**Functions Definition**

%pyspark

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

import \_\_builtin\_\_ as builtins

import sys, re

from pyspark.sql.functions import \*

from pyspark.sql.types import \*

from pyspark.ml.classification import DecisionTreeClassifier, DecisionTreeClassificationModel

from pyspark.ml.feature import VectorAssembler, StringIndexer, IndexToString, StringIndexerModel

from pyspark.sql.types import DoubleType

from pyspark.sql.context import HiveContext

import pyspark.sql.functions as sqlF

sqlCtx = HiveContext(sc)

# 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")

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")

#-\*- Daily Predictions -\*-

def prediction\_rule\_extraction(line\_test\_rdd, rules, count\_rules):

f =0

res = "not found"

for i in range(count\_rules):

c = str(rules[i]["rule\_condition"])

if eval(eval(c)):

res=rules[i]["rule\_name"]

break

else: continue

return ((list(line\_test\_rdd)+[res]))

def producePrediction(kpi\_name=None, kpi\_id=None, cassandraKeyspace=None, path=None):

"""

Produce daily predictions from loaded KPI model on Cassandra input data.

"""

getproba = sqlF.udf(lambda prob: builtins.round(builtins.max(prob), 2), DoubleType())

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

# rawDailyDF = sqlCtx.read\

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

# .options(table="daily\_tmp\_{}".format(kpi\_name), keyspace="{}".format(cassandraKeyspace)).load()

rawDailyDF = sqlCtx.read.parquet("{}/data-lake/dm\_inputs/gteq2/15days/input\_test\_tmp\_{}\_15days\_gteq2".format(path,kpi\_name))

rawDailyDF.select("reading\_date","source\_id", "source\_description", "kpi\_name", "kpi\_value")\

.write.format("org.apache.spark.sql.cassandra").mode("append").options(table="daily\_kpis\_threshold\_value", keyspace="{}".format(cassandraKeyspace)).save()

predList = [l.lower() for l in rawDailyDF.rdd.map(lambda line: sorted(line.predictedkpicollist.keys())).first()]

colList = ["kpi\_id", "reading\_date", "source\_id", "kpi\_name", "kpi\_value", "metal", "site\_id", "source\_description"] + [str(l) for l in predList] + ["actual\_class"]

outputList = colList[:8 - len(colList)]

inputDaily = rawDailyDF.rdd.map(lambda line: [line.kpi\_id, line.reading\_date, line.source\_id, line.kpi\_name, line.kpi\_value, line.metal, line.site\_id, line.source\_description] + [v for (k, v) in sorted(line.predictedkpicollist.items())] + [line.actual\_class])

predDF = sqlCtx.createDataFrame(inputDaily, colList, 0.1)

labelIndexer = StringIndexerModel.load("{}/data-lake/models/{}/batch1/cassandra\_based/labelIndexer".format(path, kpi\_name))

assembler = VectorAssembler.load("{}/data-lake/models/{}/batch1/cassandra\_based/vectorAssembler".format(path, kpi\_name))

cvModel = DecisionTreeClassificationModel.load("{}/data-lake/models/{}/batch1/cassandra\_based/model".format(path, kpi\_name))

pred = cvModel.transform(assembler.transform(predDF)).drop("features", "rawPrediction")

inverterLabel = IndexToString(inputCol="prediction", outputCol="predicted\_class", labels=labelIndexer.labels)

predOut = inverterLabel.transform(pred.withColumn("probability\_of\_prediction", getproba(pred.probability)).drop("probability"))

# Rules Computing

rules = sqlCtx.read.load("{}/data-lake/models/{}/batch1/cassandra\_based/rules\_output/temp\_rules/\*.csv".format(path, kpi\_name),

format="com.databricks.spark.csv",

header="true",

inferschema="true")

rules1 = rules.rdd.collect()

count\_rules = len(rules1)

tempList = predOut.columns + ["rules"]

predOutput = predOut.rdd.map(lambda x:prediction\_rule\_extraction(x,rules1, count\_rules)).toDF(tempList).select(outputList + ["actual\_class", "predicted\_class", "probability\_of\_prediction","rules"])

