

Face Recognition Model Using R in Spark Cluster

Zakaria Dahani, Yves Zango, Naima Touloum

MSc Big Data



**Programming with Big Data in
R using Distributed or Shared
Memory**

Outline

- Introduction
- Infrastructure - Cluster Deploiement
- Face recognition Model
- Conclusion
- Q&A

Introduction

A brief introduction 1/2

Divided in several steps:

- Find a topic theme and delimited it
- Realise the project

Objective

- Build a machine learning system on a Hadoop/Spark cluster
- Type of machine learning : face recognition

A brief introduction 2/2

Dataset :

- Training dataset: PubFig + LFW (fusion of 2 data sets)
- 35,469 training images and 11,720 test images
- 200 persons to identify

Infrastructure - Cluster Deployement

Apache Spark

Apache Spark™ is a fast and general engine for large-scale data processing.

<http://spark.apache.org/>

What is



Speed

- Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Ease of Use

- Write applications quickly in Java, Scala, Python, R.

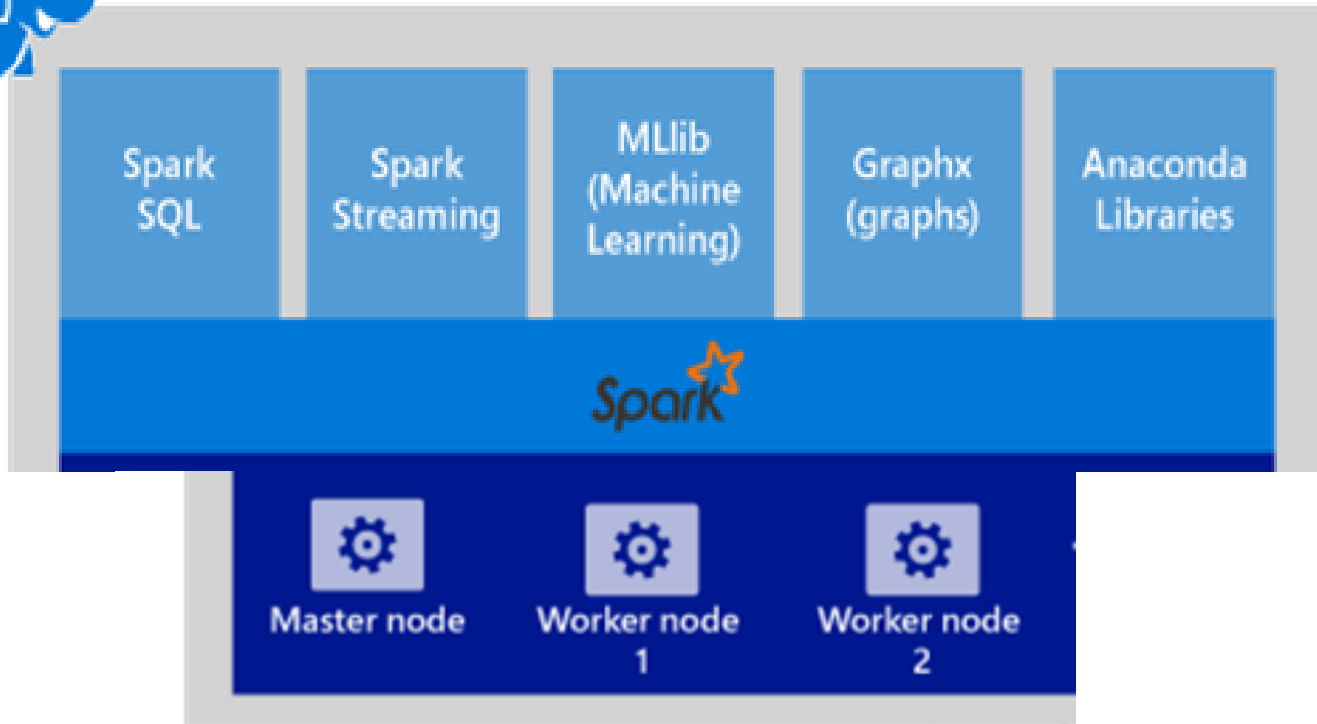
Generality

- Combine SQL, streaming, and complex analytics.

Runs Everywhere

- Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3.

Spark cluster in Hadoop



- 3 nodes with Hadoop 2.7 installed and configured
- 3 nodes with Spark 1.6 installed and configured
- SSH for passwordless communication between the master and the slaves

Sparklyr

R interface to Apache Spark, This package supports connecting to local and remote Apache Spark clusters, provides a 'dplyr' compatible back-end, and provides an interface to Spark's built-in machine learning algorithms.

