Edureka In-class Project Solution

Case Statement

Domain: Financial

Statement: A leading financial bank is trying to broaden the financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, it makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

The bank has asked you to develop a solution to ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

Understand data dictionary

The complete data dictionary is available in file HomeCredit_columns_description.csv in downloaded zip file.

For the in-class project, we are considering only following variables:

- TARGET: Target variable (1 - client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in our sample, 0 - all other cases)
- NAME_CONTRACT_TYPE: Identification if loan is cash or revolving
- CODE_GENDER: Gender of the client
- FLAG_OWN_CAR: Flag if the client owns a car
- FLAG_OWN_REALTY: Flag if client owns a house or flat
- CNT_CHILDREN: Number of children the client has
- AMT_INCOME_TOTAL: Income of the client
- AMT_CREDIT: Credit amount of the loan
- AMT_ANNUITY: Loan annuity
- NAME_INCOME_TYPE: Clients income type (businessman, working, maternity leave,)
- NAME_EDUCATION_TYPE: Level of highest education the client achieved
- NAME_FAMILY_STATUS: Family status of the client
- NAME_HOUSING_TYPE: What is the housing situation of the client (renting, living with parents, ...)
- DAYS_BIRTH: Client's age in days at the time of application
- DAYS_EMPLOYED: How many days before the application the person started current employment
- OWN_CAR_AGE: Age of client's car
- FLAG_MOBIL: Did client provide mobile phone (1=YES, 0=NO)
- FLAG_EMP_PHONE: Did client provide work phone (1=YES, 0=NO)
- FLAG_WORK_PHONE: Did client provide home phone (1=YES, 0=NO)
- FLAG_CONT_MOBILE: Was mobile phone reachable (1=YES, 0=NO)
- FLAG_PHONE: Did client provide home phone (1=YES, 0=NO)
- OCCUPATION_TYPE: What kind of occupation does the client have
- CNT_FAM_MEMBERS: How many family members does client have
- REGION_RATING_CLIENT: Our rating of the region where client lives (1,2,3)
- REGION_RATING_CLIENT_W_CITY: Our rating of the region where client lives with taking city into account (1,2,3)
- REG_REGION_NOT_LIVE_REGION: Flag if client's permanent address does not match contact address (1=different, 0=same, at region level)

- REG_REGION_NOT_WORK_REGION: Flag if client's permanent address does not match work address (1=different, 0=same, at region level)
- ORGANIZATION_TYPE: Type of organization where client works
- FLAG_DOCUMENT_2: Did client provide document 2
- FLAG_DOCUMENT_3: Did client provide document 3
- FLAG_DOCUMENT_4: Did client provide document 4
- FLAG_DOCUMENT_5: Did client provide document 5
- FLAG_DOCUMENT_6: Did client provide document 6
- FLAG_DOCUMENT_7: Did client provide document 7
- FLAG_DOCUMENT_8: Did client provide document 8
- FLAG_DOCUMENT_9: Did client provide document 9
- FLAG_DOCUMENT_10: Did client provide document 10
- FLAG_DOCUMENT_11: Did client provide document 11
- FLAG_DOCUMENT_12: Did client provide document 12
- FLAG_DOCUMENT_13: Did client provide document 13
- FLAG_DOCUMENT_14: Did client provide document 14
- FLAG_DOCUMENT_15: Did client provide document 15
- FLAG_DOCUMENT_16: Did client provide document 16
- FLAG_DOCUMENT_17: Did client provide document 17
- FLAG_DOCUMENT_18: Did client provide document 18
- FLAG_DOCUMENT_19: Did client provide document 19
- FLAG_DOCUMENT_20: Did client provide document 20
- FLAG_DOCUMENT_21: Did client provide document 21

