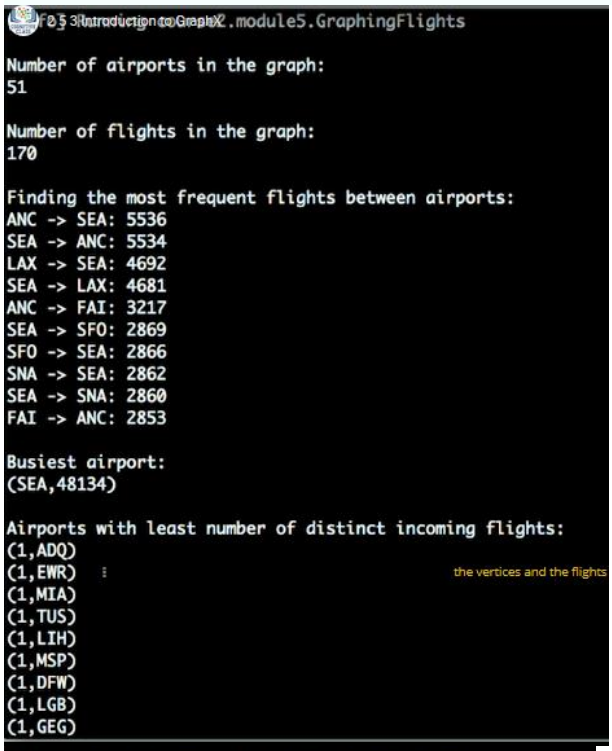
```
2008,1,1,2,2057,2052,2312,2258,AS,324,N306AS,135,126,112,14,5,SEA,SJC,697,7,16,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,703,715,958,951,AS,572,N302AS,175,156,144,7,-12,SEA,PSP,987,6,25,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,2011,1846,2248,2145,AS,511,N564AS,157,179,136,63,85,SAN,SEA,1050,7,14,0,,0,0,0,0,0,63
2008,1,1,2,2301,2300,2354,2359,AS,376,N309AS,53,59,35,-5,1,SEA,GEG,224,5,13,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1221,1221,1422,1438,AS,729,N317AS,181,197,164,-16,0,TUS,SEA,1216,6,11,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1843,1840,2110,2125,AS,283,N318AS,147,165,124,-15,3,LAX,SEA,954,7,16,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,2045,2045,2314,2330,AS,211,N305AS,149,165,126,-16,0,LAX,SEA,954,6,17,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,49,50,547,523,AS,100,N315AS,238,213,222,24,-1,ANC,PDX,1542,2,14,0,,0,0,0,24,0,0
2008,1,1,2,1719,1715,1939,1956,AS,665,N302AS,140,161,118,-17,4,LAS,SEA,866,8,14,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,613,630,815,844,AS,531,N755AS,122,134,96,-29,-17,SJC,SEA,697,7,19,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,753,720,1125,1103,AS,571,N320AS,152,163,124,22,33,SEA,DEN,1024,5,23,0,,0,22,0,0,0,0
2008,1,1,2,NA,205,NA,620,AS,154,N309AS,NA,195,NA,NA,NA,ANC,SEA,1449,NA,NA,1,A,0,NA,NA,NA,NA,NA
2008,1,1,2,741,740,1128,1139,AS,728,N317AS,167,179,150,-11,1,SEA,TUS,1216,4,13,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1702,1530,1941,1804,AS,518,N564AS,159,154,142,97,92,SEA,SAN,1050,2,15,0,,0,92,0,5,0,0
2008,1,1,2,1306,1245,1551,1525,AS,580,N307AS,165,160,142,26,21,SEA,SAN,1050,2,21,0,,0,21,0,5,0,0
2008,1,1,2,2043,2040,2308,2323,AS,85,N302AS,205,223,181,-15,3,SEA,ANC,1449,5,19,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1405,1410,1637,1633,AS,640,N302AS,152,143,123,4,-5,SEA,LAS,866,13,16,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1050,1031,1329,1301,AS,292,N319AS,159,150,131,28,19,SEA,LAX,954,10,18,0,,0,0,0,9,0,19
2008,1,1,2,1610,1555,1850,1830,AS,478,N566AS,160,155,142,20,15,SEA,PSP,987,5,13,0,,0,15,0,5,0,0
2008,1,1,2,1639,1640,1903,1925,AS,485,N577AS,144,165,124,-22,-1,LAX,SEA,954,7,13,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1554,1555,1827,1837,AS,277,N323AS,153,162,123,-10,-1,LAX,SEA,954,10,20,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,26,30,508,444,AS,198,N320AS,222,194,199,24,-4,ANC,SEA,1449,6,17,0,,0,0,0,24,0,0
2008,1,1,2,717,700,1003,930,AS,811,[illegible],0,,0,0,0,16,17,0
2008,1,1,2,629,630,856,903,AS,210,N318AS,147,153,131,-7,-1,SEA,LAX,954,6,10,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1211,1153,1408,1403,AS,539,N320AS,177,190,151,5,18,DEN,SEA,1024,4,22,0,,0,NA,NA,NA,NA,NA
2008,1,1,2,1935,1922,2130,2128,AS,295,N315AS,115,126,90,2,13,SFO,SEA,679,8,17,0,,0,NA,NA,NA,NA,NA
```

