



Sponsors for Azure Bootcamp



2019  
Global Azure  
**BOOTCAMP**



# Getting Started with Machine Learning using Spark in Azure HDInsight

Ingyu Lee  
Troy University

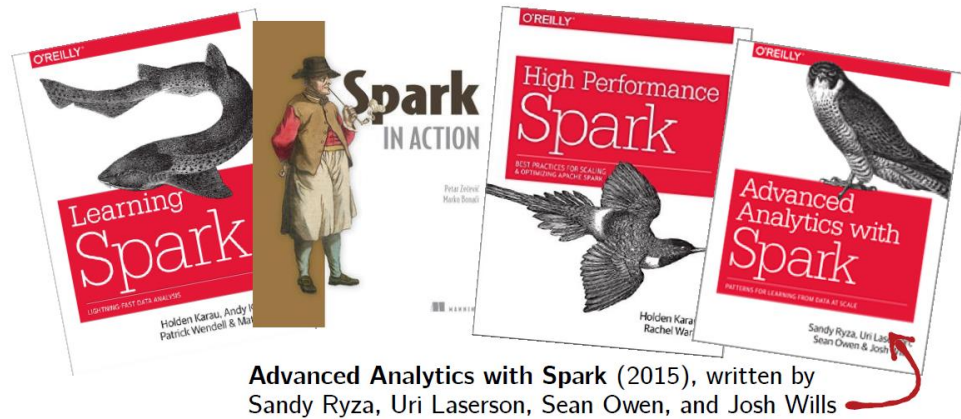
<https://github.com/Azure-Bootcamp-Troy/SparkOnHDInsight>

# Content

- What is **Spark** on **HDInsight**?
- Demo: Create Spark Cluster on Azure
- What is an **RDD, DataFrames, Spark SQL**?
- Demo: Operations of RDD, DataFrames and SQL
- What is **MLlib**?
- Demo: Machine Learning using Spark
- For Whom this talk
  - Business users to analyze Big Data using Spark on Azure Cluster.
- Prerequisite
  - None

# References

Many excellent books released in the past year or two!



**Advanced Analytics with Spark** (2015), written by Sandy Ryza, Uri Laserson, Sean Owen, and Josh Wills



**Big Data Analysis with Apache Spark**  
Learn how to apply data science techniques using parallel programming in Apache Spark to explore big data.



**Distributed Machine Learning with Apache Spark**  
Learn the underlying principles required to develop scalable machine learning pipelines and gain hands-on experience using Apache Spark.



**Introduction to Apache Spark**  
Learn the fundamentals and architecture of Apache Spark, the leading cluster-computing framework among professionals.

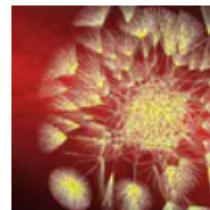
This talk is based on this course



## Big Data for Data Engineers

Yandex

[Learn More](#)



## Big Data

University of California, San Diego

[Learn More](#)

# HDInsight – What is it?

**A standard Apache Hadoop distribution offered as a managed service on Microsoft Azure**

- ❖ Based on Hortonworks Data Platform (HDP)
- ❖ Provisioned as clusters on Azure that can run on Windows or Linux servers
- ❖ Offers capacity-on-demand, pay-as-you-go pricing model
- ❖ Integrates with:
  - ❖ Azure Blob Storage and Azure Data Lake Store for Hadoop File System (HDFS)
  - ❖ Azure Portal for management and administration
  - ❖ Visual Studio for application development tooling

In addition to the core, HDInsight supports the Hadoop ecosystem



# What is Apache *Spark*?

General, open-source cluster computing engine

## Well-suited for machine learning

- Fast iterative procedures
- Efficient communication primitives

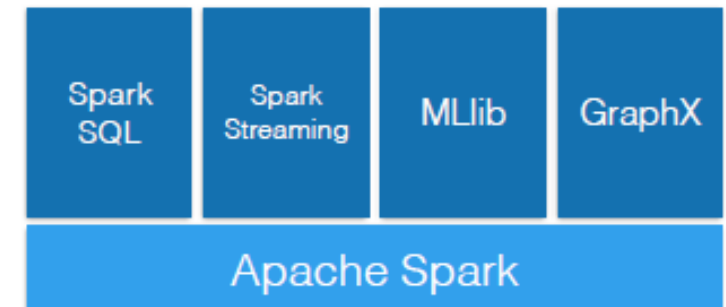
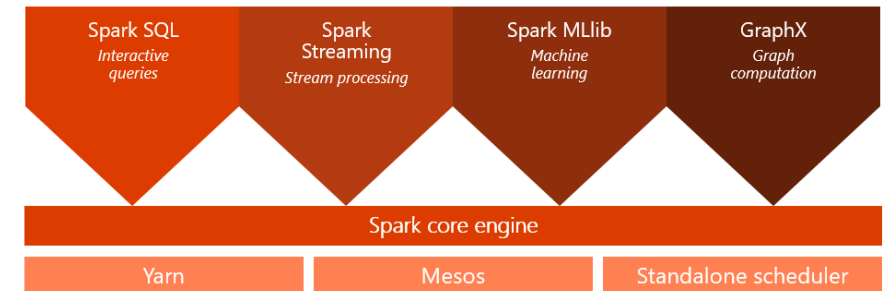
## Simple and Expressive

