

ENSF 612

Lecture – Spark Machine Learning (ML)

Dr. Gias Uddin, Assistant Professor
Electrical and Software Engineering
Schulich School of Engineering
University of Calgary

<https://giasuddin.ca/>

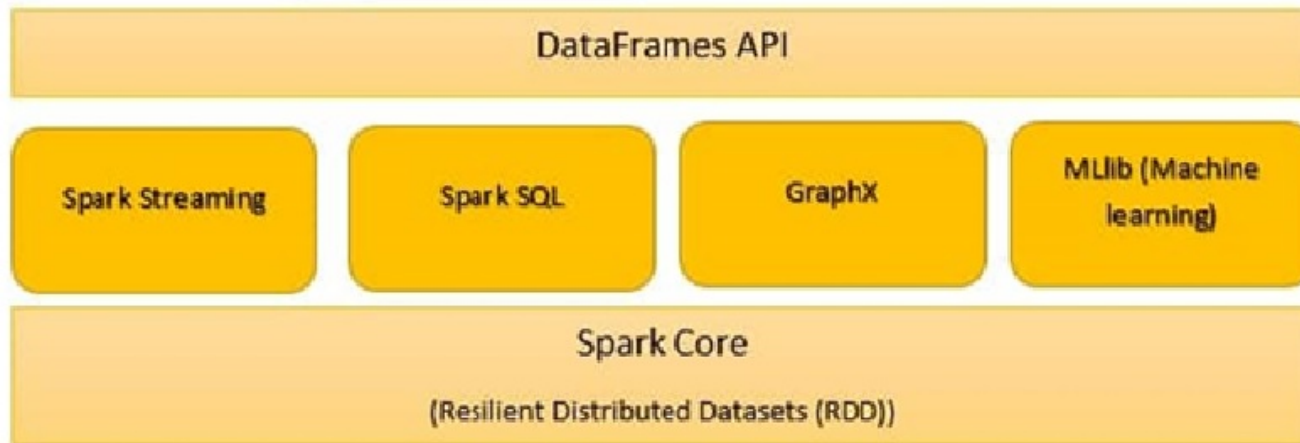
Topics

- Overview
- Pipeline
- Featurization
- Hyperparameter tuning

Overview

PySpark Architecture with MLlib

MLlib is Apache Spark's ML library for ML for big data



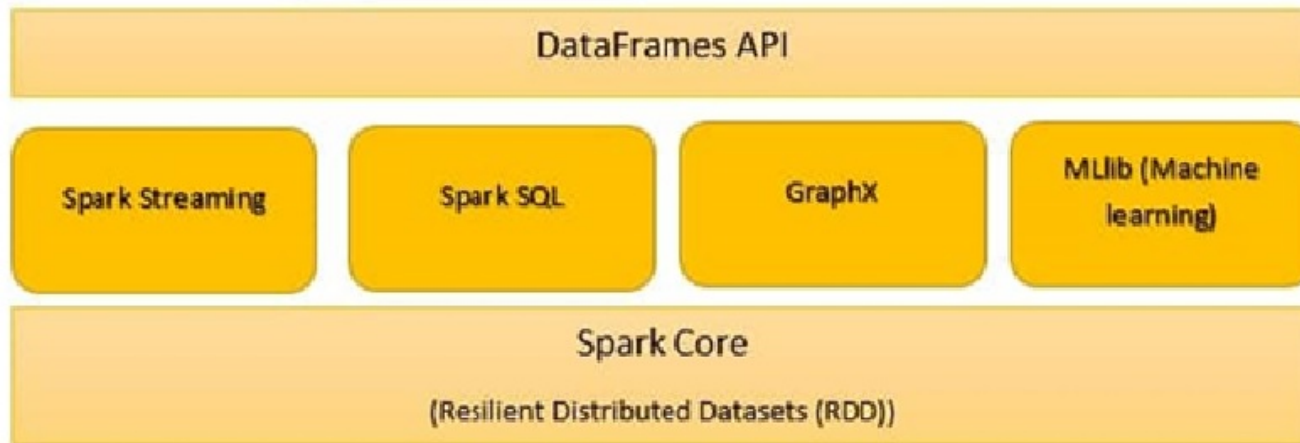
Spark Core

Manages all I/O functionalities, dispatches tasks and handles fault recovery. Data is loaded into Spark core as a special collection called RDD (Resilient Distributed Dataset). RDD handles partitioning of data across all nodes in a cluster. RDD is stored in the memory pool of the cluster as a single node. RDD has two basic operations:

1. **Transformation.** Produce new RDDs from existing RDDs.
2. **Action.** Perform a task on a dataset over an RDD.