<https://cran.r-project.org/web/packages/sparklyr/index.html>

What is



1

- New package that provides an interface between R and Apache Spark.
- Is NOT a replacement to the original apache spark but an extension to it.

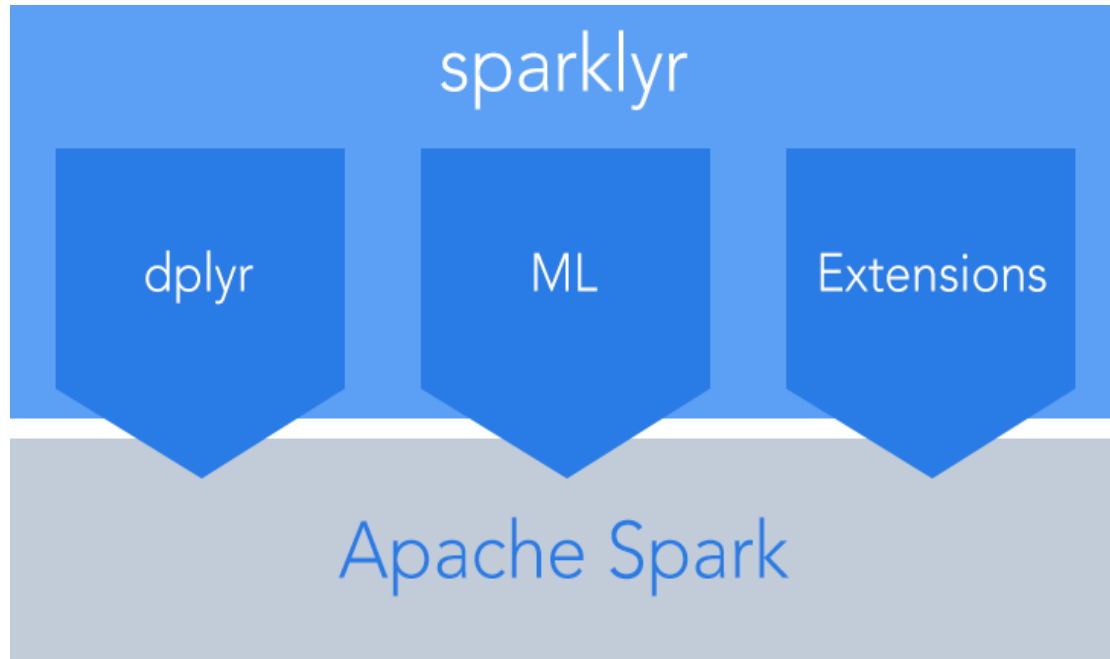
2

- Interactively manipulate Spark data using both dplyr and SQL
- Filter and aggregate Spark datasets then bring them into R for analysis and visualization.

3

- Create extensions that call the full Spark API and provide interfaces to Spark packages.

sparklyr: R interface for Apache Spark



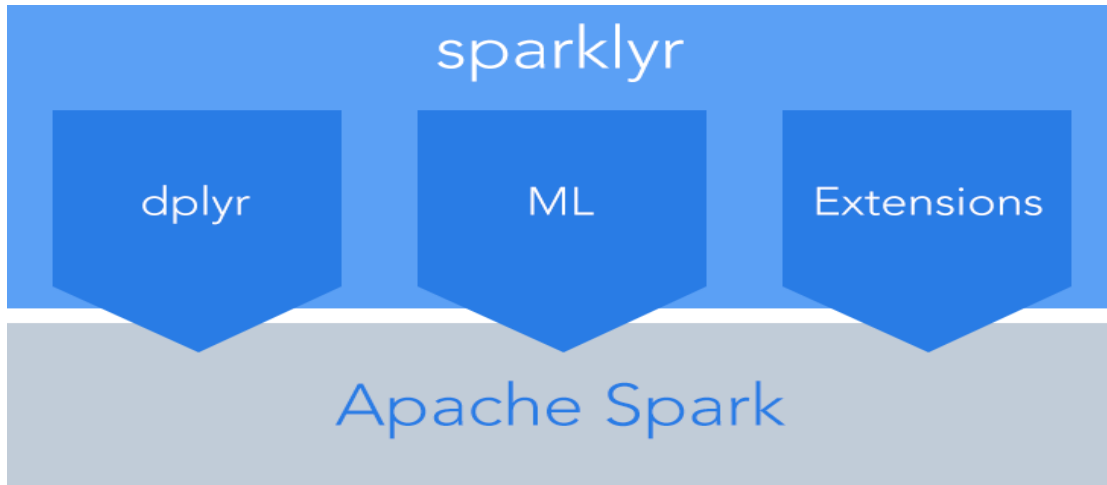
- Easy installation via devtools

```
# install sparklyr  
devtools::install_github("rstudio/sparklyr")
```