- APIs in Scala, Java, Python, R
- Interactive Shell

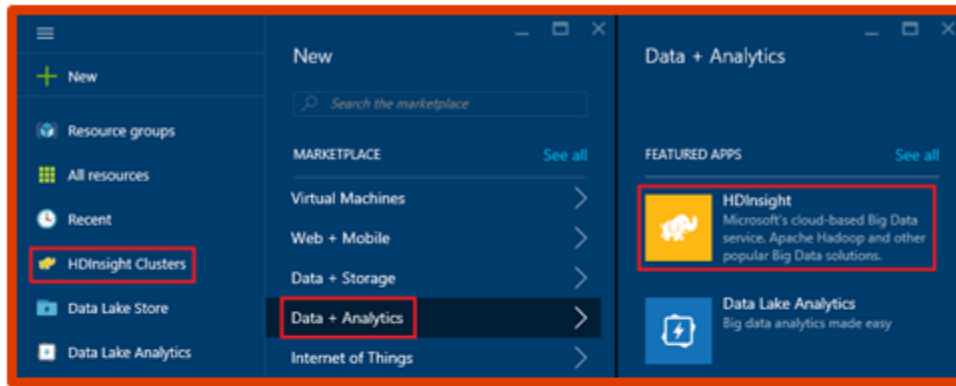
## Integrated Higher-Level Libraries

## Apache Spark: A unified framework

A unified, open source, parallel data processing framework for big data analytics



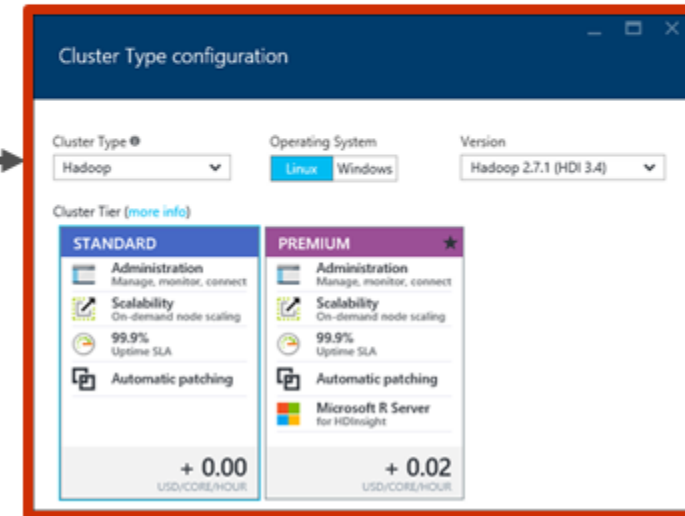
# Demo: Creating an HDInsight Spark cluster



Apache Spark use cases



The Azure console lists all types of HDInsight clusters (HBase, Storm, and Spark) currently provisioned.