PySpark Architecture with MLib

MLib is Apache Spark's ML library for ML for big data



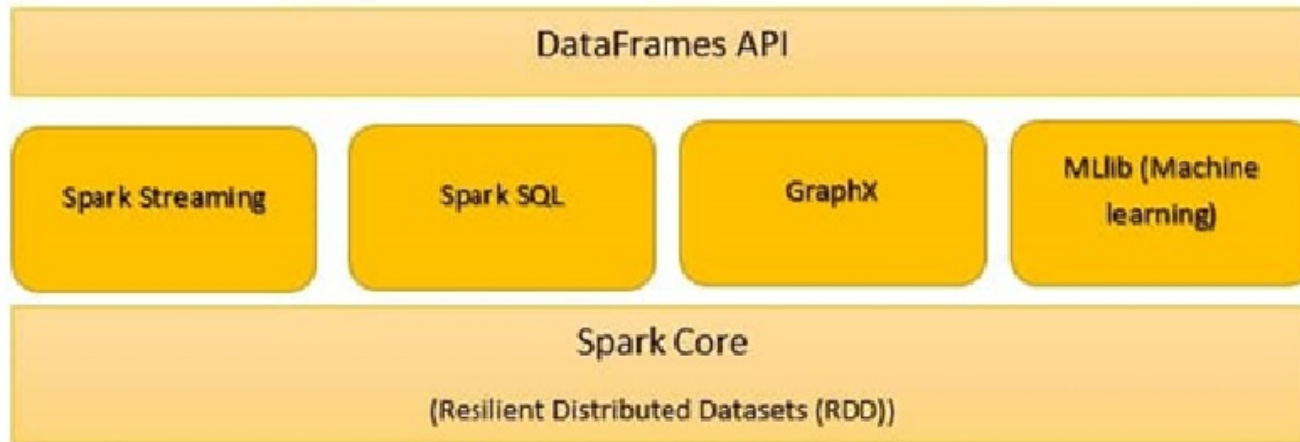
Spark SQL

A distributed framework for structured data processing. It can access structured and semi-structured information. The main features are:

1. **Cost-based Optimizer.** Uses metadata and statistics to estimate the amount of processing (memory, CPU, network traffic, I/O) required for each operation. Compares the cost of each alternative routes and then selects query-execution plan with the least cost.
2. **Mid query fault-tolerance.** Replicates query results to handle unexpected loss.

PySpark Architecture with MLib

MLib is Apache Spark's ML library for ML for big data



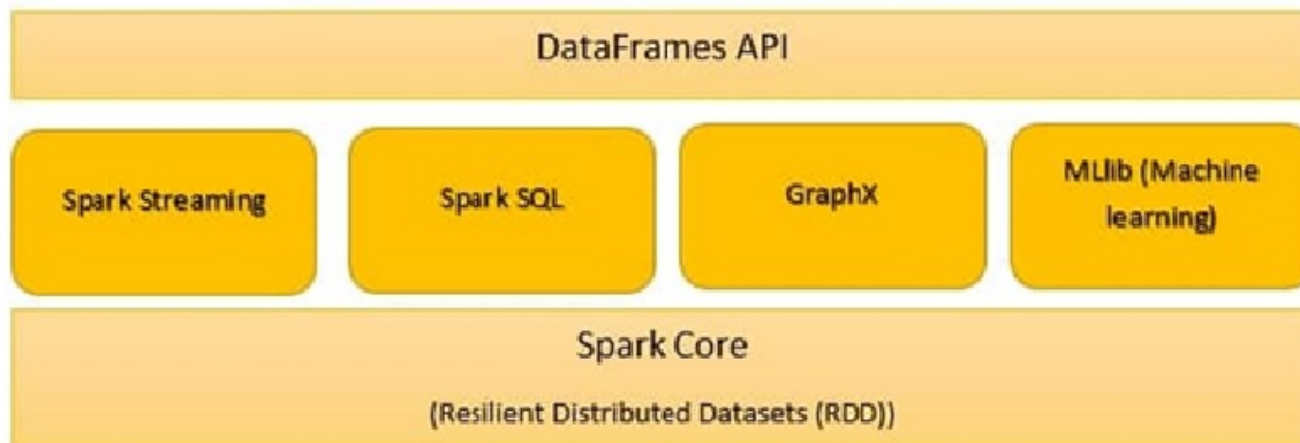
GraphX

An API for graphs and graph parallel execution. It is a network graph analytics engine and data store. Possible operations on the graph.

