STREAMER: Data Stream Framework

Getting Started Guide

SANDRA GARCIA RODRIGUEZ

MOHAMMAD ALSHAER

Index

Welcome to STREAMER! This guide will show you how to easily install and run your first use case with STREAMER. No worries, the process is straightforward. (For a detailed presentation of STREAMER read the "UserGuide.pdf".)

Let's start with our website

First of all, download STREAMER framework + environment setup from

https://github.com/streamer-framework/streamer

However, before we start the installation, you need to decide how you will use the framework. STREAMER is conceived to be used in two different ways (depending on your necessities):

1. **Development use**: (oriented to data scientists). You are interested on directly working on the code of the framework to add/develop several functionalities and test them.
2. **Product use**: (oriented to industrial use). You want to use the framework as a product (no need to get in contact with the code but execute STREAMER).
   In this case, you need to have in your computer the basic services STREAMER requires and STREAMER instance already packed.

1 Getting ready for Deployment use

1) In this case, you need to run the basic services that STREAMER requires. You can install them using the docker (recommended) following the steps of section 3.1, or install them yourself following section 4,
2) Install Eclipse (https://www.eclipse.org/downloads/) or the IDE you prefer.
3) Import the maven project: File->Import->Maven->Existing Maven Projects (and follow the steps to select the folder of STREAMER project).
4) [Optional] you are now ready to run our example use case:
   Run from eclipse the main class *ProducerMain* to launch the data ingester, or from console: java ProducerMain
   Run from eclipse the main class *LauncherMain* to launch the streaming pipeline, or from console: java LauncherMain
5) Create your first use case in STREAMER by following the steps of section 5).
6) Run your application in STREAMER as section 5 shows.

2 Getting ready for Production use

Try our example use case in STREAMER by following the steps of Section 3.

3 Using Docker

We make simple and transparent the installation of STREAMER and its services by using Docker. We provide 2 docker files that serve to:

1. **Services environment** (section 3.1): it contains all the services used by the framework (*Kafka&Zookeeper, Redis. InfluxDB, Kibana&ElasticSearch*).
2. **Production environment** (section 3.2): it contains STREAMER for production purpose.

When downloading the zip file you will find:

```
streamer_environment (folder)
|       docker-compose.yml (file)
|       streamer_environment (sub-folder)
|       |       docker-compose.yml (file)
|       |       Dockerfile_producer (file)
|       |       Dockerfile_streamer (file)
|       |       Jar file (file)
```

| | requirements.txt (file)

| | data (sub-sub-folder)

3.1 Install & run all the services from Docker (recommended)

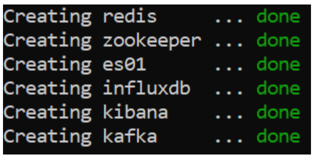
[Warning]: for Linux based systems, you may need to run all the commands in “sudo” mode as, for instance, “sudo docker-compose up --build -d “.

Follow the following steps to set all necessary services before running STREAMER:

1. Install docker on your machine. At the following link, you will find how to install the docker for all the different operating systems (Windows, Linux, Mac): https://docs.docker.com/get-docker/
2. [For Linux] also install docker-compose from https://docs.docker.com/compose/install/
3. Unzip the provided folder “streamer_environment”.
4. Open a terminal and change directory to this “streamer_environment” directory.
5. Run the following command to start the services:

```
docker-compose up --build -d
```

It should give an output similar to this:

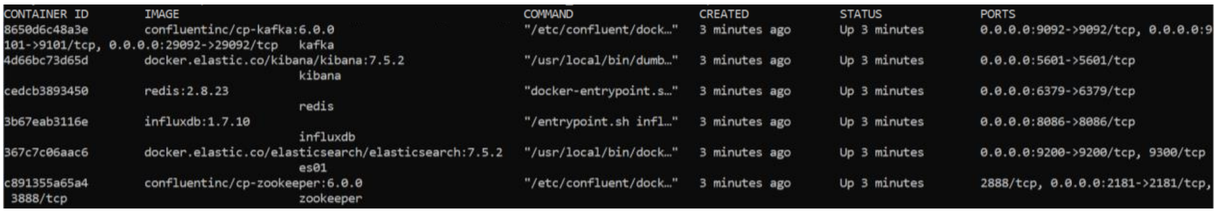


```
Creating redis      ... done
Creating zookeeper ... done
Creating es01       ... done
Creating influxdb   ... done
Creating kibana     ... done
Creating kafka      ... done
```