# Demo: Ambari Dashboard

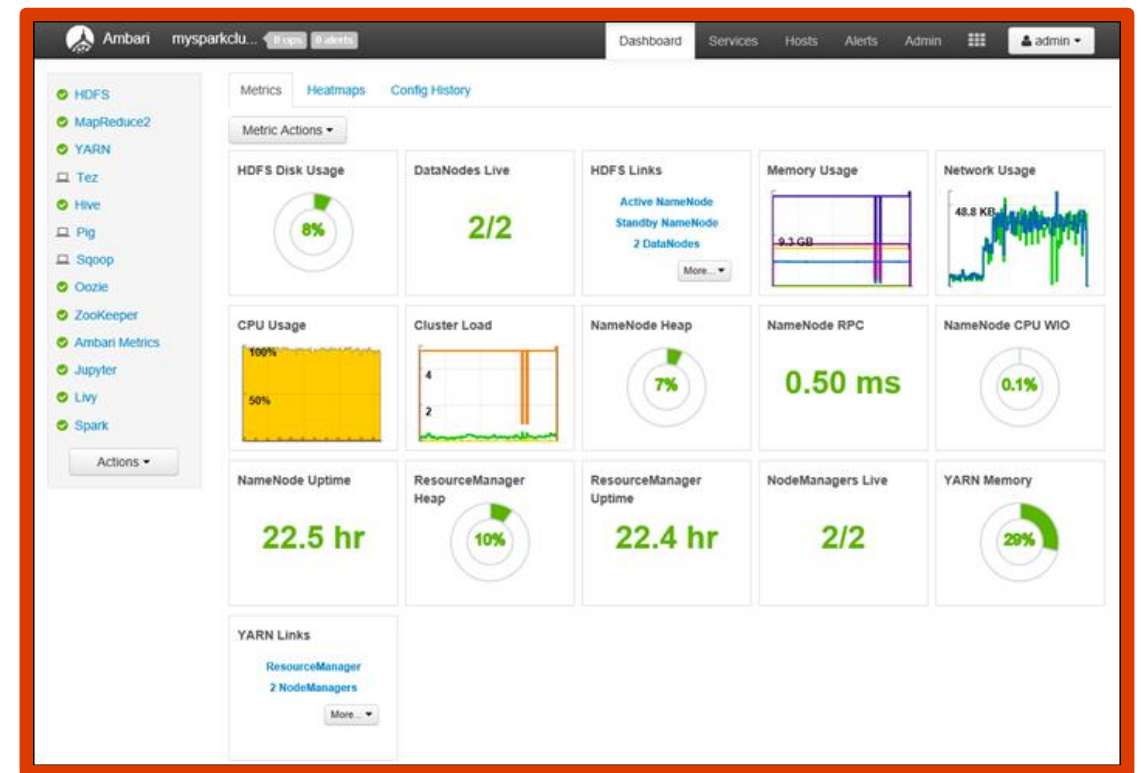


The Ambari Dashboard provides links to access:

Ambari web UI – monitors Spark clusters

**Hosts (Resource Manager)** – controls amount of resources allocated to various Spark cluster components

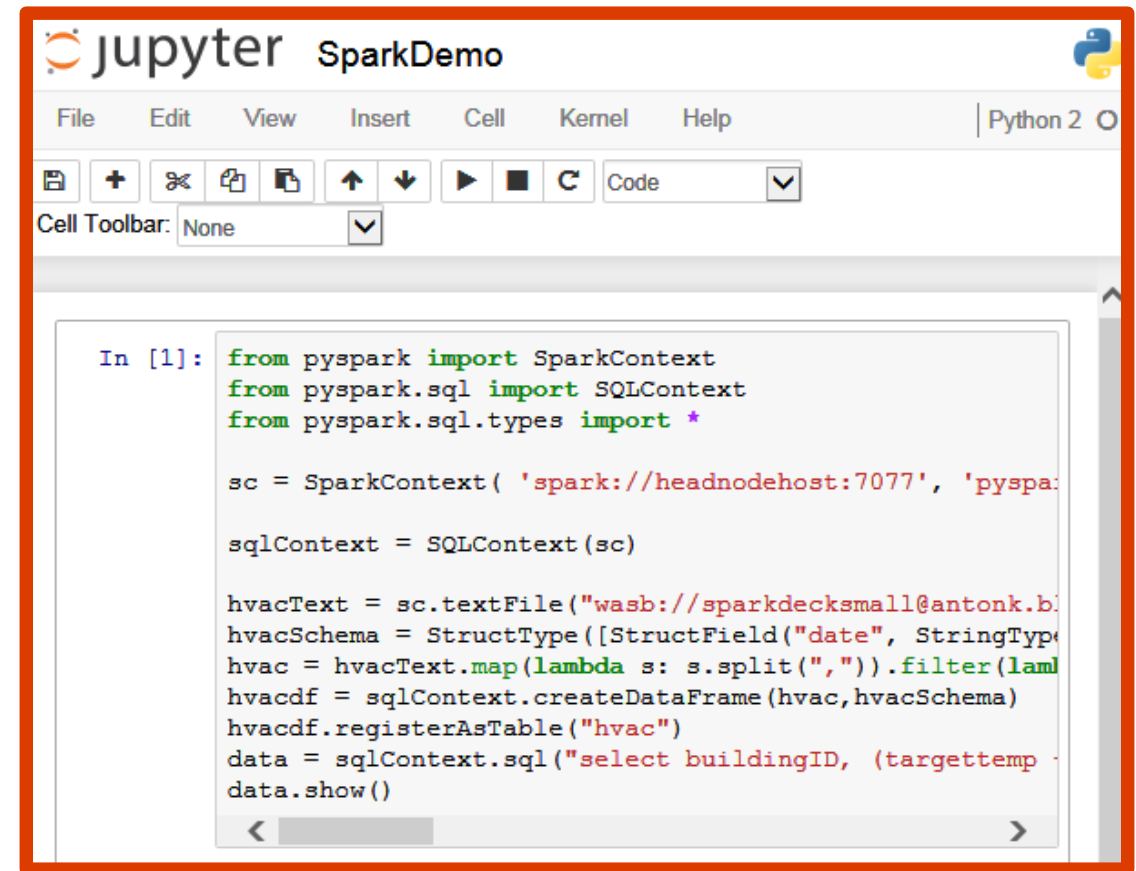
**Notebooks** – interactive web-based tools to develop and run Spark programs





# Demo: Jupyter notebooks

- Web-based interactive consoles for
  - Experimentation
  - Collaboration
- Spark HDInsight clusters include Jupyter
  - Interactive Python
  - Interactive Scala