1. **Clustering.** Extending Mlib clustering
2. **Classification.** Extending Mlib classification
3. **Graph Traversal.**
4. **Graph Searching.**
5. **Graph Pathfinding.**

PySpark Architecture with MLlib

MLib is Apache Spark's ML library for ML for big data



MLlib (Machine Learning Library)

An API to perform machine learning in Apache Spark. Mlib consists of re-implementation of popular ML algorithms with a focus on scalability (high-quality + high-speed) on top of Spark platform. It is built on top of Spark dataframes to support pipelines (discussed later). Algorithm types supported.

1. **Classification.**
2. **Regression.**
3. **Recommendation.**
4. **Clustering.**
5. Others. Topic Modeling and Frequent Itemset mining

Mlib Strengths

1. Ease of Use

- Usable in Java, Scala, Python, and R
- MLlib fits into Spark's APIs and interoperates with NumPy in Python (as of Spark 0.9) and R libraries (as of Spark 1.5). You can use any Hadoop data source (e.g. HDFS, HBase, or local files), making it easy to plug into Hadoop workflows.

2. Performance

- High-quality algorithms, 100x faster than MapReduce
- Spark excels at iterative computation, enabling MLlib to run fast. At the same time, we care about algorithmic performance: MLlib contains high-quality algorithms that leverage iteration and can yield better results than the one-pass approximations sometimes used on MapReduce.

3. Runs everywhere

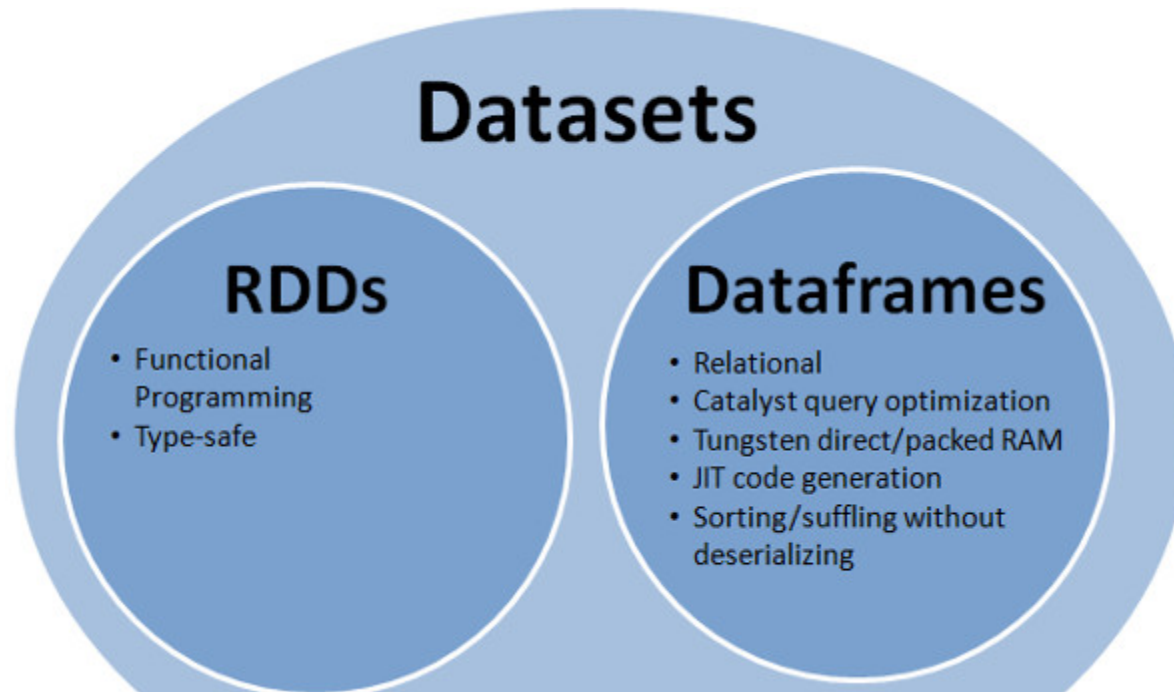
- Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud, against diverse data sources.
- You can run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, on Mesos, or on Kubernetes. Access data in HDFS, Apache Cassandra, Apache HBase, Apache Hive, and hundreds of other data sources.

