Apache Spark™ Tutorial: Getting Started with Apache Spark on Databricks

* [**INTRODUCTION**](https://databricks.com/spark/getting-started-with-apache-spark)
* [**QUICK START**](https://databricks.com/spark/getting-started-with-apache-spark/quick-start)
* [**DATAFRAMES**](https://databricks.com/spark/getting-started-with-apache-spark/dataframes)
* [**DATASETS**](https://databricks.com/spark/getting-started-with-apache-spark/datasets)
* [**MACHINE LEARNING**](https://databricks.com/spark/getting-started-with-apache-spark/machine-learning)
* [**STREAMING**](https://databricks.com/spark/getting-started-with-apache-spark/streaming)
* [**WHAT’S NEXT**](https://databricks.com/spark/getting-started-with-apache-spark/whats-next)

**MACHINE LEARNING**

* [Overview](https://databricks.com/spark/getting-started-with-apache-spark/machine-learning?utm_campaign=701610000008nLSAAY&gclid=EAIaIQobChMI-6a6xvSd4QIVRb7ACh3lsgL1EAAYASABEgIDc_D_BwE#overview)
* [Load sample data](https://databricks.com/spark/getting-started-with-apache-spark/machine-learning?utm_campaign=701610000008nLSAAY&gclid=EAIaIQobChMI-6a6xvSd4QIVRb7ACh3lsgL1EAAYASABEgIDc_D_BwE#load-sample-data)
* [Prepare and visualize data for ML algorithms](https://databricks.com/spark/getting-started-with-apache-spark/machine-learning?utm_campaign=701610000008nLSAAY&gclid=EAIaIQobChMI-6a6xvSd4QIVRb7ACh3lsgL1EAAYASABEgIDc_D_BwE#prepare-and-visualize-data-for-ml-algorithms)
* [Run the linear regression model](https://databricks.com/spark/getting-started-with-apache-spark/machine-learning?utm_campaign=701610000008nLSAAY&gclid=EAIaIQobChMI-6a6xvSd4QIVRb7ACh3lsgL1EAAYASABEgIDc_D_BwE#run-the-linear-regression-model)
* [Evaluate the model](https://databricks.com/spark/getting-started-with-apache-spark/machine-learning?utm_campaign=701610000008nLSAAY&gclid=EAIaIQobChMI-6a6xvSd4QIVRb7ACh3lsgL1EAAYASABEgIDc_D_BwE#evaluate-the-model)
* [Visualize the model](https://databricks.com/spark/getting-started-with-apache-spark/machine-learning?utm_campaign=701610000008nLSAAY&gclid=EAIaIQobChMI-6a6xvSd4QIVRb7ACh3lsgL1EAAYASABEgIDc_D_BwE#visualize-the-model)
* [Additional Resources](https://databricks.com/spark/getting-started-with-apache-spark/machine-learning?utm_campaign=701610000008nLSAAY&gclid=EAIaIQobChMI-6a6xvSd4QIVRb7ACh3lsgL1EAAYASABEgIDc_D_BwE#additional-resources)

Overview

As organizations create more diverse and more user-focused data products and services, there is a growing need for machine learning, which can be used to develop personalizations, recommendations, and predictive insights. The Apache Spark machine learning library (MLlib) allows data scientists to focus on their data problems and models instead of solving the complexities surrounding distributed data (such as infrastructure, configurations, and so on).

In this tutorial module, you will learn how to:

* [Load sample data](https://databricks.com/spark/getting-started-with-apache-spark/machine-learning?utm_campaign=701610000008nLSAAY&gclid=EAIaIQobChMI-6a6xvSd4QIVRb7ACh3lsgL1EAAYASABEgIDc_D_BwE#load-sample-data)
* [Prepare and visualize data for ML algorithms](https://databricks.com/spark/getting-started-with-apache-spark/machine-learning?utm_campaign=701610000008nLSAAY&gclid=EAIaIQobChMI-6a6xvSd4QIVRb7ACh3lsgL1EAAYASABEgIDc_D_BwE#prepare-and-visualize-data-for-ml-algorithms)
* [Run a linear regression model](https://databricks.com/spark/getting-started-with-apache-spark/machine-learning?utm_campaign=701610000008nLSAAY&gclid=EAIaIQobChMI-6a6xvSd4QIVRb7ACh3lsgL1EAAYASABEgIDc_D_BwE#run-a-linear-regression-model)
* [Evaluation a linear regression model](https://databricks.com/spark/getting-started-with-apache-spark/machine-learning?utm_campaign=701610000008nLSAAY&gclid=EAIaIQobChMI-6a6xvSd4QIVRb7ACh3lsgL1EAAYASABEgIDc_D_BwE#evaluation-a-linear-regression-model)
* [Visualize a linear regression model](https://databricks.com/spark/getting-started-with-apache-spark/machine-learning?utm_campaign=701610000008nLSAAY&gclid=EAIaIQobChMI-6a6xvSd4QIVRb7ACh3lsgL1EAAYASABEgIDc_D_BwE#visualize-a-linear-regression-model)

We also provide a [sample notebook](https://docs.databricks.com/getting-started/spark/machine-learning.html#ml-notebook) that you can import to access and run all of the code examples included in the module.

Load sample data

The easiest way to start working with machine learning is to use an example Databricks dataset available in the /databricks-datasetsfolder accessible within the Databricks workspace. For example, to access the file that compares city population to median sale prices of homes, you can access the file /databricks-datasets/samples/population-vs-price/data\_geo.csv.

# Use the Spark CSV datasource with options specifying:

# - First line of file is a header

# - Automatically infer the schema of the data

data = spark.read.format("csv")

.option("header", "true")

.option("inferSchema", "true")

.load("/databricks-datasets/samples/population-vs-price/data\_geo.csv")

data.cache() # Cache data for faster reuse

To view this data in a tabular format, instead of exporting this data to a third-party tool, you can use the display() command in your Databricks notebook.

display(data)

Prepare and visualize data for ML algorithms

In supervised learning—-such as a regression algorithm—-you typically define a label and a set of features. In this linear regression example, the label is the 2015 median sales price and the feature is the 2014 Population Estimate. That is, you use the feature (population) to predict the label (sales price).

First drop rows with missing values and rename the feature and label columns, replacing spaces with \_.

data = data.dropna() # drop rows with missing values

exprs = [col(column).alias(column.replace(' ', '\_')) for column in data.columns]

To simplify the creation of features, register a UDF to convert the feature (2014\_Population\_estimate) column vector to a VectorUDT type and apply it to the column.

from pyspark.ml.linalg import Vectors, VectorUDT

spark.udf.register("oneElementVec", lambda d: Vectors.dense([d]), returnType=VectorUDT())

tdata = data.select(\*exprs).selectExpr("oneElementVec(2014\_Population\_estimate) as features", "2015\_median\_sales\_price as label")

Then display the new DataFrame:

display(tdata)

Run the linear regression model

In this section, you run two different linear regression models using different regularization parameters to determine how well either of these two models predict the sales price (label) based on the population (feature).

Build the model

# Import LinearRegression class

from pyspark.ml.regression import LinearRegression

# Define LinearRegression algorithm

lr = LinearRegression()

# Fit 2 models, using different regularization parameters

modelA = lr.fit(data, {lr.regParam:0.0})

modelB = lr.fit(data, {lr.regParam:100.0})

Using the model, you can also make predictions by using the transform() function, which adds a new column of predictions. For example, the code below takes the first model (modelA) and shows you both the label (original sales price) and prediction (predicted sales price) based on the features (population).

# Make predictions

predictionsA = modelA.transform(data)

display(predictionsA)

Evaluate the model

To evaluate the regression analysis, calculate the root mean square error using the RegressionEvaluator. Here is the Python code for evaluating the two models and their output.

from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(metricName="rmse")

RMSE = evaluator.evaluate(predictionsA)

print("ModelA: Root Mean Squared Error = " + str(RMSE))

# ModelA: Root Mean Squared Error = 128.602026843

predictionsB = modelB.transform(data)

RMSE = evaluator.evaluate(predictionsB)

print("ModelB: Root Mean Squared Error = " + str(RMSE))

# ModelB: Root Mean Squared Error = 129.496300193

Visualize the model

As is typical for many machine learning algorithms, you want to visualize the scatterplot. Since Databricks supports pandas and ggplot, the code below creates a linear regression plot using pandas DataFrame (pydf) and ggplot to display the scatterplot and the two regression models.