The screenshot shows a Jupyter notebook window titled "jupyter SparkDemo". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". Below the menu is a toolbar with icons for saving, adding, deleting, and running code. The "Cell Toolbar" is set to "None". The main area displays a code cell with the following Python code:

```
In [1]: from pyspark import SparkContext
        from pyspark.sql import SQLContext
        from pyspark.sql.types import *

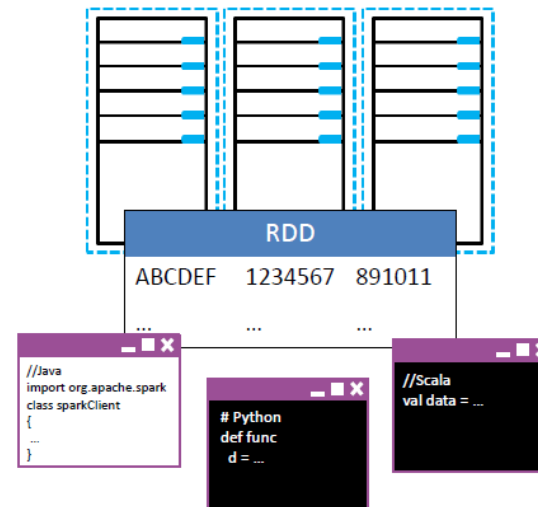
        sc = SparkContext( 'spark://headnodehost:7077', 'pyspa:
        sqlContext = SQLContext(sc)

        hvacText = sc.textFile("wasb://sparkdecksmall@antonk.b
        hvacSchema = StructType([StructField("date", StringType
        hvac = hvacText.map(lambda s: s.split(",")).filter(lam
        hvacdf = sqlContext.createDataFrame(hvac,hvacSchema)
        hvacdf.registerAsTable("hvac")
        data = sqlContext.sql("select buildingID, (targettemp
        data.show()
```

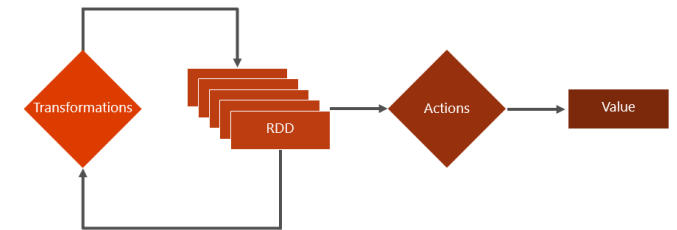
# What is RDD?

- The core abstraction for data in Spark is the *resilient distributed dataset* (RDD)
- An RDD represents a collection of items that can be distributed across compute nodes
- APIs for working with RDDs are provided for Java, Python, and Scala
  - HDInsight distribution includes Python and Scala shells

Dataset	Distributed	Resilient
Data storage created from: HDFS, S3, HBase, JSON, text, Local hierarchy of folders	Distributed across the cluster of machines	Recover from errors, e.g. node failure, slow processes
Or created transforming another RDD	Divided in partitions, atomic chunks of data	Track history of each partition, re-run



RDDs: Transformations and actions



# RDD Operations

## Some Transformations

Transformation	Description
<code>map(func)</code>	return a new distributed dataset formed by passing each element of the source through a function <i>func</i>
<code>filter(func)</code>	return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true
<code>distinct([numTasks])</code>	return a new dataset that contains the distinct elements of the source dataset
<code>flatMap(func)</code>	similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item)

Key-Value Transformation	Description
<code>reduceByKey(func)</code>	return a new distributed dataset of (K,V) pairs where the values for each key are aggregated using the given reduce function <i>func</i> , which must be of type (V,V) → V
<code>sortByKey()</code>	return a new dataset (K,V) pairs sorted by keys in ascending order
<code>groupByKey()</code>	return a new dataset of (K, Iterable<V>) pairs

## Some Actions

Action	Description
<code>reduce(func)</code>	aggregate dataset's elements using function <i>func</i> . <i>func</i> takes two arguments and returns one, and is commutative and associative so that it can be computed correctly in parallel
<code>take(n)</code>	return an array with the first <i>n</i> elements
<code>collect()</code>	return all the elements as an array <b>WARNING: make sure will fit in driver program</b>
<code>takeOrdered(n, key=func)</code>	return <i>n</i> elements ordered in ascending order or as specified by the optional key function

# What is DataFrames?

- A distributed collection of data organized into named columns.
- Similar to RDDs with schema.
- Conceptually equivalent to tables in relational database, or to DataFrames in R/Python.
- With domain-specific functions designed for common tasks:
  - Metadata
  - Sampling
  - Project, filter, aggregation, and join
  - UDFs

RDDs are a collection of opaque objects (such as internal structures unknown to Spark).

User

User

User

User

User

User

DataFrames is a collection of objects with schema that are known to Spark SQL..

Name

Age

Sex

Name

Age

Sex

Name

Age

Sex

Name

Age

Sex

# DataFrame Operations

DataFrames provide a domain-specific language for structured data manipulation in Scala, Java, and Python.

```
>>> val df = sqlContext.jsonFile("somejsonfile.json")

>>> df.show()                                // Show the contents of the DataFrame

>>> df.printSchema()                        // Print the schema in a tree format

>>> df.select("name").show()                // Select and show the name columns

>>> df.select(df("name"),df ("age") +1).show() // Select all but increment the age by 1

>>> df.filter(df("age") > 21).show()        // Select people older than 21

>>> df.groupBy("age").count().show()        // Count people by age
```

# What is Spark SQL?

1

Create an Azure storage account



2

HDInsight makes Apache Spark available as a service in cloud.



