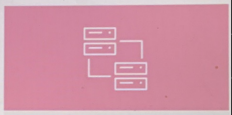
Spark for Data Engineers

What you will learn

Define Spark and list its key features



Identify the uses of Spark by data engineers

Introduction to Spark

- Is a distributed computing system
- Overcomes drawbacks of the Hadoop MapReduce framework
- Provides a unified computing engine
- Supports various data processing tasks:
  - Batch processing
  - Stream processing
  - Machine learning
  - Graph processing
- Uses distributed computing model for distributed data and processing tasks across a cluster machines

Key features of Spark

Has in-memory processing capability

- Enables caching of data in the memory
- Eliminates the high I/O costs

Support for several programming languages

- Supports Java, Scala, Python, R and others
- Offers an easy-to-use API
- Allows construction data-processing applications efficiently
- Provides built-in support for various data sources

Key features of Spark

Build scalable and efficient data processing systems

- Helps build data pipelines
- Processes and analyzes large datasets
- Integrates with other big data technologies

Data intake vs. transformation

Key tasks performed by data engineers:

Data intake

- Intake large volumes of data
- Design and implement efficient data ingestion pipelines

Data transformation

- Transform data to analyze it
- Use Spark's robust APIs and libraries
- Clean, filter, transform, and aggregate data
- Perform various data transformation tasks

Data storage vs. processing

Key tasks performed by data engineers:

Data storage

- Design and implement data storage
- Use Spark's distributed storage systems
- Build efficient data storage layers
- Use data storage systems like:
  - HDFS
  - Amazon S3
  - Apache Cassandra

Data processing

- Perform data processing tasks like:
  - Graph processing
  - Stream processing
  - Machine learning
- Design and build data processing pipelines that handle:
  - Real-time data streams
  - Large-scale batch processing

Performance tuning vs. monitoring

Key tasks performed by data engineers:

Performance tuning

- Tune various parameters like:
  - Partitioning
  - Memory usage
  - Caching
- Optimize the code and data structures
- Boost performance and reduce resource consumption

Performance monitoring

- Monitor the Spark cluster
- Troubleshoot issues that arise
- Developers use:
  - Spark's built-in monitoring tools
  - Third-party tools
- Track performance, resource usage, and overall health

Regression using SparkML

What is regression using Spark ML?

- Builds and trains regression models with Spark
- Utilizes Apache Spark libraries: Spark MLlib or Spark ML
  - Scalable and distributed ML library
  - Integrates with the Spark ecosystem

Spark ML regression algorithms

- Linear regression
- Decision tree regression
- Random forest regression
- Gradient-boosted tree regression

Assists in building models that:

Predict a continuous target variable

Based on a set of input features

Regression using Spark ML: Steps

1. Import the necessary libraries
2. Create a SparkSession
3. Read the data from a CSV file
4. Select the relevant columns
5. Create a vector assembler
6. Transform the selected data using the vector assembler
7. Split the transformed data into training and test sets

Regression using Spark ML: Steps

8. Create an instance of the linear regression model
9. Fit the model to the training data
10. Make predictions on the test data
11. Evaluate the model
12. Print the Root Mean Squared Error (RMSE)
13. Print the coefficients and intercept of the model
14. Stop the SparkSession

Regression using Spark ML: Steps

Step 1 Import necessary libraries

- Import libraries required for regression analysis

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
```

Regression using Spark ML: Steps

Step 2 Create a SparkSession

- Serves as the entry point for Spark functionality

```
spark = SparkSession.builder.appName("LinearRegressionExample").getOrCreate()
```

Step 3 Read the data from a CSV file

- Provide file path, header and inferSchema parameters

```
data = spark.read.csv("path_to_your_data.csv", header=True, inferSchema=True)
```

Regression using Spark ML: Steps

Step 4 Select the relevant columns

- Used as features and target variables

```
selected_data = data.select("feature_col1", "feature_col2", "target_col")
```

Step 5 Create a vector assembler

- Combines selected feature columns into a single vector column

```
assembler = VectorAssembler(inputCols=["feature_col1", "feature_col2"], outputCol="features")
```

Regression using Spark ML: Steps

Step 6 Transform the selected data

- Adds new column that contains the combined feature vector

```
transformed_data = assembler.transform(selected_data)
```

Step 7 Split data into training and test sets

- Train and test performance on unseen data

```
train_data, test_data = transformed_data.randomSplit([0.7, 0.3], seed=123)
```

Regression using Spark ML: Steps

Step 8 Create instance of linear regression model

- Specify feature and target variable column name

```
lr = LinearRegression(featuresCol="features", labelCol="target_col")
```

Step 9 Fit model to training data

- Allows models to learn from examples

```
model = lr.fit(train_data)
```

Regression using Spark ML: Steps

Step 10 Make predictions on the test data

- Helps evaluate how model performs

```
predictions = model.transform(test_data)
```

Step 11 Evaluate the model

- Calculate RMSE using RegressionEvaluator

```
evaluator = RegressionEvaluator(labelCol="target_col", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
```

Regression using Spark ML: Steps

Step 12 Print the RMSE

- Provides performance feedback

