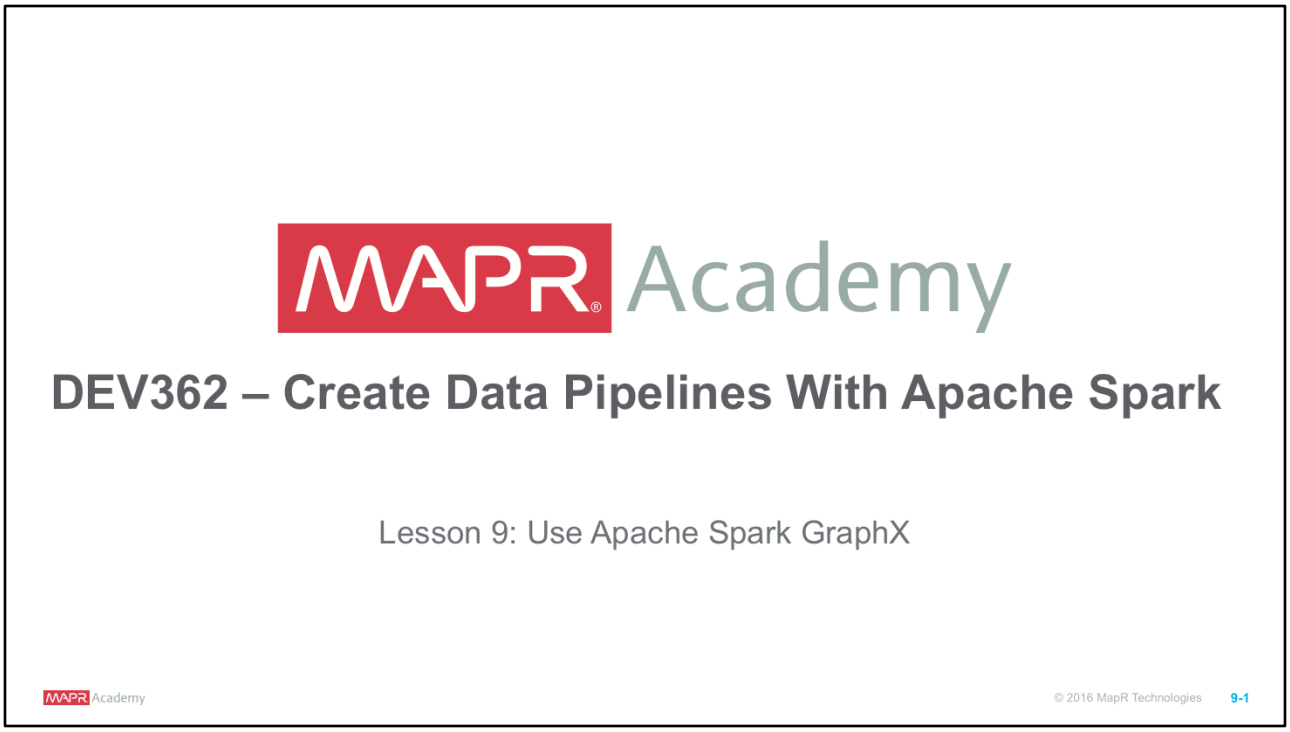


Welcome to DEV 362 lesson 9, Use Apache Spark GraphX.

Learning Goals

Learning Goals

- Describe GraphX
- Define Regular, Directed, and Property Graphs
- Create a Property Graph
- Perform Operations on Graphs

When you have finished with this lesson, you will be able to:

- Describe GraphX
- Define a regular, directed and property graph
- Create a property graph and
- Perform operations on graphs

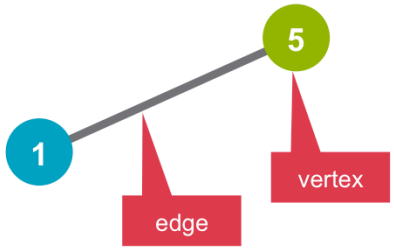
Learning Goals

- **Describe GraphX**
- Define Regular, Directed, and Property Graphs
- Create a Property Graph
- Perform Operations on Graphs

The first section introduces Apache Spark GraphX.

What is a Graph?

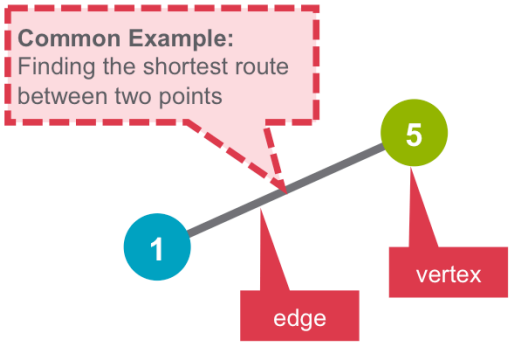
A way to represent a set of vertices that may be connected by edges.



A graph is a way of representing a set of vertices that may be connected by edges.

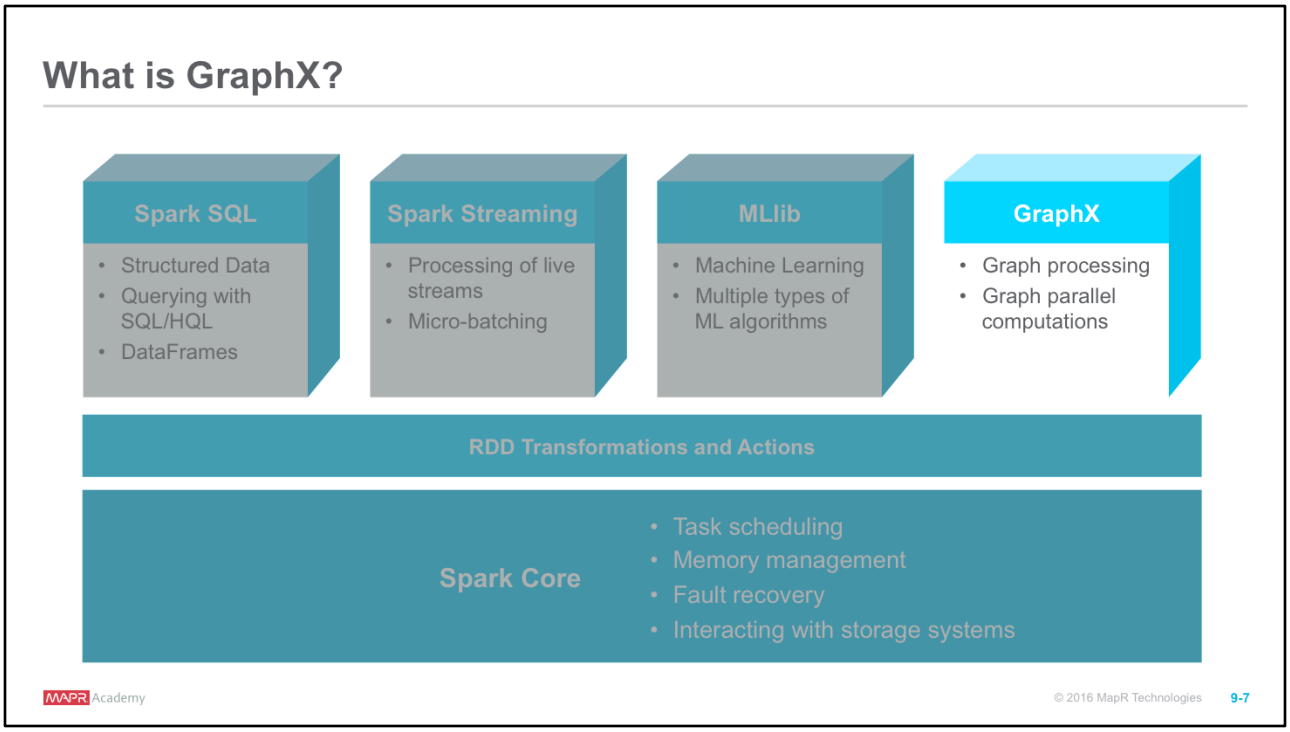
What is a Graph?

A way to represent a set of vertices that may be connected by edges.



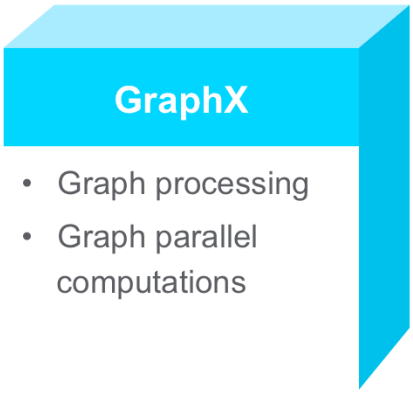
A common example of graph processing is to find the shortest route between two points where the points are the vertices and the routes are edges.

Need more info? Learn about graph theory: https://en.wikipedia.org/wiki/Graph_theory



GraphX is the Apache Spark component for graphs and graph-parallel computations. It is a distributed graph processing framework that sits on top of Spark core.

Apache Spark GraphX



- Spark component for graphs and graph-parallel computations
- Combines data parallel and graph parallel processing in single API
- View data as graphs and as collections (RDD)
  - no duplication or movement of data
- Operations for graph computation
- Provides graph algorithms and builders

GraphX combines data parallel and graph parallel processing in a single API.

We can view data as graphs and as collections without the need for duplication or movement of data.

GraphX has a number of operators that can be used for graph computations.

GraphX also provides graph algorithms and builders for graph analytics.

Knowledge Check

Knowledge Check

GraphX is the Apache Spark component for graphs and graph parallel computations. Which of the characteristics listed below are true of GraphX?

1. GraphX sits on top of Spark Core
2. GraphX provides distinct APIs for data parallel and graph parallel processing
3. GraphX provides multiple operators, including an optimized version of Pragel
4. When using GraphX, we can view data as graphs and as collections without the need for duplication or movement of data

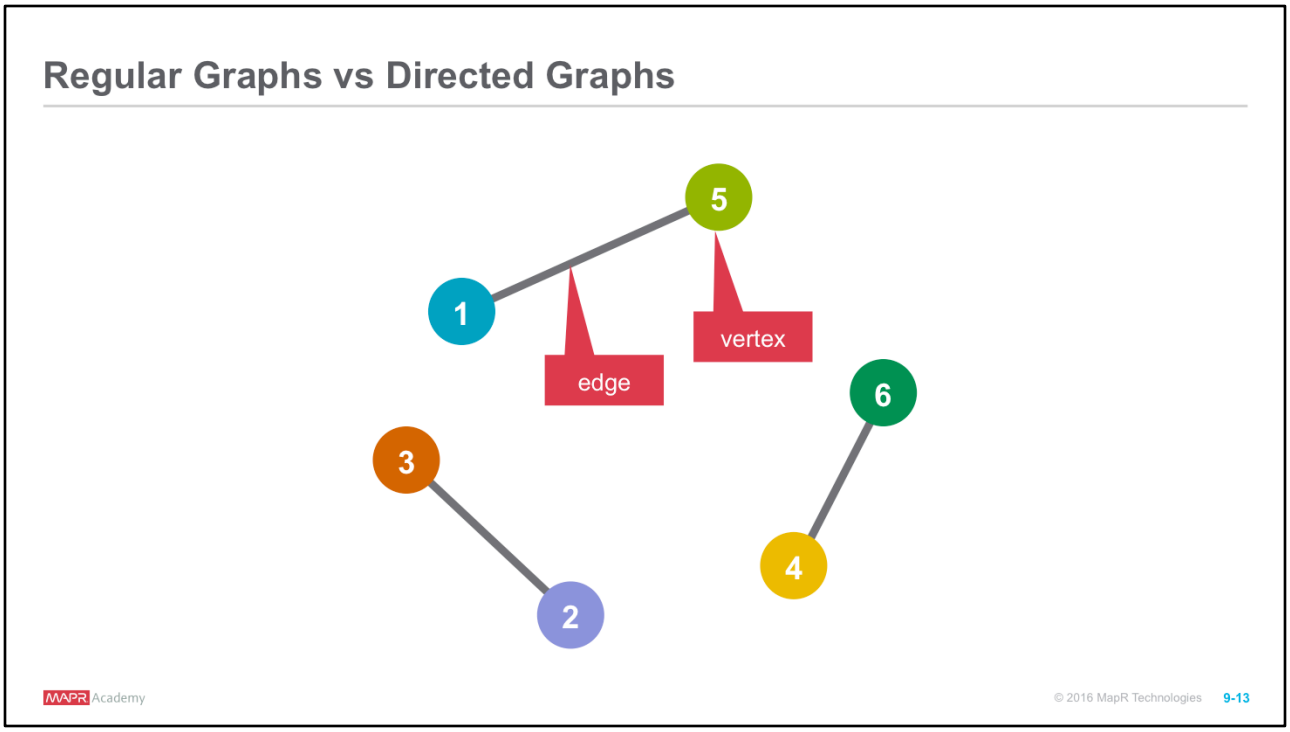
True
False – these are available in a single API
True
True

Learning Goals

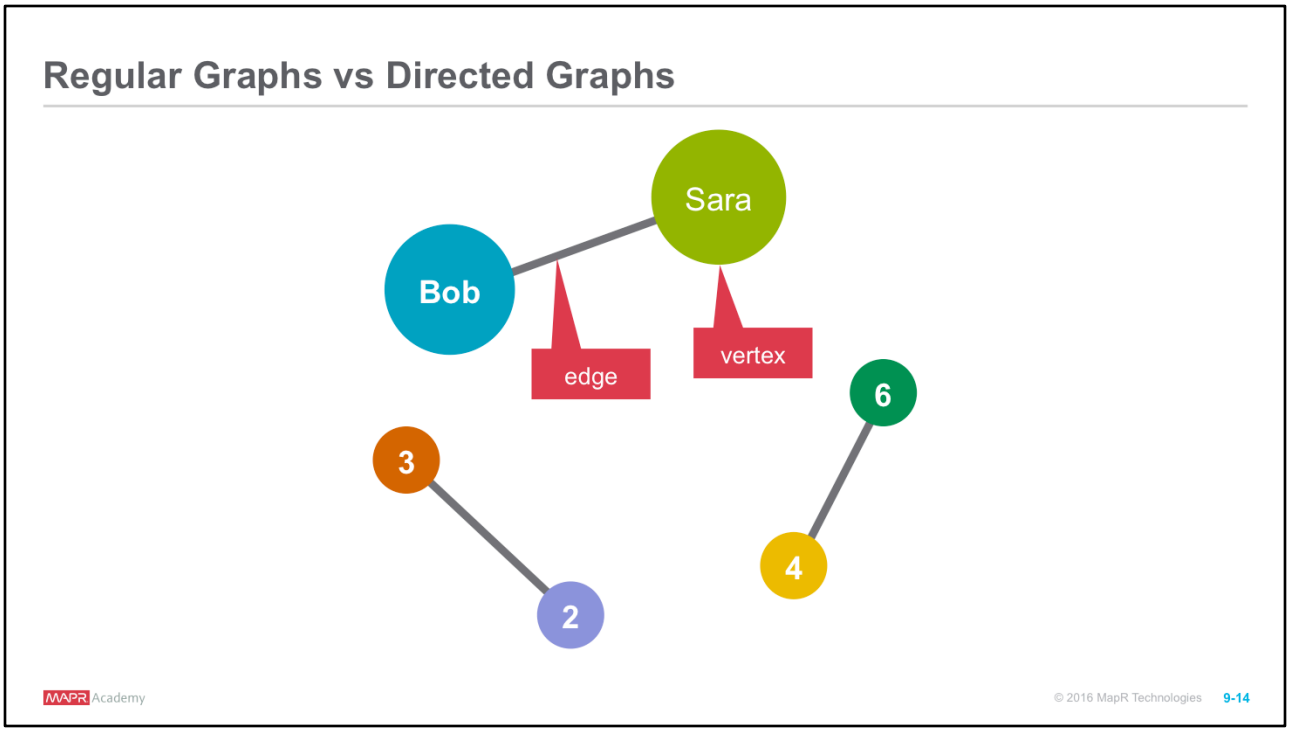
Learning Goals

- Describe GraphX
- **Define Regular, Directed, and Property Graphs**
- Create a Property Graph
- Perform Operations on Graphs

In this section, we will define and regular, directed and property graphs.

