PREDICTING HEART FAILURE WITH 125 DIMENSIONS AND 1.7M PATIENTS LEVERAGING HADOOP, SPARK AND R

CHRIS MANRODT
STAFF DATA SCIENTIST AND TECHNICAL FELLOW
EVIDENCE INNOVATION
MEDTRONIC CLINICAL RESEARCH

Add photo or leave as solid Color

Medtronic
Further, Together

ACKNOWLEDGEMENTS

- Mike Harcus – Medtronic, Sr Prin Architect,
  - Owner of our Cloudera Environment and really smart guy
- Ian Cook – Cloudera
  - Author of the implyr package
- Hadley Wickham – R Studio
  - Author of dplyr and many other awesome R packages in the so-called “Haldeyverse”
- Javier Luraschi, Kevin Kuo, Kevin Ushey, JJ Allaire
  - Authors of the sparklyr package
- Medtronic HF “EI” Research Team
  - Dana Soderlund
  - Dan Schaber
  - Verla Laager
  - Jeff Cerkvenik
  - Boyang Bian
  - Jennifer Hinnenthal
  - Damian May

DISCLOSURES

- I am an employee of Medtronic, PLC
- I have received no compensation from Cloudera or other software or analytics vendors

- I am a biomedical engineer, clinical trialist and data scientist
  - I am not an IT or database architect

ELECTRONIC HEALTH RECORDS

Pardon the cheesy clipart, this was my favorite Google Image Search result

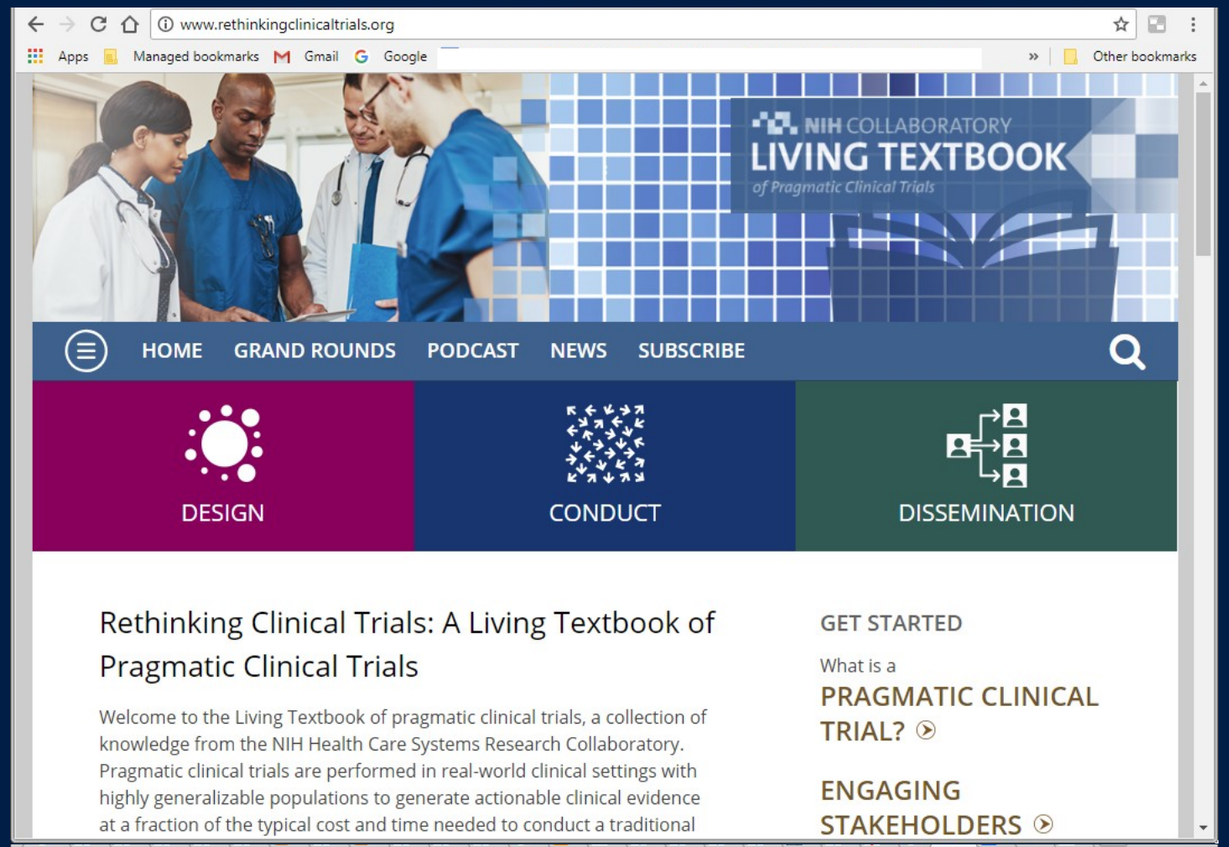
- EHR data a mix of structured and unstructured data
  - Most of the codification schemes in US healthcare data are used for billing purposes
- Most of the standards (eg. HL7 C-CDA) are designed around patient care
  - Data structure at lowest common denominator
  - Designed for viewing data 1 patient at a time
  - Analytics has had to wait a while…
- NLP is necessary to grasp clinical story
  - At best, medical codes are moderate quality meta-data
  - All the good stuff is in the notes
  - HIPAA/etc: Raw notes difficult to de-identify