Data abstractions

Spark now supports three types of data abstractions.

RDD (Spark 1.0) → Dataframe (Spark 1.3) → Dataset (Spark 1.6)

For Pyspark MLib API, dataframes are used so far



RDD vs Dataframe vs Dataset

	RDD	Dataframe	Dataset
Release version	Spark 1.0	Spark 1.3	Spark 1.6
Data representation	Distributed collection of elements.	Distributed collection of data organized into columns.	Combination of RDD and DataFrame.
Data Formats	Structured and unstructured are accepted.	Structured and semi-structured are accepted.	Structured and unstructured are accepted.
Data Sources	Various data sources	Various data sources	Various data sources
Immutability and Interoperability	Immutable partitions that easily transform into DataFrames.	Transforming into a DataFrame loses the original RDD.	The original RDD regenerates after transformation.
Compile-time type safety	Available compile-time type safety.	No compile-time type safety. Errors detect on runtime.	Available compile-time type safety.
Optimization	No built-in optimization engine. Each RDD is optimized individually.	Query optimization through the Catalyst optimizer.	Query optimization through the Catalyst optimizer, like DataFrames.
Serialization	RDD uses Java serialization to encode data and is expensive. Serialization requires sending both the data and structure between nodes.	There is no need for Java serialization and encoding. Serialization happens in memory in binary format.	Encoder handles conversions between JVM objects and tables, which is faster than Java serialization.
Garbage Collection	Creating and destroying individual objects creates garbage collection overhead.	Avoids garbage collection when creating or destroying objects.	No need for garbage collection

RDD vs Dataframe vs Dataset

	RDD	Dataframe	Dataset
Efficiency	Efficiency decreased for serialization of individual objects.	In-memory serialization reduces overhead. Operations performed on serialized data without the need for deserialization.	Access to individual attributes without deserializing the whole object.
Lazy Evaluation	Yes	Yes	Yes
Programming Language Support	Java Scala Python R	Java Scala Python R	Scala
Schema Projection	Schemas need to be defined manually.	Auto-discovery of file schemas.	Auto-discovery of file schemas.
Aggregation	Hard, slow to perform simple aggregations and grouping operations.	Fast for exploratory analysis. Aggregated statistics on large datasets are possible and perform quickly.	Fast aggregation on numerous datasets.

Spark MLib Tools

Tool	Description
ML Algorithms	<ul style="list-style-type: none">• Form the core of Mlib. The algorithms include learning algorithms like classification, regression, clustering, and collaborative filtering.• Mlib standardizes APIs to make it easier to combine multiple algorithms into a pipeline or workflow.• The key concepts are the pipelines API, where the pipeline concept is inspired by the scikit-learn project
Transformer	<ul style="list-style-type: none">• An algorithm that can transform one Dataframe into another Dataframe using the transform() method.• A feature transformer() might take is a Dataframe, read a column (e.g., text), map it into a new column (e.g., feature vectors), and output a new Dataframe with the mapped column appended• A learning model using the transformer might take a Dataframe, read the column containing the feature vectors, predict the label for each feature vector, and outputs a new Dataframe with predicted labels appended as a new column
Estimator	<ul style="list-style-type: none">• An algorithm which can fit on a Dataframe to produce a Transformer.• Technically, an estimator implements a method fit(), which accepts a Dataframe and produces a Model, which is a Transformer. For example, a learning algorithm like LogisticRegression is an Estimator, which can call a fit() method to train a LogisticRegression model.• At this moment, Transformer.transform() and Estimator.fit() are both stateless.• Each unit of a Transformer or Estimator has a unique ID (useful for specifying parameters)

Spark MLib Tools

Tool	Description
Featurization	<ul style="list-style-type: none">• Feature extraction modules allow the extraction of features from raw data• Feature transformation includes scaling, renovating, or modifying features• Feature selection involves selecting a subset of necessary features from a huge set of features
Pipeline	<ul style="list-style-type: none">• A pipeline chains multiple Transformers and Estimators together to specify an ML workflow.• A pipeline also provides tools for constructing, evaluating, and tuning ML pipelines
Persistence	<ul style="list-style-type: none">• Helps in saving and loading algorithms, models, and Pipelines.• Helps in reducing time and efforts as model is saved, it can be loaded/reused anytime
Utility	<ul style="list-style-type: none">• Any utility functions/methods that could be necessary for data exploration or transformation• Example utilities are linear algebra, statistical analysis (e.g., correlation analysis), and data cleaning• Mlib.linalg has utilities for linear algebra

MLib Pipelines

Reference Materials used in this section.