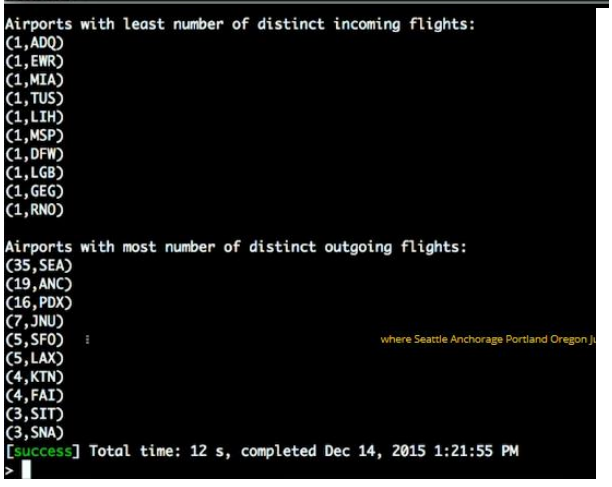
we are once again going to use the Alaska Airlines dataset for all of their

```
object GraphingFlights {

  var quiet = false

  def main(args: Array[String]) {
    val input = "data/airline-flights/alaska-airlines/2008.csv"
    val conf = new SparkConf()
      .setAppName("GraphX")
      .setMaster("local[*]")
      .set("spark.app.id", "GraphX") // To silence Metrics warning.

    val sc = new SparkContext(conf)
    try{
      val flights = for {
        line <- sc.textFile(input)
        flight <- Flight.parse(line)
      } yield flight

      // Create vertices out of airport codes for both origin and dest
      val airportCodes = flights.flatMap { f => Seq(f.origin, f.dest) }
      val airportVertices: RDD[(VertexId, String)] =
        airportCodes.distinct().map(x [illegible]

      // Create edges between origin -> dest pair and the set the edge attribute
      //   to count of number of flights between given pair of origin and dest
      val flightEdges = flights.map(f =>
        ((stringHash(f.origin), stringHash(f.dest)), 1))
        .reduceByKey(_+_)
        .map {
```

we also set up our spark context from that config once we're done we load in


.take(10).foreach(println)
println("\nBusiest airport:")
val by =
triplets.map { triplet =>
(triplet.srcAttr, triplet.attr)
}.reduceByKey(_ + _)
by.sortBy(-_._2).take(1).foreach(println)
// Which airport has the most unique flights to it?
// vertices with no "in-degree" are ignored here
val incoming: RDD[(VertexId, (PartitionID, String))] = graph.inDegrees.join(airportVertices)
println("\nAirports with least number of distinct incoming flights:")
incoming.map {
case (_, (count, airport)) => (count, airport)
}.sortByKey().take(10).foreach(println)
println("\nAirports with most number of distinct outgoing flights:")
val outgoing: RDD[(VertexId, (PartitionID, String))] = graph.outDegrees.join(airportVertices)
outgoing.map {
airport has the most unique flights to it
}.sortByKey(ascending = false).take(10).foreach(println)
} finally {
sc.stop()
}
}


Introduction to GraphX.module5.GraphingFlights
Number of airports in the graph:
51
Number of flights in the graph:
170
Finding the most frequent flights between airports:
ANC -> SEA: 5536
SEA -> ANC: 5534
LAX -> SEA: 4692
SEA -> LAX: 4681
ANC -> FAI: 3217
SEA -> SFO: 2869
SFO -> SEA: 2866
SNA -> SEA: 2862
SEA -> SNA: 2860
FAI -> ANC: 2853
Busiest airport:
(SEA,48134)
Airports with least number of distinct incoming flights:
(1,ADQ)
(1,EWR)
the vertices and the flights
(1,MIA)
(1,TUS)
(1,LIH)
(1,MSP)
(1,DFW)
(1,LGB)
(1,GEG)


Airports with least number of distinct incoming flights:
(1,ADQ)
(1,EWR)
(1,MIA)
(1,TUS)
(1,LIH)
(1,MSP)
(1,DFW)
(1,LGB)
(1,GEG)
(1,RNO)
Airports with most number of distinct outgoing flights:
(35,SEA)
(19,ANC)
(16,PDX)
(7,JNU)
(5,SFO)
where Seattle Anchorage Portland Oregon Ju
(5,LAX)
(4,KTN)
(4,FAI)
(3,SIT)
(3,SNA)
[success] Total time: 12 s, completed Dec 14, 2015 1:21:55 PM