In order to check if the services are running properly, check the following command:

```
docker ps
```

It should give an output similar to this:

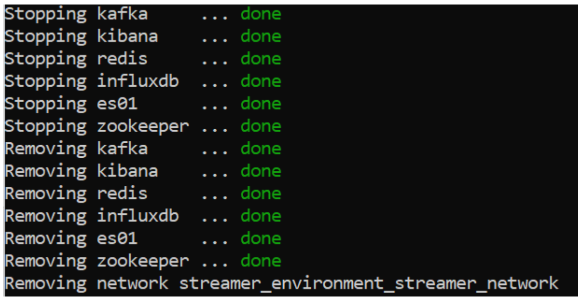


```
CONTAINER ID        IMAGE                                                   COMMAND                   CREATED             STATUS              PORTS
8650d6c48a3e        confluentinc/cp-kafka:6.0.0                             "/etc/confluent/dock…"   3 minutes ago       Up 3 minutes        0.0.0.0:9092->9092/tcp, 0.0.0.0:9
101->9101/tcp, 0.0.0.0:29092->29092/tcp   kafka
4d66bc73d65d        docker.elastic.co/kibana/kibana:7.5.2                   "/usr/local/bin/dumb…"   3 minutes ago       Up 3 minutes        0.0.0.0:5601->5601/tcp
                                          kibana
cedcb3893450        redis:2.8.23                                            "docker-entrypoint.s…"   3 minutes ago       Up 3 minutes        0.0.0.0:6379->6379/tcp
                                          redis
3b67eab3116e        influxdb:1.7.10                                         "/entrypoint.sh infl…"   3 minutes ago       Up 3 minutes        0.0.0.0:8086->8086/tcp
                                          influxdb
367c7c06aac6        docker.elastic.co/elasticsearch/elasticsearch:7.5.2     "/usr/local/bin/dock…"   3 minutes ago       Up 3 minutes        0.0.0.0:9200->9200/tcp, 9300/tcp
                                          es01
c891355a65a4        confluentinc/cp-zookeeper:6.0.0                         "/etc/confluent/dock…"   3 minutes ago       Up 3 minutes        2888/tcp, 0.0.0.0:2181->2181/tcp,
 3888/tcp                                 zookeeper
```

To stop the services, use the following command:

```
docker-compose down
```

which should give an output similar to this:



```
Stopping kafka     ... done
Stopping kibana    ... done
Stopping redis     ... done
Stopping influxdb  ... done
Stopping es01      ... done
Stopping zookeeper ... done
Removing kafka     ... done
Removing kibana    ... done
Removing redis     ... done
Removing influxdb  ... done
Removing es01      ... done
Removing zookeeper ... done
Removing network streamer_environment_streamer_network
```

Note: If after following the previous steps, you face a similar error to

ERROR: [1] bootstrap checks failed
[1]: max virtual memory areas vm.max_map_count [65530] is too low, increase to at least [262144]

you can solve it by increasing your virtual memory. Run the command:

```
sudo sysctl -w vm.max_map_count=262144
```