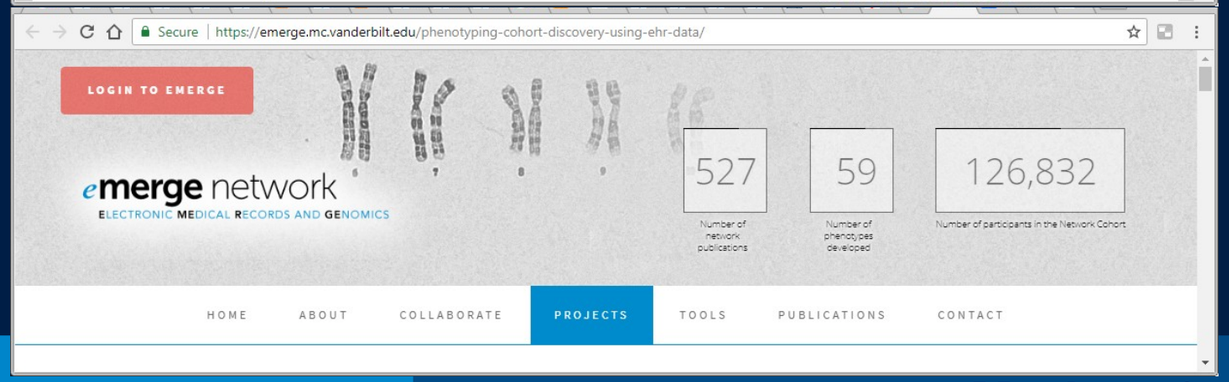
CLINICAL RESEARCH IS EVOLVING

- "Real World Evidence" (RWE) from "Pragmatic Clinical Trials"
  - FDA focus – 21st Century Cures Act, new guidance
  - NIH Collaboratory

<u>Clinical Study Process</u>

- Step 1: Experimental Design
- **Step 2: Define Cohorts, Endpoints & Outcomes**
- Step 3: Write Analysis Plan
- Step 4-to-n: Everything else