```
print("Root Mean Squared Error (RMSE):", rmse)
```

Step 13 Print the coefficients and intercept of model

- Represents the learned parameters of the model

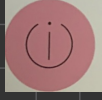
```
print("Coefficients:", model.coefficients)
print("Intercept:", model.intercept)
```

Regression using Spark ML: Steps

Step 14 Stop the SparkSession

- Releases the resources

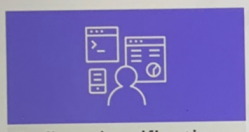
```
spark.stop()
```



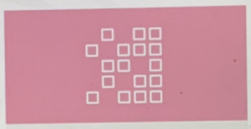
Use code as a starting point and modify and experiment according to specific dataset and requirements.

Classification using SparkML

What you will learn



Define classification using Spark ML



Identify the steps to perform classification using Spark ML

What is classification using Spark ML?

- Predicts categorical labels or classes
- Is a supervised learning technique
- Algorithms:
  - Learns from labeled training data
  - Classifies unseen data into predefined categories

Spark ML classification algorithms

- Helps perform tasks efficiently and at scale
- Leverages the distributed computing capabilities
- Handle large datasets and perform computations

Used for:

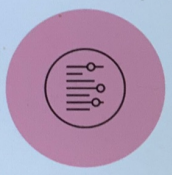
- Sentiment analysis
- Spam detection
- Fraud detection
- Image classification

Classification using Spark ML: Steps

1. Import the necessary libraries.
2. Create a SparkSession.
3. Load the dataset.
4. Select the feature columns and the target column.
5. Assemble the feature columns into a vector column.
6. Split the data into training and test sets.

Classification using Spark ML: Steps

7. Create a Logistic regression model.
8. Train the model.
9. Make predictions on the test data.
10. Evaluate the model.
11. Print the accuracy, and
12. Stop the Spark session.



Classification using Spark ML: Steps

Step 1 Import necessary libraries

- Import libraries required for ML tasks

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import
MulticlassClassificationEvaluator
```

- Provides necessary classes and functions

Classification using Spark ML: Steps

Step 2 Create a SparkSession

- Serves as the entry point for Spark functionality

```
spark = SparkSession.builder.getOrCreate()
```

Step 3 Load the dataset

- Provide file path, header and inferSchema parameters

```
data = spark.read.csv('path/to/dataset.csv', header=True,
inferSchema=True)
```

Classification using Spark ML: Steps

Step 4 Select the feature columns and the target column

- Define a list that contains column names

```
feature_cols = ["column1", "column2", "column3", "column4", "column5"]
```

Step 5 Assemble the feature columns into a vector column

- Combine values of selected features into a single vector

```
assembler = VectorAssembler(inputCols=feature_cols,
outputCol='features')
data = assembler.transform(data)
```

Classification using Spark ML: Steps

Step 6 Split the data into training and test sets

- Train and test performance on unseen data

```
train_data, test_data = data.randomSplit([0.8, 0.2], seed=42)
```

Step 7 Create a LogisticRegression model

- Specify features and target labels

```
lr = LogisticRegression(labelCol="label", featuresCol='features')
```

Classification using Spark ML: Steps

Step 8 Train the model

- Allows models to learn from labeled data

```
model = lr.fit(train_data)
```

Step 9 Make predictions on the test data

- Helps evaluate how model performs

```
predictions = model.transform(test_data)
```

Classification using Spark ML: Steps

Step 10 Evaluate the model

- Evaluates performance of classification model

```
evaluator = MulticlassClassificationEvaluator(labelCol=label,
metricName='accuracy')
accuracy = evaluator.evaluate(predictions)
```

- Compares predicted labels with the true labels

Classification using Spark ML: Steps

Step 11 Print the accuracy

- Provides performance feedback

```
print("Accuracy:", accuracy)
```

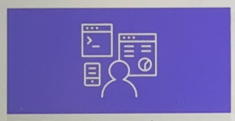
Step 12 Stop the SparkSession

- Release the resources

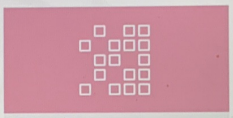
```
spark.stop()
```

Clustering using SparkML

What you will learn



Define clustering using Spark ML



Identify the steps to perform clustering using Spark ML

What is clustering using Spark ML?

- Groups similar data points together
- Is an unsupervised learning technique
- Doesn't rely on predefined labels or target variables
- Discovers patterns and structures based on similarity

Spark ML clustering algorithms

- Gaussian Mixture Models (GMM)
- Bisecting K-means
- K-means

Used for:

- Customer segmentation
- Anomaly detection
- Image segmentation
- Recommendation systems

Clustering using Spark ML: Steps

1. Import the necessary libraries.
2. Create a SparkSession.
3. Load the data into a Dataframe.
4. Select the features you want to use for clustering and assemble them into a vector.
5. Train the K-means model.
6. Make predictions.
7. Show the cluster assignments
8. Stop the Spark session.