# Output to Cassandra

print("Output to Cassandra")

predOutput.select("reading\_date", "site\_id", "metal", "source\_id", "source\_description", "kpi\_name", "actual\_class", "predicted\_class", "probability\_of\_prediction", "rules")\

.write.format("org.apache.spark.sql.cassandra").mode("overwrite").options(table="pred\_tmp\_{}".format(kpi\_name), keyspace="{}".format(cassandraKeyspace)).save()

predOutput.select("reading\_date", "source\_id", "kpi\_id", "kpi\_name", "source\_description", "actual\_class", "predicted\_class", "probability\_of\_prediction", "rules", "metal")\

.write.format("org.apache.spark.sql.cassandra").mode("append").options(table="daily\_predictions", keyspace="{}".format(cassandraKeyspace)).save()

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

#-\*- Predicted Alerts List -\*-

def applyThresholds(cassandraKeyspace=None):

""" Applied thresholds values """

if cassandraKeyspace != None:

prev1\_DF = sqlCtx.read.format("org.apache.spark.sql.cassandra").options(table="daily\_kpis\_threshold\_value", keyspace="{}".format(cassandraKeyspace)).load()

print("\*\*\*\*\*\*Start\*\*\*\*\*")

current= re.sub(r'(, )','-', re.sub(r"^(Row\(reading\_date=datetime.date\()|(\)\))$",'',str(sqlCtx.sql("select reading\_date from tempdf").first())))

print(current)

prev2\_DF =prev1\_DF.withColumn("current\_date", lit("{}".format(current))).withColumn("current", col("current\_date").cast("date")).drop("current\_date")

prev3\_DF = prev2\_DF.filter( (datediff(prev2\_DF.current, prev2\_DF.reading\_date) == 0) | (datediff(prev2\_DF.current, prev2\_DF.reading\_date) == 1) |(datediff(prev2\_DF.current, prev2\_DF.reading\_date) == 2) | (datediff(prev2\_DF.current, prev2\_DF.reading\_date) == 3) | (datediff(prev2\_DF.current, prev2\_DF.reading\_date) == 4)).withColumn("date\_values", when(datediff(prev2\_DF.current, prev2\_DF.reading\_date) == 1, "previous\_value").when(datediff(prev2\_DF.current, prev2\_DF.reading\_date) == 2, "previous\_2\_value").when(datediff(prev2\_DF.current, prev2\_DF.reading\_date) == 3, "previous\_3\_value").when(datediff(prev2\_DF.current, prev2\_DF.reading\_date) == 4, "previous\_4\_value").otherwise("last\_value")).drop("current")

sqlCtx.registerDataFrameAsTable(prev3\_DF.groupBy("source\_id", "kpi\_name").pivot("date\_values").avg("kpi\_value"), "prev\_valuesDF")

prev5\_DF = sqlCtx.sql("""

select t.\*, p.last\_value, p.previous\_value, p.previous\_2\_value, p.previous\_3\_value, p.previous\_4\_value,

case

when ((t.kpi\_threshold\_value is not null ) and (t.kpi\_threshold\_value <= p.last\_value and t.kpi\_threshold\_value <= p.previous\_value )) then "Degraded"

else 'Non-Degraded'

end as decision\_classs,

case

when ((t.kpi\_threshold\_value is not null ) and ( t.kpi\_threshold\_value <= p.last\_value)) and (lower(t.kpi\_name) != '3g\_estimated\_hsdpa\_user\_throughput\_kbps') then "Degraded"

when ((t.kpi\_threshold\_value is not null ) and ( t.kpi\_threshold\_value >= p.last\_value)) and (lower(t.kpi\_name) == '3g\_estimated\_hsdpa\_user\_throughput\_kbps') then "Degraded"

else 'Non-Degraded'

