SparkR Notebook

SparkR Notebook using a Local Mode - Standalone Spark Cluster

```
s <- Sys.getenv()
#names(s <- Sys.getenv()) # all settings (the values could be very long)
#head(s, 12)# using the Dlist print() method
s[grep("SPARK", names(s))]
```

```
## PYSPARK_DRIVER_PYTHON   /home/dhankar/anaconda3/envs/dc_dev_venv/bin/python
## PYSPARK_PYTHON          /home/dhankar/anaconda3/envs/dc_dev_venv/bin/python
## SPARK_HOME              /opt/spark
```

```
library(SparkR, lib.loc = c(file.path(Sys.getenv("SPARK_HOME"), "R", "lib")))
```

```
##
## Attaching package: 'SparkR'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, filter, lag, na.omit, predict, sd, var, window
```

```
## The following objects are masked from 'package:base':
##
##     as.data.frame, colnames, colnames<-, drop, endsWith, intersect,
##     rank, rbind, sample, startsWith, subset, summary, transform, union
```

```
sparkR.session(master = "local[2]", sparkConfig = list(spark.driver.memory = "2g"))
```

```
## Spark package found in SPARK_HOME: /opt/spark
```

```
## Launching java with spark-submit command /opt/spark/bin/spark-submit   --driver-memory "2g" sparkr-sh
```

```
## Java ref type org.apache.spark.sql.SparkSession id 1
```

```
print(as.DataFrame)
```

```
## function (data, schema = NULL, samplingRatio = 1, numPartitions = NULL)
## {
##     createDataFrame(data, schema, samplingRatio, numPartitions)
## }
## <bytecode: 0x556c4fdf5ce0>
## <environment: namespace:SparkR>
```

```
# above - we see whats the Internals of the as.DataFrame

spark_r_df <- as.DataFrame(faithful)
print(typeof(spark_r_df))
```

```
## [1] "S4"
```

```
print(class(spark_r_df))
```

```
## [1] "SparkDataFrame"
## attr(,"package")
```

```
## [1] "SparkR"
```

```
sparkR.session(sparkPackages = "org.apache.spark:spark-avro_2.12:3.1.2")
```

```
## Java ref type org.apache.spark.sql.SparkSession id 1
```

```
#coefficients in a linear model
# load libraries
#install.packages("mlbench")
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:SparkR':
##
##     histogram
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:SparkR':
##
##     expr
```

```
library(mlbench)

# load dataset
data(PimaIndiansDiabetes)
#show(PimaIndiansDiabetes) # 768 ROWS
View(PimaIndiansDiabetes[1:5,]) # 1st 5 ROWS from Total 768 ROWS
# create 80%/20% for training and validation datasets
set.seed(9)
validation_index <- createDataPartition(PimaIndiansDiabetes$diabetes, p=0.80, list=FALSE)
# Above Extracts the Index - below we do the actual Split using these Index Values
# REFER -- Page 30 -- https://cran.r-project.org/web/packages/caret/caret.pdf

validation <- PimaIndiansDiabetes[-validation_index,]
View(validation[1:5,]) # 153 ROWS
training <- PimaIndiansDiabetes[validation_index,]

# train a model and summarize model
set.seed(9)
control <- trainControl(method="cv", number=10)
# REFER - Page -170-- trainControl --
# CARET Package = https://cran.r-project.org/web/packages/caret/caret.pdf
# mehod is CROSS VALIDATION with a 10 FOLDS CV on the TRAINING Data Set.
fit.lda <- train(diabetes~., data=training, method="lda", metric="Accuracy", trControl=control)
# Above we have the TRAINING of the MODEL - we use the TRAINING Data Set , method is LDA
# This piece of Code Means == diabetes~.
# We  are Regressing to Predict the Values of the TARGET / INDEPENDENT VARIABLE ...
# diabetes and DOT NOTATION - all other VARIABLES which will be the - DEPENDENT VARIABLEs .
# No variable from the RAW DataSet is thus being Dropped
```

```
# Further Reading - COHEN's KAPPA ---  Usually used when we have an imbalance in the classes.
# CLASS - 1 - Has 70% SAMPLES
# Class - 0 - Has 30% SAMPLES
# In such a Case - we can achieve 70% accuracy by predicting all instances are for class 0.
# So target in such a Case should to have a MODEL which has ACCURACY atleast at 90% .
```

```
#

print(fit.lda)
```

```
## Linear Discriminant Analysis
##
## 615 samples
##   8 predictor
##   2 classes: 'neg', 'pos'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 554, 553, 553, 554, 554, 553, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7692491  0.4617243
```

```
print(fit.lda$finalModel)
```

```
## Call:
## lda(x, grouping = y)
##
## Prior probabilities of groups:
##       neg       pos
## 0.6504065 0.3495935
##
## Group means:
##     pregnant  glucose pressure  triceps  insulin     mass  pedigree   age
## neg 3.232500 109.6475 68.95750 19.10250 67.23500 30.43450 0.4301825 30.72
## pos 4.823256 141.2977 72.17674 22.54884 99.26512 35.51163 0.5474000 37.20
##
## Coefficients of linear discriminants:
##                   LD1
## pregnant  0.079355312
## glucose   0.028047485
## pressure -0.010731792
## triceps   0.005925179
## insulin  -0.001249797
## mass      0.058251334
## pedigree  0.656437597
## age       0.017064189
```

```
# save the model to disk
saveRDS(fit.lda$finalModel, "./final_model_diab.rds")
```

```
# Seen below we have a sample Python Code Snippet for reading the .rds file created above .
# kindly note this cant be run within R and is shared here only as a sample code

# import rpy2.robjects as robjects
```

```
# from rpy2.robjects import pandas2ri
# import numpy as np
# import pandas as pd
#
# readRDS = robjects.r['readRDS']
# py_ls_vect = readRDS('final_model_diab.rds')
# print(py_ls_vect) # OK
# print(type(py_ls_vect)) # OK --- <class 'rpy2.robjects.vectors.ListVector'>
# print(py_ls_vect.names) # OK


#  [1] "prior"       "counts"      "means"       "scaling"     "lev"
#  [6] "svd"         "N"           "call"        "xNames"      "problemType"
# [11] "tuneValue"   "obsLevels"   "param"
```

Below we run the TRAINED Model (LDA) on our VALIDATION DataSet

```
set.seed(9)
predictions <- predict(fit.lda, newdata=validation)
confusionMatrix(predictions, validation$diabetes)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction neg pos
##        neg  87  21
##        pos  13  32
##
##                Accuracy : 0.7778
##                  95% CI : (0.7036, 0.8409)
##     No Information Rate : 0.6536
##     P-Value [Acc > NIR] : 0.000586
##
##                   Kappa : 0.4912
##
##  Mcnemar's Test P-Value : 0.229949
##
##             Sensitivity : 0.8700
##             Specificity : 0.6038
##          Pos Pred Value : 0.8056
##          Neg Pred Value : 0.7111
##              Prevalence : 0.6536
##          Detection Rate : 0.5686
##    Detection Prevalence : 0.7059
##       Balanced Accuracy : 0.7369
##
##        'Positive' Class : neg
##
```

```
#Further Reading ---

#KAPPA -- https://en.wikipedia.org/wiki/Cohen%27s_kappa
#Accuracy and Precision -- https://en.wikipedia.org/wiki/Accuracy_and_precision
```