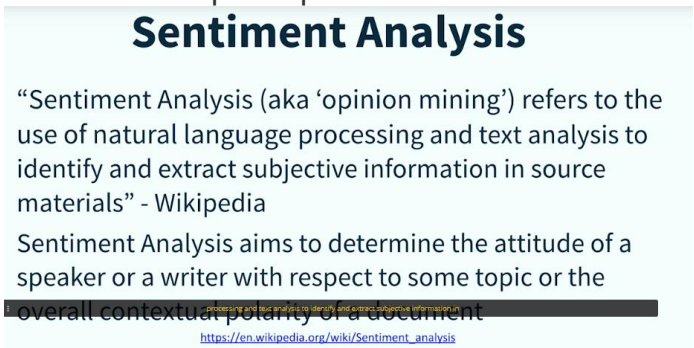
Lesson Summary

Having completed this lesson, you should be able to:

- Describe the importance of graph processing
- Discuss the GraphX library and its building blocks

nscript. Skip to the end.
welcome to introduction to grab tax after completing this lesson you should be able to describe the importance of crap processing and discuss the graphics
library and its building blocks Graham processing is becoming very popular in
big data connections between datasets give insights such as detecting fraud ranking pages on the web and social media like Twitter or Facebook graphs represent networks of data points as vertices and encode connections through
edges between pairs of vertices here's an example of a graph with vertex for a
person named Susheela another vertex for a group within a company called customer
operations and we have another vertex for another person named Jamie Susheela
could somehow be related to the customer operations vertex through an edge where
she is the head of customer operations she is also the supervisor of Jamie and
jamie is a part of customer operations the edges define how the relationship works between the vertices graphics is the grab processing layer on top of spark that brings the power of big data processing two graphs graphs that would
be too large to fit on a single machine graphics comes with the standard spark
distribution graphics implements a notion called the property graph both vertices and edges can have an arbitrary set of attributes associated with them
grab tax represents a graph using two Rd DS one per vertices and 14 edges this
allows graphics to partition large graphs graphics itself is not a database it's just a graphic processing library so grab Texas the vertex RDD containing
the vertex information and the edges RDD containing the edge information let's
look at a code example we're going to once again use the Alaska Airlines flight data to build a graph and extract various connection information and airport information from it
we are once again going to use the Alaska Airlines dataset for all of their flights from 2008 this represents a hundred and fifty one thousand one hundred and two different flights and their information again we specify the

input file and set up our spark in Fig
we also set up our spark context from that config once we're done we load in our flights and put it into a dataset that we can use we create vertices out of the airport codes for both the origin and destination and we make sure they
are distinct once we've created those vertices we then create the edges which
represent the flights between each of those airports from the vertices and the
edges we create the graph instance next we find the top 10 flights between two
airports creating graph triplets the triplets return an RDD of the edge triplet that has the source airport the destination airport and the attribute which is the flight from misinformation we can find the busiest airport out of all of them that Alaska Airlines flew to in 2008 we can also find out which airport has the most unique flights to it
vertices with no in degree are ignored in this case just as we found the airports with the most number of flights coming to it we can also find the airports with the least number of flights coming to it and then we can
find the airports with the most number of distinct outgoing flights from that outgoing data we can print out the top ten airports with the most number of distinct flights and finally we stopped
by running this application we can find that the number of airports in the graph
was 51 and the number of flights in the ground was 170 where the airports with
the vertices and the flights were the edges but can't find the most frequent flights between those airport vertices in this case Anchorage to Seattle and Seattle to Anchorage where the most frequent flights at Los Angeles to Seattle in Seattle Los Angeles for the next most frequent the busiest airport in Alaska Airlines route network with Seattle during 2008 and the airport for the most number of distinct outgoing flights for Alaska Airlines in 2008 where Seattle Anchorage Portland Oregon Juneau Alaska and San Francisco
having completed this lesson you should be able to describe the importance of
graft processing and discuss the graphics library and its building blocks
End of transcript. Skip to the start.



Naïve Bayes

"A simple probabilistic classifier which is based on the Bayes' Theorem with strong and naïve independence assumptions" - Wikipedia

It is the most basic text classification algorithm with many applications, such as spam detection, personal email sorting, language detection and more

classifier which is based on Bayes theorem with strong in naive

https://en.wikipedia.org/wiki/Naive_Bayes_classifier

Naïve Bayes

Naïve Bayes performs well in many complex real world problems

Two main variations of this algorithm are **Multinomial** and **Bernoulli**

Naïve Bayes is another supervised learning algorithm

the two main variations of the algorithm are multinomial and Bernoulli naive

The **Bernoulli** variation can be used in our problem when the absence of a particular word matters - i.e., in Spam or Adult Content Detection

The **Multinomial** variation is useful for text classification (we will use this one)

How Naïve Bayes Works

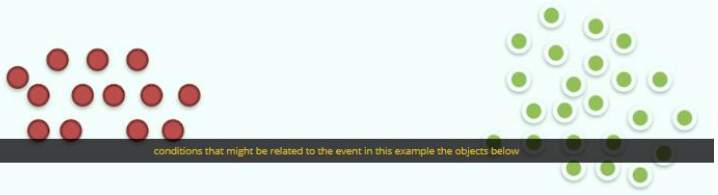
Based on Bayes' Theorem

In probability theory and statistics, Bayes' Theorem describes the probability of an event, based on conditions that might be related to the event