3

Run Spark SQL statements using notebooks

HDInsight uses Azure Blob storage account for storing data.

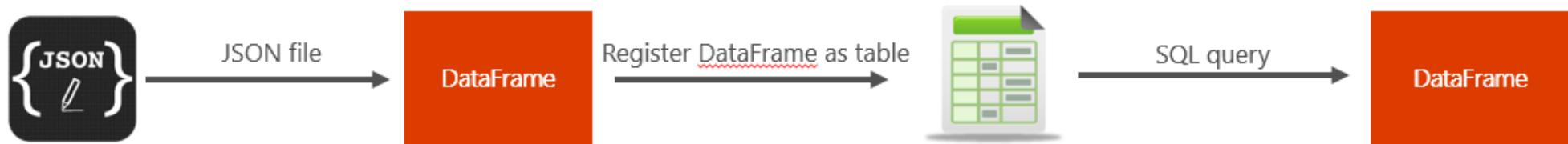
HDInsight makes Apache Spark available as a service in cloud.

You run interactive Spark SQL statements using notebooks.

# Tables and Queries

A DataFrame can be registered as a table that can then be used in SQL queries.

```
// First create a DataFrame from JSON file.  
>>> val df = sqlContext.jsonFile("Users.json")  
  
// Register the DataFrame as a temporary table. Temp tables exist only during lifetime of this SQLContext  
instance.  
>>> val usertable = sqlContext.registerDataFrameAsTable(df, "UserTable")  
  
// Alternatively, execute a SQL query on the table. The query returns a DataFrame.  
>>> val teenagers = sqlContext.sql("select Age as Years from UserTable where age > 13 and age <= 19")
```



# Spark SQL Operations

## Spark RDD API

```
rdd  
rdd1 = rdd.map(lambda x: x.split("\t"))  
rdd2 = rdd1.map(lambda x: (x[0], x[2]))
```

VS

## SQL on Spark

```
select user_id, url from access_log
```

- No data parsing
- Code optimization
- Syntax is easier
- No overhead



# What is Machine Learning?

## A Definition

Constructing and studying methods that learn from and make predictions on data

Broad area involving tools and ideas from various domains

- Computer Science
- Probability and Statistics
- Optimization
- Linear Algebra

## Some Examples

Face recognition



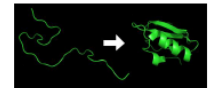
Link prediction



Text or document classification, e.g., spam detection



Protein structure prediction

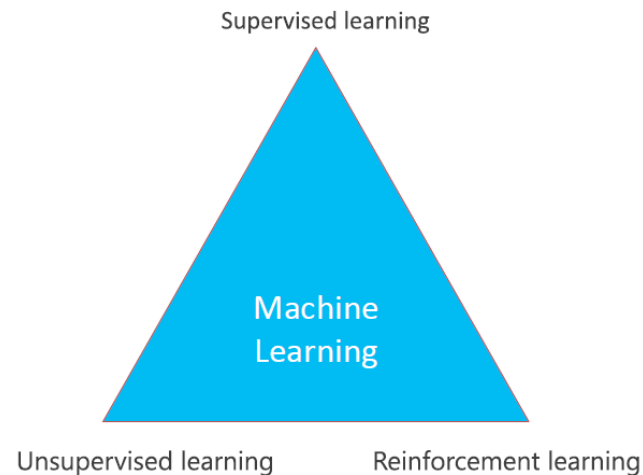


Games, e.g., Backgammon or Jeopardy



# How does Machine Learning work?

- Data labels
- Direct feedback
- Predict outcome

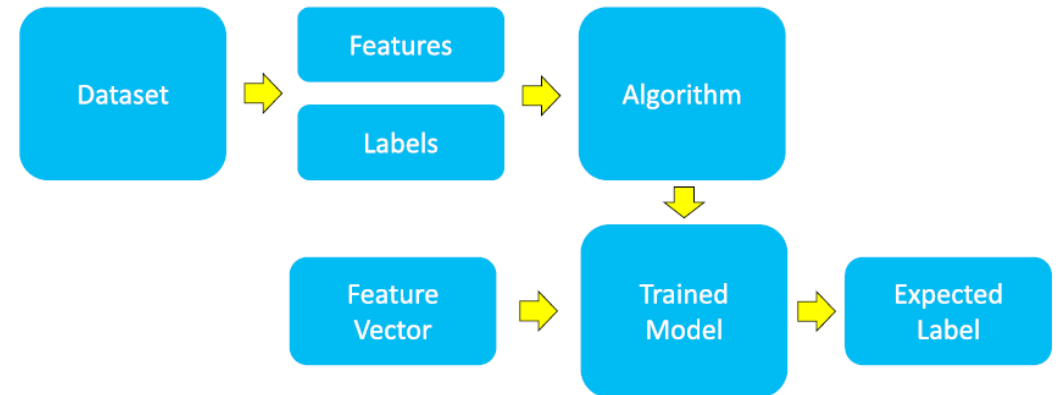


**Supervised learning.** Learning from labeled observations

- Labels 'teach' algorithm to learn mapping from observations to labels

**Unsupervised learning.** Learning from unlabeled observations

- Learning algorithm must find latent structure from features alone
- Can be goal in itself (discover hidden patterns, exploratory data analysis)
- Can be means to an end (preprocessing for supervised task)