end as current,

case

when ((t.kpi\_threshold\_value is not null ) and (t.kpi\_threshold\_value <= p.previous\_value)) and (lower(t.kpi\_name) != '3g\_estimated\_hsdpa\_user\_throughput\_kbps') then "Degraded"

when ((t.kpi\_threshold\_value is not null ) and ( t.kpi\_threshold\_value >= p.previous\_value)) and (lower(t.kpi\_name) == '3g\_estimated\_hsdpa\_user\_throughput\_kbps') then "Degraded"

else 'Non-Degraded'

end as prev,

case

when ((t.kpi\_threshold\_value is not null ) and (t.kpi\_threshold\_value <= p.previous\_2\_value)) and (lower(t.kpi\_name) != '3g\_estimated\_hsdpa\_user\_throughput\_kbps') then "Degraded"

when ((t.kpi\_threshold\_value is not null ) and ( t.kpi\_threshold\_value >= p.previous\_2\_value)) and (lower(t.kpi\_name) == '3g\_estimated\_hsdpa\_user\_throughput\_kbps') then "Degraded"

else 'Non-Degraded'

end as prev\_2\_days,

case

when ((t.kpi\_threshold\_value is not null ) and (t.kpi\_threshold\_value <= p.previous\_3\_value)) and (lower(t.kpi\_name) != '3g\_estimated\_hsdpa\_user\_throughput\_kbps') then "Degraded"

when ((t.kpi\_threshold\_value is not null ) and ( t.kpi\_threshold\_value >= p.previous\_3\_value)) and (lower(t.kpi\_name) == '3g\_estimated\_hsdpa\_user\_throughput\_kbps') then "Degraded"

else 'Non-Degraded'

end as prev\_3\_days,

case

when ((t.kpi\_threshold\_value is not null ) and (t.kpi\_threshold\_value <= p.previous\_4\_value)) and (lower(t.kpi\_name) != '3g\_estimated\_hsdpa\_user\_throughput\_kbps') then "Degraded"

when ((t.kpi\_threshold\_value is not null ) and ( t.kpi\_threshold\_value >= p.previous\_4\_value)) and (lower(t.kpi\_name) == '3g\_estimated\_hsdpa\_user\_throughput\_kbps') then "Degraded"

else 'Non-Degraded'

end as prev\_4\_days

from tempdf t join prev\_valuesdf p

on ( t.source\_id == p.source\_id and lower(t.kpi\_name) == lower(p.kpi\_name))

where (p.last\_value is not null and p.previous\_value is not null and p.previous\_2\_value is not null and p.previous\_3\_value is not null and p.previous\_4\_value is not null)

""")

sqlCtx.dropTempTable("tempdf")

prev5\_DF.printSchema()

sqlCtx.registerDataFrameAsTable(prev5\_DF.where((prev5\_DF.current == "Non-Degraded") & (prev5\_DF.prev == "Non-Degraded") & (prev5\_DF.prev\_2\_days == "Non-Degraded") & (prev5\_DF.prev\_3\_days == "Non-Degraded") & (prev5\_DF.prev\_4\_days == "Non-Degraded")), "tempdf")

#sqlCtx.registerDataFrameAsTable(prev5\_DF.where((prev5\_DF.current == "Non-Degraded")), "tempdf")

sqlCtx.dropTempTable("prev\_valuesdf")

else:

print("\nPlease give previous values table name in the keyspace {}".format(cassandraKeyspace), file=sys.stderr)

sqlCtx.dropTempTable("tempdf")

def checkPastdAlerts(current\_alert\_data=None, cassandraKeyspace=None):

"""

This function checks on a weekly basis KPIs alerts already raised.