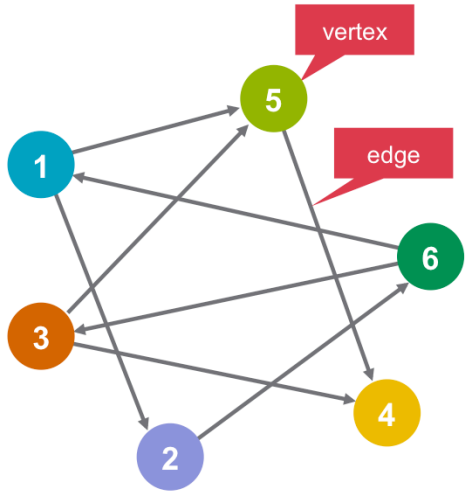


A regular graph is one where each vertex has the same number of edges.



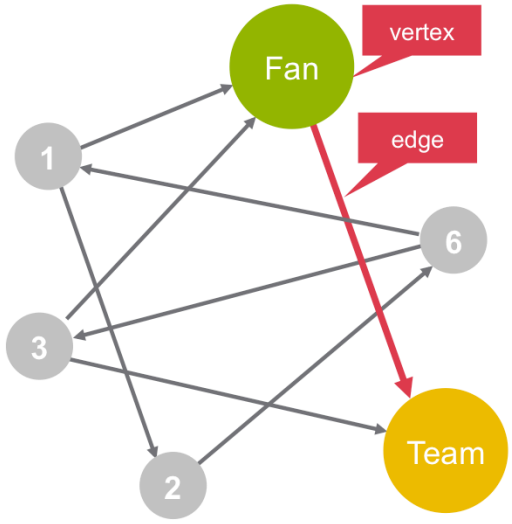
An example of a regular graph, is where the users on Facebook are vertices. If Bob is a friend of Sara, then Sara is also a friend of Bob.

Regular Graphs vs Directed Graphs



A directed graph or digraph…

Regular Graphs vs Directed Graphs



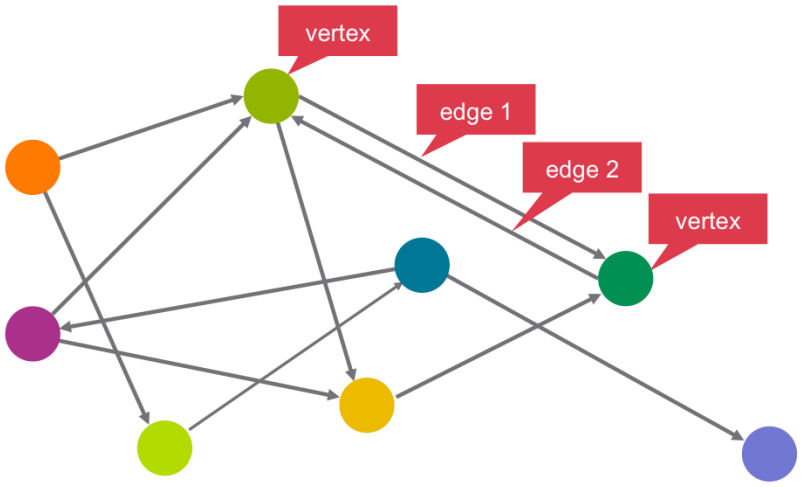
MAPR Academy

 9-16

is one in which edges run in one direction from vertex Fan to vertex Team.

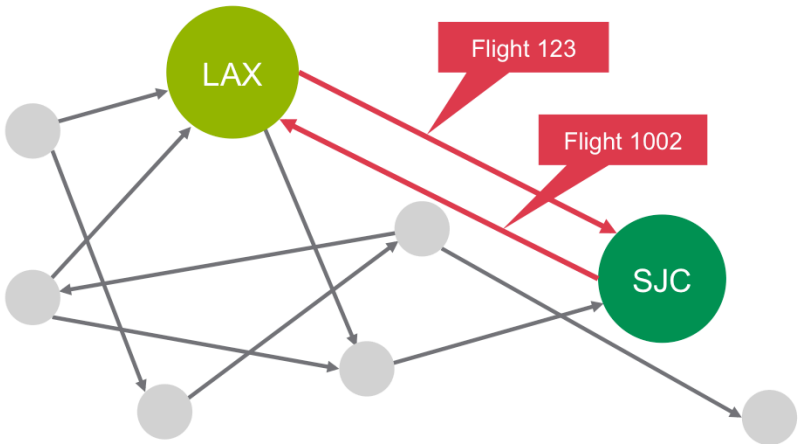
An example of a directed graph is a Twitter follower. User Fan can follow user Team without implying that user Team follows user Fan.

Property Graph



A property graph is the primary abstraction of Spark GraphX.

Property Graph



A property graph is a directed multigraph which can have multiple edges in parallel. Every edge and vertex has user defined properties associated with it, and therefore is unique.

Property graphs are immutable, distributed and fault-tolerant.

Knowledge Check

Knowledge Check

The primary abstraction in Spark GraphX is the property graph. Which characteristics below are true of a property graph?

1. Property graphs are bidirectional
2. Edges and vertices have user-defined properties associated with them
3. Property graphs are directional
4. Every edge and vertex is unique
5. Property graphs are immutable

False
True
True
True
True

Learning Goals

Learning Goals

- Define GraphX
- Define Regular, Directed, and Property Graphs
- **Create a Property Graph**
- Perform Operations on Graphs

In this section, we will create a property graph.

Create a Property Graph

1. Import required classes
2. Create vertex RDD
3. Create edge RDD
4. Create graph

To create a property graph, follow the steps listed here. We will describe each step in more detail, using flight data.

Create a Property Graph: Data Set

Airports

Vertex ID	Property (V)
org_id	(id, name, numAirlines, numFlights, numDestinations)

Routes

Source ID	Dest ID	Property (E)
org_id	dest_id	(src, dst, name, numAirlines, numFlights, distance, crsTime, majorLines)

We have the flight data streaming in. Airports are the vertices in our graph, with routes being the edges. Listed here are the properties we assign to the vertices (airports) and edges (routes).

Create a Property Graph

1. Import required classes

```
import org.apache.spark._
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD
```


When creating a property graph, first we import the GraphX related classes listed here.

Create a Property Graph

2 Create vertex RDD

```
//defining the case class first
case class Airport(id:Int, name:String, numAirlines:Int,
numFlights:Int, numDestinations:Int)
```

id	(id, name, numAirlines, numFlights, numDestinations)
14843	Airport(14843,SJU,7,2309,22)

MAPR Academy

Next we define the case class, which includes the properties that we define for the vertices.

The airports have the following properties:

- id is the ID of the airport where the flight originates
- name is the name of the airport where the flight originates
- numAirlines is the number of airlines at that airport
- numFlights is the number of flights at the airport
- And numDestinations is the number of destinations that flights go to

Create a Property Graph

2 Create vertex RDD

```
//create the vertex rdd
```

```
val airports = ardd.map(x => (x(org_id), x)).groupByKey.map(x => (x._1.toLong, {
  val id = x._1.toInt
  val name = x._2.map(y=>y(origin)).head
  val numAirlines = x._2.map(y=>y(carrier)).toList.distinct.size
  val numFlights = x._2.size
  val numDestinations = x._2.map(y=>y(dest_id)).toList.distinct.size
  Airport(id, name, numAirlines, numFlights, numDestinations)
}))
```


In this example, we are making the assumption that the data has been loaded into an RDD called ardd.

Create a Property Graph

2 Create vertex RDD

```
//create the vertex rdd
```

```
val airports = ardd.map(x => (x(org_id), x)).groupByKey.map(x => (x._1.toLong, {
  val id = x._1.toInt
  val name = x._2.map(y=>y(origin)).head
  val numAirlines = x._2.map(y=>y(carrier)).toList.distinct.size
  val numFlights = x._2.size
  val numDestinations = x._2.map(y=>y(dest_id)).toList.distinct.size
  Airport(id, name, numAirlines, numFlights, numDestinations)
}))
```

airport ID

origin

distinct airlines

total number of flights

number of destinations

We build the vertex RDD as shown here.
The result is a tuple with the origin ID, followed by an array of the properties of the vertex.

Create a Property Graph

2 Create vertex RDD

```
//create the vertex rdd
```

```
val airports = ardd.map(x => (x(org_id), x)).groupByKey.map(x => (x._1.toLong, {
  val id = x._1.toInt
  val name = x._2.map(y=>y(origin)).head
  val numAirlines = x._2.map(y=>y(carrier)).toList.distinct.size
  val numFlights = x._2.size
  val numDestinations = x._2.map(y=>y(dest_id)).toList.distinct.size
  Airport(id, name, numAirlines, numFlights, numDestinations)
}))
```

org_id

org_id	(id, name, numAirlines, numFlights, numDestinations)
14843	Airport(14843,SJU,7,2309,22)

Each vertex must have a vertex ID, which in this case is the origin airport ID.

Create a Property Graph

2 Create vertex RDD

```
//create the vertex rdd
```

```
val airports = ardd.map(x => (x(org_id), x)).groupByKey.map(x => (x._1.toLong, {
  val id = x._1.toInt
  val name = x._2.map(y=>y(origin)).head
  val numAirlines = x._2.map(y=>y(carrier)).toList.distinct.size
  val numFlights = x._2.size
  val numDestinations = x._2.map(y=>y(dest_id)).toList.distinct.size
  Airport(id, name, numAirlines, numFlights, numDestinations)
}))
```

airport ID

origin

distinct airlines

total number of flights

number of destinations

org_id	(id, name, numAirlines, numFlights, numDestinations)
14843	Airport(14843,SJU,7,2309,22)

We can define properties for the vertices. In our example, the properties for the vertices are shown in the table. We compute the distinct list of airlines from the carrier field, total number of flights, and the number of distinct destinations.

Create a Property Graph

3 Create edge RDD

```
case class Route(src:Int, dst:Int, name:String,
numAirlines:Int, numFlights:Int, distance:Int,
crsTime:Double, majorLines:Boolean)
```

org_id	dest_id	(src, dst, name, numAirlines, numFlights, distance, crsTime, majorLines)
11618	13495	Route(11618,13495,EWR->MSY,3,69,1167,50.0,true)

Now, we define the Route case class, and then create the routes RDD.

Create a Property Graph

3 Create edge RDD

```
//create the routes rdd
```

```
val routes = ardd.map(x => ((x(org_id),x(dest_id)), x)).groupByKey.map(x => (x._1, {
  val src = x._1._1.toInt
  val dst = x._1._2.toInt
  val name = x._2.map(y=>y(origin)).head + "->" + x._2.map(y=>y(dest)).head
  val numAirlines = x._2.map(y=>y(carrier)).toList.distinct.size
  val numFlights = x._2.size
  val distance = x._2.map(y=>y(dist)).head.toFloat.toInt
  val crsTime = x._2.map(y=>y(crselapsedtime)).flatMap(y=>y).head
  val majorLines = x._2.map(y=>y(carrier)).map(y => List("AA", "UA").contains(y)).reduce(_ || _)
  Route(src, dst, name, numAirlines,numFlights,distance,crsTime, majorLines)
}))
```

source ID

destination ID

route name

distinct airlines

total number of flights

distance

crs elapsed time

creating list of major airlines

An edge in a property graph will have the source ID, destination ID, and properties. The edge RDD must contain a source ID, destination ID, and edge properties. Before we create the edge RDD, we first create the routes RDD that contains the required information.

The source ID and destination IDs are numbers, and we are therefore converting them to integers. We construct the route name as shown in the example, with the origin airport code to the destination airport code. As before, we compute the distinct list of airlines and the total number of flights. The distance and elapsed time are both part of the data. We define a list of major airlines. In this case, carriers AA and UA are defined as major airlines. We check to see if the carrier is a major airline and return true or false accordingly.

Next we will use this routes RDD to create the edge RDD.

Create a Property Graph

3 Create edge RDD

```
//create the edge rdd
val edges = routes.map(x => Edge(x._1._1.toLong, x._1._2.toLong, x._2))

edges.first
Res: Edge(11618,13495,Route(11618,13495,EWR->MSY,3,69,1167,50.0,true))
```


We now use the Edge case class to map the routes RDD to an Edge RDD. The Edge class stores the edge property. To see what this RDD looks like, use edges.first, as shown here.