Subset Dataset

For in-class let's keep selected columns. We will be using these columns throughout the training.
This can be done via python:

```
- import pandas as pd
- tmp = pd.read_csv("/mnt/home/edureka_321047/av/workspace/data/in_class_project/all-2/application_train.csv")
- subset = tmp[['TARGET',
  'NAME_CONTRACT_TYPE',
  'CODE_GENDER',
  'FLAG_OWN_CAR',
  'FLAG_OWN_REALTY',
  'CNT_CHILDREN',
  'AMT_INCOME_TOTAL',
  'AMT_CREDIT',
  'AMT_ANNUITY',
  'NAME_INCOME_TYPE',
  'NAME_EDUCATION_TYPE',
  'NAME_FAMILY_STATUS',
  'NAME_HOUSING_TYPE',
  'DAYS_BIRTH',
  'DAYS_EMPLOYED',
  'FLAG_MOBIL',
  'FLAG_EMP_PHONE',
  'FLAG_WORK_PHONE',
  'FLAG_CONT_MOBILE',
  'FLAG_PHONE',
  'CNT_FAM_MEMBERS',
  'REGION_RATING_CLIENT',
```

```
  'REGION_RATING_CLIENT_W_CITY',
  'REG_REGION_NOT_LIVE_REGION',
  'REG_REGION_NOT_WORK_REGION',
  'ORGANIZATION_TYPE',
  'FLAG_DOCUMENT_2',
  'FLAG_DOCUMENT_3',
  'FLAG_DOCUMENT_4',
  'FLAG_DOCUMENT_5',
  'FLAG_DOCUMENT_6',
  'FLAG_DOCUMENT_7',
  'FLAG_DOCUMENT_8',
  'FLAG_DOCUMENT_9',
  'FLAG_DOCUMENT_10',
  'FLAG_DOCUMENT_11',
  'FLAG_DOCUMENT_12',]]
```

- subset.dropna().to_csv("/mnt/home/edureka_321047/av/workspace/data/in_class_project/all-2/subset.csv", header=None,index=None)

Load data in Mysql

As part of demo let's first put data in Mysql, which we will later transfer to HDFS using Sqoop

- Login into Mysql
- Create new database: create database inclass;
- Use the same database: use inclass;
- Create table application_train

- create table application_train (TARGET int, NAME_CONTRACT_TYPE varchar(100), CODE_GENDER varchar(100), FLAG_OWN_CAR varchar(100), FLAG_OWN_REALTY varchar(100), CNT_CHILDREN int, AMT_INCOME_TOTAL double, AMT_CREDIT double, AMT_ANNUITY double, NAME_INCOME_TYPE varchar(100), NAME_EDUCATION_TYPE varchar(100), NAME_FAMILY_STATUS varchar(100), NAME_HOUSING_TYPE varchar(100), DAYS_BIRTH int, DAYS_EMPLOYED int, FLAG_MOBIL int, FLAG_EMP_PHONE int, FLAG_WORK_PHONE int, FLAG_CONT_MOBILE int, FLAG_PHONE int, CNT_FAM_MEMBERS double, REGION_RATING_CLIENT int, REGION_RATING_CLIENT_W_CITY int, REG_REGION_NOT_LIVE_REGION int, REG_REGION_NOT_WORK_REGION int, ORGANIZATION_TYPE varchar(100), FLAG_DOCUMENT_2 int, FLAG_DOCUMENT_3 int, FLAG_DOCUMENT_4 int, FLAG_DOCUMENT_5 int, FLAG_DOCUMENT_6 int, FLAG_DOCUMENT_7 int, FLAG_DOCUMENT_8 int, FLAG_DOCUMENT_9 int, FLAG_DOCUMENT_10 int, FLAG_DOCUMENT_11 int, FLAG_DOCUMENT_12 int);

- Load data in Mysql (to be executed in Mysql)
- load data local infile '/mnt/home/edureka_321047/av/workspace/data/in_class_project/all-2/subset.csv' into table application_train fields terminated BY "," lines terminated BY "\n";

Transfer Data Using Sqoop

- List Databases:
- sqoop list-databases --connect jdbc:mysql://sqoopdb.edu.cloudlab.com --username labuser --password edureka

- Transfer Data

- sqoop import --connect jdbc:mysql://sqoopdb.edu.cloudlab.com/inclass --username labuser -password edureka --table application_train -m 1 --target-dir /user/edureka_321047/inclass/sqoop/

Now we got data in HDFS let's parse the data by reading it as text file and parsing it.

