Introduction to SparkSQL Structured Data Processing in Spark

CELEBRATING 30 YEARS
Marlan and Rosemary Bourns College of Engineering

Structured Data Processing

- A common use case in big-data is to process structured or semi-structured data
- In Spark RDD, all functions and objects are black-boxes.
- Any structure of the data has to be part of the functions which includes:
  - Parsing
  - Conversion
  - Processing

Structured data processing

- Pig/Pig Latin
  - Builds on Hadoop
  - Converts SQL-like programs to MapReduce
- Hive/HiveQL
  - Supports SQL-like queries
- Shark (Hive on Spark)
  - Translates HiveQL queries to RDD programs
  - Initial attempt to support SQL on Spark

SparkSQL

- Redesigned to consider Spark query model
- Supports all the popular relational operators
- Can be intermixed with RDD operations
- Uses the Dataframe API as an enhancement to the RDD API

Dataframe = RDD + schema

Built-in operations in SprkSQL

- Filter (Selection)
- Select (Projection)
- Join
- GroupBy (Aggregation)
- Load/Store in various formats
- Cache
- Conversion between RDD (back and forth)

SparkSQL Examples

Project Setup

```
<!--
https://mvnrepository.com/artifact/org.apache.spark/spark
-sql -->
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.12</artifactId>
  <version>2.4.5</version>
</dependency>
```

Code Setup

```
SparkSession sparkS = SparkSession
        .builder()
        .appName("Spark SQL examples")
        .master("local")
        .getOrCreate();

Dataset<Row> log_file = sparkS.read()
        .option("delimiter", "\t")
        .option("header", "true")
        .option("inferSchema", "true")
        .csv("nasa_log.tsv");
log_file.show();
```

Filter Example

```
// Select OK lines
Dataset<Row> ok_lines =
log_file.filter("response=200");
long ok_count = ok_lines.count();
System.out.println("Number of OK lines is
"+ok_count);


// Grouped aggregation using SQL
Dataset<Row> bytesPerCode =
log_file.sqlContext().sql("SELECT response,
sum(bytes) from log_lines GROUP BY response");
```

Join Example (Scala)

```scala
// For a specific time, count the number of requests
before and after that time for each response code
val filterTimestamp: Long = …
val countsBefore = input
    .filter($"time" < filterTimestamp)
    .groupBy($"response")
    .count
    .withColumnRenamed("count", "count_before")
val countsAfter = input
    .filter($"time" >= filterTimestamp)
    .groupBy($"response")
    .count
    .withColumnRenamed("count", "count_after")
val comparedResults = countsBefore
    .join(countsAfter, "response")
```

Integration

- SparkSQL is integrated with other high-level interfaces such as MLlib, PySpark, and SparkR
- SparkSQL is also integrated with the RDD interface and they can be mixed in one program

Further Reading

- Documentation
  - http://spark.apache.org/docs/latest/sql-programming-guide.html
- SparkSQL paper
  - M. Armbrust *et al*. "Spark sql: Relational data processing in spark." SIGMOD 2015

Introduction to MLlib: Machine learning in Spark

Machine Learning Algorithms

- Supervised learning
  - Given a set of features and labels
  - Builds a model that predicts the label from the features
  - E.g., classification and regression
- Unsupervised learning
  - Given a set of features without labels
  - Finds interesting patterns or underlying structure
  - E.g., clustering and association mining

Overview of MLlib

- Simple primitives
- Basic Statistics
- Extractors, transformations
- Estimators
- Evaluators
- Model tuning

