Key takeaways

- Learn about machine learning data pipelines
- How Apache Spark ML package can help implement the ML data pipelines
- Steps in data value chain process
- Spark ML Pipeline Components and API
- Text classification and spam detection use case

This is the fifth article of the "Big Data Processing with Apache Spark" series. Please see also: Part 1: Introduction, Part 2: Spark SQL, Part 3: Spark Streaming, Part 4: Spark Machine Learning, Part 6: Graph Data Analytics with Spark GraphX.

In the previous articles in the "Big Data Processing with Apache Spark" series, we looked at Apache Spark framework and its different libraries for big data processing with Spark Introduction (Part 1), Spark SQL library (Part 2), Spark Streaming (Part 3) and Spark MLlib Machine Learning library (Part 4).

In this article, we will focus on the other Machine Learning API from Spark, called Spark ML, which is the recommended solution for big data applications developed using data pipelines.

Spark ML (spark.ml) package provides machine learning API built on the DataFrames which are becoming the core part of Spark SQL library. This package can be used for developing and managing the machine learning pipelines. It also provides feature extractors, transformers, selectors and supports machine learning techniques like classification, regression, and clustering. All of these are critical for developing machine learning solutions.

We'll look at how we can use Apache Spark to perform exploratory data analysis (EDA), develop machine learning pipelines, and use the APIs and algorithms available in Spark ML package.

With support for building Machine Learning data pipelines, Apache Spark framework is a great choice for building a unified use case that combines ETL, batch analytics, real-time stream analysis, machine learning, graph processing and visualizations.

Machine Learning Data Pipelines

Machine learning pipelines are used for the creation, tuning, and inspection of machine learning workflow programs. ML pipelines help us focus more on the big data requirements and machine learning tasks in our projects instead of spending time and effort on the infrastructure and distributed computing areas. They also help us with the exploratory stages of machine learning problems where we need to develop iterations of features and model combinations.

Machine Learning (ML) workflows often involve a sequence of processing and learning stages. Machine learning data pipeline is specified as a sequence of stages where each stage is either a Transformer or an Estimator component. These stages are executed in order, and the input data is transformed as it passes through each stage in the pipeline.

ML development frameworks need to support distributed computation as well as utilities to help with assembling the pipeline components. Other requirements for building data pipelines include fault tolerance, resource management, scalability and maintainability.

The machine learning workflow solutions in real world projects also include utilities like model import/export, cross-validation to choose parameters, and aggregate data from multiple data sources. They provide data utilities like feature extraction, selection and statistics. These frameworks support machine learning pipeline persistence to save and load ML models and pipelines for future use.

The concept of machine learning workflows and the composition of dataflow operators is becoming popular in several different systems. Big data processing frameworks like scikit-learnand GraphLab use the concept of pipelines built into the system.

A typical data value chain process includes the following steps:

- Discover
- Ingest
- Process
- Persist
- Integrate
- Analyze
- Expose

A machine learning data pipeline follows a similar approach. The following table shows the different steps involved in a machine learning pipeline process.

Step #	Name	Description
ML1	Data Ingestion	Loading the data from different data sources.
ML2	Data Cleaning	Data is pre-processed to get it ready for the machine learning data analysis.
ML3	Feature Extraction	Also known as Feature Engineering, this step is about extracting the features from the data sets.
ML4	Model Training	The machine learning model is trained in the next step using the training data sets.
ML5	Model Validation	Next, the machine learning model is evaluated based on different prediction parameters, for its effectiveness. We also tune the model during the validation step. This step is used to pick the best model.
ML6	Model Testing	The next step is to test the mode before it is deployed.
ML7	Model deployment	Final step is to deploy the selected model to execute in production environment.

Table 1. Machine learning pipeline process steps

These steps are illustrated in Figure 1 below.