Load and Parse the Dataset

```
val raw = spark.read.text("inclass/sqoop/*")

case class Data
(TARGET:Int,NAME_CONTRACT_TYPE:String,CODE_GENDER:String,FLAG_OWN_CAR:String,FLAG_O
WN_REALTY:String,CNT_CHILDREN:Int,AMT_INCOME_TOTAL:Double,AMT_CREDIT:Double,AMT_AN
NUITY:Double,NAME_EDUCATION_TYPE:String)

val df = raw.map(_.getString(0).split(",")).map(d=>Data(d(0).toInt,d(1).toString, d(2).toString,
d(3).toString, d(4).toString, d(5).toInt,d(6).toDouble, d(7).toDouble, d(8).toDouble,
d(9).toString)).toDF
```

Create new column

```
import org.apache.spark.sql.functions._

val df1 =
df.withColumn("CREDIT_INCOME_PERCENT",col("AMT_CREDIT")/col("AMT_INCOME_TOTAL"))

df1 show 1
```

Load and Parse the Dataset using DataFrames

```
val columns =
Seq("TARGET","NAME_CONTRACT_TYPE","CODE_GENDER","FLAG_OWN_CAR","FLAG_OWN_REALT
Y","CNT_CHILDREN","AMT_INCOME_TOTAL","AMT_CREDIT","AMT_ANNUITY","NAME_INCOME_TYP
E","NAME_EDUCATION_TYPE","NAME_FAMILY_STATUS","NAME_HOUSING_TYPE","DAYS_BIRTH","
DAYS_EMPLOYED","FLAG_MOBIL","FLAG_EMP_PHONE","FLAG_WORK_PHONE","FLAG_CONT_MOBI
LE","FLAG_PHONE","CNT_FAM_MEMBERS","REGION_RATING_CLIENT","REGION_RATING_CLIENT_
W_CITY","REG_REGION_NOT_LIVE_REGION","REG_REGION_NOT_WORK_REGION","ORGANIZATION
_TYPE","FLAG_DOCUMENT_2","FLAG_DOCUMENT_3","FLAG_DOCUMENT_4","FLAG_DOCUMENT_5
","FLAG_DOCUMENT_6","FLAG_DOCUMENT_7","FLAG_DOCUMENT_8","FLAG_DOCUMENT_9","FLA
G_DOCUMENT_10","FLAG_DOCUMENT_11","FLAG_DOCUMENT_12")


val data = spark.read.option("inferSchema",
true).csv("inclass/sqoop/*").limit(1000).toDF(columns:_*)
```

Cache datatset

```
data.cache()
```

Exploratory Analysis

No of loans falling into each Target with percentage

```
import org.apache.spark.sql.functions._

data.groupBy("TARGET").count().withColumn("Percentage",col("count")*100/data.count()).show()
```

Number of missing values in each column

```
val nullcounts = data.select(data.columns.map(c => sum(col(c).isNull.cast("int")).alias(c)):_*)

val totcount = data.count

nullcounts.first().toSeq.zip(data.columns).map(x=>(x._1,
"%1.2f".format(x._1.toString.toDouble*100/totcount), x._2)).foreach(println)
```

View unique values in all string columns

```
import org.apache.spark.sql.types._

val exprs = data.schema.fields.filter(x => x.dataType == StringType).map(x=>x.name ->
"approx_count_distinct").toMap

data.agg(exprs).show()
```

Describe days employed

```
data.select("DAYS_EMPLOYED").describe().show()
```

Describe days birth column

```
val dfAge = data.withColumn("AGE", col("DAYS_BIRTH")/(-365))

dfAge.select("DAYS_BIRTH","AGE").describe().show()
```

Dig deep into anomalies of DAY_EMPLOYED column

```
val anom = dfAge.filter(col("DAYS_EMPLOYED").equalTo(365243))

val non_anom = dfAge.filter(col("DAYS_EMPLOYED").notEqual(365243))
```

```
val nonanomPer = 100 * non_anom.agg(avg(col("TARGET"))).first()(0).toString.toDouble

val anomPer = 100 * anom.agg(avg(col("TARGET"))).first()(0).toString.toDouble

println(f"The non-anomalies default on $nonanomPer%2.2f while anomalies default on $anomPer%2.2f ")

val anomCount = anom.count

- print(f"There are $anomCount%d anomalous days of employment")    // no of wrong employment day column
```

Create anomaly flag column

```
val anomalyDf = dfAge.withColumn("DAYS_EMPLOYED_ANOM",col("DAYS_EMPLOYED").equalTo(365243))
```

Replace anomaly value with 0

```
val anomalyFlagDf = anomalyDf.withColumn("DAYS_EMPLOYED", when(col("DAYS_EMPLOYED") === 365243, 0).otherwise(col("DAYS_EMPLOYED")))     // if anom is 365243 convert to 0
```

