Lab 12

In this lab we will combine Spark's machine learning capabilities (MLlib) and streaming functionalities (StructuredStreaming) together. We will work on data collected by a smartphone's gyroscope and accelerometer to detect which activity a person is performing. Your task consists in training a machine learning model, validate its performances using the given test data and eventually perform hyperparameter optimization; then, you are asked to apply the trained machine learning model to real-time data sent by the smartphone to detect which activity is the person performing. The initial training task must be performed offline.
The stream of data generated by the smartphone is simulated by uploading, during the execution of your application, a set of files in an input HDFS folder. Each input file contains the features extracted from the smartphone's gyroscope and accelerometer in a short time interval. Each line represents an activity to be classified.

Dataset Description

The dataset is collected through smartphone sensors held by a person when performing 6 different types of activities. For a time window of 2.56 seconds, the smartphone collects data from the gyroscope and accelerometer and the extracted signals are analyzed. A total number of 561 representative features for each time window are generated, which represent the input features. For each time window, the activity label is associated. Each line of the provided files represents a time window of 2.56 seconds with a given target label, i.e. the activity performed.
The following table contains all the possible values for the activity label:

Label Value	Description
1	Walking
2	Walking upstairs
3	Walking downstairs
4	Sitting
5	Standing
6	Laying

The dataset consists of following files:

- *train.csv,* containing the data to train your model
- *test.csv*, containing the data to verify your model's performances
- *streaming_N.csv*, where N is an integer number, containing the data to simulate the stream

Each file has the following format:

- tBodyAcc-mean()-X**\t**tBodyAcc-mean()-Y**\t**…**\t**activity

The **activity** column is the target value to predict.
The header is present **only** in *train.csv* and *test.csv* files. Files used to simulate the stream **do not contain the header**. For practical reasons, the stream files also contain the ground truth labels.

You can find the files at the following paths on the HDFS file system:

- *train.csv:* /data/students/bigdata-01QYD/Lab12_DBD/train.csv
- *test.*csv*:* /data/students/bigdata-01QYD/Lab12_DBD/test.csv

The files to simulate the data stream are uploaded on the website.
You can find on the website the zip file *Lab12_DBD_template.zip* containing two Jupyter Notebooks to be used as templates for the required tasks.

TASK 1

Write a first Spark application to load the training and test data, train a classification model to predict the activity performed and export the model.
Use the DataFrame APIs of Apache Spark to read the training and test data and train the model.
Remember (from Lab 9) that an input DataFrame for a classification model in MLlib needs to be characterized by two fields: *label* and *features* (a vector of Double). Each line of the *train.csv* and *test.csv* files represent an input record and, consequently, a record in the DataFrame.
Specifically, you need to:

- read the data from the file system
- cast the "*activity*" (i.e. the target) column to double
- export the dataframe schema for Task 2
- generate the DataFrame with the *features* and *label* columns
- train the model
- save the model for Task 2

To save the model, you can use the *save* method of the model:

```
model_export_path = "lab12/model"
model.save(model_export_path)
```

In case you want to use the VectorAssembler in your pipeline to generate the *features* column, you can use the list comprehension in python to generate the input columns by excluding the *activity* column.

```
df_columns = df.schema.names
inputCols = [x for x in df_columns if x != 'activity']
```

To export the dataframe schema, you need to use the *json* library in python and the *jsonValue()* method applied to the schema object of the dataframe.

```
import json
json_schema = df.schema.jsonValue()
with open('schema.json', 'w') as fp:
    json.dump(json_schema, fp)
```

Reminder. You can cast a column to double using the *withColumn* method:

```
df = df.withColumn('myColumn', df.myColumn.cast('double'))
```

TASK 2

A set of smartphones is used to collect and analyze data about volunteers which are performing one of the 6 activities previously described. Each smartphone is generating the 561 features introduced before and sends such data to your Spark Streaming application.
In this task, you are asked to develop a Spark Streaming application using StructuredStreaming APIs to classify in (almost) real-time the data sent by the smartphones and predict the activities performed by the volunteers. For practical purposes, we also inserted the ground truth activity label in the csv files used to simulate the data stream.
Your application must:

- load the model trained in Task 1
- load the dataframe schema exported from Task 1
- perform the prediction on new data received using the default micro-batch processing mode
- store in the output HDFS folder predictions made by your model and (optionally) the ground truth labels provided

To load the dataframe schema from a json file, you can use the following lines of code:

```
import json
from pyspark.sql.types import StructType
json_schema_path = 'dataframe_schema.json'
with open(json_schema_path, 'r') as fp:
        json_schema_dict = json.load(fp)
json_schema = StructType.fromJson(json_schema_dict)

stream = spark.readStream.format(…)\
                .option(…)\
                .schema(json_schema)\
                .load()
```

To simulate the data stream, you will upload one file with prefix *streaming_N.csv* at a time in an input HDFS folder. The upload procedure is the same as in Lab11 – Ex.2. For simplicity, the steps are reported here:

1. Create a local input folder on the local file system of jupyter.polito.it and upload the files you want to use in that folder
2. Create an empty HDFS folder (the folder associated with the input of your application)
3. Copy, one at a time, the input files from the local input folder to the HDFS input folder of your application by using the command line hdfs
   E.g., *hdfs dfs -put input_local_folder/streaming_1.csv input_HDFS_folder/*
4. Pay attention that if a file is already in the input HDFS folder and you copy other version of the same file the system will not consider the new version of the file
5. Pay attention to stop your streaming context after your tests by invoking **prediction_stream.stop()**

Note. If you run this application locally on your PC:

1. Create the input files in a folder and then copy them in the input folder of your application (one file at a time in order to simulate the stream of data)
2. Copy the files in the input folder of your application by using the command line cp
   E.g., *cp streaming_1.csv input_folder/*
3. Do not use the graphical interface to copy the file in the input folder of your application otherwise the output of your application will be empty
4. If you update the content of a file that is already in the input folder of your application, the updated version of the file will not be considered by the Spark streaming engine