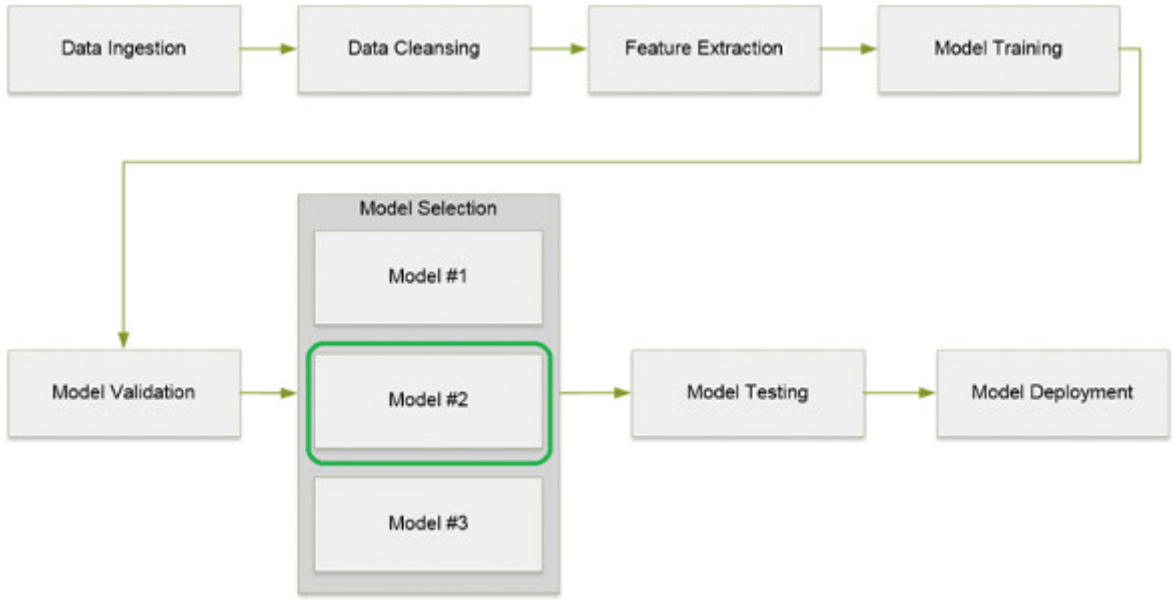


Figure 1. Machine learning data pipeline process flow diagram

Let's look at each of these steps in more detail.

Data Ingestion: The data we collect for a typical machine learning pipeline application can come from multiple data sources and can range from few hundred gigabytes to a terabyte. Also, one of the characteristics of big data applications is ingesting data in different formats.

Data Cleaning: Data cleaning is the first and critical step in the overall data analytics pipeline. Also known as data cleansing, data scrubbing, or data wrangling, it's used to structure the data to facilitate data processing and predictive analytics. Depending on the quality of data coming into the system, 60-70% of the overall time is spent in data cleaning to bring data to a suitable format so machine learning models can be applied on the data.

Data can have various quality issues like missing data or the data elements with incorrect or irrelevant values. Data cleaning process typically uses various approaches including custom transformers where the data cleansing actions are executed with custom transformers included in the pipeline.

Sparse or coarse-grained data is another challenge in data analytics. This is where lot of corner cases occur so we have to apply data cleaning techniques to make sure the data is of decent quality before feeding it to the data pipeline.

Data cleaning is usually an iterative process as we understand the problem deeper on each successive attempt and update the model iteratively. Data wrangling tools like Trifacta, OpenRefine or ActiveClean are used for data cleaning needs.

Feature Extraction: In Feature Extraction (sometimes called Feature Engineering) step, we extract specific features from the raw data using techniques like Feature Hashing (Hashing Term Frequency) and Word2Vec. The results from this step are usually combined using an Assembler component and are passed to next step in the process.

Model Training: Machine learning model training involves providing an algorithm and some training data that the model can learn from. The learning algorithm finds patterns in the training data and generates an output model.

Model Validation: This step involves evaluating and tuning the ML model to assess how effective it is with the predictions. As described in this article, for binary classification models the evaluation metric could be the area under the Receiver Operating Characteristic (ROC) curve. ROC curve illustrates the performance of a binary classifier system. It's created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

Model Selection: Model selection is done by using data to choose parameters for Transformers and Estimators. This is a critical step in the machine learning pipeline process. Classes like ParamGridBuilder and CrossValidator provide APIs for selecting the ML model.

Model Deployment: Once we select the right model, we can deploy it and start feeding new data and receive the predictive analytics results. We can also deploy machine learning models as web services.

Spark ML

The machine learning pipeline API was introduced in Apache Spark framework version 1.2. It provides the API for developers to create and execute complex ML workflows. The goal of the Pipeline API is to let users quickly and easily assemble and configure practical distributed machine learning pipelines by standardizing the APIs for different machine learning concepts. The Pipeline API is available in `org.apache.spark.ml` package.