# Import numpy, pandas, and ggplot

import numpy as np

from pandas import \*

from ggplot import \*

# Create Python DataFrame

pop = data.map(lambda p: (p.features[0])).collect()

price = data.map(lambda p: (p.label)).collect()

predA = predictionsA.select("prediction").map(lambda r: r[0]).collect()

predB = predictionsB.select("prediction").map(lambda r: r[0]).collect()

# Create a Pandas DataFrame

pydf = DataFrame({'pop':pop,'price':price,'predA':predA, 'predB':predB})

Visualizing the Model

# Create scatter plot and two regression models (scaling exponential) using ggplot

p = ggplot(pydf, aes('pop','price')) +

geom\_point(color='blue') +

geom\_line(pydf, aes('pop','predA'), color='red') +

geom\_line(pydf, aes('pop','predB'), color='green') +

scale\_x\_log10() + scale\_y\_log10()

display(p)

We also provide a [sample notebook](https://docs.databricks.com/getting-started/spark/machine-learning.html#ml-notebook) that you can import to access and run all of the code examples included in the module.

Additional Resources

* [Apache Spark MLlib: From Quick Start to Scikit-Learn](https://go.databricks.com/spark-mllib-from-quick-start-to-scikit-learn)
* [Databricks Visualizations](https://vimeo.com/156886721)
* [Mastering Advanced Analytics with Databricks](https://go.databricks.com/mastering-advanced-analytics-apache-spark-databricks)
* [Gentle Introduction to Spark and DataFrames Notebook](https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/346304/2168141618055043/484361/latest.html)
* [On-Time Flight Performance with GraphFrames for Apache Spark](https://databricks.com/blog/2016/03/16/on-time-flight-performance-with-spark-graphframes.html)

[**CONTINUE TO NEXT MODULE: STREAMING**](https://databricks.com/spark/getting-started-with-apache-spark/streaming)

Apache Spark™ Tutorial: Getting Started with Apache Spark on Databricks

* [**INTRODUCTION**](https://databricks.com/spark/getting-started-with-apache-spark)
* [**QUICK START**](https://databricks.com/spark/getting-started-with-apache-spark/quick-start)
* [**DATAFRAMES**](https://databricks.com/spark/getting-started-with-apache-spark/dataframes)
* [**DATASETS**](https://databricks.com/spark/getting-started-with-apache-spark/datasets)
* [**MACHINE LEARNING**](https://databricks.com/spark/getting-started-with-apache-spark/machine-learning)
* [**STREAMING**](https://databricks.com/spark/getting-started-with-apache-spark/streaming)
* [**WHAT’S NEXT**](https://databricks.com/spark/getting-started-with-apache-spark/whats-next)

**STREAMING**

* [Structured Streaming Overview](https://databricks.com/spark/getting-started-with-apache-spark/streaming#structured-streaming-overview)
* [Load sample data](https://databricks.com/spark/getting-started-with-apache-spark/streaming#load-sample-data)
* [Initialize the stream](https://databricks.com/spark/getting-started-with-apache-spark/streaming#initialize-the-stream)
* [Start the streaming job](https://databricks.com/spark/getting-started-with-apache-spark/streaming#start-the-streaming-job)
* [Interactively query the stream](https://databricks.com/spark/getting-started-with-apache-spark/streaming#interactively-query-the-stream)
* [Additional Resources](https://databricks.com/spark/getting-started-with-apache-spark/streaming#additional-resources)

Structured Streaming Overview

Sensors, IoT devices, social networks, and online transactions all generate data that needs to be monitored constantly and acted upon quickly. As a result, the need for large-scale, real-time stream processing is more evident than ever before. This tutorial module introduces Structured [Streaming](https://databricks.com/glossary/what-is-spark-streaming), the main model for handling streaming datasets in Apache Spark. In Structured Streaming, a data stream is treated as a table that is being continuously appended. This leads to a stream processing model that is very similar to a batch processing model. You express your streaming computation as a standard batch-like query as on a static table, but Spark runs it as an incremental query on the unbounded input table.