1. Spark Mlib official documentation

<https://spark.apache.org/docs/latest/ml-pipeline.html>

Main Concepts

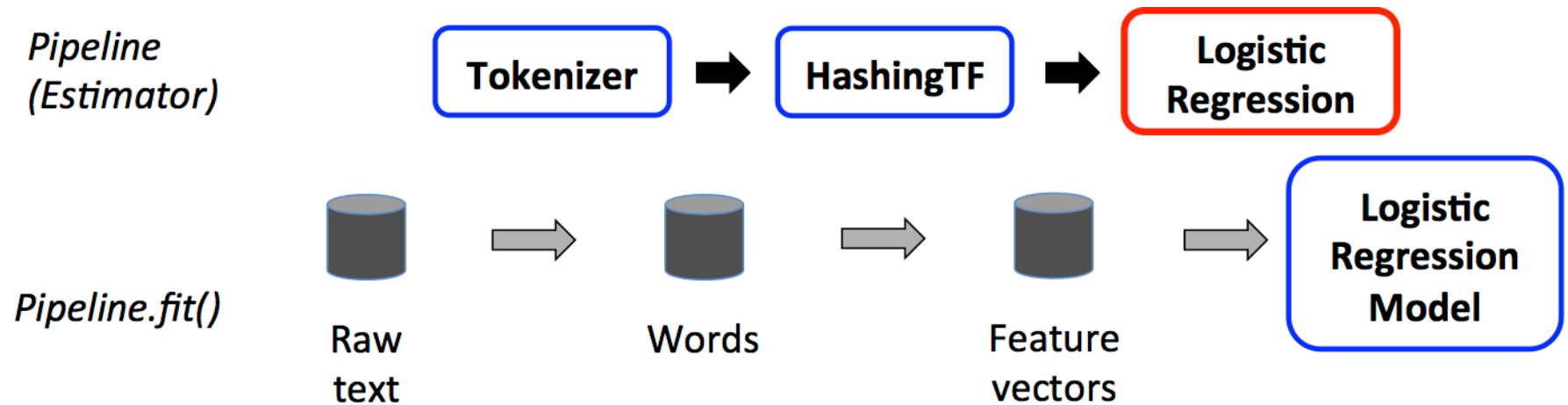
Concept/Tool	Description
DataFrame	<ul style="list-style-type: none">• Uses DataFrame from Spark SQL as an ML dataset, which can hold a variety of data types.• A DataFrame can have different columns storing text, feature vectors, true labels, and predictions• DataFrame supports many basic and structured types like numeric, string, binary, etc. See https://spark.apache.org/docs/2.4.4/sql-reference.html for details on data types supported.• Columns in a DataFrame are named• A DataFrame can be created either explicitly by reading an input file or implicitly from a regular RDD.
Transformer	<ul style="list-style-type: none">• A Transformer is an algorithm which can transform one DataFrame into another DataFrame. E.g., an ML model is a Transformer which transforms a DataFrame with features into a DataFrame with predictions.
Estimator	<ul style="list-style-type: none">• An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer. E.g., a learning algorithm is an Estimator which trains on a DataFrame and produces a model.
Parameter	<ul style="list-style-type: none">• Mlib Estimators and Transformers use a uniform API for specifying parameters• A Param is a named parameter with self-contained documentation. A ParamMap is a set of (parameter, values) pairs• Two ways to pass parameters to an MLib algorithm<ul style="list-style-type: none">• Set parameter for an instance, e.g., if lr is an instance of LogisticRegression then we call lr.maxIter(10) to make lr.fit() use at most 10 iterations.• Pass a ParamMap to fit() or transform(). Any parameters in the ParamMap will override previously set parameter values.• Parameters belong to specific instances of Estimators and Transformers. For example, if we have two LogisticRegression instances lr1 and lr2, then we can build a ParamMap with both maxIter parameters specified: ParamMap(lr1.maxIter -> 10, lr2.maxIter -> 20). This is useful if there are two algorithms with the maxIter parameter in a Pipeline.
Pipeline	<ul style="list-style-type: none">• A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow.
Persistence	<ul style="list-style-type: none">• Can be used to save an ML model as well as an entire pipeline

How a Pipeline Works

A Pipeline is simply a sequence of tasks, each corresponding to a stage (e.g, tokenization, feature vector generation, training, etc.). This sequence of tasks can be done for any record like a record in the training set as well as a record in the testing set. Thus, a pipeline , when developed, can be applied for any record for a given dataset (if specified properly).