Spark ML also helps with combining multiple machine learning algorithms into a single pipeline.

Spark machine learning API is divided into two packages called `spark.mllib` and `spark.ml`. The `spark.mllib` package contains the original API built on top of RDDs. On the other hand, the `spark.ml` package provides higher-level API built on top of DataFrames for constructing ML pipelines.

The MLlib library API which is based on RDDs is now in maintenance mode.

Spark ML is an important big data analytics library in the Apache Spark ecosystem as shown in Figure 2 below.

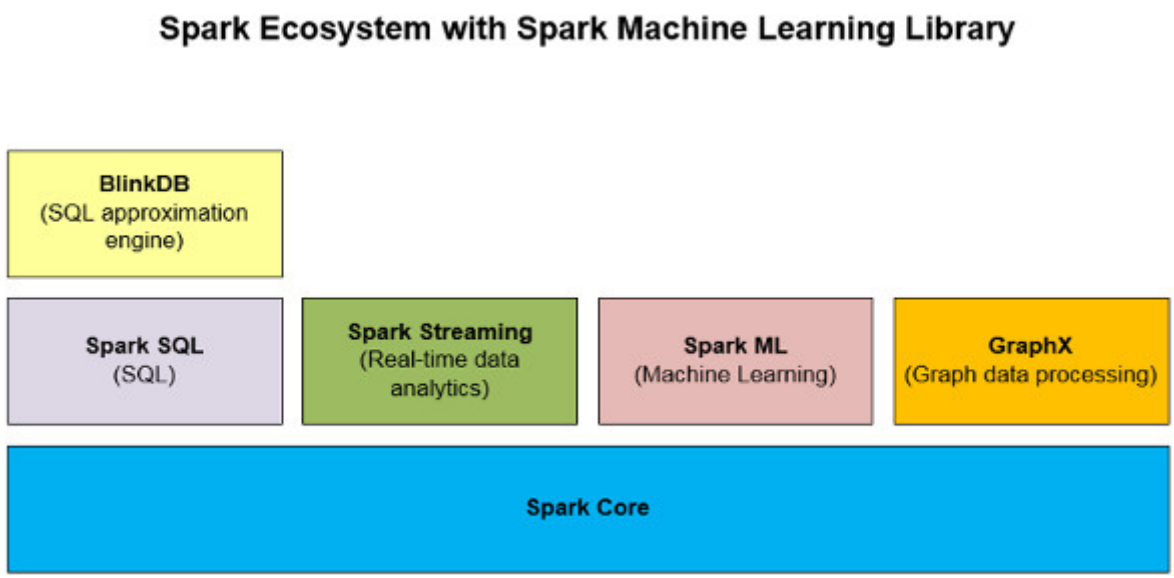


Figure 2. Spark Ecosystem with Spark ML

ML Pipeline Components

Machine learning data pipeline consists of several components to perform the data analytics. The key components of a data pipeline are listed below:

- Datasets
- Pipelines
- Pipeline Stages
- Transformers
- Estimators
- Evaluators
- Params (and ParamMaps)

Let's take a quick look at where each of these components fit into the overall process.

Datasets: DataFrame is used for representing datasets in ML pipeline. It allows storing structured data into named columns. The columns can be used for storing text, feature vectors, true labels, and predictions.

Pipelines: ML workflows are modeled as Pipelines, which consist of a sequence of stages. Each stage transforms input data to produce output for succeeding stages. A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow.

Pipeline Stages: We define two kinds of stages: Transformers and Estimators.

Transformer: An algorithm which can transform one DataFrame into another DataFrame (example: ML model is a transformer that transforms a DF with features into a DF with predictions).

A Transformer converts a DataFrame into another DataFrame with one or more added features to it. For example in Spark ML package, OneHotEncoder transforms a column with a label index into a column of vectored features. Every Transformer has a method `transform()` which is called to transform a DataFrame into another.

Estimator: Estimator is a machine learning algorithm which learns from the data provided. The input to an estimator is a DataFrame and output is a Transformer. An Estimator is used to train the model. It produces a Transformer. For example, a `LogisticRegression` estimator produces a `LogisticRegressionModel` transformer. Another example is K Means as an estimator accepts a training DataFrame and produces a K Means model which is a transformer.