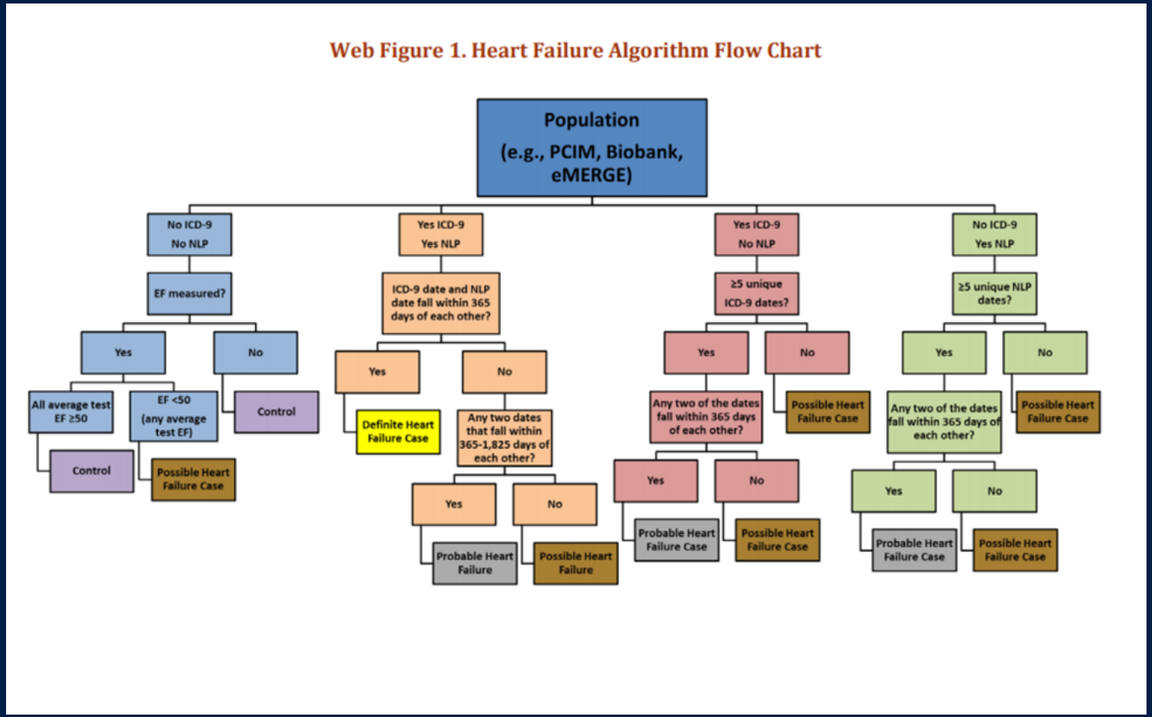




Medtronic

COMPUTABLE PHENOTYPES

- Phenotype: a composite of an organisms observable characteristics
  - Contrast with “Genotype” of heredity
- Phenotype results from *expression* of genetics as well as environmental influences
- “A computable phenotype is defined as “a clinical condition, characteristic, or set of clinical features that can be determined solely from the data in EHRs and ancillary data sources and does not require chart review or interpretation by a clinician”[1]



[1] R Richesson and M Smerek Rethinking Clinical Trials: A Living Textbook in Pragmatic Clinical Trials. NIH Health Care Systems Research Collaboratory. Published June 27, 2014
[2] Suzette J. Bielinski. Mayo Clinic. Heart Failure (HF) with Differentiation between Preserved and Reduced Ejection Fraction. PheKB; 2013 Available from: https://phekb.org/phenotype/147

Medtronic

BEFORE ANALYSIS CAN BEGIN…

Select Concepts	Define Ontology	Characterize Data	Algorithms to Clean & dim-redux	Develop and Test Apps	Pre-specify Objectives
Analysis Question(s)	• Codes • Guidelines • Terminologies	• Expected rates • Missing values • Signal/Noise	• Publications • Consensus • Medical Experts	• Computing reqt • Software eng	• “Lock” CIP/SAP

BEFORE ANALYSIS CAN BEGIN…

Select Concepts	Define Ontology	Characterize Data	Algorithms to Clean & dim-redux	Develop and Test Apps	Pre-specify Objectives
Analysis Question(s)	• Codes • Guidelines • Terminologies	• Expected rates • Missing values • Signal/Noise	• Publications • Consensus • Experts	• Computing reqt • Software eng	• “Lock” CIP/SAP

Concept	Ontologies	Algorithm	Design Inputs
Acute Myocardial Infarction	58 ICD9/10 Dx Codes for AMI 581 PCI/CABG Procedure codes Range of Troponin Lab values	1st AMI Dx code + PCI/CABG and/or Tn(+)	ACC/AHA Definitions Medical Coding Guides Published AMI rates

Medtronic

CLINICAL PHENOTYPES IN HEART FAILURE
"WE'RE GOING TO NEED A BIGGER BOAT"

Background Research

- Published computable phenotypes were not sufficient for differentiating guideline recommendations
- >250 recommendations in applicable medical guidelines
  - 900+ cited papers
  - 125 clinical concepts needed to assess EHR data for patients these categories

Phenotyping Process

- Extracted billions of observations to assess each of the 125 clinical concepts
- Each concept was characterized
  - Cleanliness, missingness, occurrence rates, concordance
- Phenotypes constructed from applicable concepts
  - Patient-level time-series constructs based on guideline care pathways