- A Pipeline is specified as a sequence of stages, and each stage is either a Transformer or an Estimator.
- These stages are run in order, and the input DataFrame is transformed as it passes through each stage.
- For Transformer stages, the transform() method is called on the DataFrame.
- For Estimator stages, the fit() method is called to produce a Transformer (which becomes part of the PipelineModel, or fitted Pipeline), and that Transformer's transform() method is called on the DataFrame.

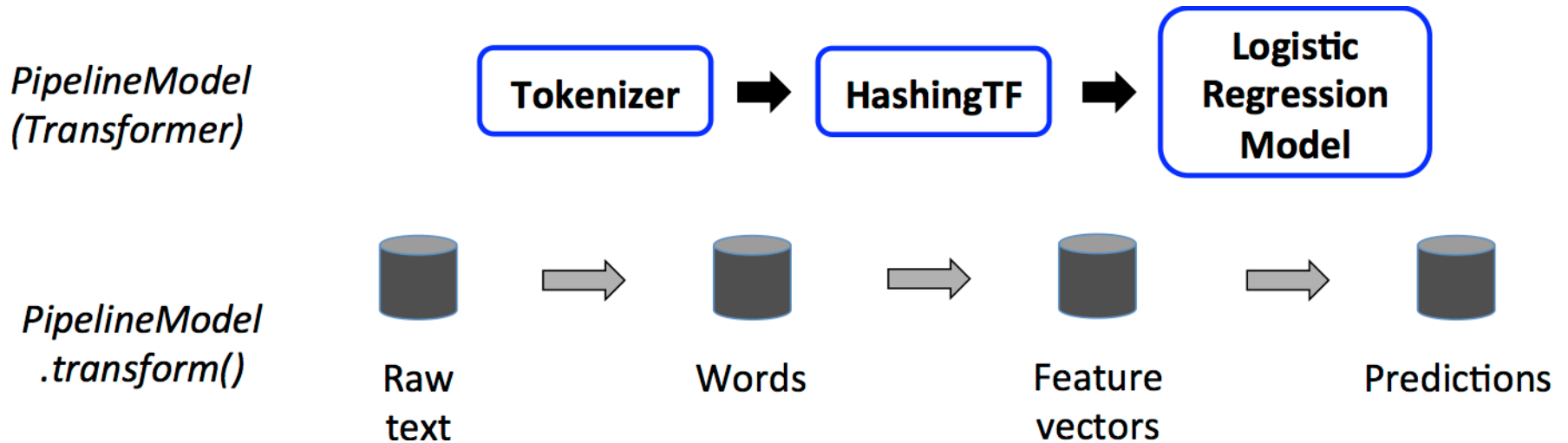
How a Pipeline Works



Above, the top row represents a Pipeline with three stages. The first two (Tokenizer and HashingTF) are Transformers (blue), and the third (LogisticRegression) is an Estimator (red). The bottom row represents data flowing through the pipeline, where cylinders indicate DataFrames.

- The `Pipeline.fit()` method is called on the original DataFrame, which has raw text documents and labels.
- The `Tokenizer.transform()` method splits the raw text documents into words, adding a new column with words to the DataFrame.
- The `HashingTF.transform()` method converts the words column into feature vectors, adding a new column with those vectors to the DataFrame.
- Now, since LogisticRegression is an Estimator, the Pipeline first calls `LogisticRegression.fit()` to produce a `LogisticRegressionModel`. If the Pipeline had more Estimators, it would call the `LogisticRegressionModel`'s `transform()` method on the DataFrame before passing the DataFrame to the next stage.

How a Pipeline Works



A Pipeline is an Estimator. Thus, after a Pipeline's `fit()` method runs, it produces a `PipelineModel`, which is a Transformer. This `PipelineModel` is used at test time; the figure above illustrates this usage.