- Connects to both local instance of Spark and to a Spark cluster

```
library(sparklyr)  
config <- spark_config()  
#connect to local instance of spark  
sc <- spark_connect(master = "local", config)  
#connect to a spark cluster  
sc <- spark_connect(master = "spark://master_node_ip:7077", config)
```

sparklyr: R interface for Apache Spark

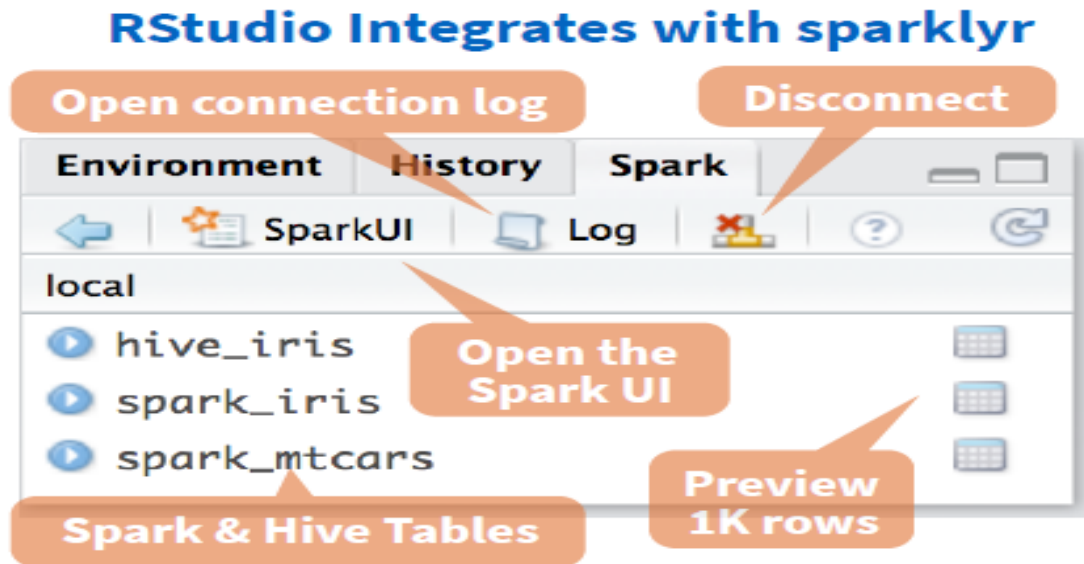


- Easy installation via devtools

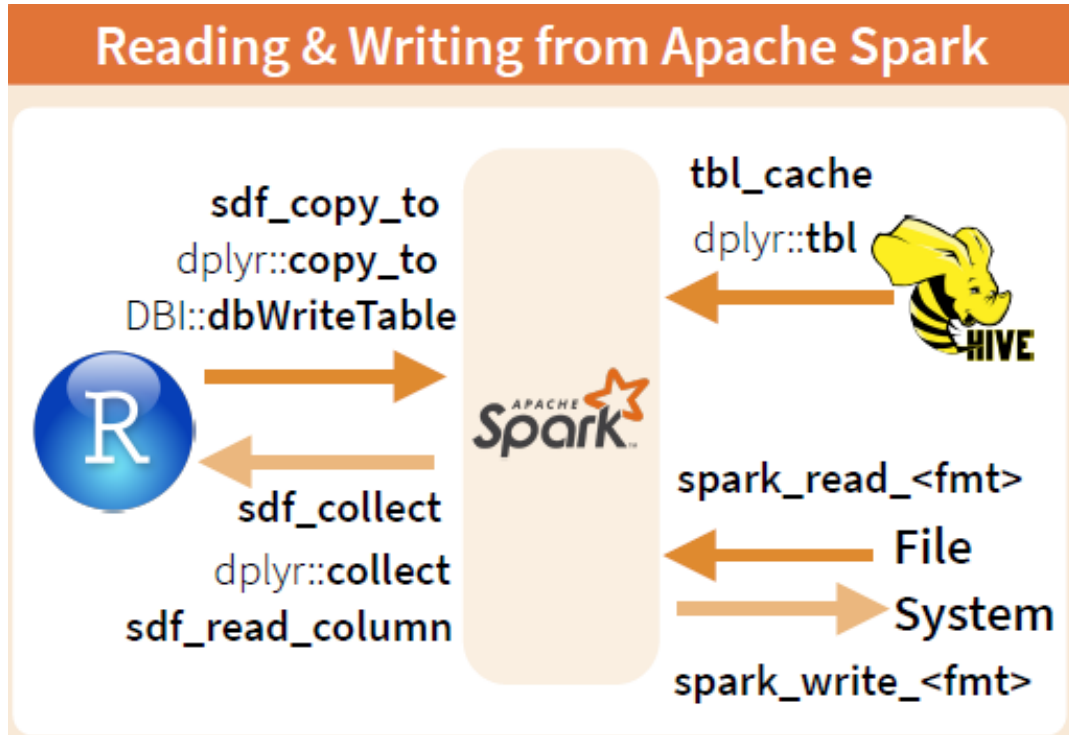
```
# install sparklyr  
devtools::install_github("rstudio/sparklyr")
```