Note: Methods and results pending publication

ASSESSMENT OF ANALYTICS OPTIONS

Solutions we tested

Option	Data Storage	Data Cleaning	Analysis	Pro's	Con's
A	MS SQL Server	SQL scripts or via PC tools to SQL	Desktop choice	• Readily available • Lower learning curve	• Slow • Limitations of desktop memory, processing, network I/O
B	MS SQL Server	MS SQL – RevolutionR	Desktop choice	• Runs serverside processing • Less exposure to I/O performance	• Not available at the time we started
C	MS SQL Server	SPSS Modeler	Desktop choice	• Graphical interface • Lots of Built-in Tools	• Our SPSS server could not handle large data tables (3+ billion rows)
D	HDFS Parquet	Impala	--	• 50x faster than MS SQL	• Design to support single queries • Not designed to support long scripts required for data cleaning procedures
E	HDFS Parquet	CDSW	CDSW	• Runs Turing-complete languages • Speed: 50-100x faster than Desktop R • Allows for collaboration	• Set up time • Kubernetes security

CLOUDERA WANTS YOUR BUSINESS

THEY SUPPORT STUFF I ACTUALLY USED, AND I GOT SOME COOL $#!+ TO HAPPEN

The good news

Data Engineering

Data has never been more plentiful

Data Science (Exploratory)

Open source data science and machine learning libraries are rapidly evolving

Production (Operational)

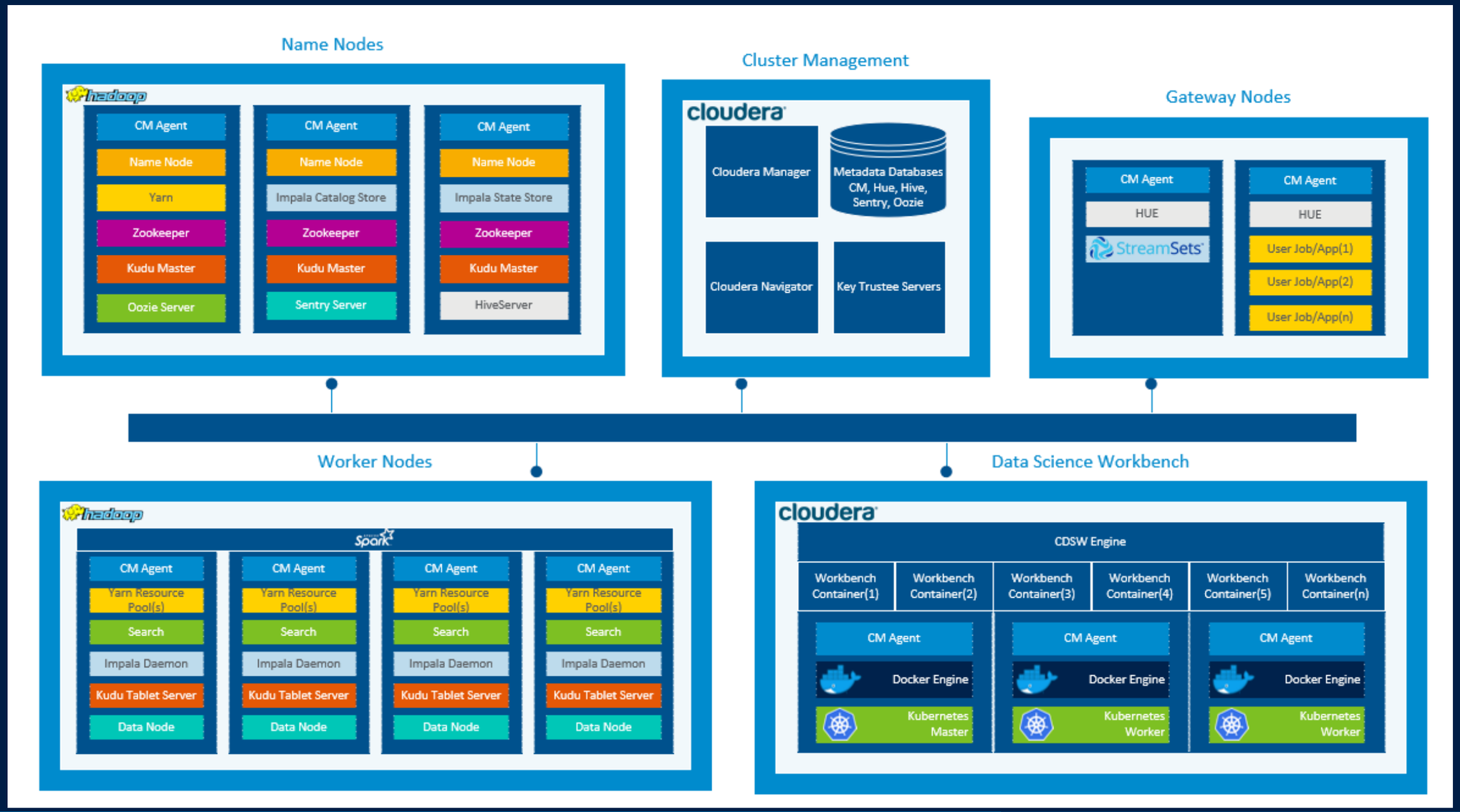
Commodity (and on-demand) compute makes scalable production machine learning affordable