and then build the docker again with:

```
sudo docker-compose up --build –d
```

3.2 Running STREAMER in production environment

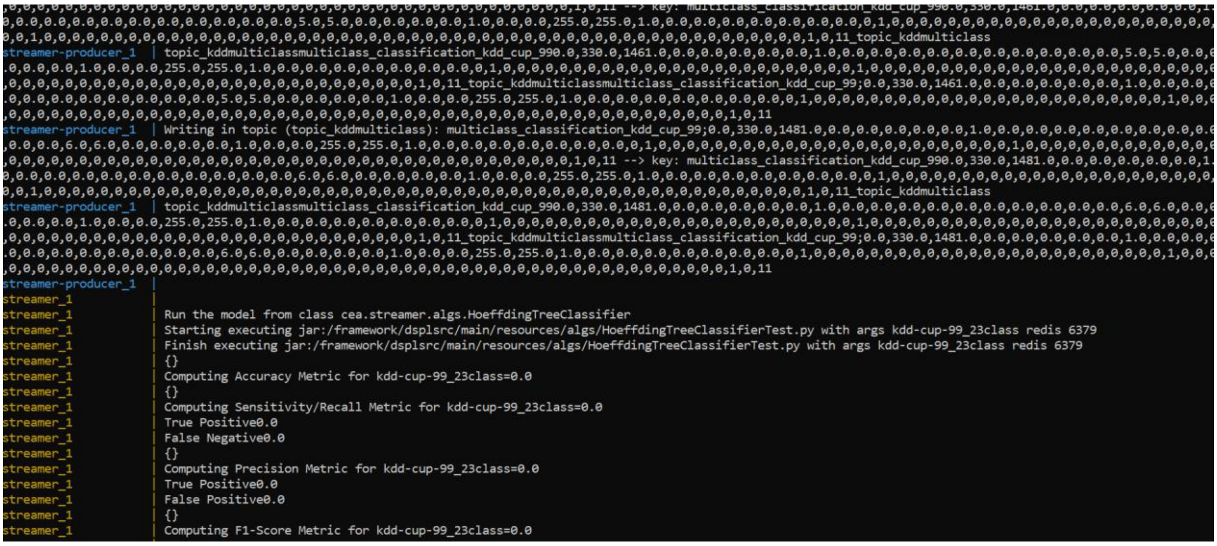
[Warning]: for Linux based systems, you may need to run all the commands in “sudo” mode as, for instance, sudo docker-compose up --build -d.

1. Before running STREAMER for production purpose, complete all the steps of case 1 above to run all the services.
2. Open a terminal and go to directory “streamer_environment/streamer_environment” directory (sub-folder of streamer_environment folder).
3. Run the following command to start the framework:

   ```
   docker-compose up --build
   ```

   If you add (-d) to the command above, it should start in the background. In case you are interested in showing the output of the framework, feel free to leave it like it-is.

   It should give an output similar to this:

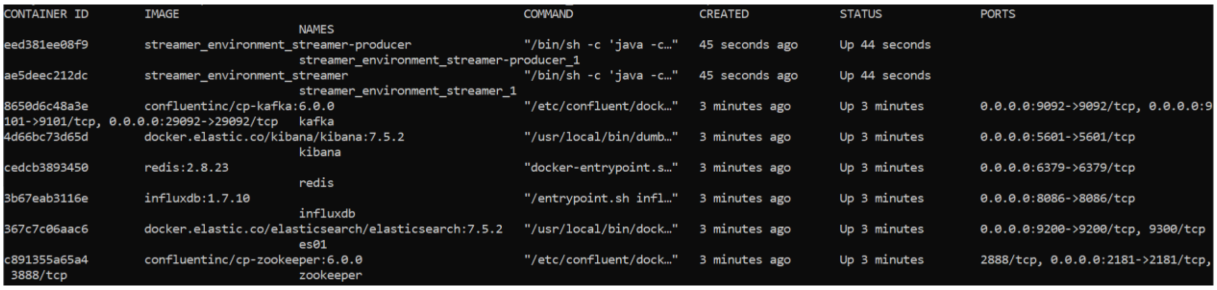


```
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,11 --> key: multiclass_classification_kdd_cup_990.0,330.0,1461.0,0.0,0.0,0.0,0.0,0.0,1.
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,5.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,255.0,255.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,11_topic_kddmulticlass
streamer-producer_1  | topic_kddmulticlassmulticlass_classification_kdd_cup_990.0,330.0,1461.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,5.0,5.0,0.0,0
.0,0.0,0.0,1.0,0.0,0.0,255.0,255.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,11_topic_kddmulticlassmulticlass_classification_kdd_cup_99;0.0,330.0,1461.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0
.0,0.0,0.0,0.0,0.0,0.0,0.0,5.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,255.0,255.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,11
streamer-producer_1  | Writing in topic (topic_kddmulticlass): multiclass_classification_kdd_cup_99;0.0,330.0,1481.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,6.0,6.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,255.0,255.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,11 --> key: multiclass_classification_kdd_cup_990.0,330.0,1481.0,0.0,0.0,0.0,0.0,0.0,1.
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,6.0,6.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,255.0,255.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,11_topic_kddmulticlass
streamer-producer_1  | topic_kddmulticlassmulticlass_classification_kdd_cup_990.0,330.0,1481.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,6.0,6.0,0.0,0
.0,0.0,0.0,1.0,0.0,0.0,255.0,255.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,11_topic_kddmulticlassmulticlass_classification_kdd_cup_99;0.0,330.0,1481.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0
.0,0.0,0.0,0.0,0.0,0.0,0.0,6.0,6.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,255.0,255.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,11
streamer-producer_1  |
streamer_1           |
streamer_1           | Run the model from class cea.streamer.algs.HoeffdingTreeClassifier
streamer_1           | Starting executing jar:/framework/dsplsrc/main/resources/algs/HoeffdingTreeClassifierTest.py with args kdd-cup-99_23class redis 6379
streamer_1           | Finish executing jar:/framework/dsplsrc/main/resources/algs/HoeffdingTreeClassifierTest.py with args kdd-cup-99_23class redis 6379
streamer_1           | {}
streamer_1           | Computing Accuracy Metric for kdd-cup-99_23class=0.0
streamer_1           | {}
streamer_1           | Computing Sensitivity/Recall Metric for kdd-cup-99_23class=0.0
streamer_1           | True Positive0.0
streamer_1           | False Negative0.0
streamer_1           | {}
streamer_1           | Computing Precision Metric for kdd-cup-99_23class=0.0
streamer_1           | True Positive0.0
streamer_1           | False Positive0.0
streamer_1           | {}
streamer_1           | Computing F1-Score Metric for kdd-cup-99_23class=0.0
```

In order to check if the framework is running properly, check the following command:

docker ps

It should give an output similar to this:



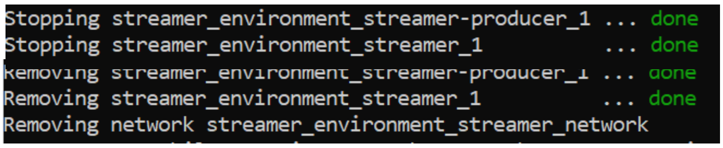
```
CONTAINER ID        IMAGE                                                 COMMAND                  CREATED             STATUS              PORTS
                                      NAMES
eed381ee08f9        streamer_environment_streamer-producer                "/bin/sh -c 'java -c…"   45 seconds ago      Up 44 seconds
                                      streamer_environment_streamer-producer_1
ae5deec212dc        streamer_environment_streamer                         "/bin/sh -c 'java -c…"   45 seconds ago      Up 44 seconds
                                      streamer_environment_streamer_1
8650d6c48a3e        confluentinc/cp-kafka:6.0.0                           "/etc/confluent/dock…"   3 minutes ago       Up 3 minutes        0.0.0.0:9092->9092/tcp, 0.0.0.0:9
101->9101/tcp, 0.0.0.0:29092->29092/tcp   kafka
4d66bc73d65d        docker.elastic.co/kibana/kibana:7.5.2                 "/usr/local/bin/dumb…"   3 minutes ago       Up 3 minutes        0.0.0.0:5601->5601/tcp
                                      kibana
cedcb3893450        redis:2.8.23                                          "docker-entrypoint.s…"   3 minutes ago       Up 3 minutes        0.0.0.0:6379->6379/tcp
                                      redis
3b67eab3116e        influxdb:1.7.10                                       "/entrypoint.sh infl…"   3 minutes ago       Up 3 minutes        0.0.0.0:8086->8086/tcp
                                      influxdb
367c7c06aac6        docker.elastic.co/elasticsearch/elasticsearch:7.5.2   "/usr/local/bin/dock…"   3 minutes ago       Up 3 minutes        0.0.0.0:9200->9200/tcp, 9300/tcp
                                      es01
c891355a65a4        confluentinc/cp-zookeeper:6.0.0                       "/etc/confluent/dock…"   3 minutes ago       Up 3 minutes        2888/tcp, 0.0.0.0:2181->2181/tcp,
 3888/tcp                             zookeeper
```