**Observations.** Items or entities used for learning or evaluation, e.g., emails

**Features.** Attributes (typically numeric) used to represent an observation, e.g., length, date, presence of keywords

**Labels.** Values / categories assigned to observations, e.g., *spam*, *not-spam*

**Training and Test Data.** Observations used to train and evaluate a learning algorithm, e.g., a set of emails along with their labels

- Training data is given to the algorithm for training
- Test data is withheld at train time

# What is MLlib?

- MLlib is a collection of machine learning algorithms optimized to run in a parallel, distributed manner on Spark clusters.
- MLlib helps lead to better performance on large data sets.
- MLlib integrates seamlessly with other Spark components.
- MLlib applications are developed in Java, Scala, and Python.

Type	Algorithms
Supervised	Classification and regression: <ul style="list-style-type: none"><li>• Linear models (SVMs) logistic regression and linear regression</li><li>• Naïve Bayes</li><li>• Decision trees</li><li>• Ensembles of trees (random forest, gradient-boosted trees)</li><li>• Isotonic regression</li></ul>
Unsupervised	Clustering: <ul style="list-style-type: none"><li>• k-means and streaming k-means</li><li>• Gaussian mixture</li><li>• Power iteration clustering (PIC)</li><li>• Latent Dirichlet Allocation (LDA)</li></ul>
Recommendation	Collaborative filtering: <ul style="list-style-type: none"><li>• Alternating least squares (ALS)</li></ul>

# How to.. Machine Learning in Apache Spark?

- All primitives in Spark Machine Learning are *Vectors*
- *Features* are represented by a Vector
- Vectors can contain other Vectors and so be Dense or Sparse
- Spark uses *LabeledPoints* to encapsulate a Vector and a Label
- RDDs are transformed into Vectors through map functions

# MLlib Terminology

## ML: Transformer

- A *Transformer* is a class which can transform one DataFrame into another DataFrame
- A Transformer implements **transform()**
- Examples
  - HashingTF
  - LogisticRegressionModel
  - Binarizer

```
val tokenizer = new Tokenizer().setInputCol("text").setOutputCol("words")
val hashingTF = new
HashingTF().setNumFeatures(1000).setInputCol(tokenizer.getOutputCol).s
etOutputCol("features")
val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.01)
val pipeline = new Pipeline().setStages(Array(tokenizer, hashingTF, lr))
```

```
val model = pipeline.fit(training)
```

```
val modelelem = model.transform(test).select("id", "label", "text",
"probability", "prediction")
```

## ML: Estimator

- An *Estimator* is a class which can take a DataFrame and produce a Transformer
- An Estimator implements **fit()**
- Examples
  - LogisticRegression
  - StandardScaler
  - Pipeline

```
val tokenizer = new Tokenizer().setInputCol("text").setOutputCol("words")
val hashingTF = new
HashingTF().setNumFeatures(1000).setInputCol(tokenizer.getOutputCol).s
etOutputCol("features")
val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.01)
val pipeline = new Pipeline().setStages(Array(tokenizer, hashingTF, lr))
```

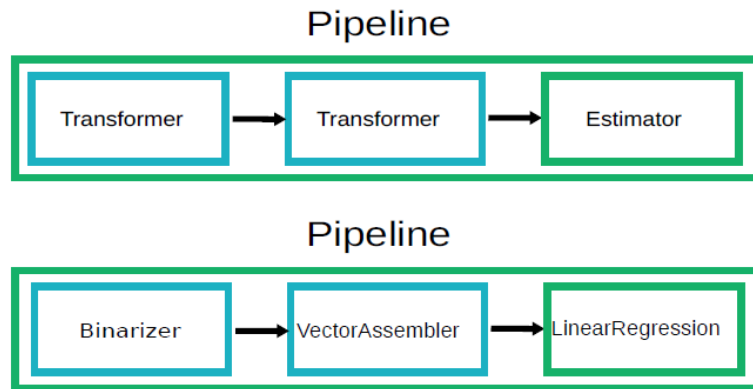
```
val model = pipeline.fit(training)
```

```
val modelelem = model.transform(test).select("id", "label", "text",
"probability", "prediction")
```

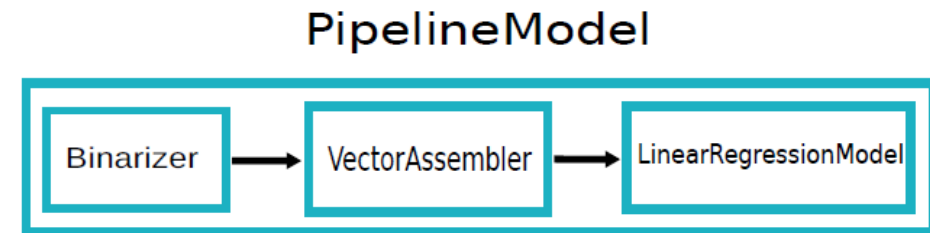
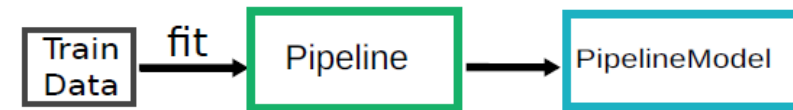
# MLlib Terminology

## ML: Pipelines

A *Pipeline* is an estimator that contains stages representing a reusable workflow. Pipeline stages can be either estimators or transformers.