cloudera

 5

Medtronic

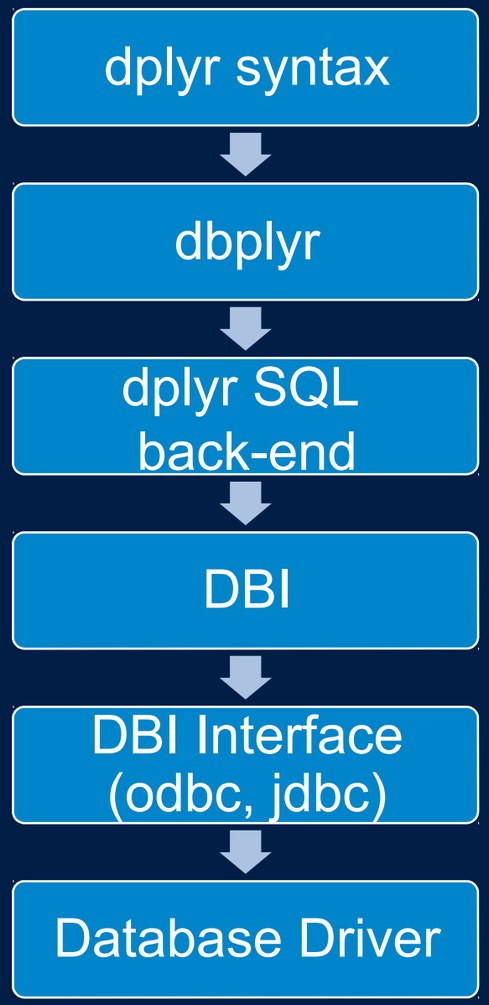
CLOUDERA ECOSYSTEM



Medtronic

Why we used dplyr…

- Cloudera Data Science Workbench supports R libraries like dplyr
- R library dplyr allows the same syntax to be applied against multiple data engines
  - MS T-SQL
  - MySQL
  - Hive
  - Impala
  - Spark
- dplyr can be “almost” DB-agnostic
  - Change the connection type at the beginning of the function
  - SQL is not case-sensitive, but everything else is…



Medtronic

CLOUDERA DATA SCIENCE WORKBENCH

File Edit View Navigate Run ▶ 3_sparklyr.R

```
## Connecting to Spark

library(sparklyr)
library(dplyr)

# The returned Spark connection (sc) provides a remote dplyr data source to the
sc

## Using dplyr
# We can now use all of the available dplyr verbs against the tables within the

# # filter by departure delay
flights_tbl %>% filter(dep_delay == 2)

# Introduction to dplyr provides additional dplyr examples you can try. For exam
delay <- flights_tbl %>%
  group_by(tailnum) %>%
  summarise(count = n(), dist = mean(distance), delay = mean(arr_delay)) %>%
  filter(count > 20, dist < 2000, !is.na(delay)) %>%
  collect()

# # Plot delays
library(ggplot2)
ggplot(delay, aes(dist, delay)) +
  geom_point(aes(size = count), alpha = 1/2) +
  geom_smooth() +
  scale_size_area(max_size = 2)

## Machine Learning
# You can orchestrate machine learning algorithms in a Spark cluster via the mac

# In this example we'll use ml_linear_regression to fit a linear regression mode

# copy mtcars into spark
mtcars_tbl <- copy_to(sc, mtcars)

# transform our data set, and then partition into 'training', 'test'
partitions <- mtcars_tbl %>%
  filter(hp >= 100) %>%
  mutate(cyl8 = cyl == 8) %>%
  sdf_partition(training = 0.5, test = 0.5, seed = 1099)

# fit a linear model to the training dataset
fit <- partitions$training %>%
  ml_linear_regression(response = "mpg", features = c("wt", "cyl"))

# For linear regression models produced by Spark, we can use summary() to learn
```