The `PipelineModel` has the same number of stages as the original Pipeline, but all Estimators in the original Pipeline have become Transformers. When the `PipelineModel`'s `transform()` method is called on a test dataset, the data are passed through the fitted pipeline in order. Each stage's `transform()` method updates the dataset and passes it to the next stage.

Pipelines and `PipelineModels` help to ensure that training and test data go through identical feature processing steps.

Mlib Pipeline Demo

By running two code examples from
<https://spark.apache.org/docs/latest/ml-pipeline.html>

MLib Featurization

Reference Materials used in this section.

1. Spark Mlib official documentation

<https://spark.apache.org/docs/latest/ml-features.html>

Algorithm Types

Type	Description
Feature Extraction	Extract features from raw data (e.g., create vectors by counting words found in texts) Some Methods: CountVectorizer, TF-IDF, Word2Vec, FeatureHasher
Feature Transformers	Scaling, converting, or modifying features Some Methods: Tokenizer, StopWordsRemover, n-gram, PCA, StandardScaler, etc.
Feature Selectors	Selection of a subset from a larger set of features Some Methods: VectorSlicer, Rformula, UnivariateFeatureSelector, etc.
Localist Sensitive hashing (LSH)	Combines aspects of feature transformation with other algorithms. Some Methods: MinHash for Jaccard Distance, Bucketed Random Projection for Euclidean Distance

CountVectorizer

CountVectorizer takes as input a list of words/items and produces a vectorized representation of the words/items as an output. The algorithm can filter based on frequency, like only show words that are found more than n times in the input.

Attributes	Description
minDF	Specifies the minimum number of different documents a term must appear in to be included in the vocabulary. If this is an integer ≥ 1 , this specifies the number of documents the term must appear in; if this is a double in $[0,1)$, then this specifies the fraction of documents. Default 1.0
maxDF	Specifies the maximum number of different documents a term could appear in to be included in the vocabulary. A term that appears more than the threshold will be ignored. If this is an integer ≥ 1 , this specifies the maximum number of documents the term could appear in; if this is a double in $[0,1)$, then this specifies the maximum fraction of documents the term could appear in. Default $(2^{63}) - 1$
minTF	Filter to ignore rare words in a document. For each document, terms with frequency/count less than the given threshold are ignored. If this is an integer ≥ 1 , then this specifies a count (of times the term must appear in the document); if this is a double in $[0,1)$, then this specifies a fraction (out of the document's token count). Note that the parameter is only used in transform of CountVectorizerModel and does not affect fitting. Default 1.0
vocabSize	max size of the vocabulary. Default $1 \ll 18$

Demo and Explanation of Output:

By running a code example from <https://spark.apache.org/docs/latest/ml-features.html#countvectorizer>

TF-IDF

TF = Term Frequency, IDF = Inverse Document Frequency

TF-IDF is a feature vectorization method used in text mining to reflect the importance of a term “t” to a document “d” in a corpus of multiple documents.

$TF(t, d)$ = number of times term t appears in document d

$DF(t, D)$ = number of documents in D that contain term t

$$IDF(t, D) = \log \frac{|D| + 1}{DF(t, D) + 1}$$

$$TF - IDF(t, d, D) = TF(t, d) \times IDF(t, D)$$

Here $|D|$ is the total number of documents in the corpus. Since logarithm is used, if a term appears in all documents, its IDF value will be zero – that means the term is not very important to distinguish among the documents.

TF: Both HashingTF and CountVectorizer can be used to generate

IDF is an Estimator which can be fit on a dataset that produces an IDFModel.

Demo and Explanation of Output:

By running a code example from <https://spark.apache.org/docs/latest/ml-features.html>