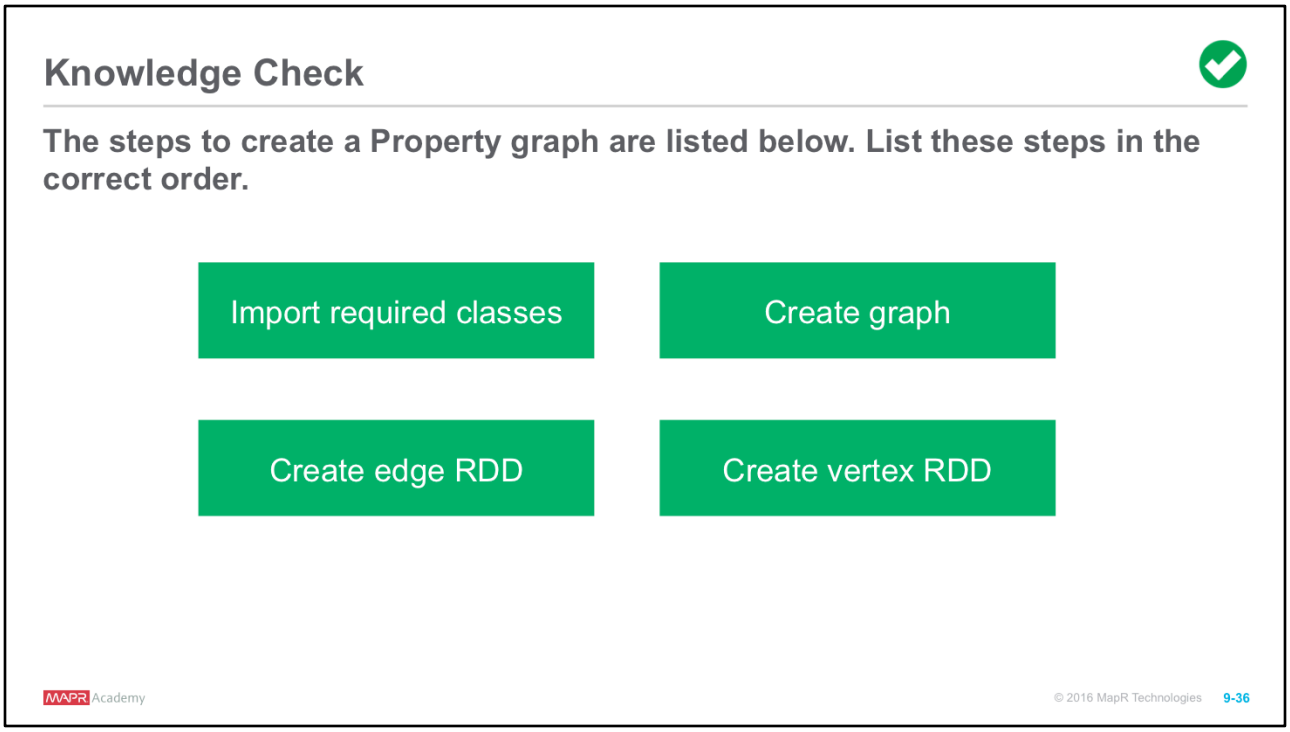
Create a Property Graph

4 Create graph

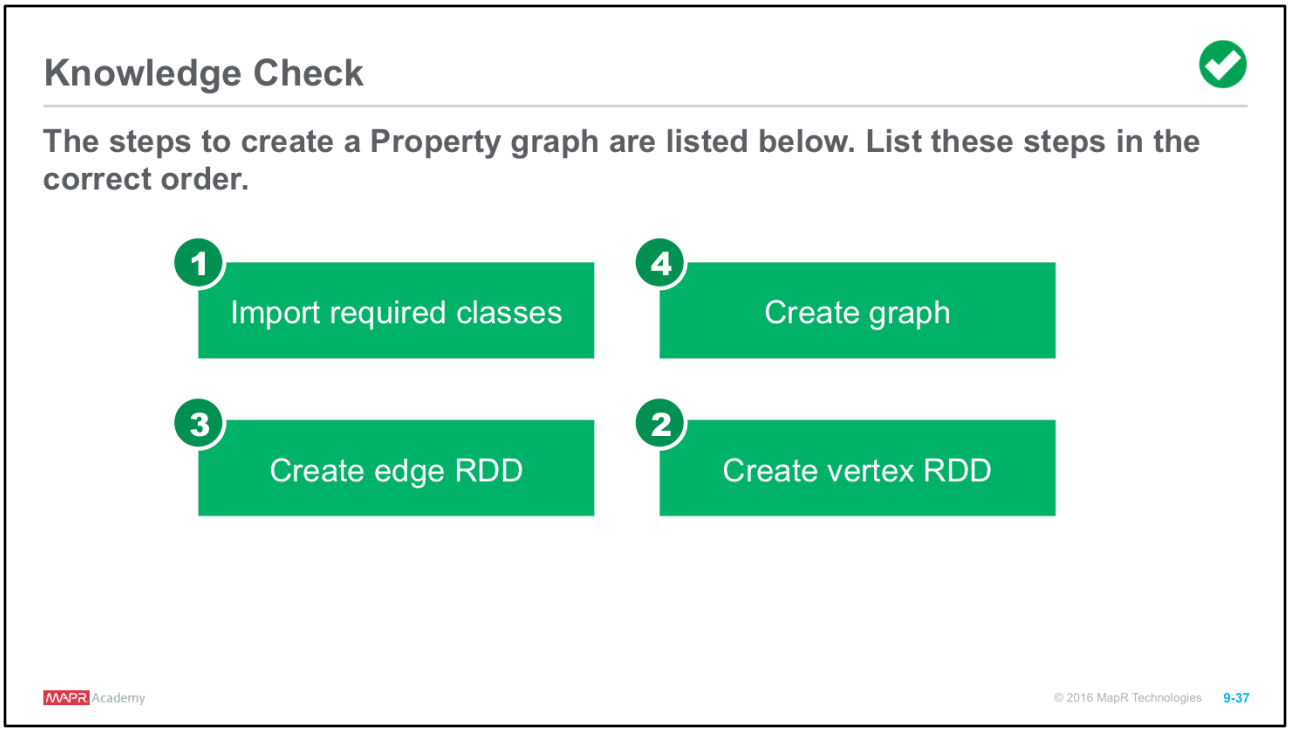
```
// define default airport
val nowhere = Airport(0,0,0,0,0)

//build initial graph
val graph = Graph(airports, edges, nowhere)
```


Once the vertex and edge RDDs are defined, we can create the property graph. To create the graph use the Graph method which accepts (vertex, edge, default vertex). We define a default vertex as shown and then create the graph.

Knowledge Check



1. Import required classes
2. Create vertex RDD
3. Create edge RDD
4. Create graph



1. Import required classes
2. Create vertex RDD
3. Create edge RDD
4. Create graph

Learning Goals

Learning Goals

- Define GraphX
- Define Regular, Directed, and Property Graphs
- Create a Property Graph
- **Perform Operations on Graphs**

In this section, we will look at the different types of operators on property graphs.

Graph Operators

To find information about the graph

Operator	Description
`numEdges`	number of edges `(Long)`
`numVertices`	number of vertices `(Long)`
`inDegrees`	The in-degree of each vertex `(VertexRDD[Int])`
`outDegrees`	The out-degree of each vertex `(VertexRDD[Int])`
`degrees`	The degree of each vertex `(VertexRDD[Int])`

Here are some operators that we can use to find more information about the graph, such as the number of edges and number of vertices.

Graph Operators

Caching Graphs

Operator	Description
`cache()`	Caches the vertices and edges; default level is `MEMORY_ONLY`
`persist(newLevel)`	Caches the vertices and edges at specified storage level; returns a reference to this graph
`unpersist(blocking)`	Uncaches both vertices and edges of this graph
`unpersistVertices(blocking)`	Uncaches only the vertices, leaving edges alone

The operators listed here can be used for caching graphs.

Graph Operators

Other Operators – Collection and Structural

Operator	Description
`vertices`	An RDD containing the vertices and associated attributes
`edges`	An RDD containing the edges and associated attributes
`subgraph(epred, vpred)`	Restricts graph to only vertices and edges satisfying the predicates
`reverse`	Reverses all edges in the graph

Some of the operators listed here can be used to view graphs as collections such as vertices and edges. The other operators listed here can be used to modify the graph structure. There are many more operators that can be used on graphs. Refer to the Spark GraphX API documentation.

Need more info?
http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.graphx.package
http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.graphx.Graph

Graph Operators

Property operators:

- Similar to map on RDD
- Result in new graph

Operator	Description
`mapVertices`	Transforms each vertex attribute using map function
`mapEdges`	Transforms each edge attribute, a partition at a time, using map function – does **not** pass the vertex value to the edge
`mapTriplets`	Transforms each edge attribute, a partition at a time, using the map function – **does** pass the vertex value to the edge

Property operators are similar to the map operators on RDDs, and result in a new graph.

Graph Operators

To answer questions such as:

- How many airports are there?
- How many routes for flights?
- Which airport has the most incoming flights?
- Which airports are covered by major airlines?

We can use the graph operators to answer the questions listed here.

Class Discussion

Class Discussion

1. How many airports are there?
   - In our graph, what represents airports?
   - Which operator could you use to find the number of airports?
2. How many routes are there?
   - In our graph, what represents routes?
   - Which operator could you use to find the number of routes?

How Many Airports are There?

How many airports are there?

- In our graph, what represents airports?

  Vertices

- Which operator could you use to find the number of airports?

  `graph.numVertices`

The first question we want to ask our data is, “How many airports are there?”

PageRank

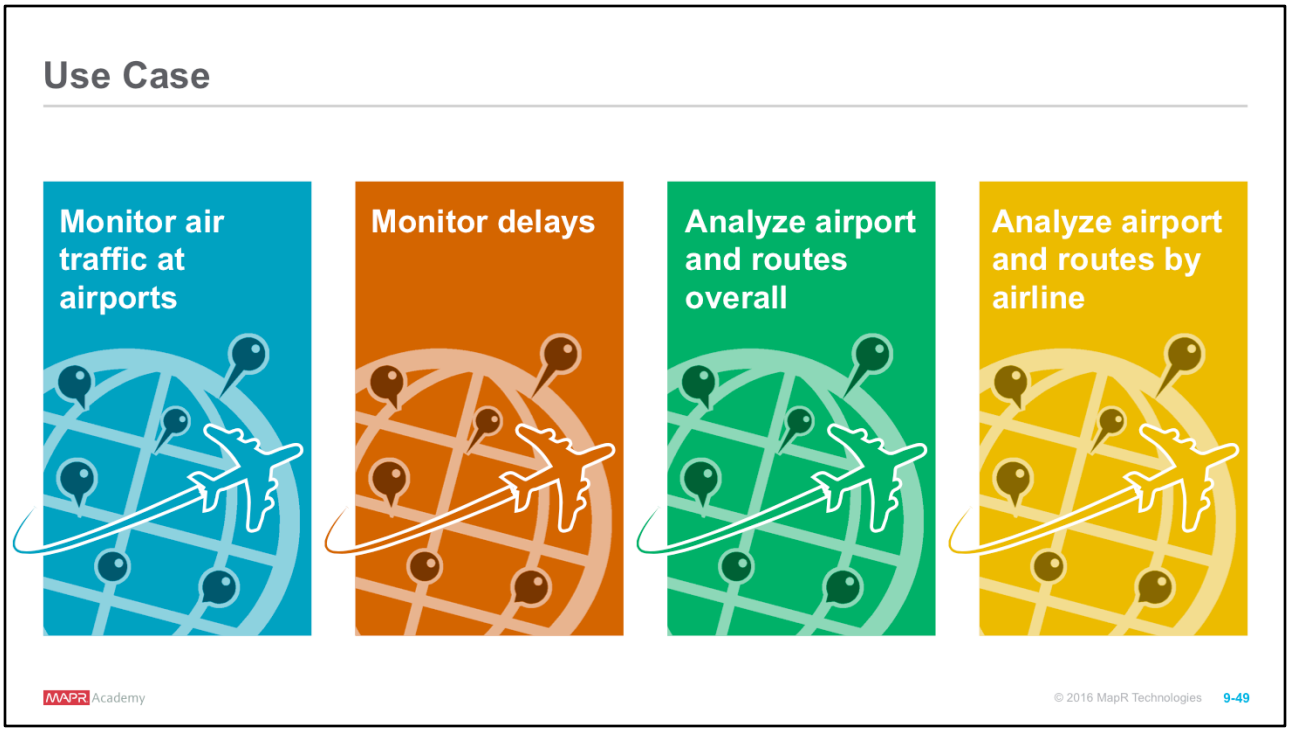
```
graph.pageRank(tolerance).vertices
```

- Measure the importance of vertices
- `tolerance` is a measure of convergence
- Returns a graph with vertex attributes

Another operator is PageRank. It is based on the Google PageRank algorithm. It can be used to measure the importance of the vertices. We have to specify the tolerance, which is the measure of convergence.

PageRank on a graph will return a graph with vertex attributes containing the normalized edge weight.

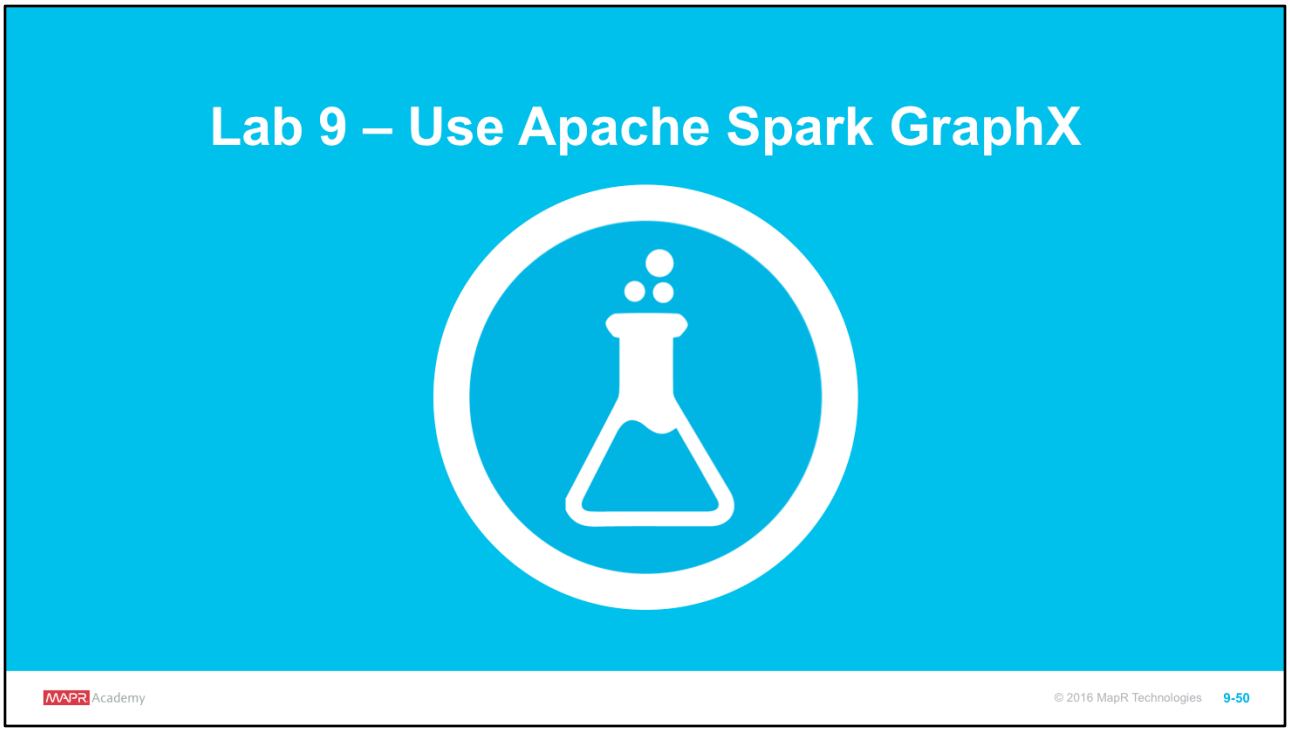
In our example, this would be to answer the question – Which are the most important airports?



You can use graphs in these different situations. For security reasons, on a particular day, you want to monitor the data in real time. In this case, you may add graph processing to a streaming application and send the output to a visualization tool.

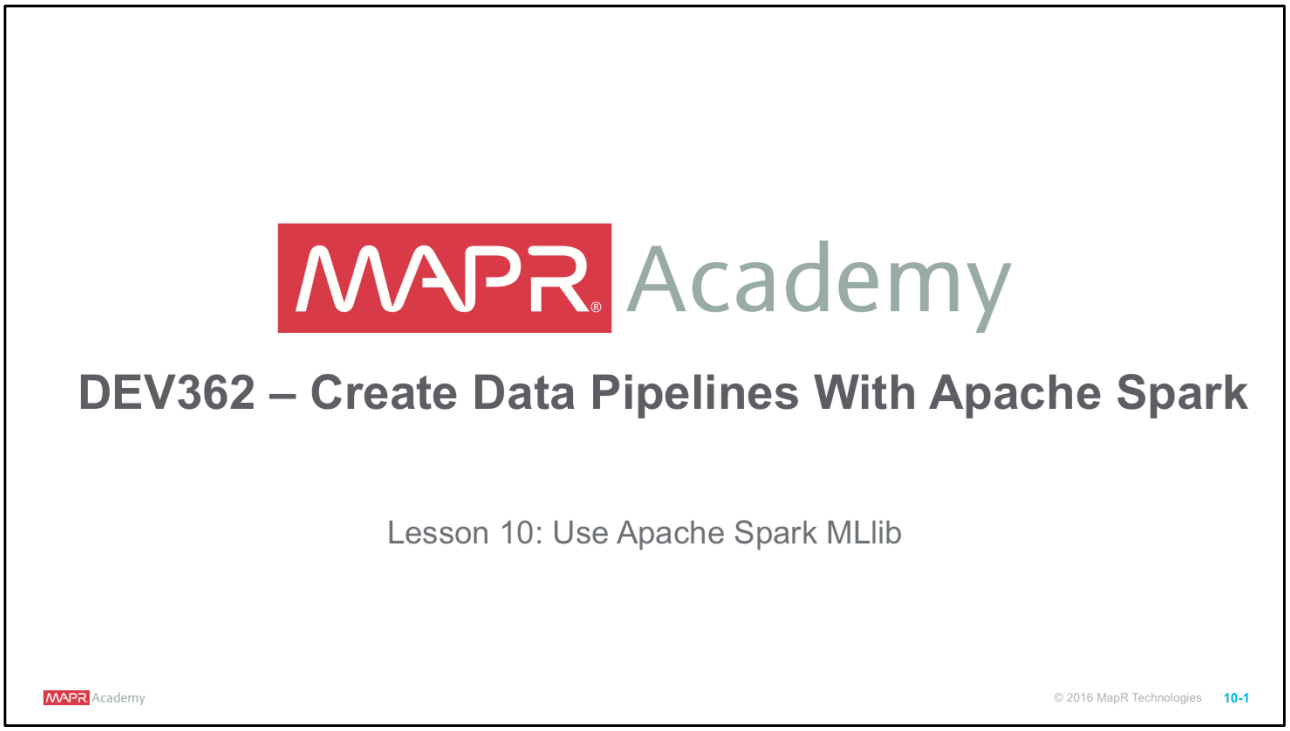
If you want to do a longer term analysis, you may save the data over a longer period of time such as a month or quarter, and then do the analysis by airline, by route, by week and so on. In this case, you may create a standalone graph processing application.

