Scalable Data Science
in
R and Apache Spark 2.0

Felix Cheung, Principal Engineer, Spark @ Microsoft
About me

- Apache Spark Committer
- Apache Zeppelin PMC/Committer
- Contributing to Spark since 1.3 and Zeppelin at the beginning of incubation
- Co-organizer of the Seattle Spark Meetup
- Was a teaching assistant of edx MOOCs, Summer 2015:
  - Introduction to Big Data with Apache Spark
  - Scalable Machine Learning
- Presented at Meetups and ApacheCon: Big Data North America
Agenda

• Spark + R
• Architecture
• What’s new in 2.0
• Options for Data Science
• Demo
• Ecosystem
• What’s coming in 2.1+
Disclaimer:
Apache Spark community contributions
1,000 contributors
Spark in 5 seconds

- Spark SQL + DataFrame/Dataset + data sources
- Streaming/Structured Streaming
- ML
- GraphX
• A programming language for statistical computing and graphics
• S - 1975
  • S4 - advanced object-oriented features
• R - 1993
  • S + lexical scoping
• Interpreted
• Matrix arithmetic
• Comprehensive R Archive Network (CRAN) - ~10k packages
Fast!
Scalable
Flexible
Statistical!
Interactive
Packages
SparkR

• R language APIs for Spark and Spark SQL
• Exposes Spark functionality in an R-friendly DataFrame API
• Runs as its own REPL `sparkR`
• or as a standard R package imported in tools like RStudio

```r
library(SparkR)
sparkR.session()
```
Brief history

• Shivaram Venkataraman & Zongheng Yang
• RDD APIs in a standalone package (Jan/2014)
• Spark SQL and SchemaRDD -> DataFrame
• 1.4.0 - first Spark release with SparkR
  ...
• 1.6.0 - ML - glm
• 2.0.0 - Spark Session APIs
Architecture

• Native R S4 classes and methods
• RBackend
• Scala “helper” methods (ML pipeline etc.)
Advantages

- R-like syntax for DataFrame APIs
- JVM processing with full access to Spark’s DAG capabilities and Catalyst engine, e.g. execution plan optimization, predicate pushdown, code generation
Advantages

• SQL
• Data source (JSON, csv, PostgreSQL)
• Catalog (external data table management)
• Spark packages
• ML
• R-native UDF (distributed processing)
• Cluster support (YARN, mesos, standalone)
• SparkSession & default session (streamlined parameter)
  as.DataFrame(iris)

What’s new in SparkR
Decisions, decisions?

- **Scalable?**
  - No: Native R UDF
  - Yes: Spark.ml
Spark.ml ML Pipeline

• Pre-processing, feature extraction, model fitting, validation stages
• Transformer
• Estimator
• Cross-validation/hyperparameter tuning
RFormula

- Specify modeling in symbolic form
  \[ y \sim f0 + f1 \]
  response \( y \) is modeled linearly by \( f0 \) and \( f1 \)
- Support a subset of R formula operators
  ‘~’, ‘.’, ‘:’, ‘+’, ‘-’
- Implemented as feature transformer in core Spark, available to Scala/Java, Python
- String label column is indexed
- String term columns are one-hot encoded
Model helpers

- **summary** - print a summary of the fitted model
- **predict** - make predictions on new data
- **write.ml/read.ml** - save/load fitted models (pipeline model plus R metadata)
Spark.ml R in Spark 2.0.0

- Generalized Linear Model (GLM)
- Naive Bayes Model
- k-means Clustering
- Accelerated Failure Time (AFT) Survival Model
Spark.ml R in Spark 2.1.0

• Generalized Linear Model (GLM)
• Naive Bayes Model
• k-means Clustering
• Accelerated Failure Time (AFT) Survival Model
• Isotonic Regression Model
• Gaussian Mixture Model (GMM)
• Latent Dirichlet Allocation (LDA)
• Alternating Least Squares (ALS)
• Multilayer Perceptron Model
• Kolmogorov-Smirnov Test (K-S test)
• Multiclass Logistic Regression
• Random Forest
• Gradient Boosted Tree
Generalized Linear Model

# R-like

```r
glm(Sepal_Length ~ Sepal_Width + Species, gaussianDF, family = "gaussian")
```

```r
spark.glm(binomialDF, Species ~ Sepal_Length + Sepal_Width, family = "binomial")
```

- “binomial” output string label, prediction
Naive Bayes

```
spark.naiveBayes(nbDF, Survived ~ Class + Sex + Age)
```

- index label, predicted label to string
k-means

spark.kmeans(kmeansDF, ~ Sepal_Length + Sepal_Width + Petal_Length + Petal_Width, k = 3)
Accelerated Failure Time (AFT) Survival Model

\texttt{spark.survreg(aftDF, Surv(futime, fustat) ~ ecog_ps + rx)}

• formula rewrite for censor
Isotonic Regression

spark.isoreg(df, label ~ feature, isotonic = FALSE)
Gaussian Mixture Model

spark.gaussianMixture(df, ~ V1 + V2, k = 2)
Latent Dirichlet Allocation

```
spark.lda(data = text, k = 20, maxIter = 25, optimizer = "em")
```

