**Proba. Ranges Computing**

%pyspark

from \_\_future\_\_ import print\_function

import sys, re

from pyspark.sql.context import HiveContext

import pyspark.sql.functions as sqlF

# Ava FP Novel

sc.\_jsc.hadoopConfiguration().set("fs.s3a.endpoint", "es-si-s3-z2.eecloud.nsn-net.net")

sc.\_jsc.hadoopConfiguration().set("fs.s3a.access.key", "9LHONDBX17WLEA8SCG1S")

sc.\_jsc.hadoopConfiguration().set("fs.s3a.secret.key", "mE39w2rXZkfNCMDDYHqslRQlBDZM9lSNp82HMvZ5")

sqlCtx = HiveContext(sc)

def setEnvVars(projectPath=None):

if projectPath:

s3ConfFiles = re.compile(r'^s3Setting$')

kpiConf = re.compile(r'^programKPIsNames$')

keyspaceName = re.compile(r'^cassandraKeyspace$')

""" Program conf files parsing"""

envSetFiles = sc.wholeTextFiles(projectPath + "/config/data\_mining")

envSetRDD = envSetFiles.map(lambda x: ([x[0].split("config/data\_mining/")[1].split(".json")[0]], (x[1].split("\n"))))

envFilesList = envSetRDD.collect()

for confFile in envFilesList:

if s3ConfFiles.match(str(confFile[0][0])):

s3Conf = eval(str(confFile[1][0]))

elif kpiConf.match(str(confFile[0][0])):

programKPIsNames = eval(str(confFile[1][0]))

elif keyspaceName.match(str(confFile[0][0])):

cassandraKeyspace = eval(str(confFile[1][0]))

return programKPIsNames, cassandraKeyspace, s3Conf

else :

print("Please enter project root path. \n")

def produceProbaRanges(kpi\_name=None, cassandraKeyspace=None):

"""

Usage: Produce probability ranges on daily prediction.

"""

print("{} range computing starts ........".format(kpi\_name))

InputPred = sqlCtx.read\

.format("org.apache.spark.sql.cassandra")\

.options(table="pred\_tmp\_{}".format(kpi\_name), keyspace="{}".format(cassandraKeyspace)).load().where("reading\_date between '2018-10-29' and '2018-11-23'")\

.withColumnRenamed("reading\_date", "days").withColumnRenamed("predicted\_class", "prediction")\

.withColumnRenamed("actual\_class", "label").withColumnRenamed("probability\_of\_prediction", "probability")

currentDate= re.sub(r'(, )','-', re.sub(r"^(Row\(\w+=datetime.date\()|(\)\))$",'',str(InputPred.select("days").first())))

sqlCtx.registerDataFrameAsTable(InputPred, "predoutput")

""" Overall proba ranges computing """

queryDF1 = sqlCtx.sql(""" select p.kpi\_name, p.days, p.prediction, p.label, p.probability,

case

when (p.probability >= 0.5 and p.probability < 0.6 ) then "Proba\_05\_06"

when (p.probability >= 0.6 and p.probability < 0.7 ) then "Proba\_06\_07"

when (p.probability >= 0.7 and p.probability < 0.8 ) then "Proba\_07\_08"

when (p.probability >= 0.8 and p.probability < 0.9 ) then "Proba\_08\_09"

when (p.probability >= 0.9 and p.probability <= 1 ) then "Proba\_09\_10"

else "NC"

end as proba\_class

from predoutput p

""").groupBy("kpi\_name", "days", "prediction", "label").pivot("proba\_class").count()

sqlCtx.registerDataFrameAsTable(queryDF1.filter(queryDF1.prediction == "Degraded"), "degradeddf")

queryTemp = """

select kpi\_name, days, sum(degradeddf.Proba\_05\_06) as Proba\_05\_06, sum(degradeddf.Proba\_06\_07) as Proba\_06\_07,

sum(degradeddf.Proba\_07\_08) as Proba\_07\_08, sum(degradeddf.Proba\_08\_09) as Proba\_08\_09, sum(degradeddf.Proba\_09\_10) as Proba\_09\_10

from degradeddf

group by kpi\_name, days

"""

firstDF = queryDF1.filter(queryDF1.prediction == "Degraded").drop("prediction").filter(queryDF1.label == "Degraded").drop("label").withColumn("analysis",sqlF.lit("Actually Predicted"))

rangeDF = sqlCtx.sql(queryTemp).withColumn("analysis",sqlF.lit("Predicted By System")).unionAll(firstDF)

sqlCtx.registerDataFrameAsTable(rangeDF, "rangeDF")

precDF = sqlCtx.sql(""" select kpi\_name, days, round(100 \* Proba\_05\_06 /lag(Proba\_05\_06,1) over (partition by kpi\_name, days order by days, analysis desc)) as Proba\_05\_06,

round(100 \* Proba\_06\_07 /lag(Proba\_06\_07,1) over (partition by kpi\_name, days order by days, analysis desc)) as Proba\_06\_07,

round(100 \* Proba\_07\_08 /lag(Proba\_07\_08,1) over (partition by kpi\_name, days order by days, analysis desc )) as Proba\_07\_08,

round(100 \* Proba\_08\_09 /lag(Proba\_08\_09,1) over (partition by kpi\_name, days order by days, analysis desc)) as Proba\_08\_09,

round(100 \* Proba\_09\_10 /lag(Proba\_09\_10,1) over (partition by kpi\_name, days order by days, analysis desc)) as Proba\_09\_10

from rangeDF """