Line 52, Column 2 ★ 52 Lines R Spaces 2

Project >_ Terminal access Clear Interrupt Stop Sessions

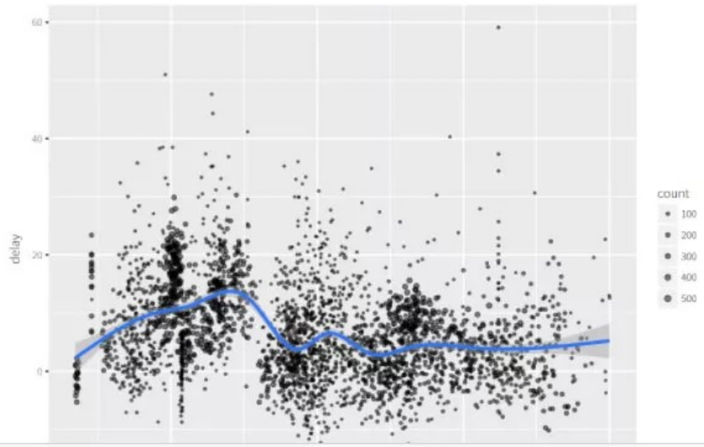
from the tutorial which plots data on flight delays:

```
> delay <- flights_tbl %>%
    group_by(tailnum) %>%
    summarise(count = n(), dist = mean(distance), delay = mean(arr_delay)) %>%
    filter(count > 20, dist < 2000, !is.na(delay)) %>%
    collect()
```

Plot delays

```
> library(ggplot2)
> ggplot(delay, aes(dist, delay)) +
    geom_point(aes(size = count), alpha = 1/2) +
    geom_smooth() +
    scale_size_area(max_size = 2)

`geom_smooth()` using method = 'gam'
```



Note: I pulled this image off the Cloudera website, this is not my data

Medtronic

PINK
FLOYD

IMPALA QUERY BENCHMARK COMPARED WITH RDBMS – TO DO

```
select ptid, obs_date, cast(obs_result as int) as obs_result
    from mytable
    where obs_type = 'SBP'
    group by ptid, obs_date, obs_result
    limit 65,000
```

Note: mytable has 5,844,983,436 rows

Database	Tools	Run time	Rows loaded in memory
SQL Server	SQL Server Mgmt Studio	1,564 sec	65,000
HDFS Parquet	Hue – Impala	76 sec	65,000
HDFS Parquet	CDSW R w/ Sparklyr	64 sec	65,000
HDFS Parquet	CDSW R w/ Implyr	15 sec	65,000

COPIED A LOT OF MY CODE AND IT RAN RIGHT AWAY

IN THIS EXAMPLE, I DROP AN SQL STATEMENT RIGHT INTO R

```
# use the connection type for the database technology
src_db  <- ...

# the query statement doesn't have to change
my_data <- dbGetQuery(conn = src_db$con
                      ,sql(paste0("select ptid,obs_date,cast(obs_result as int) as obs_result "
                      , " from ehr_obs_fix  where obs_type = 'SBP' group by ptid, obs_date,
obs_result")))

# manipulate in memory using dplyr syntax
my_data %>%  filter(!is.na(obs_result)) %>% nrow %>% print

# "lazy" option to use dplyr syntax to manipulate the data on the database side, then use the collect() function
```

SPARKLYR PACKAGE

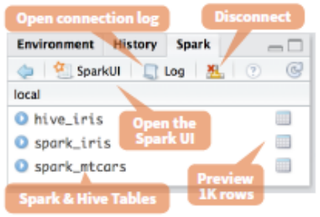
Data Science in Spark with Sparklyr : : CHEAT SHEET

Intro

sparklyr is an R interface for **Apache Spark™**, it provides a complete **dplyr** backend and the option to query directly using **Spark SQL** statement. With sparklyr, you can orchestrate distributed machine learning using either **Spark's MLlib** or **H2O** Sparkling Water.