Consider the input data stream as the input table. Every data item that is arriving on the stream is like a new row being appended to the input table.

A query on the input generates a result table. At every trigger interval (say, every 1 second), new rows are appended to the input table, which eventually updates the result table. Whenever the result table is updated, the changed result rows are written to an external sink. The output is defined as what gets written to external storage. The output can be configured in different modes:

* **Complete Mode**: The entire updated result table is written to external storage. It is up to the storage connector to decide how to handle the writing of the entire table.
* **Append Mode**: Only new rows appended in the result table since the last trigger are written to external storage. This is applicable only for the queries where existing rows in the Result Table are not expected to change.
* **Update Mode**: Only the rows that were updated in the result table since the last trigger are written to external storage. This is different from Complete Mode in that Update Mode outputs only the rows that have changed since the last trigger. If the query doesn’t contain aggregations, it is equivalent to Append mode.

In this tutorial module, you will learn how to:

* [Load sample data](https://databricks.com/spark/getting-started-with-apache-spark/streaming#load-streaming-sample-data)
* [Initialize a stream](https://databricks.com/spark/getting-started-with-apache-spark/streaming#init-stream)
* [Start a stream job](https://databricks.com/spark/getting-started-with-apache-spark/streaming#start-stream)
* [Query a stream](https://databricks.com/spark/getting-started-with-apache-spark/streaming#query-stream)

We also provide a [sample notebook](https://docs.databricks.com/getting-started/spark/streaming.html#notebook-stream) that you can import to access and run all of the code examples included in the module.

Load sample data

The easiest way to get started with Structured Streaming is to use an example Databricks dataset available in the /databricks-datasets folder accessible within the Databricks workspace. Databricks has sample event data as files in /databricks-datasets/structured-streaming/events/ to use to build a Structured Streaming application. Let’s take a look at the contents of this directory.Each line in the file contains a JSON record with two fields: time and action.

{"time":1469501675,"action":"Open"}

{"time":1469501678,"action":"Close"}{"time":1469501680,"action":"Open"}{"time":1469501685,"action":"Open"}{"time":1469501686,"action":"Open"}{"time":1469501689,"action":"Open"}{"time":1469501691,"action":"Open"}{"time":1469501694,"action":"Open"}{"time":1469501696,"action":"Close"}{"time":1469501702,"action":"Open"}{"time":1469501703,"action":"Open"}{"time":1469501704,"action":"Open"}

Initialize the stream

Since the sample data is just a static set of files, you can emulate a stream from them by reading one file at a time, in the chronological order in which they were created.

inputPath = "/databricks-datasets/structured-streaming/events/"

# Define the schema to speed up processing

jsonSchema = StructType([ StructField("time", TimestampType(), True), StructField("action", StringType(), True) ])

streamingInputDF = (

spark

.readStream

.schema(jsonSchema) # Set the schema of the JSON data

.option("maxFilesPerTrigger", 1) # Treat a sequence of files as a stream by picking one file at a time

.json(inputPath)

)

streamingCountsDF = (

streamingInputDF

.groupBy(

streamingInputDF.action,

window(streamingInputDF.time, "1 hour"))

.count()

)

Start the streaming job

You start a streaming computation by defining a sink and starting it. In our case, to query the counts interactively, set the *complete* set of 1 hour counts to be in an in-memory table.

query = (

streamingCountsDF

.writeStream

.format("memory") # memory = store in-memory table (for testing only)

.queryName("counts") # counts = name of the in-memory table

.outputMode("complete") # complete = all the counts should be in the table

.start()

)

query is a handle to the streaming query named counts that is running in the background. This query continuously picks up files and updates the windowed counts.

The command window reports the status of the stream:

When you expand counts, you get a dashboard of the number of records processed, batch statistics, and the state of the aggregation:

Interactively query the stream

We can periodically query the counts aggregation:

%sql select action, date\_format(window.end, "MMM-dd HH:mm") as time, count from counts order by time, action

As you can see from this series of screenshots, the query changes every time you execute it to reflect the action count based on the input stream of data.

We also provide a [sample notebook](https://docs.databricks.com/getting-started/spark/streaming.html#notebook-stream) that you can import to access and run all of the code examples