).dropna(thresh=3).withColumn("analysis",sqlF.lit("Precision (%)"))

rgDF1 = rangeDF.unionAll(precDF).withColumn("global\_precision", sqlF.round(sqlF.lit(100 \* float(InputPred.where((InputPred.prediction == "Degraded") & (InputPred.label == "Degraded")).count()) / InputPred.where(InputPred.prediction == "Degraded").count()), 0)).withColumn("metal",sqlF.lit("range\_overall"))

""" metal ranges prediction """

queryDF2 = sqlCtx.sql("""select p.kpi\_name, p.days, p.metal, p.prediction, p.label,

case

when (p.probability >= 0.5 and p.probability < 0.6 ) then "Proba\_05\_06"

when (p.probability >= 0.6 and p.probability < 0.7 ) then "Proba\_06\_07"

when (p.probability >= 0.7 and p.probability < 0.8 ) then "Proba\_07\_08"

when (p.probability >= 0.8 and p.probability < 0.9 ) then "Proba\_08\_09"

when (p.probability >= 0.9 and p.probability <= 1 ) then "Proba\_09\_10"

else "NC"

end as proba\_class

from predoutput p

""").groupBy("kpi\_name", "days", "metal", "prediction", "label").pivot("proba\_class").count()

sqlCtx.registerDataFrameAsTable(queryDF2.filter(queryDF2.prediction == "Degraded"), "degradeddf")

queryTemp = """

select kpi\_name, days, metal, sum(degradeddf.Proba\_05\_06) as Proba\_05\_06, sum(degradeddf.Proba\_06\_07) as Proba\_06\_07,

sum(degradeddf.Proba\_07\_08) as Proba\_07\_08, sum(degradeddf.Proba\_08\_09) as Proba\_08\_09, sum(degradeddf.Proba\_09\_10) as Proba\_09\_10

from degradeddf

group by kpi\_name, days, metal

"""

firstDF = queryDF2.filter(queryDF2.prediction == "Degraded").drop("prediction").filter(queryDF2.label == "Degraded").drop("label").withColumn("analysis",sqlF.lit("Actually Predicted"))

rangeDF = sqlCtx.sql(queryTemp).withColumn("analysis",sqlF.lit("Predicted By System")).unionAll(firstDF)

sqlCtx.registerDataFrameAsTable(rangeDF, "rangeDF")

precDF = sqlCtx.sql(""" select kpi\_name, days, metal,

round(100 \* Proba\_05\_06 /lag(Proba\_05\_06,1) over (partition by kpi\_name, days, metal order by days, metal, analysis desc)) as Proba\_05\_06,

round(100 \* Proba\_06\_07 /lag(Proba\_06\_07,1) over (partition by kpi\_name, days, metal order by days, metal, analysis desc)) as Proba\_06\_07,

round(100 \* Proba\_07\_08 /lag(Proba\_07\_08,1) over (partition by kpi\_name, days, metal order by days, metal, analysis desc)) as Proba\_07\_08,

round(100 \* Proba\_08\_09 /lag(Proba\_08\_09,1) over (partition by kpi\_name, days, metal order by days, metal, analysis desc)) as Proba\_08\_09,

round(100 \* Proba\_09\_10 /lag(Proba\_09\_10,1) over (partition by kpi\_name, days, metal order by days, metal, analysis desc)) as Proba\_09\_10

from rangeDF """

).dropna(thresh=4).withColumn("analysis",sqlF.lit("Precision (%)"))

rgDF2 = rangeDF.unionAll(precDF).withColumn("global\_precision", sqlF.round(sqlF.lit(100 \* float(InputPred.where((InputPred.prediction == "Degraded") & (InputPred.label == "Degraded")).count()) / InputPred.where(InputPred.prediction == "Degraded").count()), 0))

resultDF = rgDF2.unionAll(rgDF1[rgDF1.columns[:2] + ['metal'] + rgDF1.columns[2:9]])

sqlCtx.registerDataFrameAsTable(resultDF, "tempdf\_gb")