How Naïve Bayes Works

In this example, the objects can be classified as either GREEN or RED

We want to classify new cases as they arrive



conditions that might be related to the event in this example the objects below

Scala APACHE Spark COGNITIVE CLASS.ai

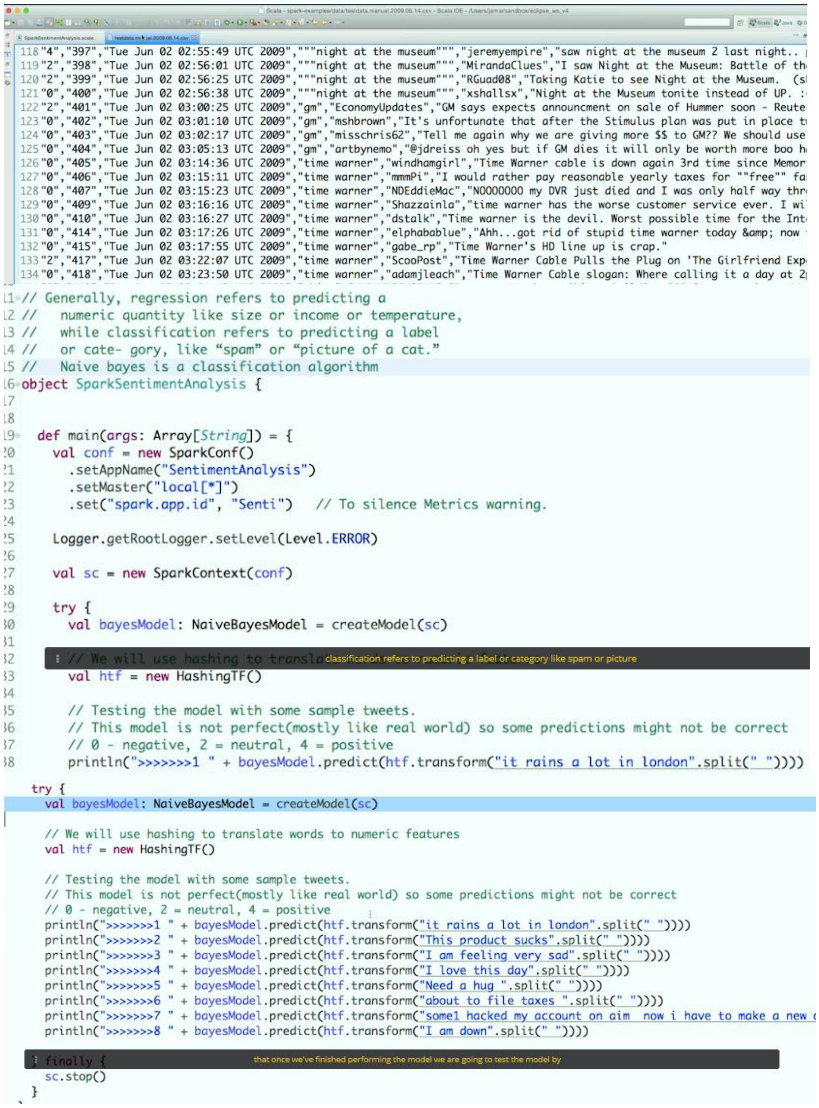
Formula

First, we calculate the likelihood of X given all green and red

Next, we combine it with the prior probability of green and red to classify whether the X will be green or red