In our activity, we will look at an example of including graph processing in our streaming application and also a stand alone graph processing application.



In this activity, you will define a graph and apply graph operators to it. In the first part of the activity, you will create the property graph and apply operators to it in the Interactive shell. In the second part, you will add some of this to the streaming application created in Lesson 8.

Congratulations! You have completed Lesson 9; Use Apache Spark GraphX.
Continue on to lesson 10 to learn about the Apache Spark Machine Learning Library.



Welcome to DEV 362 Lesson 10: Use Apache Spark machine learning library

Learning Goals

Learning Goals

- Describe Apache Spark MLlib
- Describe the Machine Learning Techniques:
  - Classification
  - Clustering
  - Collaborative Filtering
- Use Collaborative Filtering to Predict User Choice

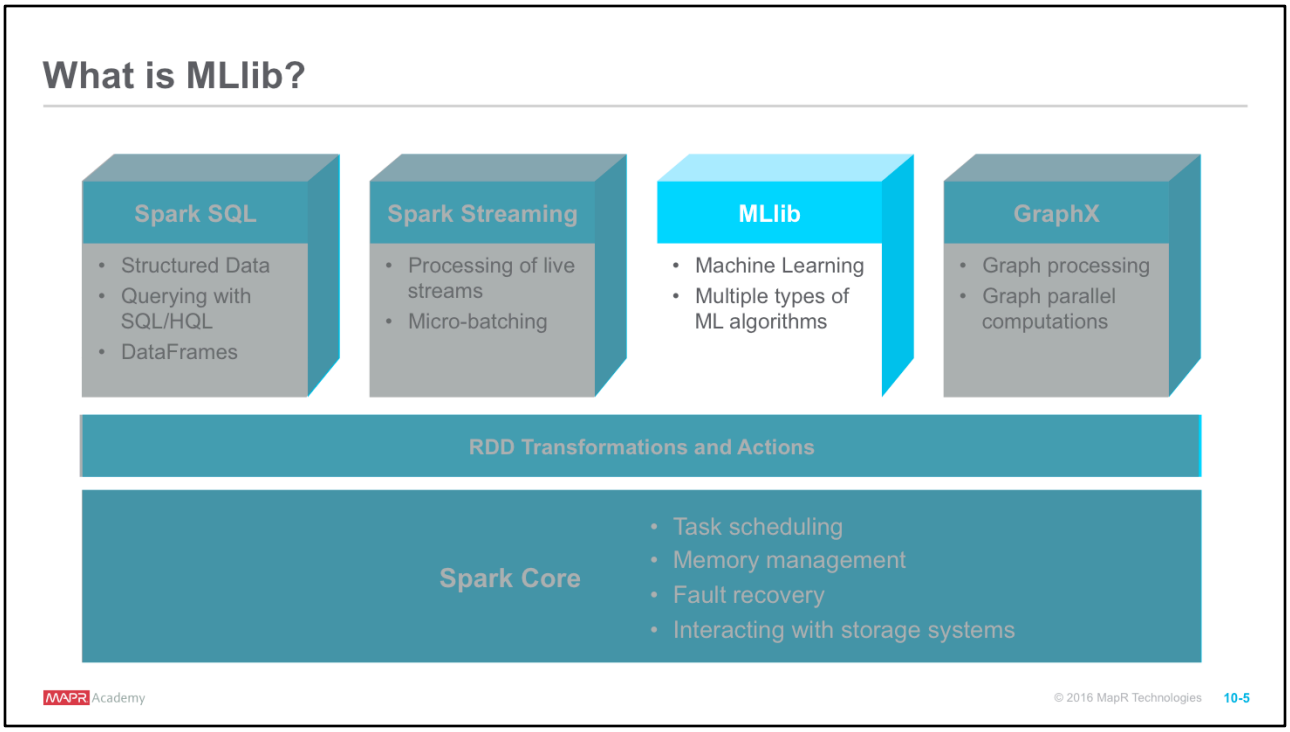
At the end of this lesson you will be able to:

- Define the Apache Spark machine learning library
- Describe three popular machine learning techniques: classification, clustering, and collaborative filtering, commonly known as recommendation
- And use collaborative filtering to predict what a user will like

Learning Goals

- **Describe Apache Spark MLlib**
- Describe the Machine Learning Techniques:
  - Classification
  - Clustering
  - Collaborative Filtering
- Use Collaborative Filtering to Predict User Choice

The first section introduces the Apache Spark machine learning library.



MLlib is the Apache Spark library of machine learning functions. It contains only those machine learning algorithms that are parallel and run on clusters. Those algorithms that were not designed to run on parallel platforms are not included.

Apache Spark machine learning library algorithms are most suitable for running on a single large data set.

MLlib Algorithms and Utilities

Algorithms and Utilities	Description
Basic statistics	Includes summary statistics, correlations, hypothesis testing, random data generation
Classification and regression	Includes methods for linear models, decision trees and Naïve Bayes
Collaborative filtering	Supports model-based collaborative filtering using alternating least squares (ALS) algorithm
Clustering	Supports K-means clustering
Dimensionality reduction	Supports dimensionality reduction on the RowMatrix class; singular value decomposition (SVD) and principal component analysis (PCA)
Feature extraction and transformation	Contains several classes for common feature transformations

MLlib provides the machine learning algorithms and utilities listed here.

MLlib Algorithms and Utilities

Algorithms and Utilities	Description
Basic statistics	Includes summary statistics, correlations, hypothesis testing, random data generation
Classification and regression	Includes methods for linear models, decision trees and Naïve Bayes
Collaborative filtering	Supports model-based collaborative filtering using alternating least squares (ALS) algorithm
Clustering	Supports K-means clustering
Dimensionality reduction	Supports dimensionality reduction on the RowMatrix class; singular value decomposition (SVD) and principal component analysis (PCA)
Feature extraction and transformation	Contains several classes for common feature transformations

MLlib includes functions for summary statistics on RDDs. There are operations for calculating correlations between two series of data using either the Pearson or Spearman methods. MLlib also provides support for hypothesis testing and random data generation.

MLlib Algorithms and Utilities

Algorithms and Utilities	Description
Basic statistics	Includes summary statistics, correlations, hypothesis testing, random data generation
Classification and regression	Includes methods for linear models, decision trees and Naïve Bayes
Collaborative filtering	Supports model-based collaborative filtering using alternating least squares (ALS) algorithm
Clustering	Supports K-means clustering
Dimensionality reduction	Supports dimensionality reduction on the RowMatrix class; singular value decomposition (SVD) and principal component analysis (PCA)
Feature extraction and transformation	Contains several classes for common feature transformations

MLlib has various methods for binary, and for multiple classification and regression problems. These include methods for linear models, decision trees and Naïve Bayes.

There is support for model-based collaborative filtering using the Alternating Least Squares or ALS algorithm.

Later in this lesson, we will use collaborative filtering to predict what movies a user will like.

MLlib Algorithms and Utilities

Algorithms and Utilities	Description
Basic statistics	Includes summary statistics, correlations, hypothesis testing, random data generation
Classification and regression	Includes methods for linear models, decision trees and Naïve Bayes
Collaborative filtering	Supports model-based collaborative filtering using alternating least squares (ALS) algorithm
Clustering	Supports K-means clustering
Dimensionality reduction	Supports dimensionality reduction on the RowMatrix class; singular value decomposition (SVD) and principal component analysis (PCA)
Feature extraction and transformation	Contains several classes for common feature transformations

MLlib also provides support for K-means clustering, often used for data mining of large data sets.

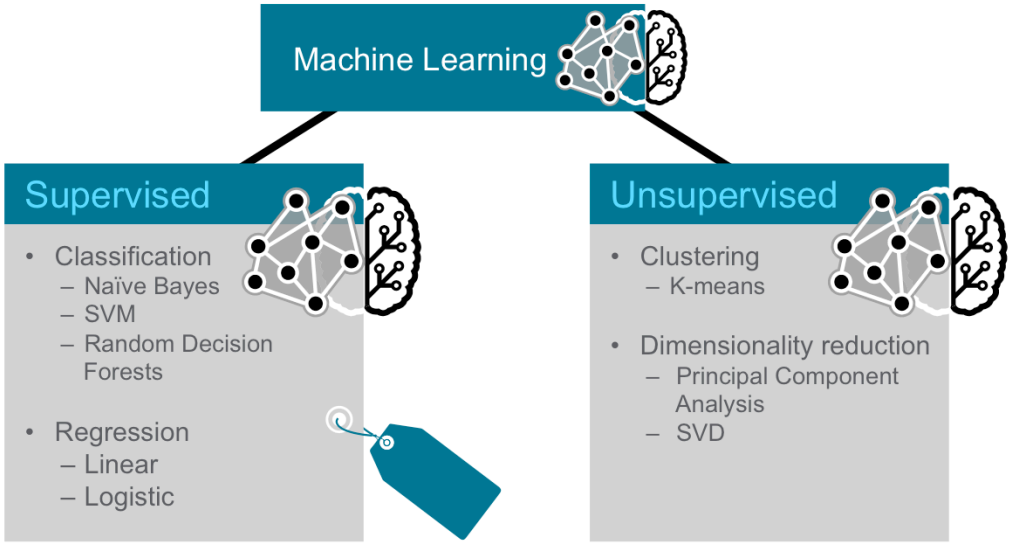
MLlib Algorithms and Utilities

Algorithms and Utilities	Description
Basic statistics	Includes summary statistics, correlations, hypothesis testing, random data generation
Classification and regression	Includes methods for linear models, decision trees and Naïve Bayes
Collaborative filtering	Supports model-based collaborative filtering using alternating least squares (ALS) algorithm
Clustering	Supports K-means clustering
Dimensionality reduction	Supports dimensionality reduction on the RowMatrix class; singular value decomposition (SVD) and principal component analysis (PCA)
Feature extraction and transformation	Contains several classes for common feature transformations

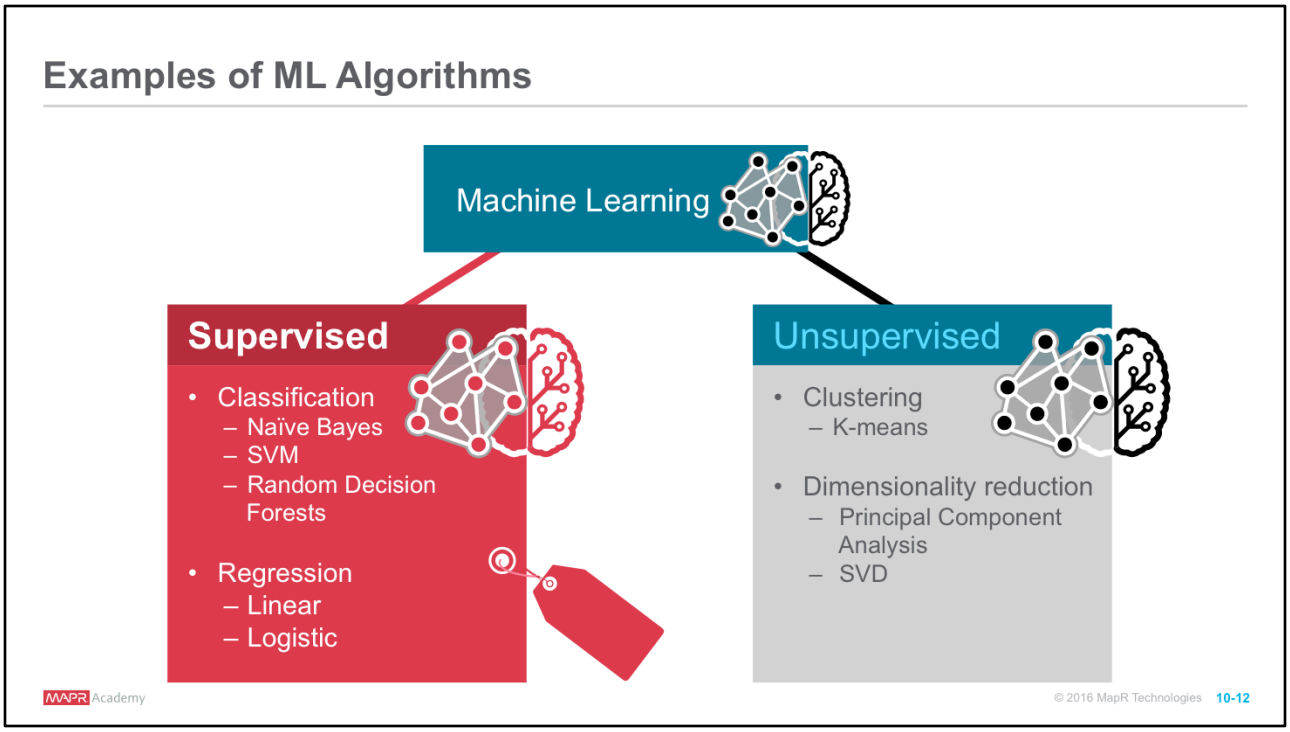
MLlib supports dimensionality reduction on the RowMatrix class. It provides functionality for Singular Value Decompositions, shown here as SVD, and Principal Component Analysis, shown as PCA.

There are also several classes for common feature transformations.

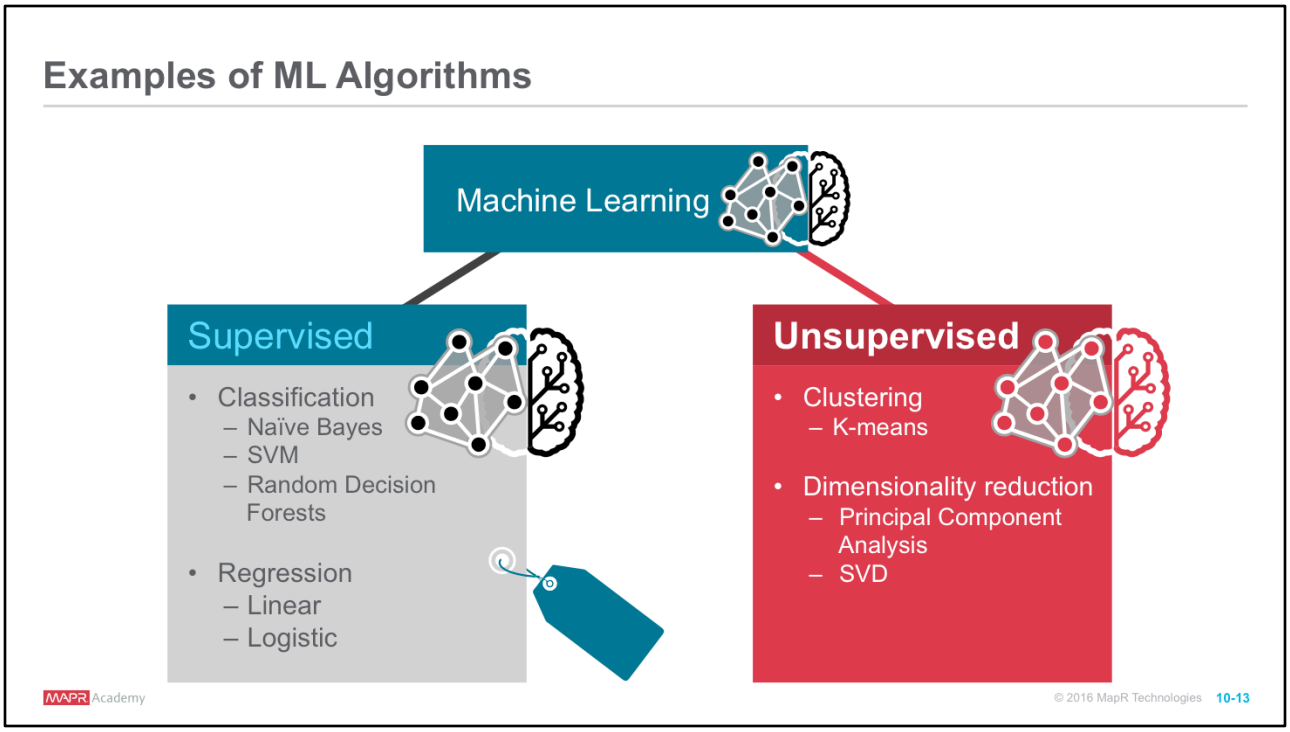
Examples of ML Algorithms



In general, machine learning may be broken down into two classes of algorithms: supervised and unsupervised.