Effect of age on repayment by binning the column and the generating pivot table

```
anomalyFlagDf.select("AGE").describe().show()
```

Create new variables based on domain knowledge

```
val tmpDf1 = anomalyFlagDf.withColumn("CREDIT_INCOME_PERCENT",col("AMT_CREDIT")/col("AMT_INCOME_TOTAL"))

val tmpDf2 = tmpDf1.withColumn("ANNUITY_INCOME_PERCENT",col("AMT_ANNUITY")/col("AMT_INCOME_TOTAL"))

val tmpDf3 = tmpDf2.withColumn("CREDIT_TERM",col("AMT_ANNUITY")/col("AMT_CREDIT"))

val tmpDf4 = tmpDf3.withColumn("DAYS_EMPLOYED_PERCENT",col("DAYS_EMPLOYED")/col("DAYS_BIRTH"))

val newDf = tmpDf4.withColumn("label",col("TARGET"))
```

Convert string column with only 2 unique values to a column of label indices to make the values readable for machine learning algorithm

ONE WAY IS BY DOING IT MANUALLY

```
import org.apache.spark.ml.feature.StringIndexer

val indexer = new
StringIndexer().setInputCol("NAME_CONTRACT_TYPE").setOutputCol("NAME_CONTRACT_TYPE_Ind
ex")

val indexed = indexer.fit(newDf).transform(newDf)
```

SMARTER WAY WILL BE TO DO THIS USING PIPELINE

```
import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.feature.StringIndexer

val indexers =
Array("NAME_CONTRACT_TYPE","CODE_GENDER","FLAG_OWN_CAR","FLAG_OWN_REALTY").map(c
=> new StringIndexer().setInputCol(c).setOutputCol(c + "_Index"))

val pipeline = new Pipeline().setStages(indexers)

val df_r = pipeline.fit(newDf).transform(newDf)
```

Convert string column with values > 2 to onehotencoder

MANUALLY:

```
import org.apache.spark.ml.feature.{OneHotEncoder, StringIndexer}

val indexer = new
StringIndexer().setInputCol("NAME_INCOME_TYPE").setOutputCol("categoryIndex").fit(df_r)

val indexed = indexer.transform(df_r)

val encoder = new
OneHotEncoder().setInputCol("NAME_CONTRACT_TYPE_Index").setOutputCol("categoryVec")

val encoded = encoder.transform(indexed)
```

THROUGH PIPELINE:

```
import org.apache.spark.ml.feature.{OneHotEncoder, StringIndexer}

val indexers1 = Array("NAME_INCOME_TYPE",
"NAME_EDUCATION_TYPE","ORGANIZATION_TYPE").map(c => new
StringIndexer().setInputCol(c).setOutputCol(c + "_Index"))

val encoder = Array("NAME_INCOME_TYPE",
"NAME_EDUCATION_TYPE","ORGANIZATION_TYPE").map(column => new
OneHotEncoder().setInputCol(column+"_Index").setOutputCol(column + "_Vec"))

val encoderPipeline = new Pipeline().setStages(indexers1 ++ encoder)

val encoded = encoderPipeline.fit(df_r).transform(df_r)

encoded.show(1)
```

Convert AGE column to bins (converting age in four categories)

```
import org.apache.spark.ml.feature.Bucketizer

val splits = Array(0, 25.0, 35.0, 55.0, 100.0)

val bucketizer = new Bucketizer().setInputCol("AGE").setOutputCol("bucketedData").setSplits(splits)

val bucketedData = bucketizer.transform(encoded)

bucketedData.groupBy("bucketedData").pivot("TARGET").count().show()     // bucketeddata is
output column name
```