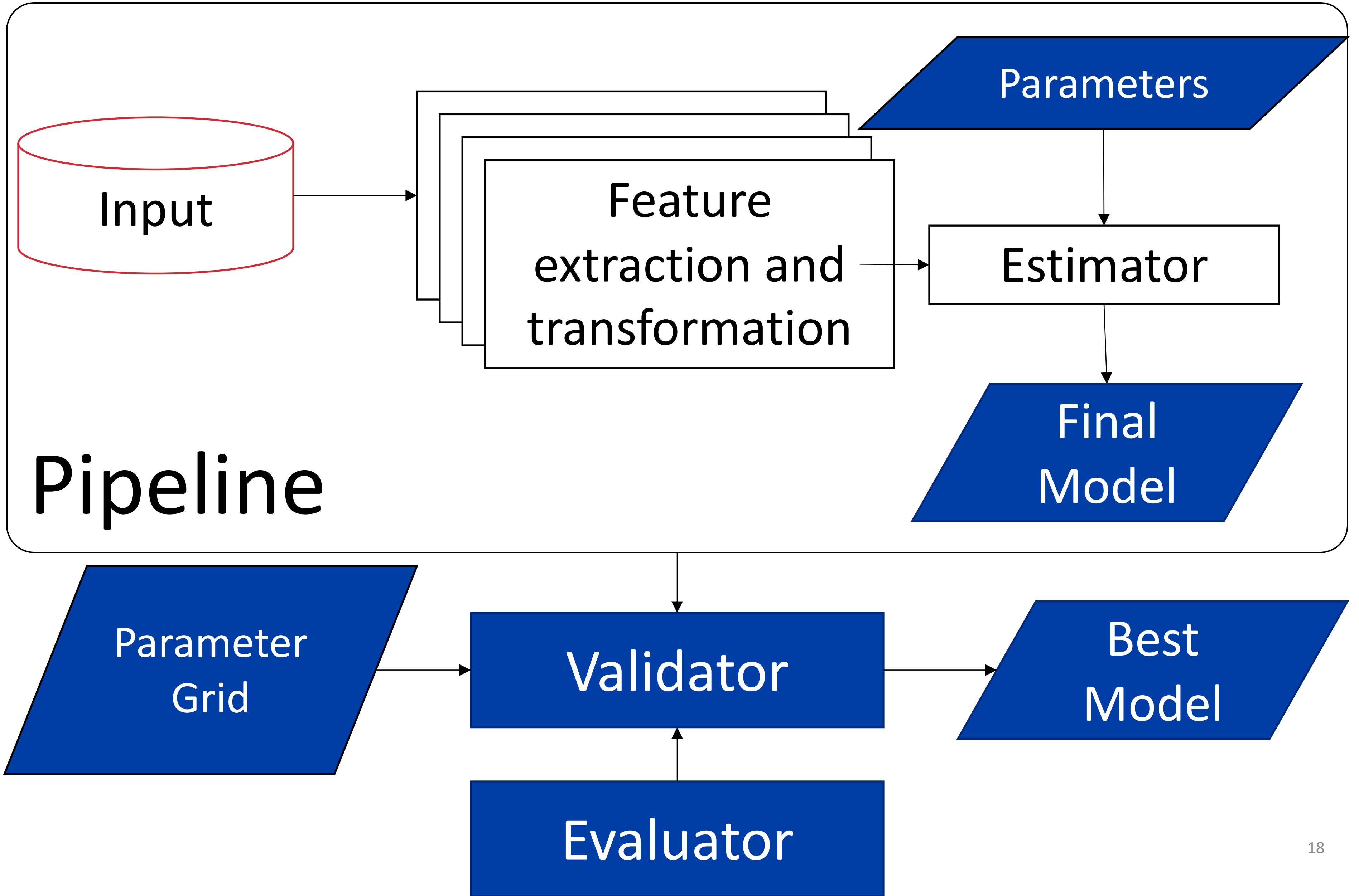
Simple Primitives

- Local Vector (Data Type)
  - To represent features
  - Example: (1.2, 0.0, 0.0, 3.4)
  - Dense vector [1.2, 0.0, 0.0, 3.4]
  - Sparse vector [0, 3], [1.2, 3.4]
- Local Matrix (Data Type)
  - Dense and Sparse
- Dataframe.randomSplit
  - Randomly splits an input dataset
  - Helps in building training and test sets

Basic Statistics

- Column statistics
  - Minimum, Maximum, count, … etc.
- Correlation
  - Pearson's and Spearman's correlation
- Hypothesis testing
  - Chi-square Test χ^2

ML Pipeline



Parameters
Input
Feature extraction and transformation
Estimator
Final Model
Pipeline
Parameter Grid
Validator
Best Model
Evaluator

Transformations

- Used in feature extraction, dimensionality reduction, or schema transformation
- Text transformations
- Encoding
- Normalization
- Hashing

TF-IDF

- Term Frequency-Inverse Document Frequency
- A measure of the importance of a term in a document
- TF: Count of a term in a document
- DF: Number of documents that contain a term
- $IDF(t, D) = \log \frac{|D|+1}{DF(t,D)+1}$
- $TFIDF(t, D) = TF(t, d) \cdot IDF(t, D)$
- Classes: HashingTF, CountVectorizer

Word2Vec

- Converts each sequence of words to a fixed-size vector
- Similar sequences of words are supposed to be mapped to nearby vectors using this model

Numeric Transformers

- Binarizer: Converts numerical values to (0/1) based on a threshold
- Bucketizer: Converts continuous values to a set of n+1 buckets based on n thresholds
- QuantileDiscretizer: Places numeric values into buckets based on quantiles
- Normalizer: normalizes each vector to have unit norm. For example,

$$[4.0 \quad 10.0 \quad 2.0] \rightarrow [0.25 \quad 0.625 \quad 0.125]$$

- MinMaxScaler: Scales each feature in a vector to a standard scale, e.g., [0.0, 1.0]