Code Example: Sentiment Analysis

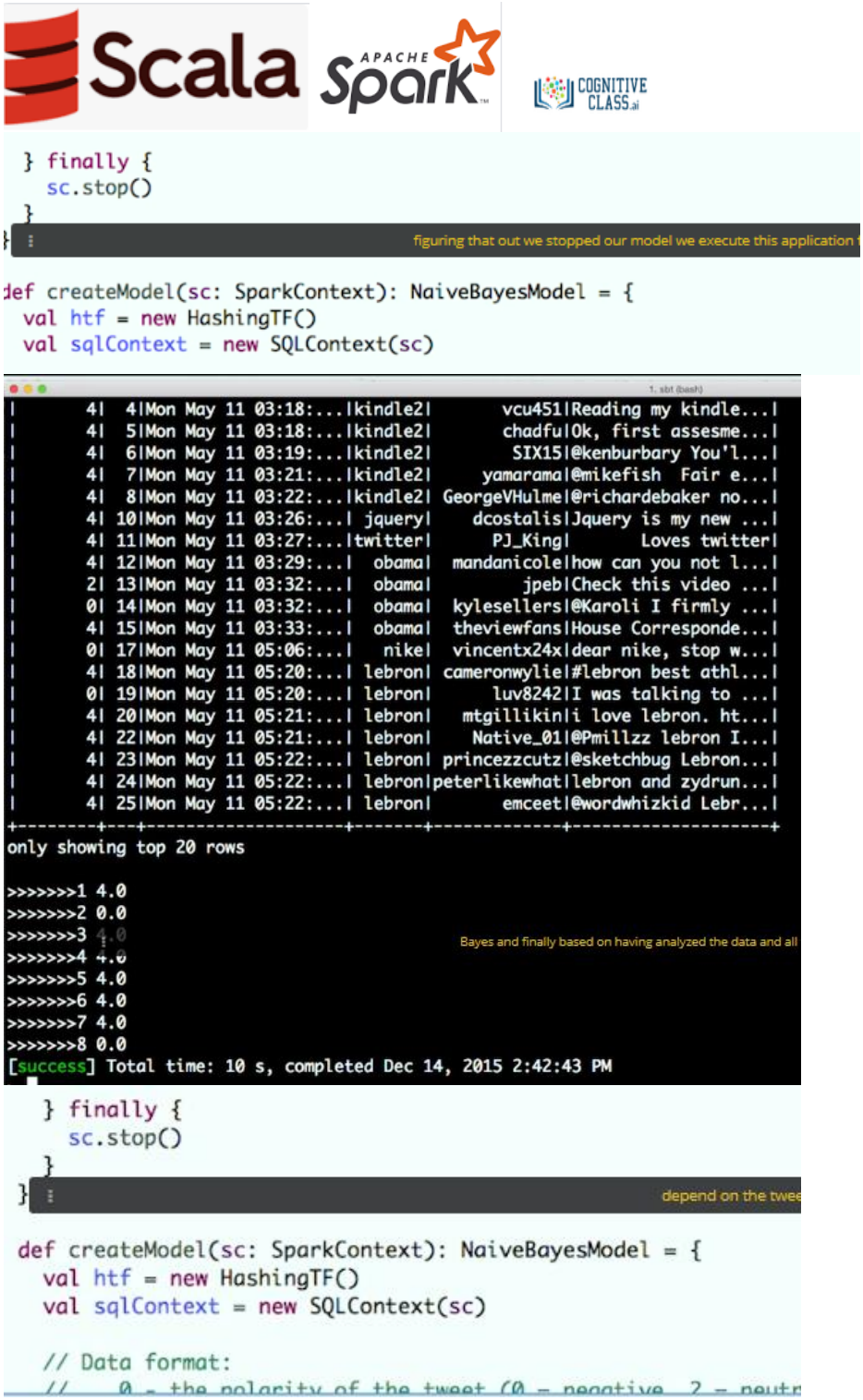
- Use the test data from sentiment140.com to train the Naïve Bayes model
- In the next lesson we will integrate it with Twitter



```
"4","397","Tue Jun 02 02:55:49 UTC 2009","""night at the museum""","jeremyempire","saw night at the museum 2 last night..
"2","398","Tue Jun 02 02:56:01 UTC 2009","""night at the museum""","MirandaClues","I saw Night at the Museum: Battle of th
"2","399","Tue Jun 02 02:56:25 UTC 2009","""night at the museum""","RGuad08","Taking Katie to see Night at the Museum.  (s
"0","400","Tue Jun 02 02:56:38 UTC 2009","""night at the museum""","xshallsx","Night at the Museum tonite instead of UP. :
"2","401","Tue Jun 02 03:00:25 UTC 2009","gm","EconomyUpdates","GM says expects announcment on sale of Hummer soon - Reute
"0","402","Tue Jun 02 03:01:10 UTC 2009","gm","mshbrown","It's unfortunate that after the Stimulus plan was put in place t
"0","403","Tue Jun 02 03:02:17 UTC 2009","gm","misschris62","Tell me again why we are giving more $$ to GM?? We should use
"0","404","Tue Jun 02 03:05:13 UTC 2009","gm","artbynemo","@jdreiss oh yes but if GM dies it will only be worth more boo h
"0","405","Tue Jun 02 03:14:36 UTC 2009","time warner","windhamgirl","Time Warner cable is down again 3rd time since Memor
"0","406","Tue Jun 02 03:15:11 UTC 2009","time warner","mmmPi","I would rather pay reasonable yearly taxes for ""free"" fa
"0","407","Tue Jun 02 03:15:23 UTC 2009","time warner","NDEddieMac","NOOOOOOO my DVR just died and I was only half way thr
"0","409","Tue Jun 02 03:16:16 UTC 2009","time warner","Shazzainla","time warner has the worse customer service ever. I wi
"0","410","Tue Jun 02 03:16:27 UTC 2009","time warner","dstalk","Time warner is the devil. Worst possible time for the Int
"0","414","Tue Jun 02 03:17:26 UTC 2009","time warner","elphabablue","Ahh...got rid of stupid time warner today &amp;amp; now
"0","415","Tue Jun 02 03:17:55 UTC 2009","time warner","gabe_rp","Time Warner's HD line up is crap."
"2","417","Tue Jun 02 03:22:07 UTC 2009","time warner","ScooPost","Time Warner Cable Pulls the Plug on 'The Girlfriend Exp
"0","418","Tue Jun 02 03:23:50 UTC 2009","time warner","adamjleach","Time Warner Cable slogan: Where calling it a day at 2
```

```
// Generally, regression refers to predicting a
//   numeric quantity like size or income or temperature,
//   while classification refers to predicting a label
//   or cate- gory, like “spam” or “picture of a cat.”
//   Naive bayes is a classification algorithm
object SparkSentimentAnalysis {


  def main(args: Array[String]) = {
    val conf = new SparkConf()
      .setAppName("SentimentAnalysis")
      .setMaster("local[*]")
      .set("spark.app.id", "Senti")   // To silence Metrics warning.

    Logger.getRootLogger.setLevel(Level.ERROR)

    val sc = new SparkContext(conf)

    try {
      val bayesModel: NaiveBayesModel = createModel(sc)

      // We will use hashing to transla classification refers to predicting a label or category like spam or picture
      val htf = new HashingTF()

      // Testing the model with some sample tweets.
      // This model is not perfect(mostly like real world) so some predictions might not be correct
      // 0 - negative, 2 = neutral, 4 = positive
      println(">>>>>>>1 " + bayesModel.predict(htf.transform("it rains a lot in london".split(" "))))
```

```
try {
  val bayesModel: NaiveBayesModel = createModel(sc)

  // We will use hashing to translate words to numeric features
  val htf = new HashingTF()

  // Testing the model with some sample tweets.
  // This model is not perfect(mostly like real world) so some predictions might not be correct
  // 0 - negative, 2 = neutral, 4 = positive
  println(">>>>>>>1 " + bayesModel.predict(htf.transform("it rains a lot in london".split(" "))))
  println(">>>>>>>2 " + bayesModel.predict(htf.transform("This product sucks".split(" "))))
  println(">>>>>>>3 " + bayesModel.predict(htf.transform("I am feeling very sad".split(" "))))
  println(">>>>>>>4 " + bayesModel.predict(htf.transform("I love this day".split(" "))))
  println(">>>>>>>5 " + bayesModel.predict(htf.transform("Need a hug ".split(" "))))
  println(">>>>>>>6 " + bayesModel.predict(htf.transform("about to file taxes ".split(" "))))
  println(">>>>>>>7 " + bayesModel.predict(htf.transform("some1 hacked my account on aim  now i have to make a new
  println(">>>>>>>8 " + bayesModel.predict(htf.transform("I am down".split(" "))))

} finally {                    that once we've finished performing the model we are going to test the model by
  sc.stop()
}
```