Generate feature columns (discarded string only index columns)

```
val feature_cols =
Array("CNT_CHILDREN","AMT_INCOME_TOTAL","AMT_CREDIT","AMT_ANNUITY","DAYS_EMPLOYE
D","FLAG_MOBIL","FLAG_EMP_PHONE","FLAG_WORK_PHONE","FLAG_CONT_MOBILE","FLAG_PHO
NE","CNT_FAM_MEMBERS","REGION_RATING_CLIENT","REGION_RATING_CLIENT_W_CITY","REG_R
EGION_NOT_LIVE_REGION","REG_REGION_NOT_WORK_REGION","FLAG_DOCUMENT_2","FLAG_DO
CUMENT_3","FLAG_DOCUMENT_4","FLAG_DOCUMENT_5","FLAG_DOCUMENT_6","FLAG_DOCUME
NT_7","FLAG_DOCUMENT_8","FLAG_DOCUMENT_9","FLAG_DOCUMENT_10","FLAG_DOCUMENT_1
1","FLAG_DOCUMENT_12","NAME_CONTRACT_TYPE_Index","CODE_GENDER_Index","FLAG_OWN_
CAR_Index","FLAG_OWN_REALTY_Index","NAME_INCOME_TYPE_Vec","NAME_EDUCATION_TYPE_
Vec","ORGANIZATION_TYPE_Vec","AGE","DAYS_EMPLOYED_ANOM","bucketedData","CREDIT_INCO
ME_PERCENT","ANNUITY_INCOME_PERCENT","CREDIT_TERM","DAYS_EMPLOYED_PERCENT")
```

Assemble features (assemble all features in one vector)

```
import org.apache.spark.ml.feature.VectorAssembler

val assembler = new VectorAssembler().setInputCols(feature_cols).setOutputCol("features")

val output = assembler.transform(bucketedData)
```

Train logistic Regression model (creating initializing and fitting model**)**

```
import org.apache.spark.ml.classification.LogisticRegression

val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)

val lrModel = lr.fit(output)

println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")
```

Get model Accuracy

```
import org.apache,spark.sql.types._

import org.apache.spark.mllib.evaluation.MulticlassMetrics

val transformed = lrModel.transform(output)

val results = transformed.select("prediction", "label").withColumn("label",
col("label").cast(DoubleType))

val predictionAndLabels=results.rdd.map(row => (row(0).toString.toDouble,
row(1).toString.toDouble))

val metrics = new MulticlassMetrics(predictionAndLabels)

println("Confusion matrix:")

println(metrics.confusionMatrix)

val accuracy = metrics.accuracy

println("Summary Statistics")

println(s"Accuracy = $accuracy")
```