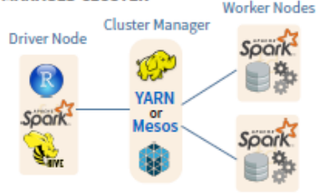
Starting with **version 1.044, RStudio Desktop, Server and Pro include integrated support for the sparklyr package**. You can create and manage connections to Spark clusters and local Spark instances from inside the IDE.

RStudio Integrates with sparklyr



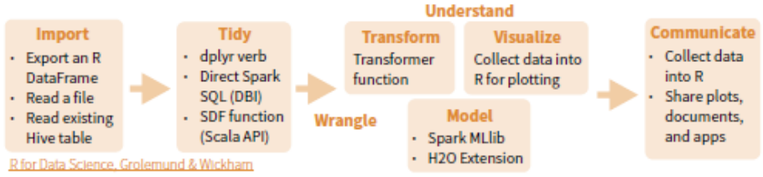
Cluster Deployment

MANAGED CLUSTER



STAND ALONE CLUSTER

Data Science Toolchain with Spark + sparklyr



R for Data Science, Grolemund & Wickham

Getting Started

LOCAL MODE (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 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(method = "livy", master = "http://host:port")**

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 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())**

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 *512m*
- spark.network.timeout *120s*
- spark.executor.memory *1g*
- spark.executor.cores *1*
- spark.executor.instances
- spark.executor.extraJavaOptions
- spark.executor.heartbeatInterval *10s*
- sparklyr.shell.executor-memory
- sparklyr.shell.driver-memory

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(tidyr);
set.seed(100)
```

Install Spark locally

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

Connect to local version

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

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

Copy data to Spark memory

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

Partition data

```
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"))

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

Create reference to Spark table

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

Bring data back into R memory for plotting

```
pred_iris %>%
  inner_join(data.frame(prediction=0:2,
  lab=model_iris$model.parameters$labels)) %>%
  ggplot(aes(Petal_Length, Petal_Width, col=lab)) +
  geom_point()

spark_disconnect(sc)
```

Disconnect

Model (MLlib)

ml_decision_tree(my_table, response = "Species", features = c("Petal_Length" , "Petal_Width"))

ml_als_factorization(x, user.column = "user", rating.column = "rating", item.column = "item", rank = 10L, regularization.parameter = 0.1, iter.max = 10L, ml.options = ml_options())

ml_decision_tree(x, response, features, max.bins = 32L, max.depth = 5L, type = c("auto", "regression", "classification"), ml.options = ml_options()) **Same options for: ml_gradient_boosted_trees**

ml_generalized_linear_regression(x, response, features, intercept = TRUE, family = gaussian(link = "identity"), iter.max = 100L, ml.options = ml_options())

ml_kmeans(x, centers, iter.max = 100, features = dplyr::tbl_vars(x), compute.cost = TRUE, tolerance = 1e-04, ml.options = ml_options())

ml_lda(x, features = dplyr::tbl_vars(x), k = length(features), alpha = (50/k) + 1, beta = 0.1 + 1, ml.options = ml_options())

ml_linear_regression(x, response, features, intercept = TRUE, alpha = 0, lambda = 0, iter.max = 100L, ml.options = ml_options()) **Same options for: ml_logistic_regression**

ml_multilayer_perceptron(x, response, features, layers, iter.max = 100, seed = sample(.Machine$integer.max, 1), ml.options = ml_options())

ml_naive_bayes(x, response, features, lambda = 0, ml.options = ml_options())

ml_one_vs_rest(x, classifier, response, features, ml.options = ml_options())

ml_pca(x, features = dplyr::tbl_vars(x), ml.options = ml_options())

ml_random_forest(x, response, features, max.bins = 32L, max.depth = 5L, num.trees = 20L, type = c("auto", "regression", "classification"), ml.options = ml_options())

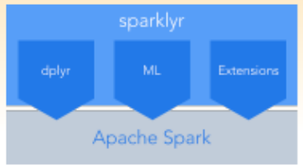
ml_survival_regression(x, response, features, intercept = TRUE,censor = "censor", iter.max = 100L, ml.options = ml_options())

ml_binary_classification_eval(predicted_tbl_spark, label, score, metric = "areaUnderROC")

ml_classification_eval(predicted_tbl_spark, label, predicted_lbl, metric = "f1")

ml_tree_feature_importance(sc, model)