"""

if (current\_alert\_data != None) & (cassandraKeyspace != None):

dataDate = current\_alert\_data.select(current\_alert\_data.reading\_date, date\_sub(current\_alert\_data.reading\_date, 7).alias('b\_date'), date\_sub(current\_alert\_data.reading\_date, 1).alias('e\_date')).limit(3)

start\_date= str(dataDate.select(date\_format('b\_date', 'yyy-MM-dd').alias('b\_date')).rdd.map(lambda l: l.b\_date).take(1)[0])

end\_date= str(dataDate.select(date\_format('e\_date', 'yyy-MM-dd').alias('e\_date')).rdd.map(lambda l: l.e\_date).take(1)[0])

alertHistData1 = sqlCtx.read.format("org.apache.spark.sql.cassandra").options(table="alert\_data", keyspace=cassandraKeyspace).load().where(col("reading\_date").between(start\_date, end\_date))

if alertHistData1.limit(1).count() == 1:

sqlCtx.registerDataFrameAsTable(alertHistData1.select("source\_description"), "alerthist")

sqlCtx.registerDataFrameAsTable(current\_alert\_data, "current\_alert\_data")

alertDF = sqlCtx.sql("""select c.\*

from current\_alert\_data c

where c.source\_description not in

(select source\_description

from alerthist) """)

sqlCtx.dropTempTable("alerthist")

sqlCtx.dropTempTable("current\_alert\_data")

return alertDF

else:

return current\_alert\_data

def predictedAlerts(kpi\_name=None, cassandraKeyspace=None, selectedClass=None, selectedThreshold=None, selectedRange=None, selectedCategory=None, selectedRules=None,start\_date=None, end\_date=None):

"""

Process predicted alert lists on each KPI.

"""

inputDF = sqlCtx.read.format("org.apache.spark.sql.cassandra").options(table="pred\_tmp\_{}".format(kpi\_name), keyspace="{}".format(cassandraKeyspace)).load()\

.where("reading\_date between \'"+ start\_date +"\' and \'"+ end\_date + "\' ")

""" Filtering on Class and Rules """

if (selectedClass != None):

if (selectedRules != None):

if isinstance(selectedRules, str):

sqlCtx.registerDataFrameAsTable(inputDF.filter("predicted\_class = {} and rules in ('{}')".format(selectedClass, selectedRules).replace( "\'",'"' )), "inputTable")

else:

sqlCtx.registerDataFrameAsTable(inputDF.filter("predicted\_class = {} and rules in {}".format(selectedClass, selectedRules).replace( "\'",'"' )), "inputTable")

else:

sqlCtx.registerDataFrameAsTable(inputDF.filter("predicted\_class = {}".format(selectedClass)), "inputTable")

else :

sqlCtx.registerDataFrameAsTable(inputDF, "inputTable")

""" Filtering on range probability of prediction """

if isinstance(selectedRange[0], float) :

if selectedRange[1] == 1.0 :

queryClass = "select reading\_date, site\_id, metal, source\_id,source\_description, kpi\_name, actual\_class, predicted\_class, Cast(probability\_of\_prediction as double) as probability\_of\_prediction, rules from inputTable where probability\_of\_prediction >= {} and probability\_of\_prediction <= {}".format(selectedRange[0], selectedRange[1])

queryDF = sqlCtx.sql(queryClass)

else:

queryClass = "select reading\_date, site\_id, metal, source\_id,source\_description, kpi\_name, actual\_class, predicted\_class, Cast(probability\_of\_prediction as double) as probability\_of\_prediction, rules from inputTable where probability\_of\_prediction >= {} and probability\_of\_prediction < {}".format(selectedRange[0], selectedRange[1])

queryDF = sqlCtx.sql(queryClass)

else :

if selectedRange[0][1] == 1.0 :

queryClass = "select reading\_date, site\_id, metal, source\_id, source\_description, kpi\_name, actual\_class, predicted\_class, Cast(probability\_of\_prediction as double) as probability\_of\_prediction, rules from inputTable where probability\_of\_prediction >= {} and probability\_of\_prediction <= {}".format(selectedRange[0][0], selectedRange[0][1])

queryDF = sqlCtx.sql(queryClass)

else:

queryClass = "select reading\_date, site\_id, metal, source\_id, source\_description, kpi\_name, actual\_class, predicted\_class, Cast(probability\_of\_prediction as double) as probability\_of\_prediction, rules from inputTable where probability\_of\_prediction >= {} and probability\_of\_prediction < {}".format(selectedRange[0][0], selectedRange[0][1])