- Regex tokenizer, stopwords, count vectorize
Alternating Least Squares

spark.als(df, "rating", "user", "item", rank = 20, reg = 0.1, maxIter = 10, nonnegative = TRUE)
Multilayer Perceptron Model

```
spark.mlp(df, blockSize = 128, layers = c(4, 3), solver = "l-bfgs", maxIter = 100, tol = 0.5, stepSize = 1)
```
Kolmogorov-Smirnov Test

```
spark.ktest(df, "test", "norm", c(0, 1))
```
Multiclass Logistic Regression

```
spark.logit(df, label ~ ., regParam = 0.3, elasticNetParam = 0.8, family = "multinomial", thresholds = c(0, 1, 1))
```

• binary or multiclass
Random Forest

```
spark.randomForest(df, Employed ~ ., type = "regression", maxDepth = 5, maxBins = 16)

spark.randomForest(df, Species ~ Petal_Length + Petal_Width, "classification", numTree = 30)

• “classification” index label, predicted label to string
```
Gradient Boosted Tree

spark.gbt(df, Employed ~ ., type = "regression", maxDepth = 5, maxBins = 16)

spark.gbt(df, IndexedSpecies ~ ., type = "classification", stepSize = 0.1)

• “classification” index label, predicted label to string
• Binary classification
Modeling parameters

spark.randomForest

function(data, formula, type = c("regression", "classification"),
    maxDepth = 5, maxBins = 32, numTrees = 20, impurity = NULL,
    featureSubsetStrategy = "auto", seed = NULL, subsamplingRate = 1.0,
    minInstancesPerNode = 1, minInfoGain = 0.0, checkpointInterval = 10,
    maxMemoryInMB = 256, cacheNodeIds = FALSE, probabilityCol = "probability"
Spark.ml challenges

• Converting spark.ml API to R API
  • Pipeline flexibility
• Catching up to changes
  • Almost all spark.ml models (except One vs Rest)
Native-R UDF

- User-Defined Functions - custom transformation
- Apply by Partition
- Apply by Group
UDF: Apply by Partition

• Function to process each partition of a DataFrame
• Mapping of Spark/R data types

dapply(carsSubDF,
    function(x) {
      x <- cbind(x, x$mpg * 1.61)
    },
    schema)
UDF: Apply by Partition + Collect

• No schema

```r
out <- dapplyCollect(
carsSubDF,
function(x) {
    x <- cbind(x, "kmpg" = x$mpg * 1.61)
})
```
UDF: Apply by Group

• By grouping columns
• Mapping of Spark/R data types

gapply(carsDF, "cyl",
    function(key, x) {
      y <- data.frame(key, max(x$mpg))
    },
    schema)
UDF: Apply by Group + Collect

• No Schema

out <- gapplyCollect(carsDF, "cyl", function(key, x) {
    y <- data.frame(key, max(x$mpg))
    names(y) <- c("cyl", "max_mpg")
    y
})
# UDF: data type mapping

* not a complete list

<table>
<thead>
<tr>
<th>R</th>
<th>Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>byte</td>
</tr>
<tr>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>float</td>
<td>float</td>
</tr>
<tr>
<td>double, numeric</td>
<td>double</td>
</tr>
<tr>
<td>character, string</td>
<td>string</td>
</tr>
<tr>
<td>binary, raw</td>
<td>binary</td>
</tr>
<tr>
<td>logical</td>
<td>boolean</td>
</tr>
<tr>
<td>POSIXct, POSIXlt</td>
<td>timestamp</td>
</tr>
<tr>
<td>Date</td>
<td>date</td>
</tr>
<tr>
<td>array, list</td>
<td>array</td>
</tr>
<tr>
<td>env</td>
<td>map</td>
</tr>
</tbody>
</table>
UDF: challenges

• “struct”
• scale - what if partition or group too big to fit executor?
• performance - serialization/deserialization, data transfer costs
UDF: lapply

• Like R `lapply` or `doParallel`
• Good for “embarrassingly parallel” tasks
• Such as hyperparameter tuning
UDF: parallel distributed processing

- Output is a list - needs to fit in memory at the driver

```r
costs <- exp(seq(from = log(1), to = log(1000), length.out = 5))
train <- function(cost) {
  model <- e1071::svm(Species ~ ., iris, cost = cost)
  summary(model)
}
summaries <- spark.lapply(costs, train)
```
Remember this?

Scalable?

No → Native R UDF

Yes → Spark.ml
Demo!
Other stuff
SparkR as a Package (target Spark 2.1.0)

• Goal: install SparkR/Spark from CRAN
• Spark JVM installs automatically (`install.spark()` in Spark 2.0.0)
• R vignettes
• Community can write package that depends on SparkR package
• Advanced Spark JVM interop APIs

  `sparkR.newJObject`
  `sparkR.callJMethod`
  `sparkR.callJStatic`
Ecosystem

- RStudio **sparklyr**
- **RevoScaleR**/RxSpark, R Server
- **H2O R**
- IBM BigInsights Big R (not Spark!)
Recap: What’s coming with Spark 2.1.0

• More, richer ML
• SparkR as a Package

Next - Spark 2.2.0!