- Connects to both local instance of Spark and to a Spark cluster

```
library(sparklyr)  
config <- spark_config()  
#connect to local instance of spark  
sc <- spark_connect(master = "local", config)  
#connect to a spark cluster  
sc <- spark_connect(master = "spark://master_node_ip:7077", config)
```



sparklyr: R interface for Apache Spark



- Loads data into Spark DataFrames from:
- Local R data frames (Small Data Set)
- Hive tables (Large Data Sets)
- Types: CSV, JSON, and Parquet files.

dplyr and ML in sparklyr

- Provides a complete dplyr backend for data manipulation, analysis and visualization

```
#manipulate data with dplyr
library(dplyr)
partitions <- data_spark %>%
  sdf_partition(training = 0.7, test = 0.3, seed = 1099)
```

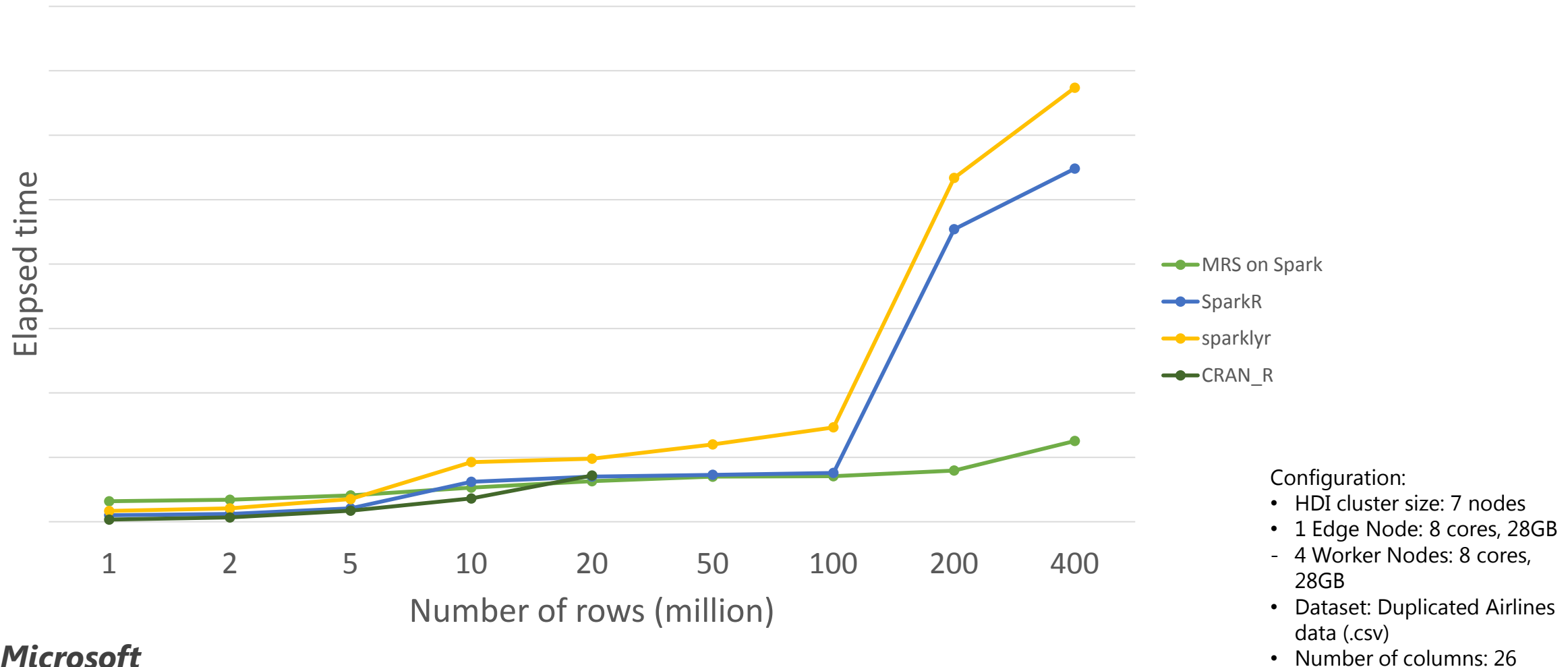
%>%

- Includes 3 family of functions for machine learning
 - ml_***: Machine learning algorithms for analyzing data provided by the spark.ml package.
 - ft_***: Feature transformers for manipulating individual features.
 - sdf_***: Functions for manipulating SparkDataFrames.

```
#Train RandomForest Model
rf_model <- partitions$training %>%
  ml_random_forest(response = class_name, features = features_names,
    type = "classification", num.trees = 300, max.depth = 3)
```