welcome to sentiment analysis of a Twitter stream part 1 after completing this lesson we should be able to describe naive Bayes algorithm and discuss how to build and train a machine learning algorithm from scratch from
Wikipedia sentiment analysis aka opinion mining refers to use of natural language
processing and text analysis to identify and extract subjective information in source materials sentiment analysis aims to determine the attitude of a speaker
or writer with respect to some topic or the overall contextual polarity of a document Wikipedia also says that naive Bayes is a simple probabilistic

classifier which is based on Bayes theorem with strong in naive independence assumptions is the most basic text classification algorithm with
many applications such as spam detection personal e mail sorting language detection and more naive Bayes performs well in many complex real-world problems
the two main variations of the algorithm are multinomial and Bernoulli naive Bayes is another supervised learning algorithm much like als as we saw before
the Bernoulli variation can be used in our problem when the absence of a particular word matters for example in spam or adult content detection the multinomial variation is useful for text classification and we will use this one in our example naive Bayes is based on Bayes theorem in probability theory and
statistics based theorem describes the probability of an event based on conditions that might be related to the event in this example the objects below
can be classified as either green or red we want to classify new cases as they
arrive so how do we classify and new element has come into our dataset first we calculate the likelihood of X given all green and red next we combine it with the prior probability of green and red to classify whether the ex will be green or red let's look at a code example using the test data from sentiment 140 dot com to train are naive Bayes model and the next lesson we will
integrate it with Twitter for live data analysis as an input to are naive Bayes model we were going to use a comma separated values file that we received from sentiment 140 dot com is going to include a bunch of tweets that were sent
out during a specific time frame back in 2009 now generally regression refers to
predicting a numeric quantity like size or income or temperature while classification refers to predicting a label or category like spam or picture of a cat naive Bayes is a classification algorithm we're going to set up our spark application as usual using a spark in Fig
and creating a new spark context now inside of our trying and finally block the person we're going to do is create are naive Bayes model and then after that once we've finished performing the model we are going to test the model by
printing out some information and finding out whether or not it could predict whether these statements are positive or negative let's look at the way we create our actual model first we're going to create are hashing tee eff and then we're going to create a sequel context of that we can use our data frames next we're going to read in the data for our spark CSB information
from sentiment 140 and going to give us some polarity information about the tweet as well as the Tweed idea that date the user and between information itself we're going to create a labeled RDD which is going to allow us to give

headings to the data inside of our RDD in this case polarity and tweet and then
we cashed the data we then asked the naive Bayes implementation to train on
this data and result in a new
base model that is returned from this function when we return from the create
model function we now have a base model that we can use to figure out whether or
not a set of statements are positive or negative the model has been trained based upon the data that it used in the tweets from our CSV file to figure out whether the following statements are positive or negative when we are done figuring that out we stopped our model we execute this application first we see
the scheme of the CSV data that we were bringing in and next we see a representation of the first 20 rows from which we did our analysis and are naive
Bayes and finally based on having analyzed the data and all the tweets that were reviewed in the model we now figure out whether there are tests tweet
are positive or negative if they are positive about you of four shows up for its polarity if they're negative about you 0 shows up and if they were somewhere in between a value of two shows up in this case some of our tweets
were really positive or negative but the model was not able to ascertain that based upon the data that was given to it let's have a look for the first element
we were told this is positive we see that right here however when we look at the source we see that the tweet said it rains a lot in London that may be a positive thing for someone but it may not be positive for others this can depend on the tweeter
having completed this lesson you should be able to describe the naive Bayes
algorithm and discuss how to build and train a machine learning algorithm
End of transcript. Skip to the start.