Generating Pipeline

```
// scalastyle:off println
package com.edureka.training.inclass

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types._
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.Pipeline
import org.apache.spark.sql.functions._
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.feature.{OneHotEncoder, StringIndexer}
import org.apache.spark.ml.feature.Bucketizer

object ModelGenerator {
  def main(args: Array[String]) {
    if (args.length < 1) {
      System.err.println("Usage: ModelGenerator <data path> <model persitence path>")
      System.exit(1)
    }

    val spark = SparkSession
      .builder
      .appName("Inclass ModelGenerator")
      .getOrCreate()

    // data and model path
    val dataPath = args(0)
```

```
val modelPath = args(1)


// load data
val raw = spark.read.option("inferSchema",true).csv(dataPath)


// Add header
val columns = Seq(
"TARGET",
"NAME_CONTRACT_TYPE",
"CODE_GENDER",
"FLAG_OWN_CAR",
"FLAG_OWN_REALTY",
"CNT_CHILDREN",
"AMT_INCOME_TOTAL",
"AMT_CREDIT",
"AMT_ANNUITY",
"NAME_INCOME_TYPE",
"NAME_EDUCATION_TYPE",
"NAME_FAMILY_STATUS",
"NAME_HOUSING_TYPE",
"DAYS_BIRTH",
"DAYS_EMPLOYED",
"FLAG_MOBIL",
"FLAG_EMP_PHONE",
"FLAG_WORK_PHONE",
"FLAG_CONT_MOBILE",
"FLAG_PHONE",
"CNT_FAM_MEMBERS",
"REGION_RATING_CLIENT",
"REGION_RATING_CLIENT_W_CITY",
"REG_REGION_NOT_LIVE_REGION",
```

```
  "REG_REGION_NOT_WORK_REGION",
  "ORGANIZATION_TYPE",
  "FLAG_DOCUMENT_2",
  "FLAG_DOCUMENT_3",
  "FLAG_DOCUMENT_4",
  "FLAG_DOCUMENT_5",
  "FLAG_DOCUMENT_6",
  "FLAG_DOCUMENT_7",
  "FLAG_DOCUMENT_8",
  "FLAG_DOCUMENT_9",
  "FLAG_DOCUMENT_10",
  "FLAG_DOCUMENT_11",
  "FLAG_DOCUMENT_12"
  )

  val data = raw.limit(10000).toDF(columns:_*)
  data.cache

  // Add age columns
  val dfAge = data.withColumn("AGE", col("DAYS_BIRTH")/(-365))

  // Add anomaly flag and replace it with 0
  val anomalyFlagDf =
dfAge.withColumn("DAYS_EMPLOYED_ANOM",col("DAYS_EMPLOYED").equalTo(365243))
  val anomalyDf = anomalyFlagDf.withColumn("DAYS_EMPLOYED", when(col("DAYS_EMPLOYED")
=== 365243, 0).otherwise(col("DAYS_EMPLOYED")))

  // Rename column TARGET to label
  val labelDf = anomalyDf.withColumn("label",col("TARGET"))

  // create domain features
```

```
   val tmpDf1 =
labelDf.withColumn("CREDIT_INCOME_PERCENT",col("AMT_CREDIT")/col("AMT_INCOME_TOTAL"))

   val tmpDf2 =
tmpDf1.withColumn("ANNUITY_INCOME_PERCENT",col("AMT_ANNUITY")/col("AMT_INCOME_TOT
AL"))

   val tmpDf3 = tmpDf2.withColumn("CREDIT_TERM",col("AMT_ANNUITY")/col("AMT_CREDIT"))

   val df =
tmpDf3.withColumn("DAYS_EMPLOYED_PERCENT",col("DAYS_EMPLOYED")/col("DAYS_BIRTH"))


   // define columns that will be used as feature variables in model training

   val feature_cols = Array(

   "CNT_CHILDREN",

   "AMT_INCOME_TOTAL",

   "AMT_CREDIT",

   "AMT_ANNUITY",

   "DAYS_EMPLOYED",

   "FLAG_MOBIL",

   "FLAG_EMP_PHONE",

   "FLAG_WORK_PHONE",

   "FLAG_CONT_MOBILE",

   "FLAG_PHONE",

   "CNT_FAM_MEMBERS",

   "REGION_RATING_CLIENT",

   "REGION_RATING_CLIENT_W_CITY",

   "REG_REGION_NOT_LIVE_REGION",

   "REG_REGION_NOT_WORK_REGION",

   "FLAG_DOCUMENT_2",

   "FLAG_DOCUMENT_3",

   "FLAG_DOCUMENT_4",

   "FLAG_DOCUMENT_5",

   "FLAG_DOCUMENT_6",

   "FLAG_DOCUMENT_7",
```

```
  "FLAG_DOCUMENT_8",
  "FLAG_DOCUMENT_9",
  "FLAG_DOCUMENT_10",
  "FLAG_DOCUMENT_11",
  "FLAG_DOCUMENT_12",
  "NAME_CONTRACT_TYPE_Index",
  "CODE_GENDER_Index",
  "FLAG_OWN_CAR_Index",
  "FLAG_OWN_REALTY_Index",
  "NAME_INCOME_TYPE_Vec",
  "NAME_EDUCATION_TYPE_Vec",
  "ORGANIZATION_TYPE_Vec",
  "AGE",
  "DAYS_EMPLOYED_ANOM",
  "bucketedData",
  "CREDIT_INCOME_PERCENT",
  "ANNUITY_INCOME_PERCENT",
  "CREDIT_TERM",
  "DAYS_EMPLOYED_PERCENT")

  // Convert string to label index
  val indexers =
Array("NAME_CONTRACT_TYPE","CODE_GENDER","FLAG_OWN_CAR","FLAG_OWN_REALTY","NAM
E_INCOME_TYPE","NAME_EDUCATION_TYPE","ORGANIZATION_TYPE").map(c => new
StringIndexer().setInputCol(c).setOutputCol(c + "_Index"))

  //val indexers = Array("CODE_GENDER","NAME_INCOME_TYPE").map(c => new
StringIndexer().setInputCol(c).setOutputCol(c + "_In
dex"))
  println("==> Indexed")

  // convert string columns to binary columns
```

```
   val encoder =
Array("NAME_INCOME_TYPE","NAME_EDUCATION_TYPE","ORGANIZATION_TYPE").map(column =>
new OneHotEncoder().setInp

utCol(column+"_Index").setOutputCol(column + "_Vec"))


   // convert continuous variable to category

   val splits = Array(0, 25.0, 35.0, 55.0, 100.0)

   val bucketizer = new
Bucketizer().setInputCol("AGE").setOutputCol("bucketedData").setSplits(splits)


   // Assemble features

   val assembler = new VectorAssembler().setInputCols(feature_cols).setOutputCol("features")


   // LogisticRegression Model

   val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)


   // Define pipeline

   //val pipeline = new Pipeline().setStages(Array(bucketizer) ++ indexers ++ encoder ++
Array(assembler, lr))

   val pipeline = new Pipeline().setStages(Array(bucketizer) ++ indexers ++ encoder ++
Array(assembler, lr))

   val model = pipeline.fit(df)


   println(model.transform(df).show(1))


   // save model

   model.write.overwrite().save(modelPath)

   spark.stop()

  }

}

// scalastyle:on println
```