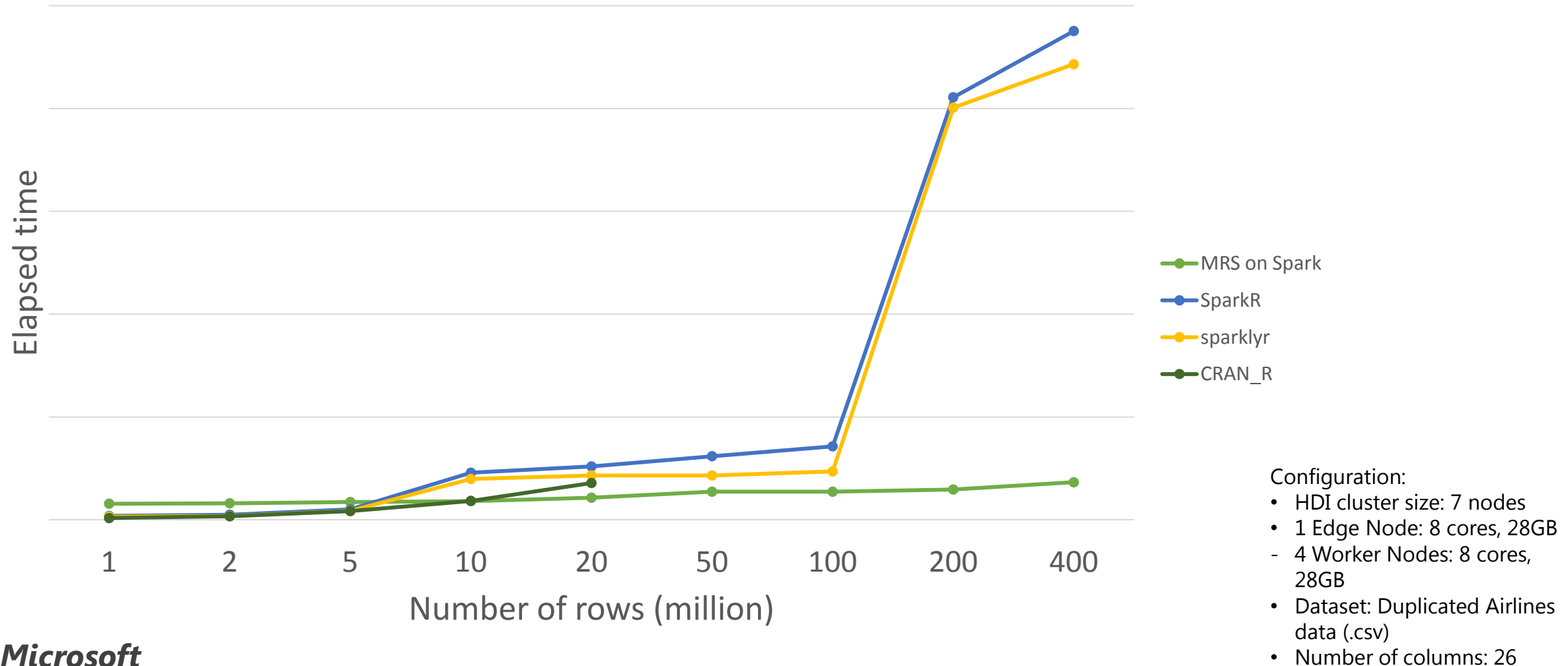
R on Spark – faster and more scalable

Logistic Regression (Reading from csv files)



R on Spark – substantially faster


Logistic Regression (Executing models)



Sparklyr cheat sheet

<http://spark.rstudio.com/images/sparklyr-cheatsheet.pdf>

Data Science in Spark with sparklyr Cheat Sheet



Import

- Export an R DataFrame
- Read a file
- Read existing Hive table

Tidy

- dplyr verb
- Direct Spark SQL (DBI)
- SDF function (Scala API)

Transform

- Transform function

Visualize

- Collect data into R for plotting

Model

- Spark MLlib
- H2O Extension

Communicate

- Collect data into R
- Share plots, documents, and apps

R for Data Science, Grolemund & Wickham

Using sparklyr

A brief example of a data analysis using Apache Spark, R and sparklyr in local mode

```
library(sparklyr); library(dplyr); library(ggplot2); library(tidy); set.seed(100)
```