Spark Streaming (recap)

A minibatch model, data is captured in fixed time intervals

Each minibatch is stored in an RDD

The sequence of RDDs is held in a DStream

DStream provides window functions

Connecting to Twitter

Setup Twitter account to authorize the application

Use OAuth to connect to Twitter and read the stream data

Add the `spark-streaming-twitter` dependency

Import `TwitterUtils` and create a DStream

towards to connect to Twitter first we set up a Twitter account to authorize

```
1 import sbt.complete.Parsers._
2
3 name := "spark-examples"
4 version := "0.1.0"
5 scalaVersion := "2.10.5"
6
7 //default Spark version
8 val sparkVersion = "1.5.2"
9
10 libraryDependencies ++= Seq(
11   "org.apache.spark" %% "spark-core"              % sparkVersion withSources(),
12   "org.apache.spark" %% "spark-streaming"         % sparkVersion withSources(),
13   "org.apache.spark" %% "spark-sql"               % sparkVersion withSources(),
14   "org.apache.spark" %% "spark-hive"              % sparkVersion withSources(),
15   "org.apache.spark" %% "spark-mllib"             % sparkVersion withSources(),
16   "org.apache.spark" %% "spark-streaming-twitter" % sparkVersion withSources(),
17   "org.twitter4j"    %  "twitter4j-core"          % "3.0.3"      withSources(),
18   "com.databricks"   %% "spark-csv"               % "1.3.0"      withSources()
19 )
20
21 unmanagedResourceDirectories in Compile += baseDirectory.value / "conf"
22 unmanagedResourceDirectories in Test += baseDirectory.value / "conf"
23
24 initialCommands += """
25   import org.apache.spark.{SparkConf, SparkContext}
26   import org.apache.spark.SparkContext._
27   import org.apache.spark.sql.SQLContext
28   val conf = new SparkConf().
29     setMaster("local[*]").
```

streaming Twitter and the second will be the Twitter forge a core so that we can

```
21 object SparkSentimentAnalysisTweets {
22
23
24   def main(args: Array[String]): Unit = {
25     //read twitter tokens from arguments
26     val Array(consumerKey, consumerSecret, accessToken, accessTokenSecret) = args
27
28     val conf = new SparkConf()
29       .setAppName("SentimentAnalysis")
30       .setMaster("local[*]")
31       .set("spark.app.id", "Senti")   // To silence Metrics warning.
32
33     Logger.getRootLogger.setLevel(Level.ERROR)
34
35     val sc = new SparkContext(conf)
36     val ssc = new StreamingContext(sc, Seconds(5))
37
38     try {
39       val model: NaiveBayesModel = SparkSentimentAnalysis.createModel(sc)
40
41       val stream: DStream[Status] = TwitterUtils.createStream(ssc,
42         Twitter[illegible](consumerKey, consumerSecret, accessToken, accessTokenSecret),
43         Seq("programming", "computer", "crash") //filter criteria
44       )
45       stream
46         .filter(s => s.getUser.getLang == "en")
47         .filter(s => !s.isPossiblySensitive)
48         .map(s => (s.getUser.getName, s.getText))
49         .foreachRDD { rdd =>
```

OAuth specific information so that we can access an account in this case we

```
stream
  .filter(s => s.getUser.getLang == "en")
  .filter(s => !s.isPossiblySensitive)
  .map(s => (s.getUser.getName, s.getText))
  .foreachRDD { rdd =>
    val htf: HashingTF = new HashingTF()
    //this is the power of Spark where we are able to use algo. written for bat
    //addresses the Lambda architecture problems
    rdd.map {
      case (username, text) => (model.predict(htf.transform(text.split(" "))),
    }.foreach(println)
  }
```