Parameter: Machine learning components use a common API for specifying parameters. An example of a parameter is the maximum number of iterations that the model should use.

The components of the data pipeline for text classification use case are shown in the following diagram.

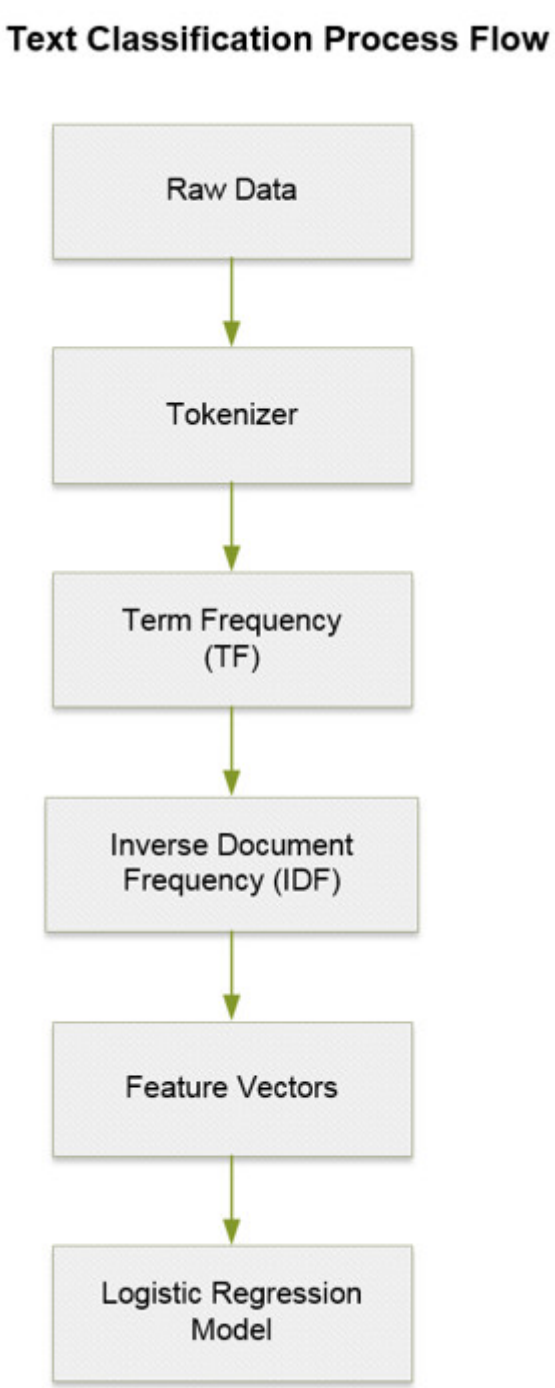


Figure 3. Data Pipelines using Spark ML

Use Cases

One of the use cases for using machine learning pipelines is text categorization. This use case typically includes the following steps:

- clean the text data
- transform data into feature vectors, and
- train the classification model

In text categorization or classification, data preprocessing steps like n-gram extraction and TF-IDF feature weighting are used before the training of a classification model (like SVM).

Another machine learning pipeline use case is the image classification as described in this article.

There are several other machine learning use cases that include fraud detection (uses classification model which is part of supervised learning), user segmentation (clustering model which is part of unsupervised learning).

TF-IDF

Term Frequency - Inverse Document Frequency (TF-IDF), is a statistical measure to evaluate how important a word is to a document in a given corpus. It's an information retrieval algorithm used to rank how important a word is to a collection of documents.

TF: If a word appears frequently in a doc, it's important. This is calculated as:

```
TF = (# of times word X appears in a document) / (Total # of
words in the document)
```

IDF: But if a word appears in many docs (like "the", "and", "of"), the word is not meaningful, so lower its score.

Sample Application

Let's look at a sample application to see how the Spark ML package can be used in a big data processing system. We'll develop a document classification application to identify spam content in the datasets provided to the application. The dataset examples are Documents, Email messages, or other content received from external systems that can contain spam content.

We'll use the Spam Detection example discussed in "Building machine-learning apps with Spark" at Strata Hadoop World Conference workshop to build our sample application.

Use Case

This use case includes analyzing different messages sent to our system. Some of these messages contain spam whereas the messages we get without any spam are called ham data. Our goal is to find the messages that contain spam using Spark ML API.