Install Spark locally

```
spark_install("2.0.1")
```

Connect to local version

```
sc <- spark_connect(master = "local")
```

Copy data to Spark memory

```
import_iris <- copy_to(sc, iris, "spark_iris", overwrite = TRUE)
```

Partition data

```
partition_iris <- sdf_partition(import_iris, training=0.5, testing=0.5)
```

Sdf register

```
sdf_register(partition_iris, c("spark_iris_training", "spark_iris_test"))
```

Create a hive metadata for each partition

```
tidy_iris <- tbl(sc, spark_iris_training) %>% select(Species, Petal_Length, Petal_Width)
```

Spark ML Decision Tree Model

```
model_iris <- tidy_iris %>% ml_decision_tree(response="Species", features=c("Petal_Length", "Petal_Width"))
```

Create reference to Spark table

```
test_iris <- tbl(sc, "spark_iris_test")
```

Bring data back into R memory for plotting

```
pred_iris <- sdf_predict(model_iris, test_iris) %>% collect
```

```
pred_iris %>% inner_join(data.frame(prediction=0.2, label=model_iris$model.parameters$label) %>% ggplot(aes(Petal_Length, Petal_Width, col=label)) + geom_point())
```

Disconnect

```
spark_disconnect(sc)
```

Getting started

Local Mode

Easy setup, no cluster required

1. Install a local version of Spark: `spark_install("2.0.1")`
2. Open a connection: `sc <- spark_connect(master = "local")`

On a YARN Managed Cluster

1. Install RStudio Server or RStudio Pro on one of the existing nodes, preferably an edge node
2. Locate path to the cluster's Spark Home Directory, it normally is "/usr/lib/spark"
3. Open a connection: `spark_connect(master="yarn-client", version = "1.6.2", spark_home = [Cluster's Spark path])`

On a Mesos Managed Cluster

1. Install RStudio Server or Pro on one of the existing nodes
2. Locate path to the cluster's Spark directory
3. Open a connection: `spark_connect(master="[mesos URL]", version = "1.6.2", spark_home = [Cluster's Spark path])`

Using Livy (Experimental)

1. The Livy REST application should be running on the cluster
2. Connect to the cluster: `sc <- spark_connect(master = "http://host:port", method = "livy")`

On a Spark Standalone Cluster

1. Install RStudio Server or RStudio Pro on one of the existing nodes or a server in the same LAN
2. Install a local version of Spark: `spark_install(version = "2.0.1")`
3. Open a connection: `spark_connect(master="spark://host:port", version = "2.0.1", spark_home = spark_home_dir())`

Cluster Deployment

Cluster Deployment Options

Managed Cluster

- Cluster Manager
- Worker Nodes

Stand Alone Cluster

- Driver Node
- Worker Nodes

Spark is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rds.com

Tuning Spark

Example Configuration

```
config <- spark_config()
config$spark.executor.cores <- 2
config$spark.executor.memory <- "4G"
sc <- spark_connect(master = "yarn-client", config = config, version = "2.0.1")
```

Important Tuning Parameters with defaults

- spark.yarn.am.cores
- spark.yarn.am.memory

Important Tuning Parameters with defaults continued

- spark.executor.heartsbeathinterval 10s
- spark.network.timeout 120s
- spark.executor.memory 1g
- spark.executor.cores 1
- spark.executor.extraJavaOptions
- spark.executor.instances
- sparklyr.shell.executor-memory
- sparklyr.shell.driver-memory

Learn more at spark.rstudio.com • package version 0.5 • Updated: 12/21/16

Face recognition Model

Computer vision and its fields of application

Face recognition: a subfield of computer vision

Computer vision fields of application:

- Health: medical imaging
- Precision Agriculture: usage of computer vision for yields maximisation
- Optical Character Recognition (OCR): digitalization of handwritten documents
- Security: smart video surveillance: Intruder recognition