and we were set up our spark streaming such that we can access data coming

```
} finally {
  ssc.stop(stopSparkContext = true, stopGracefully = true)
}
```

```
|      4| 12|Mon May 11 03:29:...| obama|  mandanicole|how can you not l...|
|      2| 13|Mon May 11 03:32:...| obama|         jpeb|Check this video ...|
|      0| 14|Mon May 11 03:32:...| obama| kylesellers|@Karoli I firmly ...|
|      4| 15|Mon May 11 03:33:...| obama| theviewfans|House Corresponde...|
|      0| 17|Mon May 11 05:06:...|  nike| vincentx24x|dear nike, stop w...|
|      4| 18|Mon May 11 05:20:...|lebron| cameronwylie|#lebron best athl...|
|      0| 19|Mon May 11 05:20:...|lebron|      luv8242|I was talking to ...|
|      4| 20|Mon May 11 05:21:...|lebron|  mtgillikin|i love lebron. ht...|
|      4| 22|Mon May 11 05:21:...|lebron|   Native_01|@Pmillzz lebron I...|
|      4| 23|Mon May 11 05:22:...|lebron| princezzcutz|@sketchbug Lebron...|
|      4| 24|Mon May 11 05:22:...|lebron|peterlikewhat|lebron and zydrun...|
|      4| 25|Mon May 11 05:22:...|lebron|       emceet|@wordwhizkid Lebr...|
+-------+---+--------------------+------+-------------+--------------------+
only showing top 20 rows

[Stage 6:>                                          (0 + 0) / 3](4.0,Two vehicle crash injures two -
imes https://t.co/ef5jzqYAwu #dodge #avenger)
(0.0,aww sis stepdad needs the computer toooooo)
[Stage 7:>                                          (0 + 0) / 8](4.0,Offers : https://t.co/7MBr9yUmk
ter Gaming Mouse Bekhic Optical Wired Metal Colors LED pad keybo… https://t.co/hYZtcTGqj7)
(4.0,Employers fined $20,000 after manager's quad bike crash https://t.co/ntLe4pxHTA)
(4.0,RT @ThePoqito: HEY GUYS WE ARE GIVING AWAY A COMPUTER, $200 RP, AND RAZER SWAG. RT RIGHT NOW AND ENJOY THE HOLI
EuVglqUDEY)
(4.0,RT @Maxgschneider: Found these on @MTV computer today @HoodieAllen #Priceless https://t.co/fGxhXfOSGS)
(0.0,We're stuck in the back up!  https://t
(4.0,Offers : https://t.co/NoWzHTYRgI #8392 #5624 6pc Multi-Functional Handheld Wired Trackball 1600dpi PC Computer
xN4foTPFF)
(0.0,A shuttle van crash has caused Old Wilmington Rd. to be closed from Fisher St. To Eastern Blvd.)
(4.0,A man died in a car crash while he was in an uber, the same road I always take ubers from damn)
(4.0,"2003 WROTE ON COMPUTER I I WAS ANGRY AT WENDY GA
#+++•^^^*$$$• W TO FELLOWS  YOU•••|
 IS SAME AS LUMP STAR WARS DAMON BNY.")
```

there we used to learn from and then we begin receiving the actual tweet data

Lesson Summary

Having completed this lesson, you should be able to:

- Describe how to connect Spark Streaming to Twitter
- Discuss steps required to integrate a machine learning algorithm to streaming

d.
welcome to sentiment analysis of a Twitter stream part 2 after completing this lesson we should be able to describe how to connect spark streaming to Twitter and discuss steps required to integrate a machine learning algorithm
to streaming data for a quick recap of spark streaming spark streaming involves
a mini badge model where data is captured and fixed time intervals each many batches stored in an RDD and the sequence of our DD's is held in addi
stream 2d stream provides windowing functions that we can perform algorithms
towards to connect to Twitter first we set up a Twitter account to authorize

the application next we used to connect to Twitter and read the stream data we
add the spark streaming Twitter dependency and we import Twitter utils and create a D stream we will use the naive Bayes model built in the previous
lesson and running on live tweets for example we will first add to dependencies to are billed for our project the first will be the spark streaming Twitter and the second will be the Twitter forge a core so that we can
parse JSON data from Twitter we want again perform naive Bayes on the data
coming from Twitter at the beginning of our application we have to pass in the
OAuth specific information so that we can access an account in this case we need to consumer key the consumer secret the access token and the access token
secret we once again create a spark configuration and from their Creator new
spark context and from there we will create a new streaming context it would take place within the rules of five seconds we will reuse our spark sentiment analysis create model function to generate our new naive Bayes model
and we were set up our spark streaming such that we can access data coming
directly from Twitter next we will stream the data through a series of transformations resulting in information about how positive or negative these tweets might be we start and then we run the streaming of data for 60 seconds finally when we are finished we stop the spark context and the streaming
of data gracefully the result of running this application means that we get the exact same data format that we had before the polarity between idea that date and its user and between information in this case we still have
our naive Bayes model showing how positive or negative two tweets were there we used to learn from and then we begin receiving the actual tweet data
live from Twitter note that the state is unfiltered and we apologize for any data
that may be deemed inappropriate but you'll notice that the left hand side of every tweet is a sentiment analysis saying whether or not this is a positive or negative tweet in some cases the model may not be correct such as a person positively saying I forgot to back up my life split files when I wipe my computer this is likely not a positive tweet when using a model such as naive Bayes the amount of data that you use your input to train the model is
important for the accuracy of how perceives the data in this case we use the small dataset and as a result the naive Bayes modeling of positivity of Tweed's may not be as accurate as it would be if we used a larger dataset having completed this lesson you should be able to describe how to connect spark

streaming to Twitter and discuss steps required to integrate a machine learning
algorithm to streaming data
End of transcript. Skip to the start.