## ML: PipelineModel



# Examples: Classification

## Prepare the Data

### Import Spark SQL and Spark ML Libraries

First, import the libraries you will need:

```
from pyspark.sql.types import *
from pyspark.sql.functions import *

from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler
```

```
data = csv.select("DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID", "DepDelay",
                  |((col("ArrDelay") > 15).cast("Int").alias("Late")))
data.show()
```

DayofMonth	DayOfWeek	OriginAirportID	DestAirportID	DepDelay	Late
19	5	11433	13303	-3	0
19	5	14869	12478	0	0
19	5	14057	14869	-4	0

## Split the Data

You will load this data into a DataFrame and display it.

### Load Source Data

```
csv = spark.read.csv('wasb:///data/flights.csv', inferSchema=True, header=True)
csv.show()
```

DayofMonth	DayOfWeek	Carrier	OriginAirportID	DestAirportID	DepDelay	ArrDelay
19	5	DL	11433	13303	-3	1
19	5	DL	14869	12478	0	-8

```
splits = data.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1]
train_rows = train.count()
test_rows = test.count()
print "Training Rows:", train_rows, " Testing Rows:", test_rows
```

```
assembler = VectorAssembler(inputCols = ["DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID", "DepDelay"],
                             outputCol="features")
training = assembler.transform(train).select(col("features"), col("Late").alias("label"))
training.show()
```

features	label
[1.0,1.0,10140.0,...]	0

## Prepare the Training Data

# Examples: Classification

## Train a Classification Model

```
lr = LogisticRegression(labelCol="label",featuresCol="features",maxIter=10,regParam=0.3)
model = lr.fit(training)
print "Model trained!"
```

Model trained!

## Prepare the Testing Data

```
testing = assembler.transform(test).select(col("features"), col("label").alias("trueLabel"))
testing.show()
```

features	trueLabel
[1.0,1.0,10140.0,...]	0
[1.0,1.0,10140.0,...]	0

## Test the Model

```
prediction = model.transform(testing)
predicted = prediction.select("features", "prediction", "probability", "trueLabel")
predicted.show(100)
```

features	prediction	probability	trueLabel
[1.0,1.0,10140.0,...]	0.0	[0.83121028425635...	0
[1.0,1.0,10140.0,...]	0.0	[0.82357817040192...	0



# Example: Regression

## Import Spark SQL and Spark ML Libraries

First, import the libraries you will need:

```
from pyspark.sql.types import *
from pyspark.sql.functions import *

from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler
```

Starting Spark application

## Load Source Data

```
csv = spark.read.csv('wasb:///data/flights.csv', inferSchema=True, header=True)
csv.show()
```

```
+-----+-----+-----+-----+-----+-----+
|DayofMonth|DayOfWeek|Carrier|OriginAirportID|DestAirportID|DepDelay|ArrDelay|
+-----+-----+-----+-----+-----+-----+
|      19|        5|    DL|      11433|      13303|      -3|        1|
|      19|        5|    DL|      14869|      12478|        0|       -8|
|      19|        5|    DL|      1405|             |         |         |
```

```
assembler = VectorAssembler(inputCols = ["DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID", "DepDelay"],
                             outputCol="features")
training = assembler.transform(train).select(col("features"), (col("ArrDelay").cast("Int").alias("label")))
training.show()
```

```
+-----+-----+
|      features|label|
+-----+-----+
|[1.0,1.0,10140.0,...]| -11|
|[1.0,1.0,10140.0,...]| -17|
```

## Prepare the Data

```
data = csv.select("DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID", "DepDelay", "ArrDelay")
data.show()
```

```
+-----+-----+-----+-----+-----+-----+
|DayofMonth|DayOfWeek|OriginAirportID|DestAirportID|DepDelay|ArrDelay|
+-----+-----+-----+-----+-----+-----+
|      19|        5|      11433|      13303|      -3|        1|
```

## Split the Data

```
splits = data.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1]
train_rows = train.count()
test_rows = test.count()
print "Training Rows:", train_rows, " Testing Rows:", test_rows
```

Training Rows: 1891270 Testing Rows: 810948

## Prepare the Training Data

# Example: Regression

```
: lr = LinearRegression(labelCol="label", featuresCol="features", maxIter=10, regParam=0.3)
  model = lr.fit(training)
  print "Model trained!"
```

Model trained!