Face Recognition Architecture

1. Generic Architecture

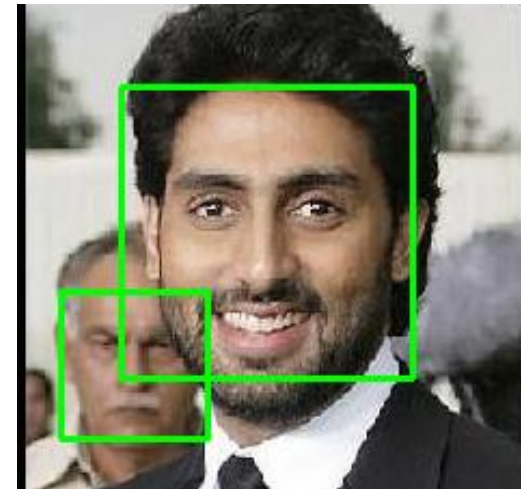
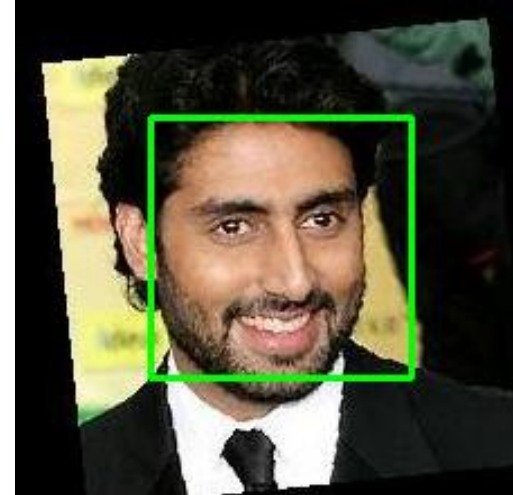


Face Recognition Architecture

2. Face Detection

Several techniques in the literature

- But, usually based on a supervised classifier model
- In our case: we used the OpenCV frontal-face model based on Haar features
- Training phase: we dropped the observations with 2 faces



Face Recognition Architecture

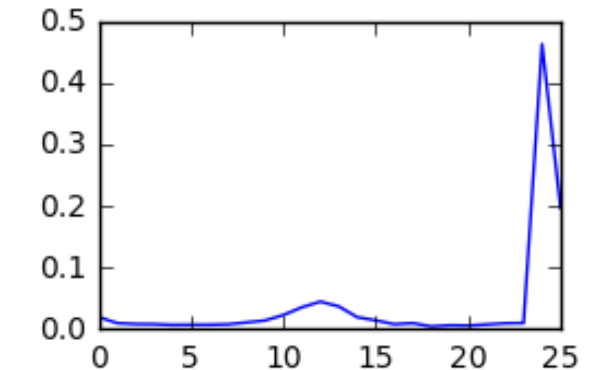
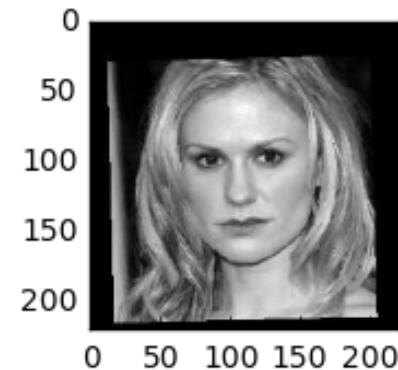
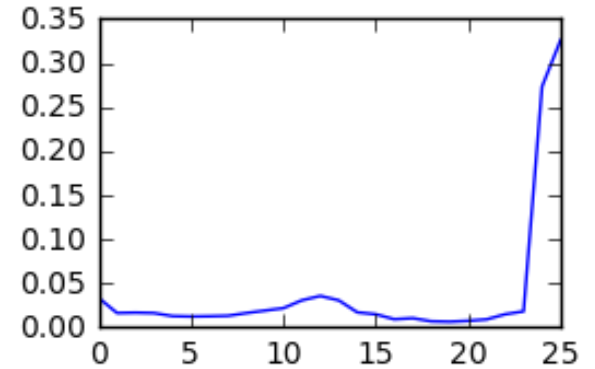
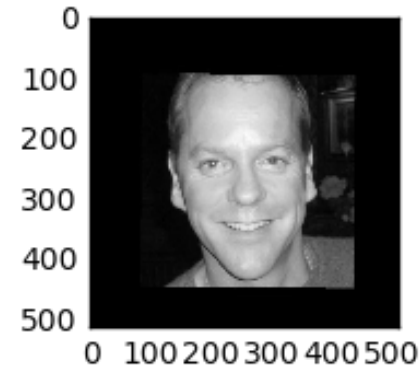
3. Features extraction (1/2)

- Goal: Extraction of discriminant information to feed a machine learning algorithm
- Features extraction \approx Feature engineering applied to some attributes of the pixels
- Face recognition \rightarrow Local features are efficient
- Examples of Local features
 - HoG (Histogram of Oriented Gradient) \rightarrow Human image detection
 - Gabor filter \rightarrow Convolution of the image with Gabor kernels
 - Local Binary Pattern \rightarrow Useful for Texture description