Streaming

Create kafka topic

kafka-topics --create --zookeeper ip-20-0-21-161.ec2.internal:2181 --replication-factor 1 --partitions 1 --topic inclass

Test the kafka topic

kafka-console-producer --broker-list ip-20-0-31-4.ec2.internal:9092 --topic inclass

kafka-console-consumer --zookeeper ip-20-0-21-161.ec2.internal:2181 --topic inclass -from-beginning

Start Streaming application

Create the scala file named InclassStreaming.scala in package com.edureka.training.inclass

```
package com.edureka.training.inclass

import org.apache.spark.sql.SparkSession

import org.apache.spark.sql.types._

import org.apache.spark.sql.functions._


import org.apache.spark._

import org.apache.spark.streaming._

import org.apache.spark.sql.Encoders

import org.apache.spark.ml._

import org.apache.spark.streaming.kafka._


case class Data1(NAME_CONTRACT_TYPE:String, CODE_GENDER:String, FLAG_OWN_CAR:String, FLAG_OWN_REALTY:String, CNT_CHILDREN:Int, AMT_INCOME_TOTAL:Double, AMT_CREDIT:Double, AMT_ANNUITY:Double, NAME_INCOME_TYPE:String, NAME_EDUCATION_TYPE:String, NAME_FAMILY_STATUS:String, NAME_HOUSING_TYPE:String, DAYS_BIRTH:Int, DAYS_EMPLOYED:Int, FLAG_MOBIL:Int, FLAG_EMP_PHONE:Int, FLAG_WORK_PHONE:Int, FLAG_CONT_MOBILE:Int, FLAG_PHONE:Int, CNT_FAM_MEMBERS:Double,
```

```
REGION_RATING_CLIENT:Int, REGION_RATING_CLIENT_W_CITY:Int,
REG_REGION_NOT_LIVE_REGION:Int, REG_REGION_NOT_WORK_REGION:Int,
ORGANIZATION_TYPE:String, FLAG_DOCUMENT_2:Int, FLAG_DOCUMENT_3:Int,
FLAG_DOCUMENT_4:Int, FLAG_DOCUMENT_5:Int, FLAG_DOCUMENT_6:Int,
FLAG_DOCUMENT_7:Int, FLAG_DOCUMENT_8:Int, FLAG_DOCUMENT_9:Int,
FLAG_DOCUMENT_10:Int, FLAG_DOCUMENT_11:Int, FLAG_DOCUMENT_12:Int)


object InclassStreaming {

  def main(args: Array[String]) {

    val conf  = new SparkConf().setAppName("InclassStreaming")

    val ssc = new StreamingContext(conf, Seconds(10))


    val topicMap = "inclass".split(",").map((_, 1)).toMap

    val lines = KafkaUtils.createStream(ssc, " ip-20-0-21-161.ec2.internal:2181 ","spark-streaming-
consumer", topicMap).map(_._2)

    //val lines = ssc.textFileStream("tmp/kafka/spam_message")


    lines.foreachRDD { rdd =>


      if (!rdd.isEmpty) {

      println("====================================")

      val spark=SparkSession.builder().getOrCreate()

      import spark.implicits._

      val rawRdd = rdd.map(_.split(",")).map(d=>Data1(d(0).toString, d(1).toString, d(2).toString,
d(3).toString, d(4).toInt, d(5).toDouble, d(6).toDouble, d(7).toDouble, d(8).toString, d(9).toString,
d(10).toString, d(11).toString, d(12).toInt, d(13).toInt, d(14).toInt, d(15).toInt, d(16).toInt,
d(17).toInt, d(18).toInt, d(19).toDouble, d(20).toInt, d(21).toInt, d(22).toInt, d(23).toInt,
d(24).toString, d(25).toInt, d(26).toInt, d(27).toInt, d(28).toInt, d(29).toInt, d(30).toInt,d(31).toInt,