Supervised algorithms use labeled data in which both the input and output are provided to the algorithm.



Unsupervised algorithms do not have the outputs in advance. These algorithms are left to make sense of the data without any hints.

Class Discussion

Class Discussion

What is the difference between supervised and unsupervised machine learning algorithms?

Why is classification considered supervised, while clustering is considered unsupervised?

Supervised algorithms require known information about the results, to which to compare the sample data.

Unsupervised algorithms have no known information, and must determine all of their results from the sample data.

Classification has a known set of definitions that the sample data is compared against. The sample data is determined to either be the same as, or different from the known definitions.

Clustering organizes a set of data into groups, based only on the data itself. It does not require any known prior information about the data.

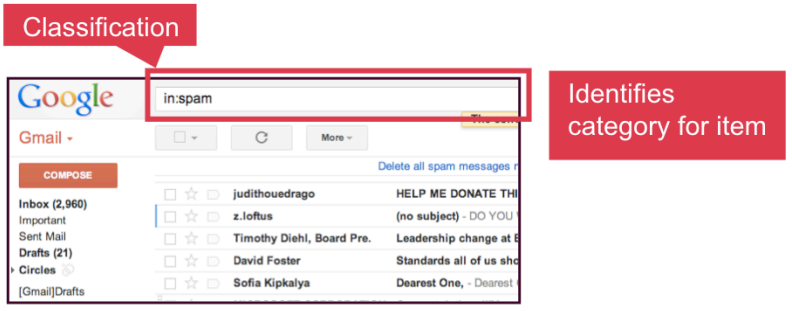
Learning Goals

Learning Goals

- Describe Apache Spark MLlib
- Describe the Machine Learning Techniques:
  - Classification
  - Clustering
  - Collaborative Filtering
- Use Collaborative Filtering to Predict User Choice

In the next section, we will describe three common categories of machine learning techniques, starting with Classification.

Machine Learning: Classification



Gmail uses a machine learning technique called classification to designate if an email is spam or not, based on the data of an email: the sender, recipients, subject, and message body.

Classification is a supervised algorithm meaning it uses labeled data in which both the input and output are provided to the algorithm.

Classification takes a set of data with known labels and learns how to label new records based on that information. The algorithm identifies which category an item belongs to, based on labeled examples of known items. For example, it identifies whether an email is spam or not based on emails known to be spam or not.

Classification: Definition

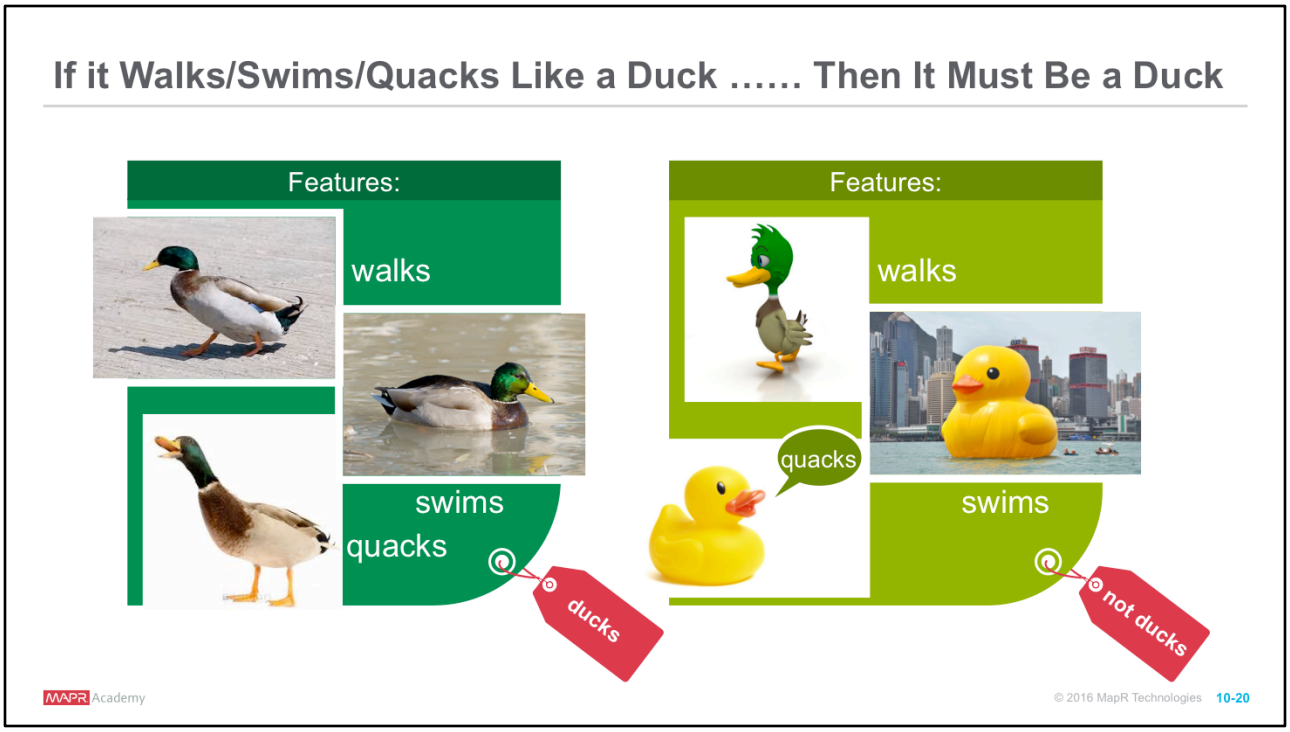
Form of ML that:

- Identifies which category an item belongs to
- Uses supervised learning algorithms
  - Data is labeled

Classification is a family of supervised machine learning algorithms that designate input as belonging to one of several pre-defined classes.

Some common use cases for classification include credit card fraud detection and email spam detection, both of which are binary classification problems.

Classification data is labeled, for example, as spam/non-spam or fraud/non-fraud. Machine learning assigns a label or class to new data.



You can classify something based on pre-determined features. Features are the “if questions” that you ask. The label is the answer to those questions**.** In this example, if it walks, swims, and quacks like a duck, then the label is "duck".

In this simplistic example, the classification is binary but classification can extend to any number of pre-defined classes.

Building and Deploying a Classifier Model

Spam:
free money now!
buy this money
free savings $$$

Non-spam:
how are you?
that Spark job
that Spark job

Training Data

Image reference O'Reilly Learning Spark

To build a classifier model, first extract the features that most contribute to the classification. In our email example, we find features that define an email as spam, or not spam.

Building and Deploying a Classifier Model

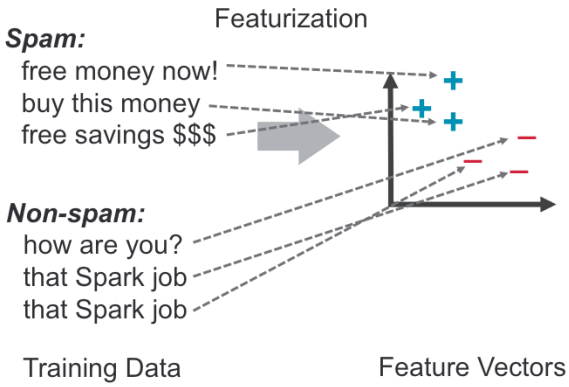


Image reference O'Reilly Learning Spark

The features are transformed and put into Feature Vectors, which is a vector of numbers representing the value for each feature. We rank our features by how strongly they define an email as spam, or not spam.

Building and Deploying a Classifier Model

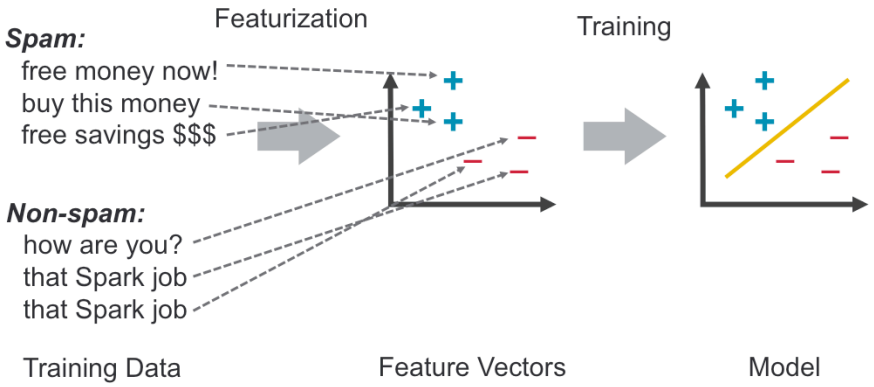
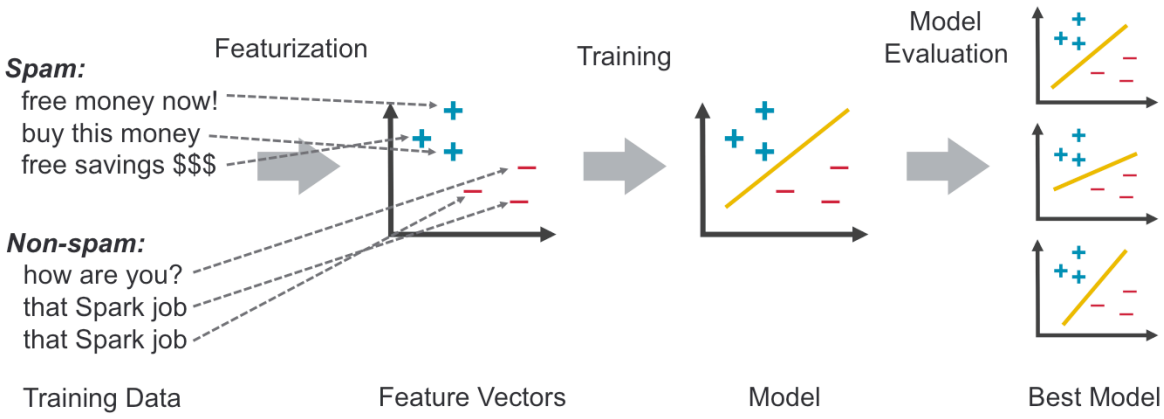


Image reference O'Reilly Learning Spark

We train our model by making associations between the input features and the labeled output associated with those features.

Building and Deploying a Classifier Model



MAPR Academy

Image reference O'Reilly Learning Spark

Then at runtime, we deploy our model by extracting the same set of features, and ask the model to classify that set of features as one of the pre-defined set of labeled classes.

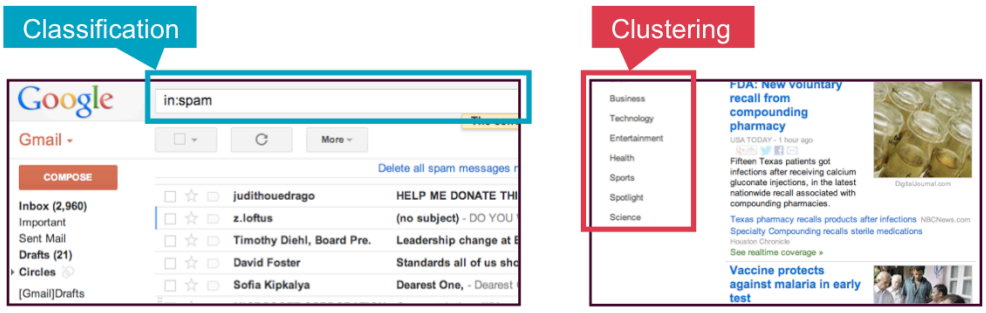
Learning Goals

Learning Goals

- Describe Apache Spark MLlib
- Describe the Machine Learning Techniques:
  - Classification
  - Clustering
  - Collaborative Filtering
- Use Collaborative Filtering to Predict User Choice

Next, we will discuss Clustering as a machine learning model.

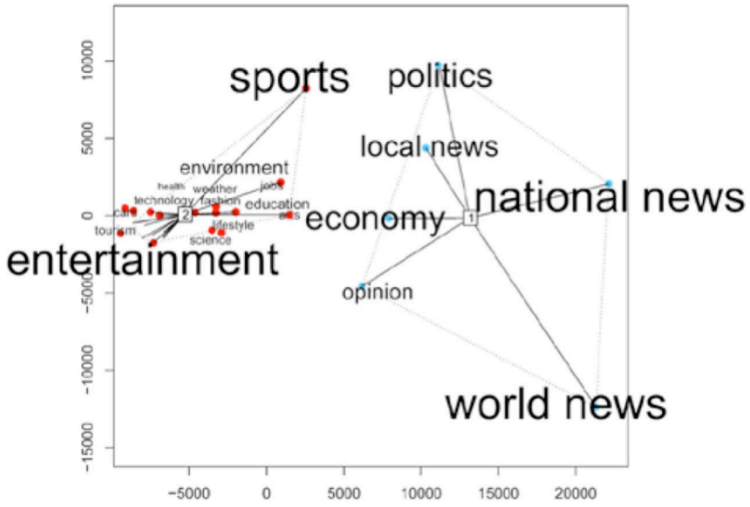
Machine Learning: Clustering



Google News uses a technique called clustering to group news articles into different categories, based on title and content.

Clustering algorithms discover groupings that occur in collections of data.

Clustering: Definition

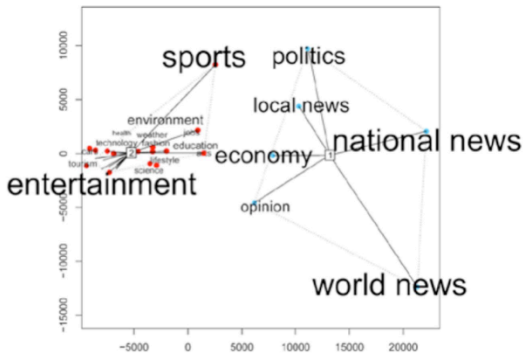


Marco Toledo Bastos, and Gabriela Zago SAGE

In clustering, an algorithm classifies inputs into categories by analyzing similarities between input examples.

Clustering: Definition

- Unsupervised learning task
- Groups objects into **clusters of high similarity**



Clustering uses unsupervised algorithms, which do not have the outputs in advance. No known classes are used as a reference, as with a supervised algorithm like classification.

Clustering: Definition

- Unsupervised learning task
- Groups objects into **clusters of high similarity**
  - **Search results** grouping
  - **Grouping of customers**
  - **Anomaly** detection
  - Text categorization

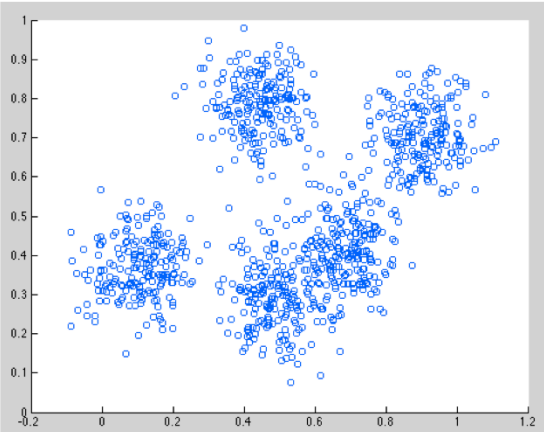
Clustering can be used for many purposes, for example, search results grouping.