To stop the services, use the following command:

If you started the framework without using (-d) property, first press (ctrl + c) and then the following (with (-d) or not):

docker-compose down

It should give an output similar to this:



```
Stopping streamer_environment_streamer-producer_1 ... done
Stopping streamer_environment_streamer_1          ... done
Removing streamer_environment_streamer-producer_1 ... done
Removing streamer_environment_streamer_1          ... done
Removing network streamer_environment_streamer_network
```

4 Installing STREAMER services yourself (from sources)

STREAMER use the following services that you can install yourself:

[Compulsory]:

- apache kafka / soft / Apache License 2.0 / kafka.version=2.7.0

- zookeeper / soft / Apache License 2.0 / zookeeper.version=3.5.9
https://kafka.apache.org/quickstart or apache kafka & zookeeper confluent 6.2.1

- redis server / soft / BSD / Redis version=6.2 https://redis.io/

- influxdb / soft / MIT / version=1.8.9 https://portal.influxdata.com/

[Optional]:

- elasticsearch / soft / Apache License 2.0 / version=7.15.0
https://www.elastic.co/elasticsearch

- kibana / soft / Elastic License/Apache License / version=7.15.0
https://www.elastic.co/kibana

5 How to run a STREAMER use case

1) Set up all the properties in the files *.props* within *src/main/resources.*

2) For using the **data producer simulator** functionality, run *ProducerMain.java* (placed in folder *streamer/src/test/java/cea/streamer*):

   ProducerMain.java [origin1 … originN]

   A producer is now writing in Kafka topic(s).
   [*origin1 … originN*] Arguments are optional. Each of them runs a problem in a separate process; its name *originX* indicates the folder where the properties files are located for this specific execution. If no arguments indicated, the system considers the properties files directly placed in *src/main/resources/setup* folder.

3) For using the **streaming pipeline** functionality, run *LauncherMain.java* (placed in folder *streamer\src\test\java\cea\streamer)* as:

   LauncheMain.java [origin1 … originN]

   [*origin1 … originN*] Arguments are optional. Each of them runs a problem in a separate process; its name *originX* indicates the folder where the properties files are located for this specific execution. If no arguments indicated, the system considers the properties files directly placed in *src/main/resources/setup* folder.

4) For using the **learning API** in **offline** mode (algorithms test and/or train) run

AlgInvokerMain_offline.java (placed in folder *streamer\src\test\java\cea\streamer*) as
AlgInvoker_Main.java arg1 arg2 arg3 arg4 [arg5]

Arguments are:

a) *arg1*. Data source type: From where data must be retrieved. Option: [*influx* or *file*]. It could be either "influx" to build the model using the data stored in InfluxDB or "file" to directly use the data from the file specify in algs.properties (data.training).
b) *arg2.* Origin: [influx-data-base-name or origin]. *origin* indicates the folder where the properties files are located for this specific execution. If no arguments indicated, the system considers the properties files directly placed in *src/main/resources/setup* folder.
c) *arg3.* Perform training [*true* or *false*] (train).
d) *arg4.* Perform model running [*true* or *false*] (test).
e) [*arg5*]. [Optional] Path from where to store/retrieve the trained model.

5) For using **kibana graphical interface** open a web browser and type http://localhost:5601. The dashboard will appear.