queryDF = sqlCtx.sql(queryClass)

for i in range(1, len(selectedRange)):

if selectedRange[i][1] == 1.0 :

queryTemp = "select reading\_date, site\_id, metal, source\_id,source\_description, kpi\_name, actual\_class, predicted\_class, Cast(probability\_of\_prediction as double) as probability\_of\_prediction, rules from inputTable where probability\_of\_prediction >= {} and probability\_of\_prediction <= {}".format(selectedRange[i][0], selectedRange[i][1])

queryDF = queryDF.unionAll(sqlCtx.sql(queryTemp))

else:

queryTemp = "select reading\_date, site\_id, metal, source\_id,source\_description, kpi\_name, actual\_class, predicted\_class, Cast(probability\_of\_prediction as double) as probability\_of\_prediction, rules from inputTable where probability\_of\_prediction >= {} and probability\_of\_prediction < {}".format(selectedRange[i][0], selectedRange[i][1])

queryDF = queryDF.unionAll(sqlCtx.sql(queryTemp))

sqlCtx.dropTempTable("inputTable")

sqlCtx.registerDataFrameAsTable(queryDF, "df\_tofilter")

""" Filtering on category """

for i in range(len(selectedCategory.values())):

if i == 0:

if selectedCategory.values()[i] == None:

queryTemp = "select \* from df\_tofilter where metal = '{}'".format(str(selectedCategory.keys()[i]))

sqlCtx.registerDataFrameAsTable(sqlCtx.sql(queryTemp), "temptab")

else:

rgIter = iter(selectedCategory.values()[i]).next()

if isinstance(rgIter, tuple):

if float(selectedCategory.values()[i][0][1]) == 1.0 :

queryTemp = "select \* from df\_tofilter where metal = '{}' and ( probability\_of\_prediction >= {} and probability\_of\_prediction <= {} )".format( selectedCategory.keys()[i], float(selectedCategory.values()[i][0][0]), float(selectedCategory.values()[i][0][1]))

sqlCtx.registerDataFrameAsTable(sqlCtx.sql(queryTemp), "temptab")

else:

queryTemp = "select \* from df\_tofilter where metal = '{}' and ( probability\_of\_prediction >= {} and probability\_of\_prediction < {} )".format( selectedCategory.keys()[i], float(selectedCategory.values()[i][0][0]), float(selectedCategory.values()[i][0][1]))

sqlCtx.registerDataFrameAsTable(sqlCtx.sql(queryTemp), "temptab")

for k in range(1, len(selectedCategory.values()[i])):

if float(selectedCategory.values()[i][k][1]) == 1.0 :

queryTemp = "select \* from df\_tofilter where metal = '{}' and probability\_of\_prediction >= {} and probability\_of\_prediction <= {}".format( selectedCategory.keys()[i], float(selectedCategory.values()[i][k][0]), float(selectedCategory.values()[i][k][1]))

queryDF = sqlCtx.sql("select \* from temptab").unionAll(sqlCtx.sql(queryTemp))

queryDF.createOrReplaceTempView("temptab")

else:

queryTemp = "select \* from df\_tofilter where metal = '{}' and probability\_of\_prediction >= {} and probability\_of\_prediction < {}".format( selectedCategory.keys()[i], float(selectedCategory.values()[i][k][0]), float(selectedCategory.values()[i][k][1]))

queryDF = sqlCtx.sql("select \* from temptab").unionAll(sqlCtx.sql(queryTemp))

queryDF.createOrReplaceTempView("temptab")

else:

if float(selectedCategory.values()[i][1]) == 1.0 :

queryTemp = "select \* from df\_tofilter where metal = '{}' and ( probability\_of\_prediction >= {} and probability\_of\_prediction <= {} )".format( selectedCategory.keys()[i], selectedCategory.values()[i][0], selectedCategory.values()[i][1])