Other examples include:

- partitioning data into groups, such as grouping similar customers
- anomaly detection, such as fraud detection
- and text categorization, such as sorting books into genres

Clustering: Example

- Group similar objects

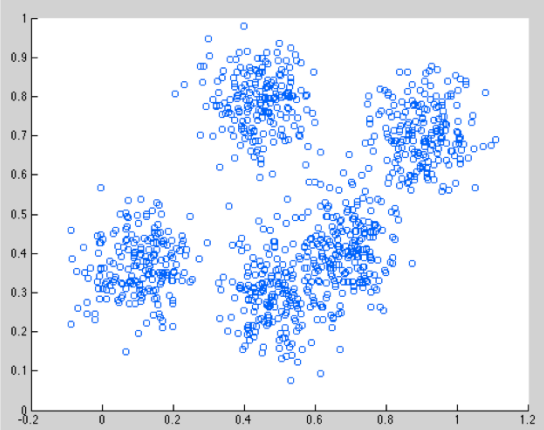


In this example, we are given a set of raw data points. The objective is to create some number of clusters that group these data points with those that are most similar, or closest.

Clustering: Example

- Group similar objects
- Use MLlib K-means algorithm

1. Initialize coordinates to center of clusters (centroid)

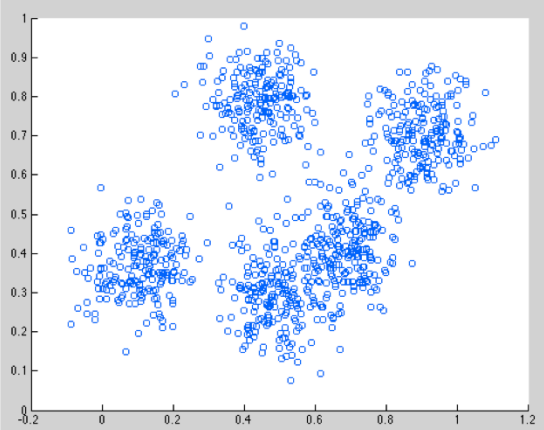


Clustering using the K-means algorithm begins by initializing all the coordinates to centroids. There are various ways you can initialize the points, including randomly.

Clustering: Example

- Group similar objects
- Use MLlib K-means algorithm

1. Initialize coordinates to center of clusters (centroid)
2. Assign all points to nearest centroid

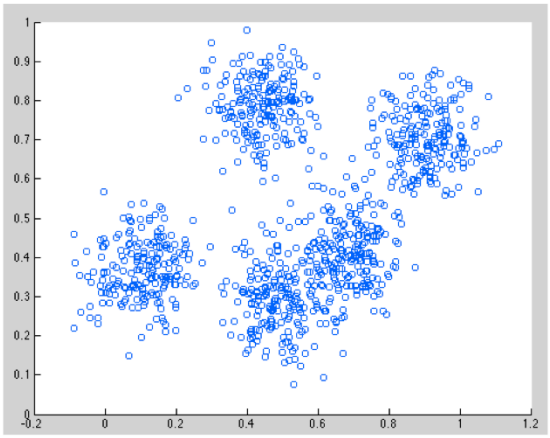


With every pass of the algorithm, each point is assigned to its nearest centroid based on some distance metric, usually Euclidean distance.

Clustering: Example

- Group similar objects
- Use MLlib K-means algorithm

1. Initialize coordinates to center of clusters (centroid)
2. Assign all points to nearest centroid
3. Update centroids to center of points

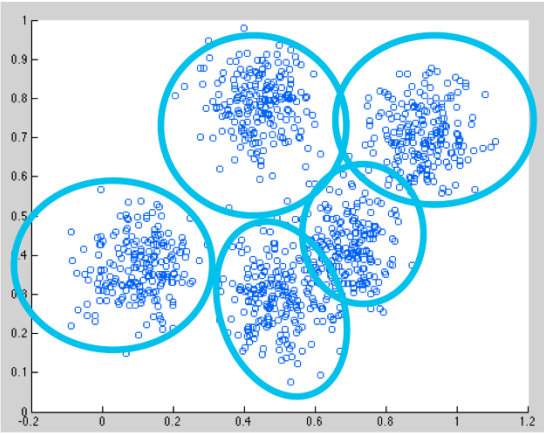


The centroids are then updated to be the “centers” of all the points assigned to it in that pass.

Clustering: Example

- Group similar objects
- Use MLlib K-means algorithm

1. Initialize coordinates to center of clusters (centroid)
2. Assign all points to nearest centroid
3. Update centroids to center of points
4. Repeat until conditions met



The algorithm stops according to various programmable conditions, such as:

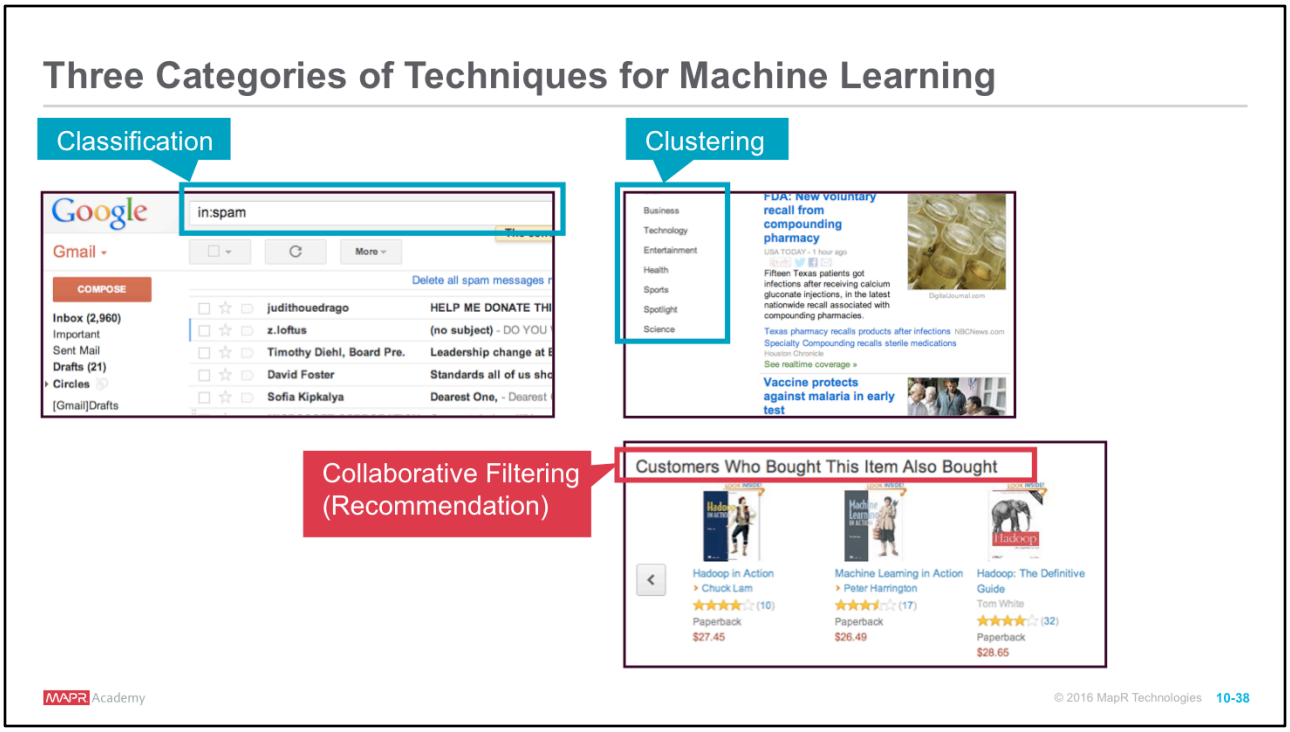
- A minimum change of median from the last iteration
- A sufficiently small intra-cluster distance
- A sufficiently large inter-cluster distance

Learning Goals

Learning Goals

- Describe Apache Spark MLlib
- Describe the Machine Learning Techniques:
  - Classification
  - Clustering
  - Collaborative Filtering
- Use Collaborative Filtering to Predict User Choice

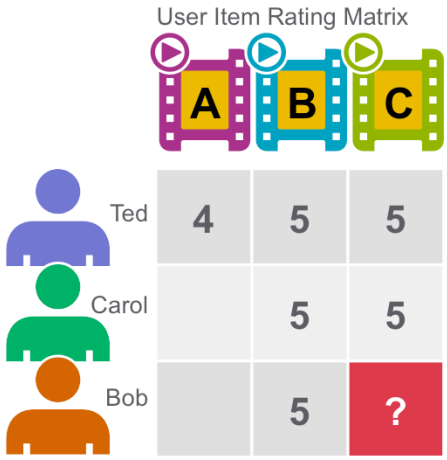
The final machine learning technique we will look at is collaborative filtering, more commonly known as recommendation. We will describe collaborative filtering briefly here, and then show an example of a movie recommendation engine.



Amazon uses a machine learning technique called collaborative filtering commonly referred to as recommendation, to determine products users will like based on their history and similarity to other users.

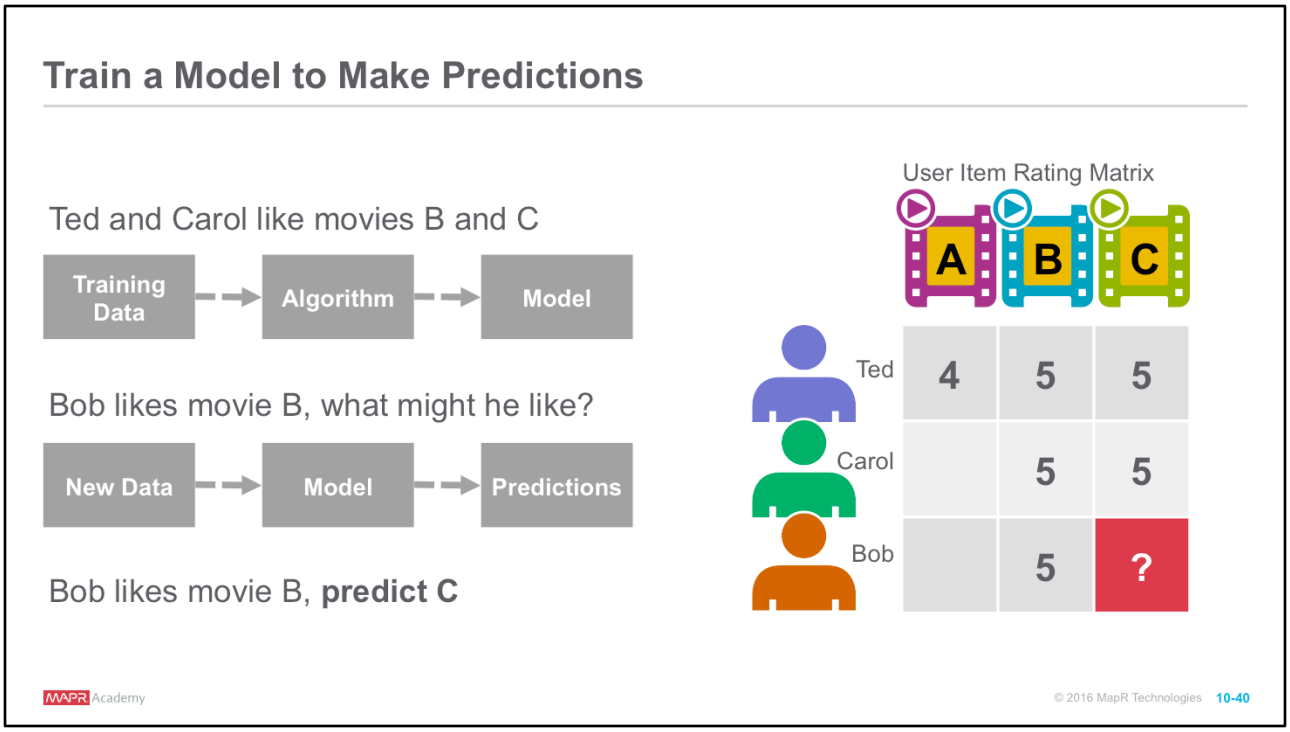
Collaborative Filtering with Spark

- Recommend items
  - (Filtering)
- Based on user preferences data
  - (Collaborative)



Collaborative filtering algorithms recommend items based on preference information from many users. The collaborative filtering approach is based on similarity; people who liked similar items in the past will like similar items in the future.

In the example shown, Ted likes movies A, B, and C. Carol likes movies B and C. Bob likes movie B. To recommend a movie to Bob, we calculate that users who liked B also liked C, so C is a possible recommendation for Bob.



The goal of a collaborative filtering algorithm is to take preferences data from users, and to create a model that can be used for recommendations or predictions.

Ted likes movies A, B, and C. Carol likes movies B and C. We take this data and run it through an algorithm to build a model.

Then when we have new data such as Bob likes movie B, we use the model to predict that C is a possible recommendation for Bob.

Knowledge Check

Knowledge Check

Match the scenarios listed here with the machine learning technique that would provide the best results:

Classification, Clustering or Collaborative Filtering

You want to determine what restaurant someone may like, based on restaurants they have previous liked.

You want to detect fraudulent attempts to log into your website.

You need to list which students have passed, and which have failed an exam.

You want to organize your music collection based on genre metadata.

Restaurant recommendations::Collaborative Filtering. Suggesting restaurants that people liked, who also like the restaurants that our user liked.
Login fraud detection::Clustering. Looking for anomalies in failed login attempts.
List passed and failed students::Classification
Organizing music::Clustering.

Class Discussion

Class Discussion

Given the three machine learning techniques we've discussed:

- Classification
- Clustering
- Collaborative Filtering

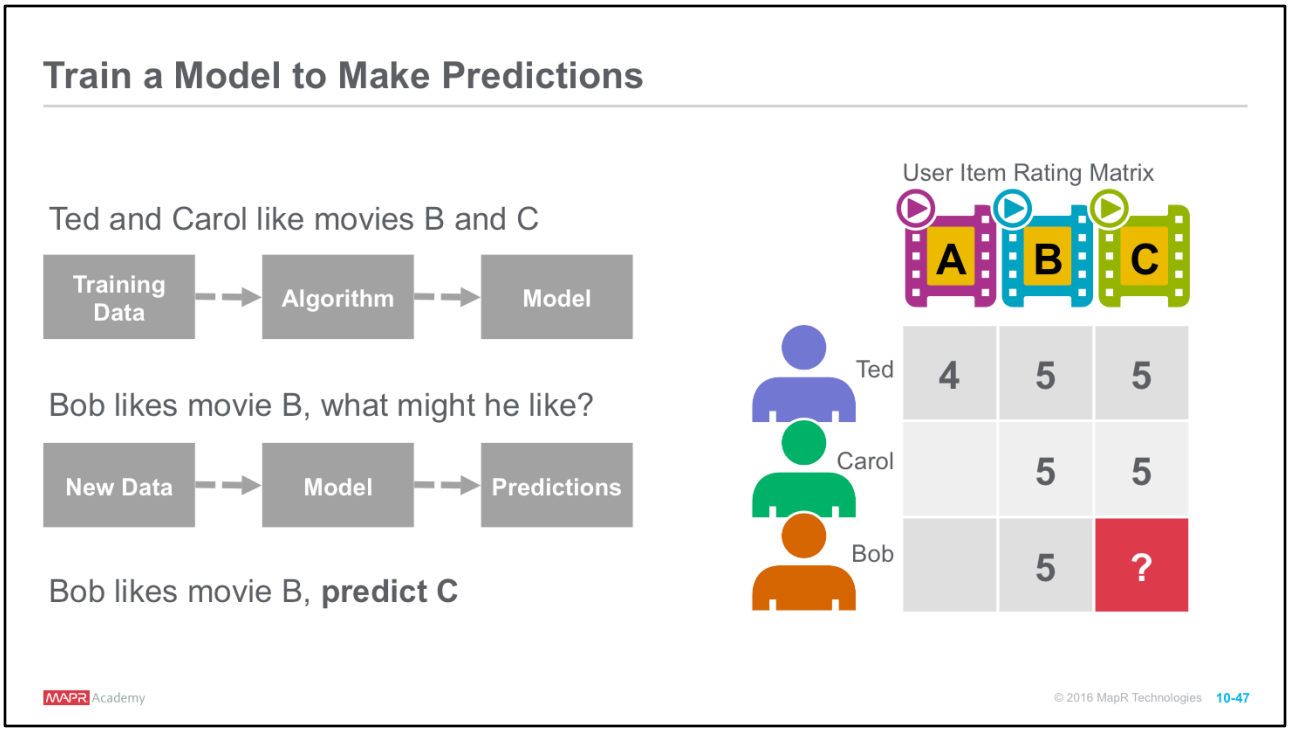
What are some scenarios at your company that use one or more of these techniques? Fraud detection? Organization of data? Spam filters?