## Train a Regression Model

```
testing = assembler.transform(test).select(col("features"), (col("ArrDelay")).cast("Int").alias("trueLabel"))
testing.show()
```

features	trueLabel
[1.0, 1.0, 10140.0, ...]	-5
[1.0, 1.0, 10140.0, ...]	-25

## Prepare the Testing Data

```
: prediction = model.transform(testing)
  predicted = prediction.select("features", "prediction", "trueLabel")
  predicted.show()
```

features	prediction	trueLabel
[1.0, 1.0, 10140.0, ...]	-3.7579519642834542	-5
[1.0, 1.0, 10140.0, ...]	-9.743936006015753	-25

## Test the Model

# Example: Pipeline

## Define the Pipeline

```
strIdx = StringIndexer(inputCol = "Carrier", outputCol = "CarrierIdx")
catVect = VectorAssembler(inputCols = ["CarrierIdx", "DayOfMonth", "DayOfWeek", "OriginAirportID", "DestAirportID"],
                          outputCol="catFeatures")
catIdx = VectorIndexer(inputCol = catVect.getOutputCol(), outputCol = "idxCatFeatures")
numVect = VectorAssembler(inputCols = ["DepDelay"], outputCol="numFeatures")
minMax = MinMaxScaler(inputCol = numVect.getOutputCol(), outputCol="normFeatures")
featVect = VectorAssembler(inputCols=["idxCatFeatures", "normFeatures"], outputCol="features")
dt = DecisionTreeClassifier(labelCol="label", featuresCol="features")
pipeline = Pipeline(stages=[strIdx, catVect, catIdx, numVect, minMax, featVect, dt])
```

```
pipelineModel = pipeline.fit(train)
print "Pipeline complete!"
```

## Run the Pipeline as an Estimator

```
prediction = pipelineModel.transform(test)
predicted = prediction.select("features", "prediction", "trueLabel")
predicted.show(100, truncate=False)
```

## Test the Pipeline Model

# More Example: Text Analysis

## Define the Pipeline

The pipeline for the model consist of the following stages:

- A Tokenizer to split the tweets into individual words.
- A StopWordsRemover to remove common words such as "a" or "the" that have little predictive value.
- A HashingTF class to generate numeric vectors from the text values.
- A LogisticRegression algorithm to train a binary classification model.

```
tokenizer = Tokenizer(inputCol="SentimentText", outputCol="SentimentWords")
swr = StopWordsRemover(inputCol=tokenizer.getOutputCol(), outputCol="MeaningfulWords")
hashTF = HashingTF(inputCol=swr.getOutputCol(), outputCol="features")
lr = LogisticRegression(labelCol="label", featuresCol="features", maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, swr, hashTF, lr])
```

```
: pipelineModel = pipeline.fit(train)
print "Pipeline complete!"
```

## Run the Pipeline as an Estimator

## Test the Pipeline Model

```
prediction = pipelineModel.transform(test)
predicted = prediction.select("SentimentText", "prediction", "trueLabel")
predicted.show(100, truncate = False)
```

# More Example: Clustering

## Create the K-Means Model

```
assembler = VectorAssembler(inputCols = ["Age", "MaritalStatus", "IncomeRange", "Gender", "TotalChildren",  
                                          "ChildrenAtHome", "Education", "Occupation", "HomeOwner", "Cars"], outputCol="features")  
train = assembler.transform(customers)  
  
kmeans = KMeans(featuresCol=assembler.getOutputCol(), predictionCol="cluster", k=5, seed=0)  
model = kmeans.fit(train)  
print "Model Created!"
```

## Get the Cluster Centers

```
centers = model.clusterCenters()  
print("Cluster Centers: ")  
for center in centers:  
    print(center)
```

## Predict Clusters

```
prediction = model.transform(train)  
prediction.groupBy("cluster").count().orderBy("cluster").show()
```

```
prediction.select("CustomerName", "cluster").show(50)
```

# More Example: Recommendation

## Import the ALS class

In this exercise, you will use the Alternating Least Squares (ALS) class.

```
from pyspark.ml.recommendation import ALS
```

## Load Source Data

The source data for the recommender is in two files - one containing numeric IDs for movies and users, and another containing ratings of the movies.

```
ratings = spark.read.csv('wasb:///data/ratings.csv', inferSchema=True, header=True)
movies = spark.read.csv('wasb:///data/movies.csv', inferSchema=True, header=True)
ratings.join(movies, "movieId").show()
```

## Build the Recommender

The ALS class is an estimator, so you can use its `fit` method to train a model, or you can include it in a pipeline. Regardless of the method, the ALS algorithm requires a numeric user ID, item ID, and rating.

```
als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="label")
model = als.fit(train)
```

## Test the Recommender

Now that you've trained the recommender, you can see how accurately it predicts known ratings in the test set.

## Prepare the Data

To prepare the data, split it into a training set and a test set.

```
data = ratings.select("userId", "movieId", "rating")
splits = data.randomSplit([0.7, 0.3])
train = splits[0].withColumnRenamed("rating", "label")
test = splits[1].withColumnRenamed("rating", "trueLabel")
train_rows = train.count()
test_rows = test.count()
print "Training Rows:", train_rows, " Testing Rows:", test_rows
```

```
prediction = model.transform(test)
prediction.join(movies, "movieId").select("userId", "title", "prediction", "trueLabel").show(100, truncate=False)
```

# Questions