sparklyr is an R interface for Spark



Medtronic

SPARK CONNECTION TO TAP INTO MACHINE LEARNING

I PROMISE, THIS IS MY LAST SLIDE WITH CODE ON IT

```
sc <- spark_connect(master = "yarn", version ='2.1.0', config = config, spark_home = '…')
db_tbl <- tbl(sc, from  = 'my_database')
my_data_sp <- db_tbl %>%
                    filter(obs_type == 'SBP') %>%
                    select(ptid, obs_date, obs_result) %>%
                    mutate(obs_result = as.numeric(obs_result)) %>%
                    distinct %>%
                    collect %>%                                    # I collect here because I
needed to clean data more
                    group_by(…) %>%                                # You can use tbl_cache() if
your data is clean enough
                    summarise(…) %>%
                    ungroup %>%
                    copy_to(sc, name,…)
my_model  <- my_data_sp %>% ml_decision_tree(…)          # spark machine learning libraries
```


Medtronic

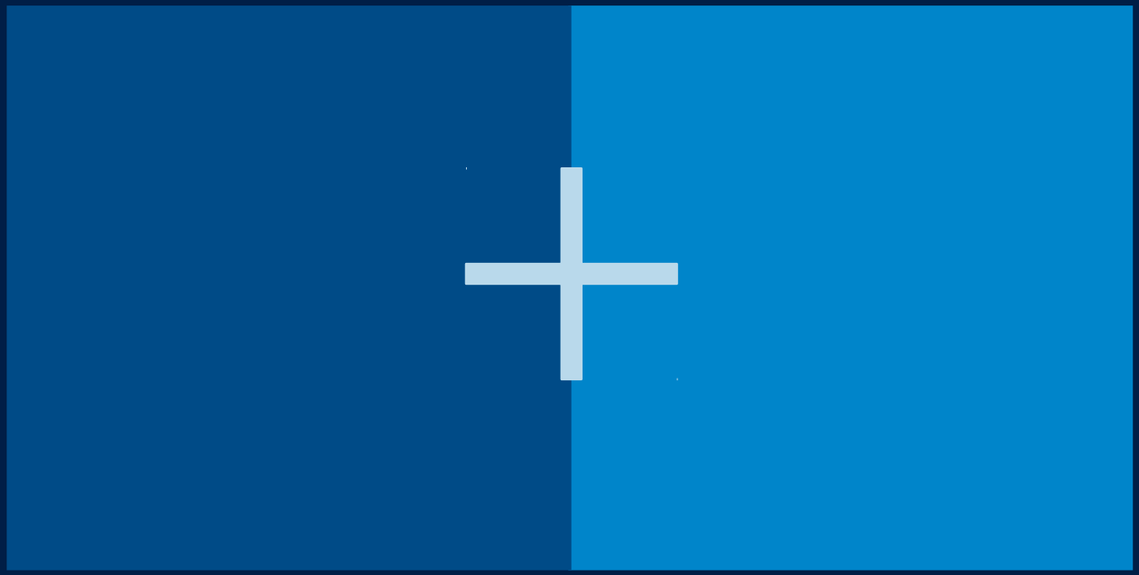
SUMMARY

(+)

- Fast
  - Phenotyping on 16-core workstation: 196 hours
  - Same R code in CDSW: 4 hours
- Very flexible
  - Data access
  - Data Storage and retrieval
  - Processing
- Cost structure vs. cloud services
- Machine Learning libraries

(Δ)

- Takes time to configure
- CDSW is not really a full IDE yet
  - Limited feature set for supporting syntax
  - Debug is like base R
  - New release coming soon
- Not a huge user community using CDSW in R



CLINICAL PHENOTYPES IN HEART FAILURE
WE WERE DEALING WITH A LOT OF DATA

Background Research

- Published computable phenotypes were not sufficient for differentiating guideline recommendations
- 133 recommendations in the ACC/AHA guidelines for the management of Heart Failure
  - 383 cited papers
- 135 recommendations in the ACC/AHA guidelines for devices in the treatment of cardiac arrhythmias
  - 595 cited papers
- Identified 14 categories of recommendations specific to our hypothesis
  - 124 clinical concepts needed to assess EHR data for patients these categories

Phenotyping Process

- Extracted billions of observations to assess each of the 124 clinical concepts
- Each concept was assessed individually for cleanliness, occurrence rates, concordance across multiple tables
- Each concept was also assessed in patient-level time-series constructs based on guideline care pathways
  - Methods and results pending publication