sqlCtx.registerDataFrameAsTable(sqlCtx.sql(queryTemp), "temptab")

else:

queryTemp = "select \* from df\_tofilter where metal = '{}' and ( probability\_of\_prediction >= {} and probability\_of\_prediction < {} )".format( selectedCategory.keys()[i], selectedCategory.values()[i][0], selectedCategory.values()[i][1])

sqlCtx.registerDataFrameAsTable(sqlCtx.sql(queryTemp), "temptab")

else:

if selectedCategory.values()[i] == None:

queryTemp = "select \* from df\_tofilter where metal = '{}'".format(str(selectedCategory.keys()[i]))

queryDF = sqlCtx.sql("select \* from temptab").unionAll(sqlCtx.sql(queryTemp))

queryDF.createOrReplaceTempView("temptab")

else:

rgIter = iter(selectedCategory.values()[i]).next()

if isinstance(rgIter, tuple):

if float(selectedCategory.values()[i][0][1]) == 1.0 :

queryTemp = "select \* from df\_tofilter where metal = '{}' and ( probability\_of\_prediction >= {} and probability\_of\_prediction <= {} )".format( selectedCategory.keys()[i], float(selectedCategory.values()[i][0][0]), float(selectedCategory.values()[i][0][1]))

queryDF = sqlCtx.sql("select \* from temptab").unionAll(sqlCtx.sql(queryTemp))

queryDF.createOrReplaceTempView("temptab")

else:

queryTemp = "select \* from df\_tofilter where metal = '{}' and ( probability\_of\_prediction >= {} and probability\_of\_prediction < {} )".format( selectedCategory.keys()[i], float(selectedCategory.values()[i][0][0]), float(selectedCategory.values()[i][0][1]))

queryDF = sqlCtx.sql("select \* from temptab").unionAll(sqlCtx.sql(queryTemp))

queryDF.createOrReplaceTempView("temptab")

for k in range(1, len(selectedCategory.values()[i])):

if float(selectedCategory.values()[i][k][1]) == 1.0 :

queryTemp = "select \* from df\_tofilter where metal = '{}' and probability\_of\_prediction >= {} and probability\_of\_prediction <= {}".format( selectedCategory.keys()[i], float(selectedCategory.values()[i][k][0]), float(selectedCategory.values()[i][k][1]))

queryDF = sqlCtx.sql("select \* from temptab").unionAll(sqlCtx.sql(queryTemp))

queryDF.createOrReplaceTempView("temptab")

else:

queryTemp = "select \* from df\_tofilter where metal = '{}' and probability\_of\_prediction >= {} and probability\_of\_prediction < {}".format( selectedCategory.keys()[i], float(selectedCategory.values()[i][k][0]), float(selectedCategory.values()[i][k][1]))

queryDF = sqlCtx.sql("select \* from temptab").unionAll(sqlCtx.sql(queryTemp))

queryDF.createOrReplaceTempView("temptab")

else:

if float(selectedCategory.values()[i][1]) == 1.0 :

queryTemp = "select \* from df\_tofilter where metal = '{}' and ( probability\_of\_prediction >= {} and probability\_of\_prediction <= {} )".format( selectedCategory.keys()[i], selectedCategory.values()[i][0], selectedCategory.values()[i][1])

queryDF = sqlCtx.sql("select \* from temptab").unionAll(sqlCtx.sql(queryTemp))

queryDF.createOrReplaceTempView("temptab")

else:

queryTemp = "select \* from df\_tofilter where metal = '{}' and ( probability\_of\_prediction >= {} and probability\_of\_prediction < {} )".format( selectedCategory.keys()[i], selectedCategory.values()[i][0], selectedCategory.values()[i][1])

queryDF = sqlCtx.sql("select \* from temptab").unionAll(sqlCtx.sql(queryTemp))

queryDF.createOrReplaceTempView("temptab")