d(32).toInt, d(33).toInt, d(34).toInt, d(35).toInt))


      val raw = spark.createDataFrame(rawRdd)

      // Add age columns

      val dfAge = raw.withColumn("AGE", col("DAYS_BIRTH")/(-365))

      // Add anomaly flag and replace it with 0
```

```
val anomalyFlagDf = dfAge.withColumn("DAYS_EMPLOYED_ANOM",col("DAYS_EMPLOYED").equalTo(365243))

val anomalyDf = anomalyFlagDf.withColumn("DAYS_EMPLOYED", when(col("DAYS_EMPLOYED") === 365243, 0).otherwise(col("DAYS_EMPLOYED")))

// Rename column TARGET to label

val labelDf = anomalyDf//.withColumn("label",col("TARGET"))

// create domain features

val tmpDf1 = labelDf.withColumn("CREDIT_INCOME_PERCENT",col("AMT_CREDIT")/col("AMT_INCOME_TOTAL"))

val tmpDf2 = tmpDf1.withColumn("ANNUITY_INCOME_PERCENT",col("AMT_ANNUITY")/col("AMT_INCOME_TOTAL"))

val tmpDf3 = tmpDf2.withColumn("CREDIT_TERM",col("AMT_ANNUITY")/col("AMT_CREDIT"))

val df = tmpDf3.withColumn("DAYS_EMPLOYED_PERCENT",col("DAYS_EMPLOYED")/col("DAYS_BIRTH"))

// define columns that will be used as feature variables in model training

val feature_cols = Array(

"CNT_CHILDREN",

"AMT_INCOME_TOTAL",

"AMT_CREDIT",

"AMT_ANNUITY",

"DAYS_EMPLOYED",

"FLAG_MOBIL",

"FLAG_EMP_PHONE",

"FLAG_WORK_PHONE",

"FLAG_CONT_MOBILE",

"FLAG_PHONE",

"CNT_FAM_MEMBERS",

"REGION_RATING_CLIENT",

"REGION_RATING_CLIENT_W_CITY",

"REG_REGION_NOT_LIVE_REGION",

"REG_REGION_NOT_WORK_REGION",

"FLAG_DOCUMENT_2",
```

```
"FLAG_DOCUMENT_3",
"FLAG_DOCUMENT_4",
"FLAG_DOCUMENT_5",
"FLAG_DOCUMENT_6",
"FLAG_DOCUMENT_7",
"FLAG_DOCUMENT_8",
"FLAG_DOCUMENT_9",
"FLAG_DOCUMENT_10",
"FLAG_DOCUMENT_11",
"FLAG_DOCUMENT_12",
"NAME_CONTRACT_TYPE_Index",
"CODE_GENDER_Index",
"FLAG_OWN_CAR_Index",
"FLAG_OWN_REALTY_Index",
"NAME_INCOME_TYPE_Vec",
"NAME_EDUCATION_TYPE_Vec",
"ORGANIZATION_TYPE_Vec",
"AGE",
"DAYS_EMPLOYED_ANOM",
"bucketedData",

"CREDIT_INCOME_PERCENT",
"ANNUITY_INCOME_PERCENT",
"CREDIT_TERM",
"DAYS_EMPLOYED_PERCENT")

val pipeline = PipelineModel.read.load("inclass/scalamodel.model")

val predictions = pipeline.transform(df)
println("==========================")
println(predictions.show(1))
```

```
    }
  }

  ssc.start()
  ssc.awaitTermination()

 }
}
```

Compile and package the jar using sbt package

Execute the scala program:

spark2-submit --jars /opt/cloudera/parcels/SPARK2/lib/spark2/kafka-0.9/spark-streaming-kafka-0-8_2.11-2.1.0.cloudera2.jar --class com.edureka.training.inclass.InclassStreaming --deploy-mode client target/scala-2.11/sparkme-project_2.11-1.0.jar

Send sample messages from kafka console producer

Cash loans,M,N,Y,0,202500.0,406597.5,24700.5,Working,Secondary / secondary special,Single / not married,House / apartment,-9461,-637,1,1,0,1,1,1.0,2,2,0,0,Business Entity Type 3,0,1,0,0,0,0,0,0,0,0,0