#Computing of global range

globalRG1 = sqlCtx.sql("""select kpi\_name,

round(100 \* Proba\_05\_06 /lag(Proba\_05\_06,1) over (partition by kpi\_name, global\_precision order by analysis desc)) as Proba\_05\_06,

round(100 \* Proba\_06\_07 /lag(Proba\_06\_07,1) over (partition by kpi\_name, global\_precision order by analysis desc)) as Proba\_06\_07,

round(100 \* Proba\_07\_08 /lag(Proba\_07\_08,1) over (partition by kpi\_name, global\_precision order by analysis desc)) as Proba\_07\_08,

round(100 \* Proba\_08\_09 /lag(Proba\_08\_09,1) over (partition by kpi\_name, global\_precision order by analysis desc)) as Proba\_08\_09,

round(100 \* Proba\_09\_10 /lag(Proba\_09\_10,1) over (partition by kpi\_name, global\_precision order by analysis desc)) as Proba\_09\_10, analysis, global\_precision

from (

select kpi\_name, sum(tempdf\_gb.Proba\_05\_06) as Proba\_05\_06, sum(tempdf\_gb.Proba\_06\_07) as Proba\_06\_07,

sum(tempdf\_gb.Proba\_07\_08) as Proba\_07\_08, sum(tempdf\_gb.Proba\_08\_09) as Proba\_08\_09, sum(tempdf\_gb.Proba\_09\_10) as Proba\_09\_10, analysis, global\_precision

from tempdf\_gb

where metal == "range\_overall" and analysis != "Precision (%)"

group by kpi\_name, analysis, global\_precision ) """

).dropna(thresh = 4).withColumn("analysis", sqlF.lit("Precision (%)"))

globalRG2 = sqlCtx.sql("""select kpi\_name, sum(tempdf\_gb.Proba\_05\_06) as Proba\_05\_06, sum(tempdf\_gb.Proba\_06\_07) as Proba\_06\_07,

sum(tempdf\_gb.Proba\_07\_08) as Proba\_07\_08, sum(tempdf\_gb.Proba\_08\_09) as Proba\_08\_09, sum(tempdf\_gb.Proba\_09\_10) as Proba\_09\_10, analysis, global\_precision

from tempdf\_gb

where metal == "range\_overall" and analysis != "Precision (%)"

group by kpi\_name, analysis, global\_precision""")

globalRG3 = globalRG2.unionAll(globalRG1).withColumn("days", sqlF.lit(currentDate).cast("date")).withColumn("metal", sqlF.lit("range\_global"))

resultFinalTemp = resultDF.unionAll(globalRG3[['kpi\_name'] + globalRG3.columns[8:10] + globalRG3.columns[1:8]])

resultFinal = resultFinalTemp.orderBy(resultFinalTemp.days.asc(), resultFinalTemp.metal.asc(), resultFinalTemp.analysis.asc())

resultFinal.toDF(\*[c.lower() for c in resultFinal.columns]).write.format("org.apache.spark.sql.cassandra").mode("append")\

.options(table="predictions\_ranges", keyspace="{}".format(cassandraKeyspace)).save()

"""

resultFinal.coalesce(1).write.format('com.databricks.spark.csv').mode('overwrite').save("s3a://data-analytics/po-csd/data\_mining/Batch\_2/Batch\_2\_cassandra\_based/{}/backup/output/range\_computing/{}".format(kpi\_name, re.sub(r'\-','/', currentDate)))"""

print("{} range computing completed! \n".format(kpi\_name))

**Output Proba. Ranges Results to Cassandra**

%pyspark

path = "s3a://csd-cell-site-degradation/development/csd-telefonica"

kpi\_names, cassandraKeyspace, s3Setting = setEnvVars(path)

kpi\_names= {"3g\_bpcr\_ps\_and\_hsdpa":901,"3g\_bpcr\_voice":902,"3g\_dpcr\_hsdpa":903,"3g\_dpcr\_voice":904,"3g\_estimated\_hsdpa\_user\_throughput\_kbps":905}

for k in range(len(kpi\_names.keys())):

produceProbaRanges(kpi\_names.keys()[k], cassandraKeyspace.get("keyspace"))

%pyspark

rawDailyDF = spark.read\

.format("org.apache.spark.sql.cassandra")\

.options(table="predictions\_ranges", keyspace="ava\_ks\_350").load()

rawDailyDF.coalesce(1).write.format('com.databricks.spark.csv').mode('overwrite').save(path + "/data-lake/data\_science/Results/predictions\_ranges", header='true')

%pyspark

rawDailyDF = spark.read\

.format("org.apache.spark.sql.cassandra")\

.options(table="pred\_tmp\_hsdpa\_user\_throughput\_kbps\_test", keyspace="ava\_ks\_350").load()

rawDailyDF.coalesce(1).write.format('com.databricks.spark.csv').mode('overwrite').save(path + "/data-lake/models/temp/predictions\_hsdpa\_user\_throughput\_kbps", header='true')

%pyspark

rawDailyDF = spark.read\

.format("org.apache.spark.sql.cassandra")\

.options(table="pred\_tmp\_hsdpa\_user\_throughput\_kbps\_test", keyspace="ava\_ks\_350").load()

%cassandra

USE ava\_ks\_350;

truncate table predictions\_ranges;

%pyspark

rawDailyDF = spark.read\

.format("org.apache.spark.sql.cassandra")\

.options(table="pred\_tmp\_3g\_bpcr\_ps\_and\_hsdpa", keyspace="ava\_ks\_350").load().show()