""" Update of kpi alert lists table """

if ('tempdf' not in sqlCtx.tableNames()):

sqlCtx.registerDataFrameAsTable(sqlCtx.sql("select \* from temptab").withColumn("kpi\_threshold\_value", lit("{}".format(selectedThreshold))), "tempdf")

sqlCtx.dropTempTable("temptab")

sqlCtx.dropTempTable("df\_tofilter")

else:

finalDF = sqlCtx.sql("select \* from tempdf")

finalDF = finalDF.unionAll(sqlCtx.sql("select \* from temptab").withColumn("kpi\_threshold\_value", lit("{}".format(selectedThreshold))))

sqlCtx.dropTempTable("tempdf")

sqlCtx.dropTempTable("temptab")

sqlCtx.dropTempTable("df\_tofilter")

sqlCtx.registerDataFrameAsTable(finalDF, "tempdf")

**Run Daily Prediction on all KPIs**

%pyspark

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

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

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

producePrediction(kpi\_names.keys()[k], str(kpi\_names.values()[k]), cassandraKeyspace.get("keyspace"), path)

**Run Predicted Alerts List**

%pyspark

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

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}

start\_date="2018-10-29"

end\_date="2018-11-23"

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

pclass = re.compile(r'^class=')

pkpi\_threshold\_value = re.compile(r'^kpi\_threshold\_value=')

pthreshold\_sign = re.compile(r'^threshold\_sign=')

pproba\_pred\_range = re.compile(r'^proba\_pred\_range=')

pcategory = re.compile(r'^category=')

prules= re.compile(r'^rules=')

""" Mapping of conf files for alert list """

confFiles = sc.wholeTextFiles("{}/config/alert\_list/daily\_kpis\_conf".format(path))

confRDD = confFiles.map(lambda kip: (kip[0].split("daily\_kpis\_conf/")[1].split(".conf")[0], (kip[1].split("\n"))))

confList = confRDD.collect()

mapDataConf = []

for i in range(len(confList)):

s\_cl = str(confList[i][0])

mapDataConf.append([s\_cl.lower(), confList[i][1]])

""" Processing of each file data following its conf file """

for dataCf in mapDataConf:

for conff in dataCf[1]:

s\_conf = str(conff)

if pclass.match(s\_conf):

selectedClass = pclass.sub('', s\_conf).strip()

elif pkpi\_threshold\_value.match(s\_conf):

selectedThreshold = eval(pkpi\_threshold\_value.sub('', s\_conf))

elif pthreshold\_sign.match(s\_conf):

selectedThreshold\_sign = eval(pthreshold\_sign.sub('', s\_conf))

elif pproba\_pred\_range.match(s\_conf):

selectedRange = eval(pproba\_pred\_range.sub('', s\_conf))

elif pcategory.match(s\_conf):

selectedCategory = eval(pcategory.sub('', s\_conf))

elif prules.match(s\_conf):

selectedRules = eval(prules.sub('', s\_conf))

exec('predictedAlerts("{}", cassandraKeyspace.get("keyspace"), selectedClass, selectedThreshold, selectedRange, selectedCategory, selectedRules, start\_date, end\_date)'.format(dataCf[0]))

selectedClass=None

selectedRange=None

selectedCategory=None

selectedRules=None

selectedThreshold=None

print("{} finished!".format(dataCf[0]))

finalDat = sqlCtx.sql("select \* from tempdf")

sqlCtx.registerDataFrameAsTable(finalDat, "golden\_tempdf")

sqlCtx.dropTempTable("tempdf")

%pyspark

# Ranking processing

dateRange = sorted(finalDat.select(date\_format("reading\_date", "yyy-MM-dd").alias("date")).distinct().rdd.map(lambda l: str(l.date)).collect())[1:]

for dat in dateRange[3:]:

sqlCtx.registerDataFrameAsTable(sqlCtx.sql("select \* from golden\_tempdf where reading\_date == '{}' ".format(dat)), "tempdf")

applyThresholds(cassandraKeyspace.get("keyspace"))

sqlCtx.registerDataFrameAsTable(checkPastdAlerts(sqlCtx.sql("select \* from tempdf"), cassandraKeyspace.get("keyspace")), "tempdfbis")

query = """ SELECT tp.reading\_date, tp.site\_id, tp.metal, tp.source\_id, tp.source\_description, tp.kpi\_name, tp.predicted\_class, tp.actual\_class, tp.probability\_of\_prediction, cntdf.kpi\_count, tp.prev, tp.prev\_2\_days, tp.prev\_3\_days,tp.prev\_4\_days, tp.current,

dense\_rank() over (PARTITION BY reading\_date ORDER BY kpi\_count desc, site\_id) rank, tp.rules