Learning Goals

Learning Goals

- Describe Apache Spark MLlib
- Describe the Machine Learning Techniques:
  - Classification
  - Clustering
  - Collaborative Filtering
- **Use Collaborative Filtering to Predict User Choice**

In this next section, we will use collaborative filtering to predict what a user will like.



Let's look more deeply at our movie recommendation example.

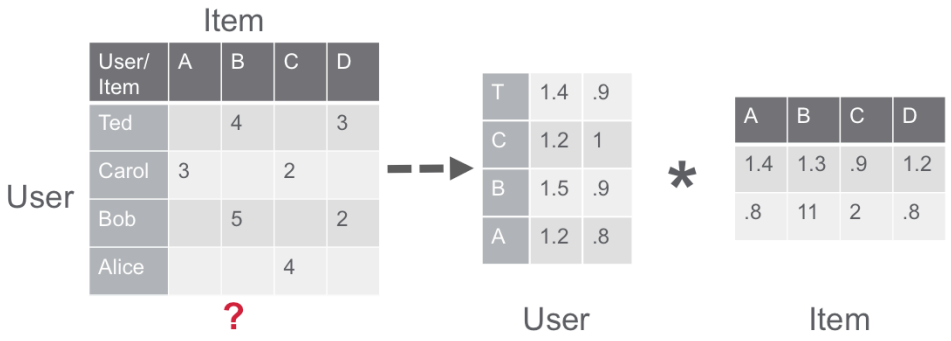
To quickly review, the goal of a collaborative filtering algorithm is to take preferences data from users, and to create a model that can be used for recommendations or predictions.

Ted likes movies A, B, and C. Carol likes movies B and C. We use this data to build a model, and predict what Bob will like.

Alternating Least Squares

Approximates sparse user item rating matrix

- as product of two dense matrices, User and Item factor matrices



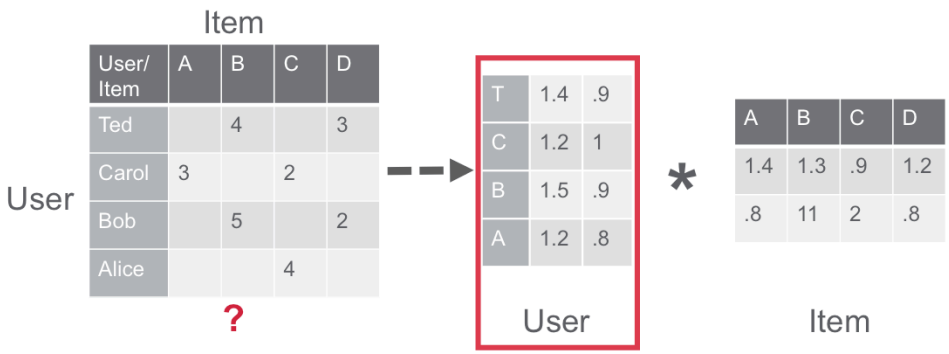
Alternating Least Squares, or ALS, approximates a sparse user item rating matrix of dimension K, as the product of two dense matrices.

We start with User and Item factor matrices of size U×K and I×K. The factor matrices are also called latent feature models, and represent hidden features which the algorithm tries to discover.

Alternating Least Squares

Approximates sparse user item rating matrix

- as product of two dense matrices, User and Item factor matrices
- tries to learn the hidden features of each user and item

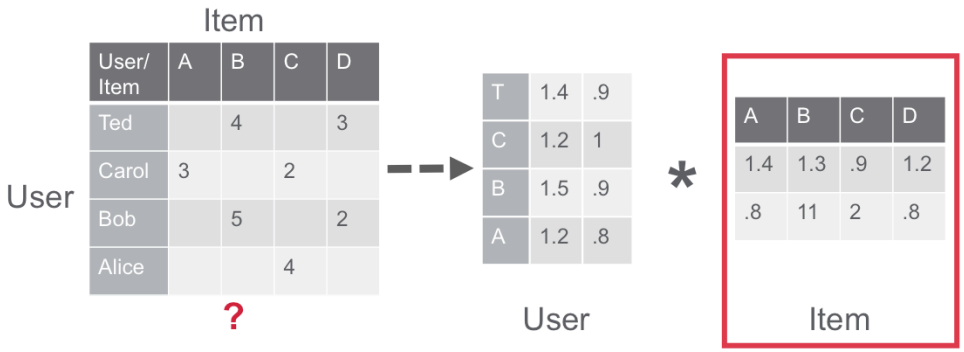


One matrix tries to describe the latent or hidden features of each user

Alternating Least Squares

Approximates sparse user item rating matrix

- as product of two dense matrices, User and Item factor matrices
- tries to learn the hidden features of each user and item

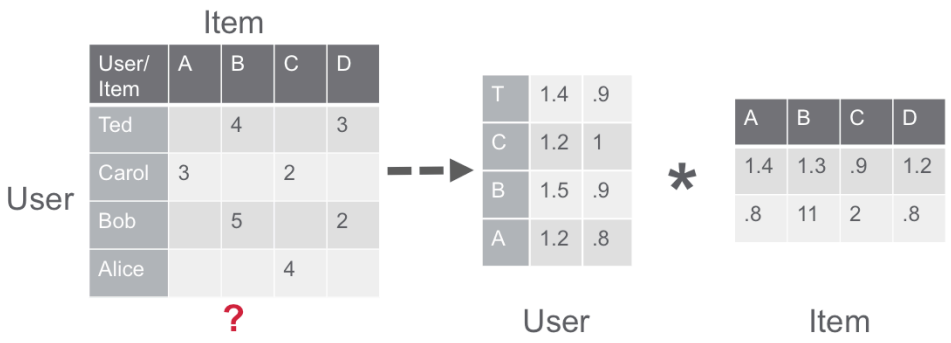


and one tries to describe latent properties of each movie.

Alternating Least Squares

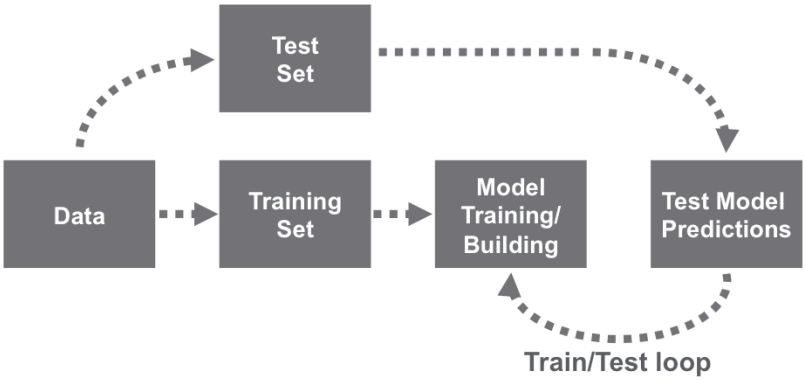
Approximates sparse user item rating matrix

- as product of two dense matrices, User and Item factor matrices
- tries to learn the hidden features of each user and item
- algorithm alternatively fixes one factor matrix and solves for the other

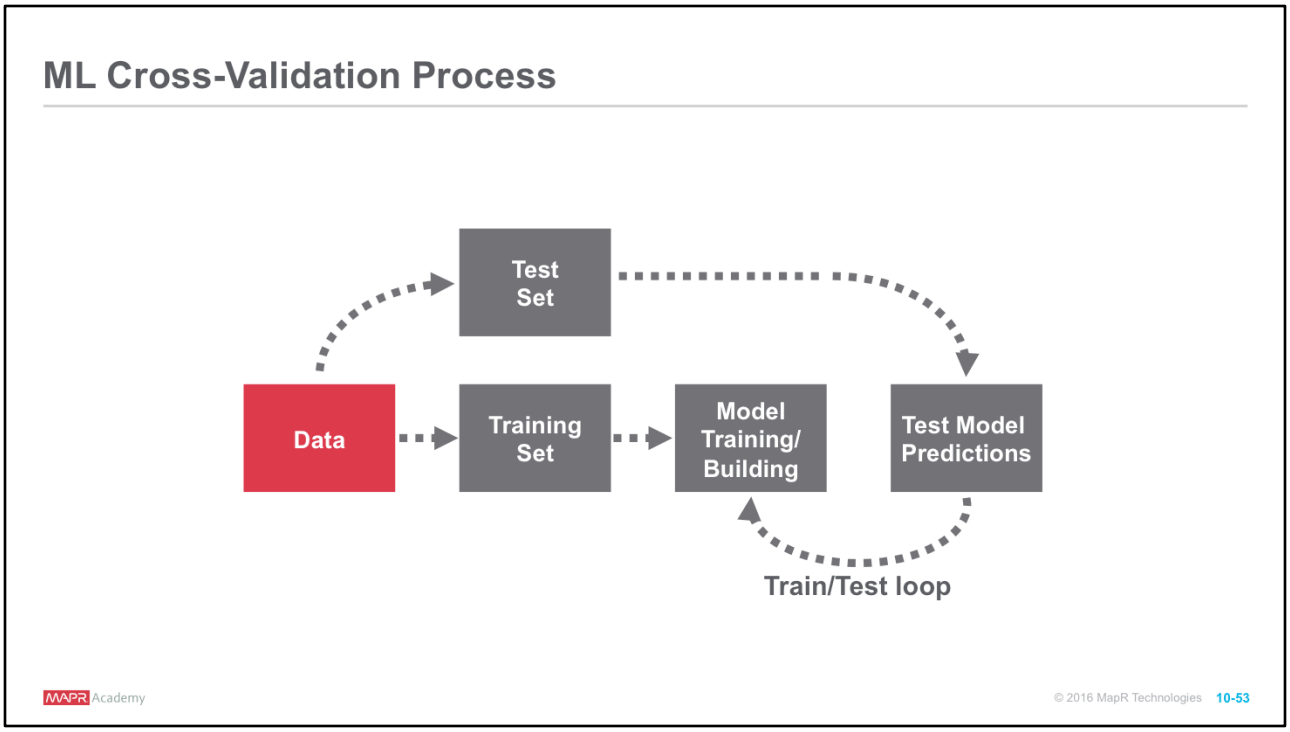


ALS is an iterative algorithm. During each iteration, the algorithm alternatively fixes one factor matrix and solves for the other. This process continues until it converges. The practice of alternating between which matrix to optimize is why the process is called Alternating Least Squares.

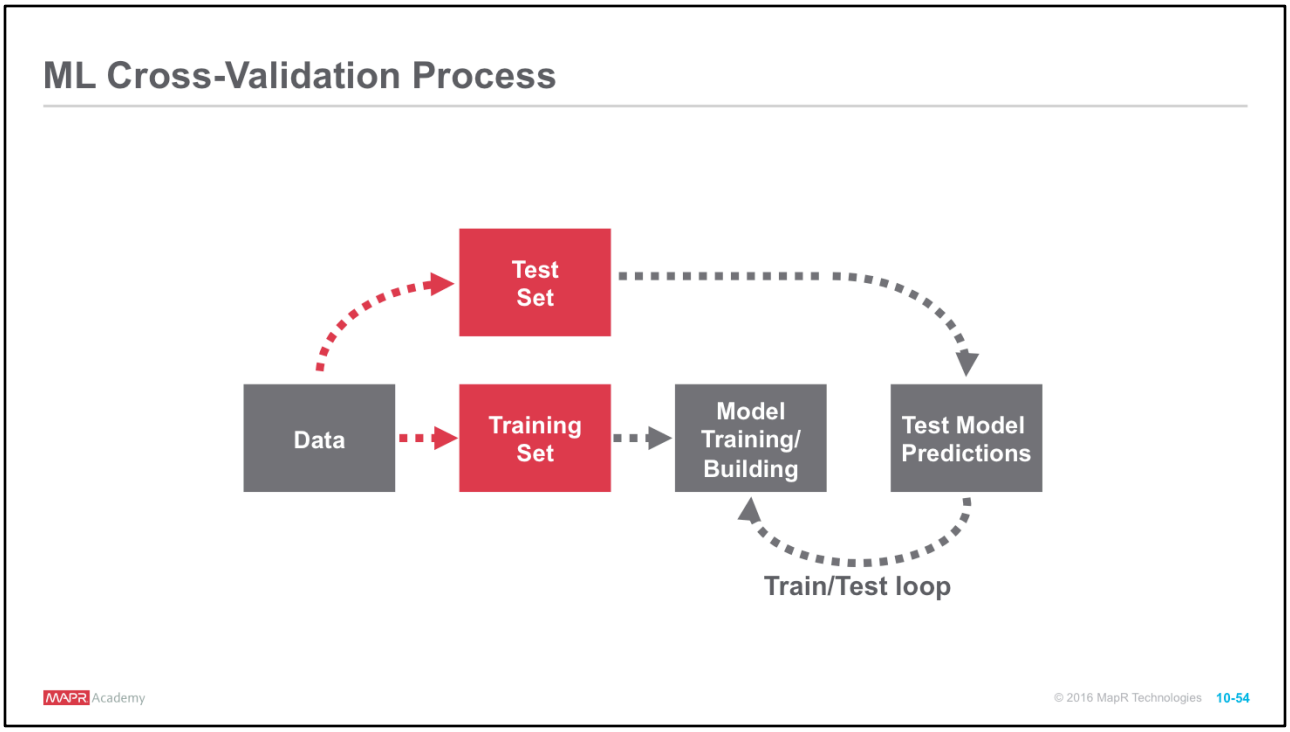
ML Cross-Validation Process



A typical machine learning workflow is shown here. To make our predictions, we will perform the following steps:

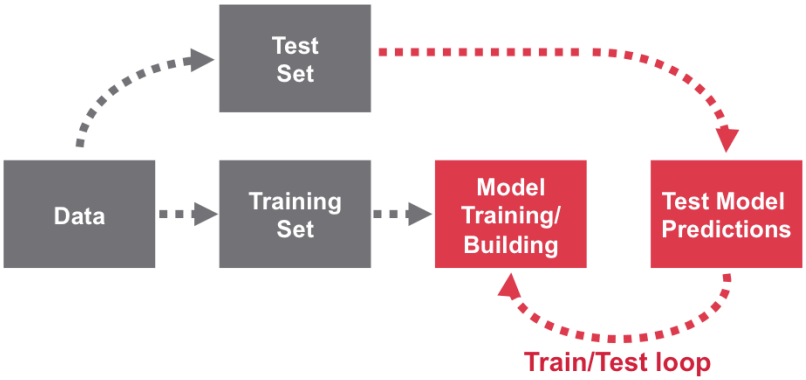


Load the sample data, and parse the data into the input format for the ALS algorithm.