Model Selection and Hyperparameter Tuning

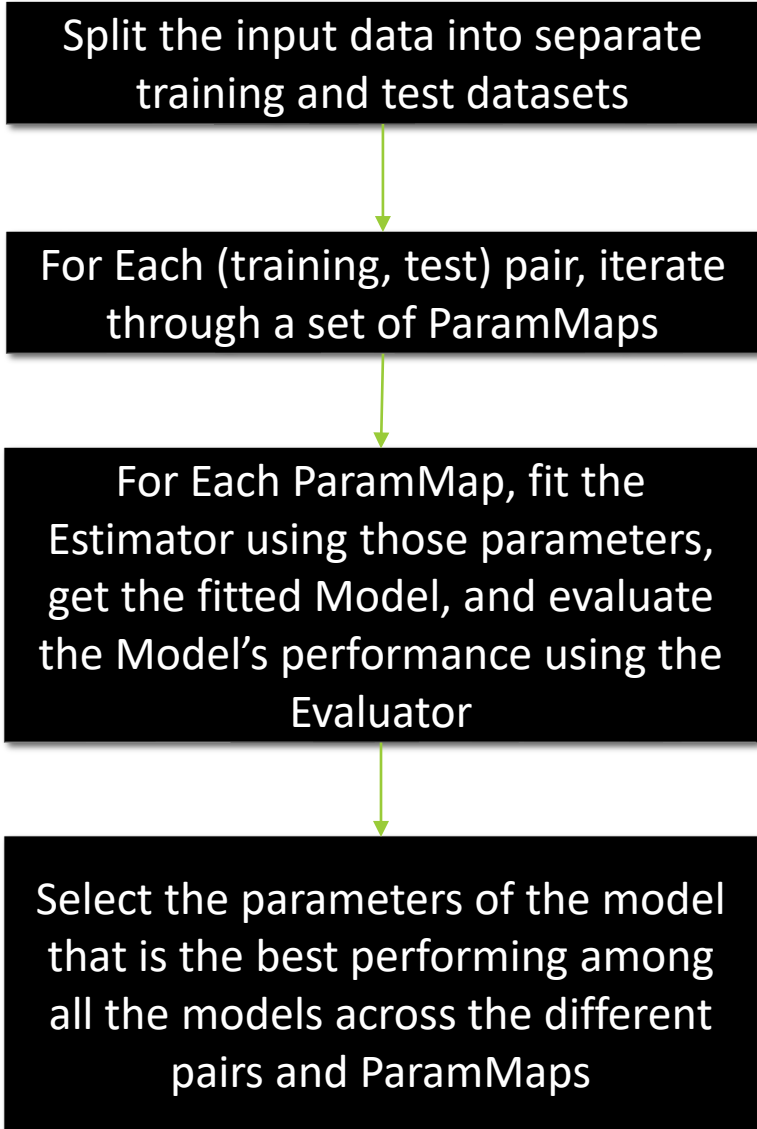
Reference Materials used in this section.

1. Spark Mlib official documentation

<https://spark.apache.org/docs/latest/ml-tuning.html>

Major Steps

Split the input data into separate training and test datasets



For Each (training, test) pair, iterate through a set of ParamMaps

For Each ParamMap, fit the Estimator using those parameters, get the fitted Model, and evaluate the Model's performance using the Evaluator

Select the parameters of the model that is the best performing among all the models across the different pairs and ParamMaps

- **Estimator:** algorithm or Pipeline
- **Set of ParamMaps:** parameters to choose from, also called a “parameter grid”
- **Evaluator:** metric to measure how well a fitted model does on held-out test data

Two types of tuning supported in Spark Mlib:

1. Cross-Validation:

Types

Type	Description
Cross-Validation	<ul style="list-style-type: none">• CrossValidator begins by splitting the dataset into a set of <i>folds</i> which are used as separate training and test datasets (e.g., $k = 3$ will denote 3 folds)• To evaluate a particular ParamMap, CrossValidator computes the average evaluation metric for the 3 Models (if $k = 3$) produced by fitting the Estimator on the 3 different (training, test) dataset pairs.• After identifying the best ParamMap, CrossValidator finally re-fits the Estimator using the best ParamMap and the entire dataset.
Train-Validation	<ul style="list-style-type: none">• Unlike CrossValidator, TrainValidationSplit creates a single (training, test) dataset pair. It splits the dataset into these two parts using the trainRatio parameter. For example with <i>trainRatio=0.75</i>• TrainValidationSplit only evaluates each combination of parameters once, as opposed to k times in the case of CrossValidator. It is, therefore, less expensive, but will not produce as reliable results when the training dataset is not sufficiently large.• TrainValidationSplit will generate a training and test dataset pair where 75% of the data is used for training and 25% for validation. Like CrossValidator, TrainValidationSplit finally fits the Estimator using the best ParamMap and the entire dataset.

Demo and Explanation of Output:

By running a code example from <https://spark.apache.org/docs/latest/ml-tuning.html>