Clustering using Spark ML: Steps

Step 1 Import necessary libraries

- Import libraries required for clustering tasks

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans
```

- Provides necessary classes and functions

Clustering using Spark ML: Steps

Step 2 Create a SparkSession

- Serves as the entry point for Spark functionality

```
spark = SparkSession.builder.appName("ClusteringExample").getOrCreate()
```

Step 3 Load the data into a DataFrame

- Provide file path, header, and inferSchema parameters

```
data = spark.read.csv("path_to_your_data.csv", header=True,
inferSchema=True)
```

Clustering using Spark ML: Steps

Step 4 Select the features you want to use for clustering and assemble them into a vector

- Define a list of feature columns

```
assembler = VectorAssembler(inputCols=["feature1", "feature2"],
outputCol="features")
data = assembler.transform(data)
```

Step 5 Train the K-means model

- Train K-means algorithm with cluster number and seed

```
kmeans = KMeans(k=3)
model = kmeans.fit(data)
```

Clustering using Spark ML: Steps

Step 6 Make predictions

- Helps evaluate how the model performs

```
predictions = model.transform(data)
```

Step 7 Show the cluster assignments

- Display cluster assignments and print silhouette score

```
predictions.select("features", "prediction").show()
```

Clustering using Spark ML: Steps

Step 8 Stop the SparkSession

- Release the resources

```
spark.stop()
```

That's it! You now have a basic understanding of how to perform clustering using Spark ML.

GraphFrames on Apache Spark

Objectives

After watching this video, you will be able to:

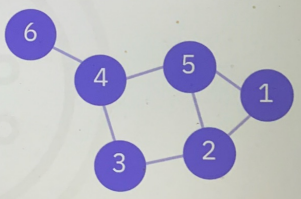
- Define graph theory
- Describe Apache Spark GraphFrames
- Identify data suitable for GraphFrames

Graph Theory

Graph Theory is the mathematical study of modeling pairwise relationships between objects

Graphs consist of

- Vertices
- Edges that connect one vertex to another

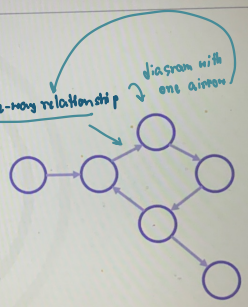


Graph Theory

Graphs can either directed or undirected

Directed graphs

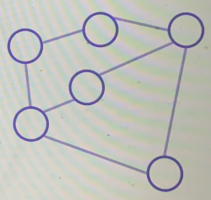
- Contain edges with a single direction between two vertices
- Examples: manufacturing optimization, project scheduling, Train and airline route analysis, traffic recommendations, and others



Graph Theory

Undirected graphs

- Contain edges with no defined directions
- Examples: social relationship analysis, marketing analysis, genomics analysis, knowledge repositories, and others



GraphFrames – What is it?

An Apache Spark extension for graph processing

- Based on Spark DataFrames
- Runs queries on graphs of vertices and edges and represents data
- Contains built-in algorithms
- Exists as a separate, downloadable package

Downloading GraphFrames

- Available at the Spark-packages.org website
- Specify the argument `--packages` followed by the package group, name and version as shown onscreen when submitting your Spark application with `spark-submit'
- **Important!** Choose the GraphFrames package that matches your installed Spark and Scala versions

```
--packages graphframes:graphframes:0.8.1-spark3.0-s_2.12
```

spark scala

Using GraphFrames

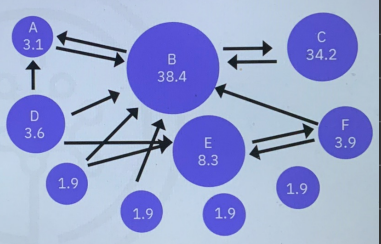
- Runs SQL queries on vertex and edge DataFrames
- Requires that you to set a directory for checkpoints
- Performs Motif finding, which searches the graph for structural patterns
  - that uses domain specific language, DSL, to specify query in terms of edges and vertices

Supported Graph Algorithms

- Breadth-first search (BFS)
- Connected components (strongly connected components)
- Label Propagation Algorithm (LPA)
- PageRank
- Shortest paths
- Triangle count

PageRank Graph Algorithms

Developed by Google to measure importance and rank web pages for their search engine



Supported types of data

- Ideal for modeling data with connecting relationships
- Computes relationship strength and direction

Supported types of data

Friend Example:

```
// Vertex DataFrame: unique id

g.vertices.show()
// +--+-------+---+
// |id|   name|age|
// +--+-------+---+
// | a|  Alice| 34|
// | b|    Bob| 36|
// | c|Charlie| 30|
// | d|  David| 29|
// | e| Esther| 32|
// | f|  Fanny| 36|
// | g|  Gabby| 60|
// +--+-------+---+
```

show name, id

relationship between both by id

can query by Spark SQL or any GraphFrames

```
// Edge DataFrame: src & dst vertex

g.edges.show()
// +---+---+------------+
// |src|dst|relationship|
// +---+---+------------+
// |  a|  b|      friend|
// |  b|  c|      follow|
// |  c|  b|      follow|
// |  f|  c|      follow|
// |  e|  f|      follow|
// |  e|  d|      friend|
// |  d|  a|      friend|
// |  a|  e|      friend|
// +---+---+------------+
```