Split the data into two parts, one for building the model and one for testing the model.

ML Cross-Validation Process



We then run the ALS algorithm to build and train a user product matrix model.

We make predictions with the training data, and observe the results.

Then, test the model with the test data.

Parse Input

```
// load ratings data into a RDD
val ratingText = sc.textFile("/data/ratings.dat")

// create an RDD of Ratings objects
val ratingsRDD = ratingText.map(parseRating).cache()
```


In the first step, we load the ratings data into the ratingText RDD.

Then we use the map transformation on the ratingText RDD. This will apply the parseRating function to each element in ratingText, and return a new RDD of Rating objects. We cache ratingsRDD, since we will use this data to build the matrix model.

Parse Input Lines into Ratings Objects

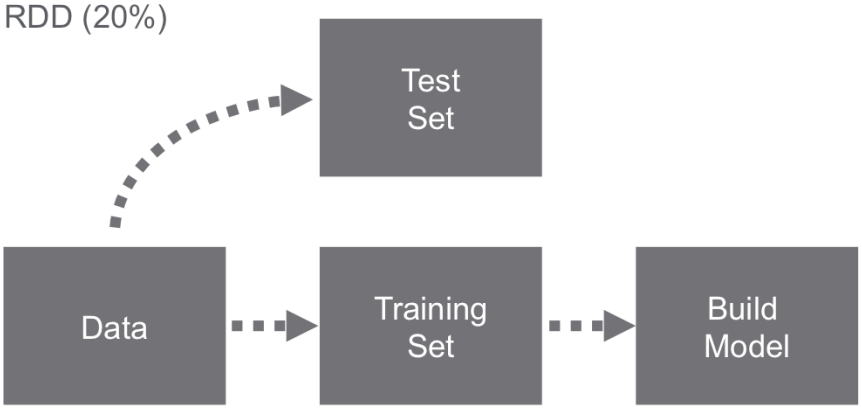
```
// Import mllib recommendation data types
import org.apache.spark.mllib.recommendation.{ALS,
MatrixFactorizationModel, Rating}


// parse string: UserID::MovieID::Rating
def parseRating(str: String): Rating= {
    val fields = str.split("::")
    Rating(fields(0).toInt, fields(1).toInt,
     fields(2).toDouble)
}
```


The parseRating function parses a line from the ratings data file into the MLlib Rating class. We will use this as input for the ALS run method.

Build Model

Split ratings RDD into:

- Training data RDD (80%)
- Test data RDD (20%)

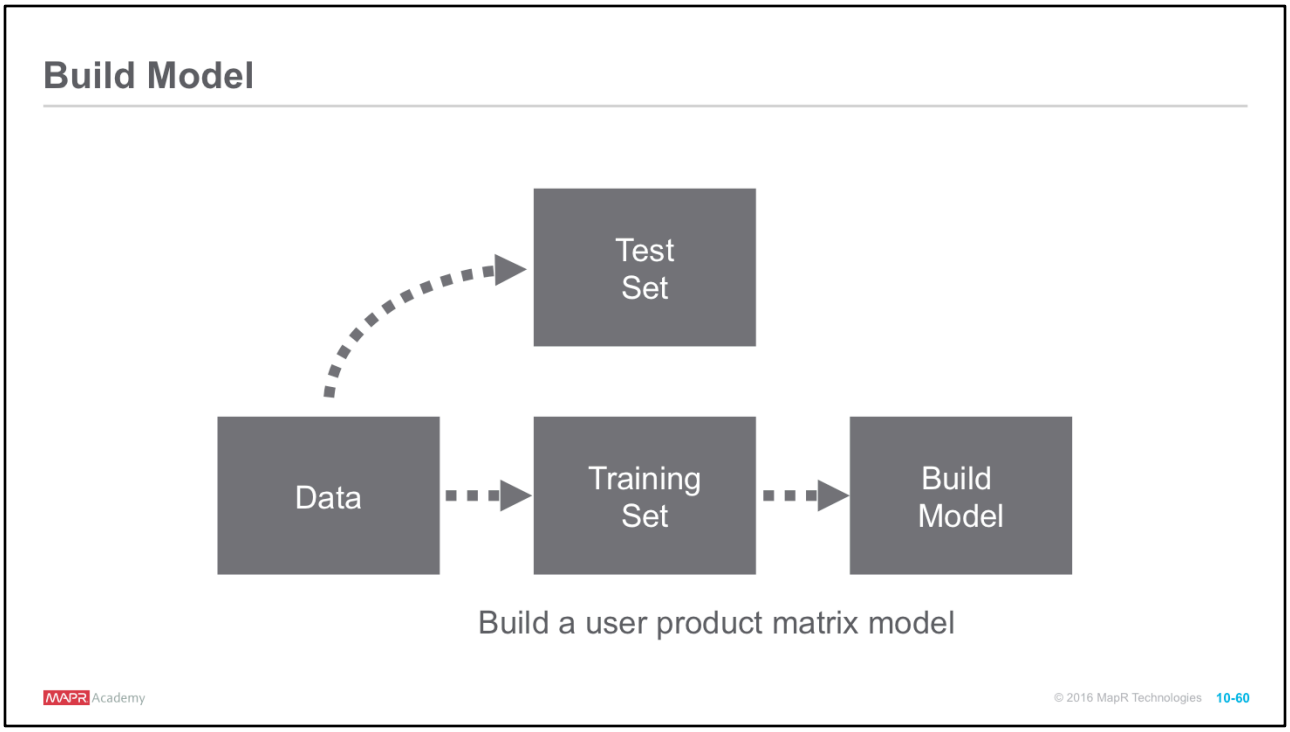


Next we split the data into two parts, one for building the model and one for testing the model.

Build Model

```
// Randomly split ratings RDD into training data RDD (80%)
and test data RDD (20%)
val splits = ratingsRDD.randomSplit(Array(0.8, 0.2), 0L)

val trainingRatingsRDD = splits(0).cache()
val testRatingsRDD = splits(1).cache()

 // build a ALS user product matrix model with rank=20,
iterations=10
val model = (new ALS().setRank(20).setIterations(10)
   .run(trainingRatingsRDD))
```


The top line in the code shown here applies the 80-20 split to our data.



Then run the ALS algorithm to build and train a user product matrix model.

Build Model

```
// Randomly split ratings RDD into training data RDD (80%)
and test data RDD (20%)
val splits = ratingsRDD.randomSplit(Array(0.8, 0.2), 0L)

val trainingRatingsRDD = splits(0).cache()
val testRatingsRDD = splits(1).cache()

 // build a ALS user product matrix model with rank=20,
iterations=10
val model = (new ALS().setRank(20).setIterations(10)
   .run(trainingRatingsRDD))
```


The last line builds the new product matrix model.

Get Predictions

Now that we have trained our model, we want to get predicted movie ratings for the test data.

Get Predictions

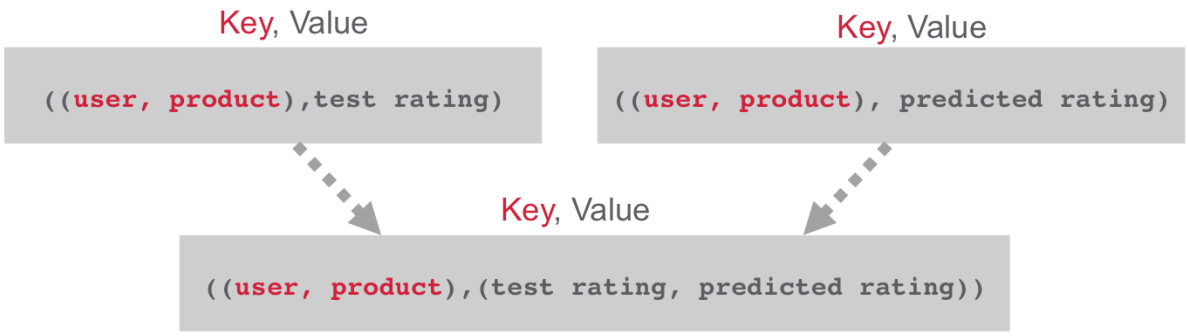
```
// get predicted ratings to compare to test ratings
val testUserProductRDD = testRatingsRDD.map {
  case Rating(user, product, rating) => (user, product)
}
// call  model.predict with test Userid, MovieId input data
val predictionsForTestRDD = model.predict(testUserProductRDD)
```


First, we use a map on the testRatingsRDD to create a new RDD of test User IDs and Movie IDs, without any ratings.

We then call the model.predict method, with the new testUserProductRDD, to get predicted ratings for each test User ID and Movie ID pair.

Compare Predictions to Tests

Join predicted ratings to test ratings in order to compare



Next we compare test User ID, Movie ID rating pairs to the predicted User ID, Movie ID rating pairs.

Test Model

```
// prepare  predictions for comparison
val predictionsKeyedByUserProductRDD = predictionsForTestRDD.map{
  case Rating(user, product, rating) => ((user, product), rating)
}
// prepare  test for comparison
val testKeyedByUserProductRDD = testRatingsRDD.map{
  case Rating(user, product, rating) => ((user, product), rating)
}
//Join the  test with  predictions
val testAndPredictionsJoinedRDD = testKeyedByUserProductRDD
  .join(predictionsKeyedByUserProductRDD)
```


Here we create User ID, Movie ID ratings key-value pairs, so that we can compare the test ratings to the predicted ratings.

Compare Predictions to Tests

Find false positives where:

```
test rating <= 1  and  predicted rating >= 4
```

Key, Value

```
((user, product),(test rating, predicted rating))
```

MAPR Academy

We compare the ratings predictions to the actual rating to find false positives where the predicted rating was high, and the actual rating was low.

Test Model

```
val falsePositives =(testAndPredictionsJoinedRDD.filter{
  case ((user, product), (ratingT, ratingP)) =>
       (ratingT <= 1 && ratingP >=4)
  })
falsePositives.take(2)

Array[((Int, Int), (Double, Double))] =
((3842,2858),(1.0,4.106488210964762)),
((6031,3194),(1.0,4.790778049100913))
```


The code shown here compares actual ratings to predicted ratings by filtering on ratings where the test rating<=1, and the predicted rating is >=4.

Test Model Mean Absolute Error

```
//Evaluate the model using Mean Absolute Error (MAE) between
test and predictions
val meanAbsoluteError = testAndPredictionsJoinedRDD.map {
  case ((user, product), (testRating, predRating)) =>
    val err = (testRating - predRating)
    Math.abs(err)
}.mean()
meanAbsoluteError: Double = 0.7244940545944053
```


The model can also be evaluated by calculating the mean absolute error, or MAE. The MAE is the mean of the absolute error between the actual test rating and the predicted rating.

Get Predictions for New User

```
val newRatingsRDD=sc.parallelize(Array(Rating(0,260,4),Rating(0,1,3))
// union
val unionRatingsRDD = ratingsRDD.union(newRatingsRDD)
// build a ALS user product matrix model
val model = (new ALS().setRank(20).setIterations(10)
   .run(unionRatingsRDD))

//  get 5 recs for userid 0
val topRecsForUser = model.recommendProducts(0, 5)
```

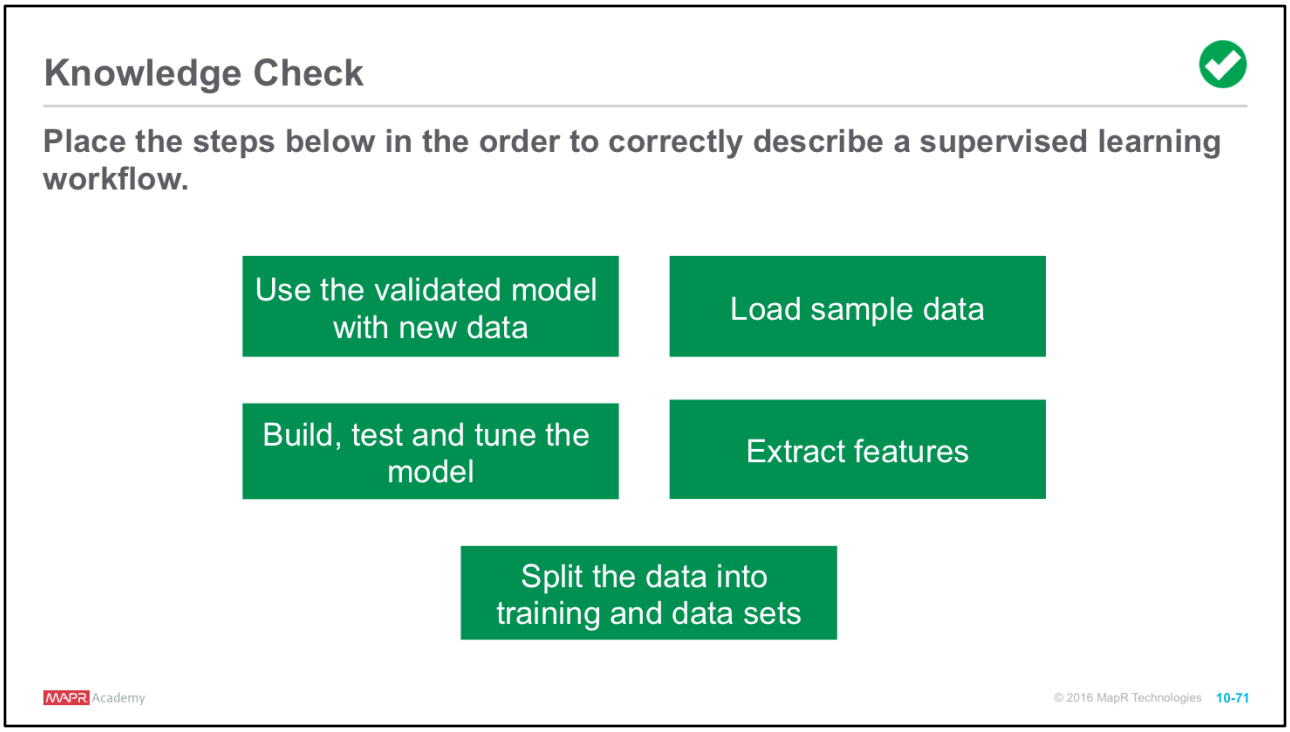

In this example, we get ratings for a new user, with the ID of 0.

First, we create an RDD with the User ID, Movie ID, and ratings for some movies.

Then, we add this RDD to the existing ratings data RDD with a union. We call an ALS run using the new unionRatingsRDD, which will return a new recommendation model.

We can now use this model to get recommendations by calling model.recommendProducts with the userid 0, and the number of recommendations desired.

Knowledge Check

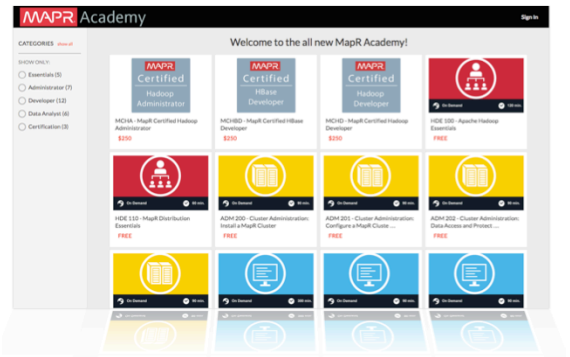


1. Load sample data
2. Extract features
3. Split the data into training and test sets.
4. Build, Test, and Tune the model
5. Use the validated model with new data

Lab 10 – Use Apache Spark MLlib to Make Recommendations

Load and inspect data using the Spark shell
Use Spark to make movie recommendations

Congratulations!



You have completed this course.

Register for more at: **learn.mapr.com**

Congratulations! You have completed DEV 362 Create Data Pipelines with Apache Spark. Visit the MapR Academy to find more courses about big data processing and analysis, and the Hadoop ecosystem.