FROM(

SELECT tp.\*, cntdf.kpi\_count

FROM (SELECT reading\_date, site\_id, count(source\_id, kpi\_name) as kpi\_count

FROM tempdfbis

GROUP BY reading\_date, site\_id

) as cntdf RIGHT OUTER JOIN tempdfbis as tp ON (cntdf.reading\_date = tp.reading\_date and cntdf.site\_id = tp.site\_id)) """

#finalDF = sqlCtx.sql(query).orderBy(['reading\_date', 'metal','site\_id', 'rank'], ascending=[0, 0, 1, 1]).drop('kpi\_count').where("rank < 21")

finalDF = sqlCtx.sql(query).orderBy(['reading\_date', 'metal','site\_id', 'rank'], ascending=[0, 0, 1, 1]).drop('kpi\_count').where("rank < 21")

#finalDF = sqlCtx.sql(query).orderBy(['reading\_date', 'metal','site\_id', 'rank'], ascending=[0, 0, 1, 1]).drop('kpi\_count')

"""Writing results to Cassandra """

finalDF.drop("actual\_class").write\

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

.mode("append")\

.options(table="alert\_data", keyspace="{}".format(cassandraKeyspace.get("keyspace")))\

.save()

if dateRange[0] == dat:

finalDF.write\

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

.mode("overwrite")\

.options(table="pred\_list", keyspace="{}".format(cassandraKeyspace.get("keyspace")))\

.save()

else:

finalDF.write\

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

.mode("append")\

.options(table="pred\_list", keyspace="{}".format(cassandraKeyspace.get("keyspace")))\

.save()

sqlCtx.dropTempTable("tempdf")

sqlCtx.dropTempTable("tempdfbis")

print("{}\n".format(dat))

sqlCtx.dropTempTable("golden\_tempdf")

**"""Writing results to parquet"""**

**finalDF.drop("actual\_class").write\**

**.mode("append")\**

**.parquet(path + "/data-lake/data\_science/predicted\_alerts5parquet/alert\_hist\_data/")**

**if dateRange[0] == dat:**

**finalDF.partitionBy("reading\_date").write\**

**.mode("overwrite")\**

**.parquet(path + "/data-lake/data\_science/predicted\_alerts5parquet/predicted\_alerts\_list/")**

**else:**

**finalDF.write\**

**.mode("append")\**

**.parquet(path + "/data-lake/data\_science/predicted\_alerts5parquet/predicted\_alerts\_list/")**

%pyspark

rawDailyDF = spark.read\

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

.options(table="daily\_kpis\_threshold\_value", keyspace="ava\_ks\_350").load().show()

%pyspark

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

rawDailyDF = spark.read\

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

.options(table="pred\_list", keyspace="ava\_ks\_350").load()

rawDailyDF.coalesce(1).write.format('com.databricks.spark.csv').mode('overwrite').save(path + "/data-lake/data\_science/Results/alerts\_pred\_5days\_sucess/", header='true')

%cassandra

use ava\_ks\_350;

TRUNCATE ava\_ks\_350.pred\_list;

TRUNCATE ava\_ks\_350.alert\_data;

%spark

var df = spark.read.parquet("s3a://csd-cell-site-degradation/development/csd-verizon/data-lake/data\_science/output/predicted\_alerts\_list").show()

var df = spark.read.parquet("s3a://csd-cell-site-degradation/development/csd-verizon/data-lake/data\_science/output/predicted\_alerts\_list").count()