Face Recognition Architecture

3. Features extraction (2/2) (LPB Histogram Example)

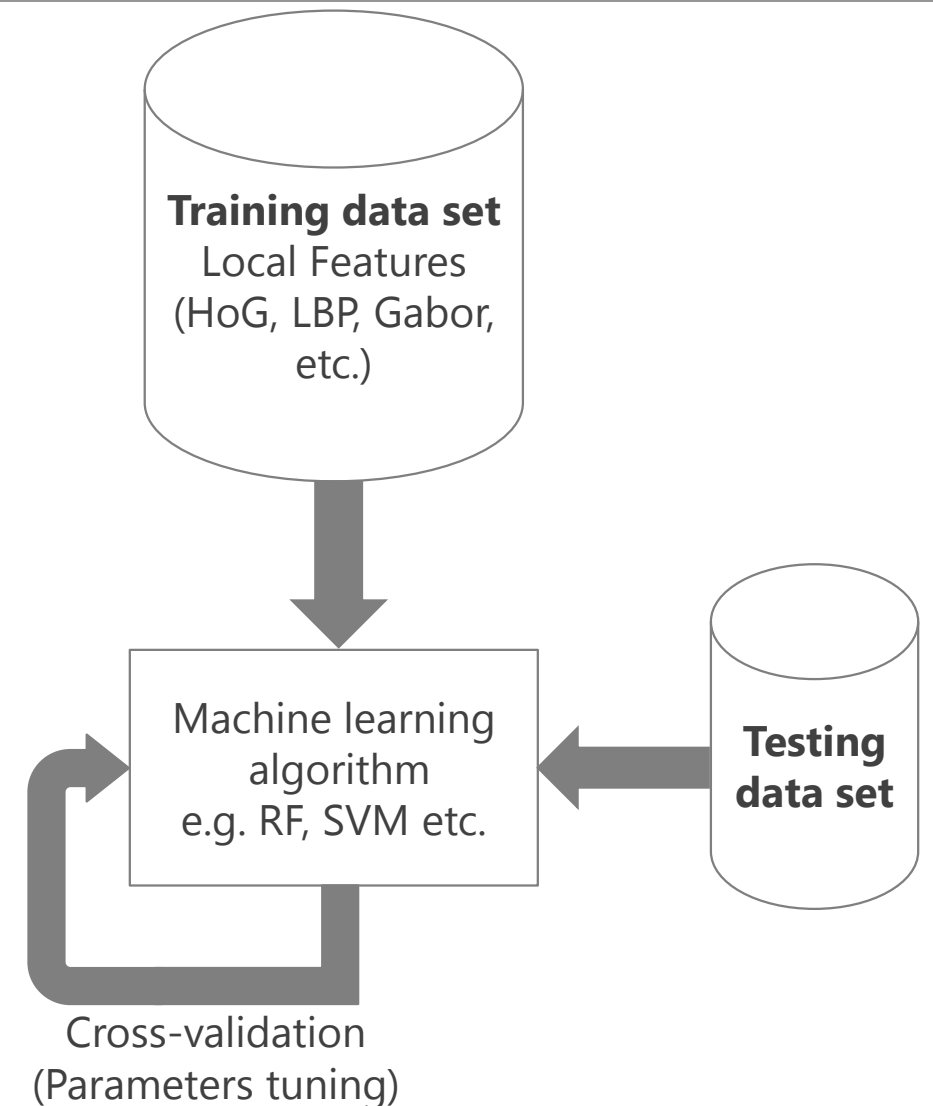
- LBP assumes: Face \approx Set of micro-patterns
- We used histograms of LBP as features:
 - \Rightarrow Each observation: 26 features (variables)



Face Recognition Architecture

4. Supervised Learning

- Feed a machine learning algorithm with features extracted:
 - SVM, Random Forests, Deep learning etc.
- Training/optimize a model using cross-validation process
- Test the model



Face Recognition Experimentation

1. Training the model

- Train and test data merged \Rightarrow global data set
- Training phase \Rightarrow split the global data set:
 - 70% for the train set and 30% for the test set
- Choice of machine learning algorithm:
 - \Rightarrow Non parametric, Scalability and straightforward parameter tuning
 - \Rightarrow Random forest (RF)
- We used SparklyR package to run Spark RF function

Face Recognition Experimentation

2. Testing the model

- Unfortunately: useless model obtained
- Probable reasons:
 - Inefficiency of the LBP features extracted (most important)
 - Lack of parameters tuning
- Need more times to investigate on the features extraction part:
 - A crucial component of the workflow
 - Combine several features ?
 - Automated features extraction using a deep learning algorithm ?

A dark gray triangle pointing downwards, located on the left side of the slide.

Conclusion

Conclusion

- Hadoop cluster : successfully achieved
- Designed a face recognition engine : not really satisfactory
- Perspectives :
 - Combination of several features
 - Use a deep learning algorithm

THANK YOU

Q&A