Algorithm

We'll use Logistic Regression algorithm in our machine learning program. Logistic Regression is a regression analysis model and is used to predict the probability of a binary response (yes or no) based on one or more independent variables.

Solution Details

Let's look at the details of the sample application and the steps we will be running as part of the Spark ML program.

Data Ingestion: We'll load the datasets (text files) for the content that has the spam data as well as the data that doesn't contain any spam (called ham data).

Data Cleansing: In our sample application, we don't perform any specific data cleaning. We just aggregate all the data into a single DataFrame object.

We create an array object by randomly selecting the data from both training and test datasets. In our example we divide the data sets into 70% of training data and 30% of test data.

We use these two data objects later in the pipeline process to train the model and to make predictions respectively.

Our ML data pipeline includes four steps:

- Tokenizer
- HashingTF
- IDF
- LR

Create a pipeline object and set the above stages in the pipeline. Then we create a Logistic Regression model based on training data in our example.

Now, we can make predictions on the model using the Test data (new datasets).

Figure 4 below shows the architecture diagram of the sample application.

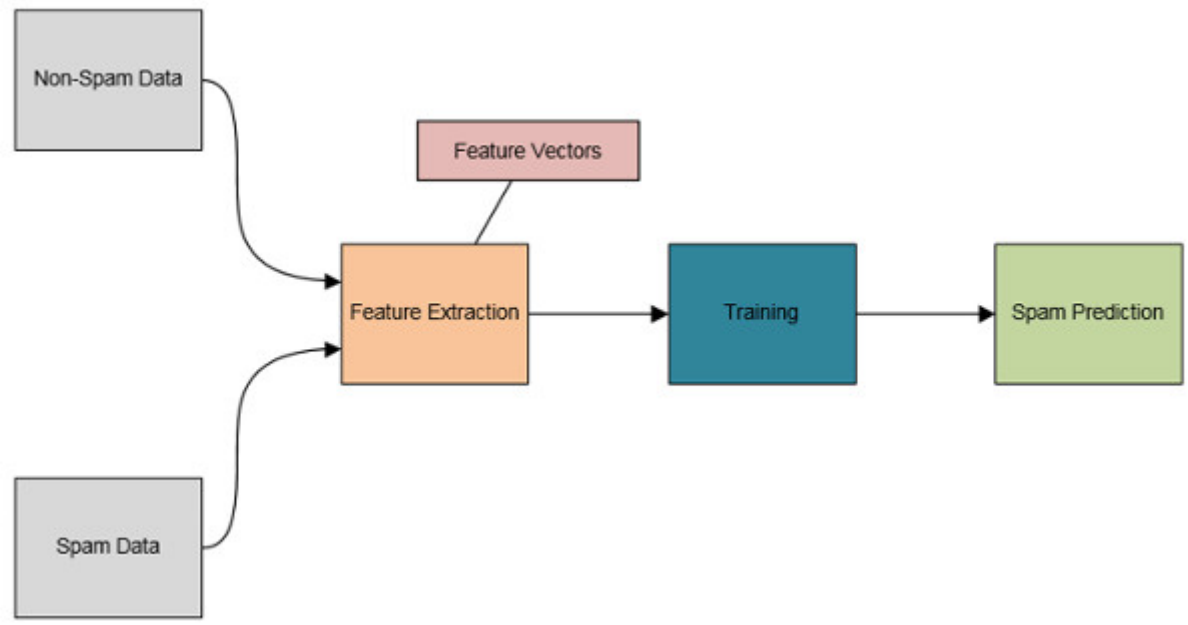


Figure 4. Data classification application architecture diagram

Technologies

We'll use the following technologies in implementing the machine learning pipeline solution.

Technology	Version

Apache Spark	2.0.0
JDK	1.8
Maven	3.3

Table 2. Technologies and tools used in machine learning sample application.

Spark ML Program

The sample machine learning code, from the workshop example, is written in the Scala programming language and we can run the program using Spark Shell console.

Spam detection Scala Code snippets:

Step 1: Create a custom class to store the details of spam content

```
case class SpamDocument(file: String, text: String, label:
Double)
```

Step 2: Initialize SQLContext and import the implicits methods to convert Scala objects into DataFrames. Then load the datasets from the specified directory where the files are located, which returns the RDD objects. Create DataFrame objects from the RDD's for both datasets.