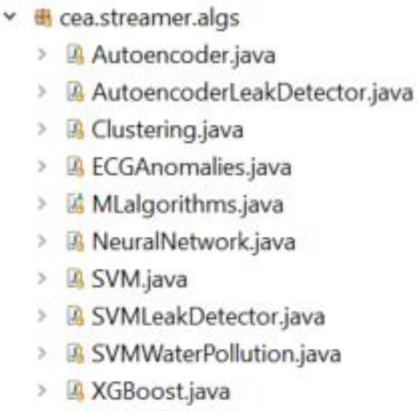
Important: User can follow the evolution of STREAMER at any time by just checking the log files STREAMER generates. Placed in *log* folder, user can find>

5.1. Logs reporting the execution of the algorithms as *displayLogTrain, displayLogTest, errorLog, infoLog, temp_[id]* (temporary log while algorithm is running), etc.
5.2. *metricsLog* where each metrics evaluations are stored.

6 How to Integrate your Algorithm in STREAMER

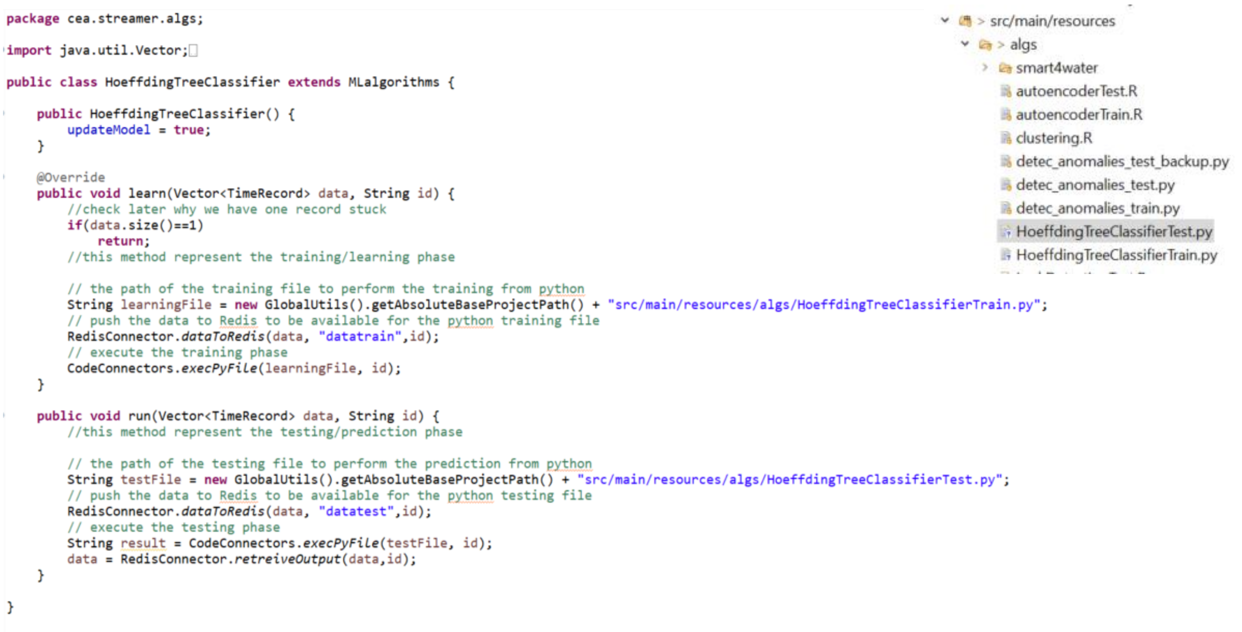
In order to create your running application in STREAMER, two main phases are required:

6.1 PHASE 1: INTEGRATE OR CREATE YOUR ALGORITHM



```
cea.streamer.algs
  Autoencoder.java
  AutoencoderLeakDetector.java
  Clustering.java
  ECGAnomalies.java
  MLalgorithms.java
  NeuralNetwork.java
  SVM.java
  SVMLeakDetector.java
  SVMWaterPollution.java
  XGBoost.java
```