Applying Transformers

- Simple transformers
  - Can be applied by looking at each individual record
  - E.g., `Bucketizer`, or `VectorAssembler`
  - Applied by calling the `transform` method
  - E.g., `outdf = model.transform(indf)`
- Holistic transformers
  - Need to see the entire dataset first before they can work
  - e.g., `MinMaxScaler`, `HashingTF`, `StringIndexer`
  - To apply them, you need to call `fit` then `transform`
  - e.g., `outdf = model.fit(indf).transform(indf)`

Estimators

- An estimator is a machine learning algorithm that fits a model on the data
- Classification
  - Classifies data points into discrete points (categories)
- Regression
  - Estimates a continuous numeric
- Clustering
  - Groups similar records together into clusters
- Collaborative filtering (Recommendation)
  - Predicts (missing) user ratings for items
- Frequent Pattern Mining

Classification and regression

- Supervised learning algorithms
- Classification
  - Logistic regression
  - Decision tree
  - Naïve Bayes
  - …
- Regression
  - Linear regression
  - Decision tree regression
  - Random forest regression
  - …

Clustering

- Unsupervised learning method
- K-means clustering. Clustering based on distance between vectors
- Latent Dirichlet allocation (LDA). Groups vectors based on some latent (hidden) variables
- Bisecting k-means. Hierarchical clustering
- Gaussian Mixture Model (GMM). Breaks down data distribution into multiple Gaussian distributions

Evaluators

- An Evaluator takes a model and produces numeric values that measure the goodness of the model for a specific dataset
- BinaryClassificationEvaluator evaluates binary classifiers using precision, recall, F-measure, area under ROC curve, … etc.
- MulticlassClassificationEvaluator evaluates multiclass classifiers using confusion matrix, accuracy, precision, recall … etc.

Evaluators

- ClusteringEvaluator evaluates clustering algorithms using sum of squared distances
- RegressionEvaluator evaluates regression models using Mean Squared Error (MSE), Root Mean Squared Error (RMSE) … etc.

Validators

- Each model has its own parameters that are usually no intuitive to tune
- A validator takes a pipeline, an evaluator, and a set of parameters and it tries all possible combinations of parameters to find the best model, i.e., the model that gives the best numeric evaluation metric
- Examples, CrossValidator and TrainValidationSplit

Code Example

Input Data

House ID	Bedrooms	Area (sqft)	…	Price
1	2	1,200		$200,000
2	3	3,200		$350,000
…				

- Goal: Build a model that estimates the price given the house features, e.g., # of bedrooms and area

Initialization

- Similar to SparkSQL

```
val spark = SparkSession
  .builder()
  .appName(”SparkSQL Demo")
  .config(conf)
  .getOrCreate()

// Read the input
val input = spark.read
  .option("header", true)
  .option("inferSchema", true)
  .csv(inputfile)
```

Transformations

```
// Create a feature vector
val vectorAssembler = new VectorAssembler()
  .setInputCols(Array("bedrooms", "area"))
  .setOutputCol("features")

val linearRegression = new LinearRegression()
  .setFeaturesCol("features")
  .setLabelCol("price")
  .setMaxIter(1000)
```

Create a Pipeline

```
val pipeline = new Pipeline()
  .setStages(Array(vectorAssembler, linearRegression))

// Hyper parameter tuning
val paramGrid = new ParamGridBuilder()
  .addGrid(linearRegression.regParam,
      Array(0.3, 0.1, 0.01))
  .addGrid(linearRegression.elasticNetParam,
      Array(0.0, 0.3, 0.8, 1.0))
  .build()
```

Cross Validation

```
val crossValidator = new CrossValidator()
  .setEstimator(pipeline)
  .setEvaluator(new
RegressionEvaluator().setLabelCol("price"))
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(5)
  .setParallelism(2)

val Array(trainingData, testData) =
input.randomSplit(Array(0.8, 0.2))
val model = crossValidator.fit(trainingData)
```

Apply the model on test data

```scala
val predictions = model.transform(testData)
// Print the first few predictions
predictions.select("price", "prediction").show(5)

val rmse = new RegressionEvaluator()
  .setLabelCol("price")
  .setPredictionCol("prediction")
  .setMetricName("rmse")
  .evaluate(predictions)
println(s"RMSE on test set is $rmse")
```

Further Reading

- Documentation
  - http://spark.apache.org/docs/latest/ml-guide.html
- MLlib paper
  - X. Meng et al, “MLlib: Machine Learning in Apache Spark”, Journal of Machine Learning Research 17:34:1-34:7 (2016)