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

//
// Load the data files with spam
//
val rddSData = sc.wholeTextFiles("SPAM_DATA_FILE_DIR", 1)
val dfSData = rddSData.map(d => SpamDocument(d._1, d._2,
1)).toDF()
dfSData.show()

//
// Load the data files with no spam
//
val rddNSData = sc.wholeTextFiles("NO_SPAM_DATA_FILE_DIR",
1)
val dfNSData = rddNSData.map(d => SpamDocument(d._1,
d._2, 0)).toDF()
dfNSData.show()
```

Step 3: Now, aggregate the datasets and split the whole data into training and test datasets (with 70% to 30% ratio)

```
//
// Aggregate both data frames
//
val dfAllData = dfSData.unionAll(dfNSData)
dfAllData.show()

//
// Split the data into 70% training data and 30% test data
//
val Array(trainingData, testData) =
dfAllData.randomSplit(Array(0.7, 0.3))
```

Step 4: We can configure the machine learning data pipeline now which includes creating the components that we discussed earlier in the article, `Tokenizer`, `HashingTF`, and `IDF`. Then create the regression model, in this case, LogisticRegression, using the training data.

```
//
// Configure the ML data pipeline
//

//
// Create the Tokenizer step
//
val tokenizer = new Tokenizer()
  .setInputCol("text")
  .setOutputCol("words")

//
// Create the TF and IDF steps
//
val hashingTF = new HashingTF()
  .setInputCol(tokenizer.getOutputCol)
  .setOutputCol("rawFeatures")

val idf = new
IDF().setInputCol("rawFeatures").setOutputCol("features")

//
// Create the Logistic Regression step
//
val lr = new LogisticRegression()
  .setMaxIter(5)
lr.setLabelCol("label")
lr.setFeaturesCol("features")
```

```
//
// Create the pipeline
//
val pipeline = new Pipeline()
  .setStages(Array(tokenizer, hashingTF, idf, lr))

val lrModel = pipeline.fit(trainingData)
println(lrModel.toString())
```

Step 5: Finally, we can call the `transform` method in logistic regression model to make the predictions on the test data.

```
//
// Make predictions.
//
val predictions = lrModel.transform(testData)

//
// Display prediction results
//
predictions.select("file", "text", "label", "features",
"prediction").show(300)
```

Conclusions

Spark Machine Learning library is one of the critical libraries in Apache Spark framework. It's used for implementing data pipelines. In this article, we learned about how to use Spark ML package API and how to use it in a text classification use case.

What's Next

Graph data models are about the connected data and relationships between different entities in the data model. Graph data processing techniques are getting a lot of attention lately because they can solve problems like fraud detection and develop recommendation engines.

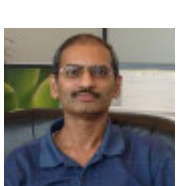
Spark framework provides a library specialized for graph data analytics. We'll learn about this library, called Spark GraphX, in the next article in this series. We'll develop a sample application to perform graph data processing and analytics using Spark GraphX.

References

- Big Data Processing using Apache Spark - Part 1: Introduction
- Big Data Processing using Apache Spark - Part 2: Spark SQL
- Big Data Processing using Apache Spark - Part 3: Spark Streaming
- Big Data Processing using Apache Spark - Part 4: Spark Machine Learning
- Apache Spark Main Website
- Spark Machine Learning Website
- Spark Machine Learning Programming Guide

- Spark Workshop Exercise on Spam Detection

About the Author



Srini Penchikala currently works as Senior Software Architect and is based out of Austin, Texas. He has over 22 years of experience in software architecture, design and development. Penchikala is currently authoring a book on Apache Spark. He is also the co-author of Spring Roo in Action book from Manning Publications. He has presented at conferences like JavaOne, SEI Architecture Technology Conference (SATURN), IT Architect Conference (ITARC), No Fluff Just Stuff, NoSQL Now, Enterprise Data World, and Project World Conference. Penchikala also published several articles on software architecture, security & risk management, NoSQL and Big Data topics on websites like InfoQ, The ServerSide, OReilly Network (ONJava), DevX Java, java.net and JavaWorld. He is the Lead Editor for Data Science community at InfoQ.

This is the fifth article of the "Big Data Processing with Apache Spark" series. Please see also: Part 1: Introduction, Part 2: Spark SQL, Part 3: Spark Streaming, Part 4: Spark Machine Learning, Part 6: Graph Data Analytics with Spark GraphX.