1) We add the main algorithm class to the cea.streamer.algs package :
   a. Implement both methods learn and run for training and testing phases (if necessary).
   b. If your model is not coded in Java, you will have to use Redis as an intermediary data storage to save the arriving records to be accessible from the algorithm (in the example below, the algorithm is coded in Python).
   c. If your algorithm is an online algorithm that does not require previous records (updatable model), you should specify “updateModel=true”, this option will help clean the influxDb and remove previous data.



```java
package cea.streamer.algs;

import java.util.Vector;

public class HoeffdingTreeClassifier extends MLalgorithms {

    public HoeffdingTreeClassifier() {
        updateModel = true;
    }

    @Override
    public void learn(Vector<TimeRecord> data, String id) {
        //check later why we have one record stuck
        if(data.size()==1)
            return;
        //this method represent the training/learning phase

        // the path of the training file to perform the training from python
        String learningFile = new GlobalUtils().getAbsoluteBaseProjectPath() + "src/main/resources/algs/HoeffdingTreeClassifierTrain.py";
        // push the data to Redis to be available for the python training file
        RedisConnector.dataToRedis(data, "datatrain",id);
        // execute the training phase
        CodeConnectors.execPyFile(learningFile, id);
    }

    public void run(Vector<TimeRecord> data, String id) {
        //this method represent the testing/prediction phase

        // the path of the testing file to perform the prediction from python
        String testFile = new GlobalUtils().getAbsoluteBaseProjectPath() + "src/main/resources/algs/HoeffdingTreeClassifierTest.py";
        // push the data to Redis to be available for the python testing file
        RedisConnector.dataToRedis(data, "datatest",id);
        // execute the testing phase
        String result = CodeConnectors.execPyFile(testFile, id);
        data = RedisConnector.retreiveOutput(data,id);
    }

}
```

```
src/main/resources
  algs
    smart4water
    autoencoderTest.R
    autoencoderTrain.R
    clustering.R
    detec_anomalies_test_backup.py
    detec_anomalies_test.py
    detec_anomalies_train.py
    HoeffdingTreeClassifierTest.py
    HoeffdingTreeClassifierTrain.py
```

2) Add the training and testing python files to the path src/main/resources/algs
   a. While coding the algorithms in Python, pay attention for the training file to read the data from Redis, and write the model in Redis.
   b. Likewise, for the testing file, pay attention to read the data and the model from Redis, and write the predictions as a list in Redis by invoking the rpush method. These predictions are read by the framework in order to do the realtime evaluation of the model’s performance.

Note: *the prediction of each record may have multiple outputs (e.g., multi-labelling, interval predictions). Those outputs are by default separated by space, for changing the separator, just set the value of the redis key “separator+id” to the new customized separator.*

An example of coding in python for the training file:

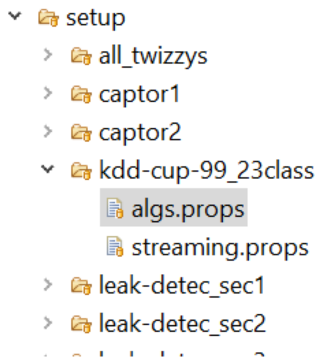
```
    redis_key = 'kdd-cup-99'
    if(len(sys.argv)>1):
        redis_key = sys.argv[1]

    redis_key_data = 'datatrain' + redis_key
    redis_key_target = 'datatrain' + redis_key + 'target'
    redis_key_model = redis_key + 'model'

    # Redis connection to connect the python program to Redis
    redisConn = configuringRedis()
    if(len(sys.argv)>3):
        redisConn = configuringRedis(host=sys.argv[2], port=int(sys.argv[3]))

    #read the training data from redis
    list_data = redisConn.lrange(redis_key_data, 0, -1)
    if not list_data:
        print("Please check if the training data is stored properly in Redis!")
        return
    # read the labels from redis for the training data
    list_output = redisConn.lrange(redis_key_target, 0, -1)
    if not list_data:
        print("Please check if the labels of the training data are stored properly in Redis!")
        return
        .
        .
        .
        .
        .
    #print("The model's training phase is done!")

    # saving the model in redis
    pickled_model = pickle.dumps(model)
    redisConn.set(redis_key_model, pickled_model)
```

An example of coding in python for the testing file:

```
    redis_key = 'kdd-cup-99'
    if(len(sys.argv)>1):
        redis_key = sys.argv[1]

    redis_key_data = 'datatest' + redis_key
    redis_key_target = 'datatest' + redis_key + 'target'
    redis_key_model = redis_key + 'model'
    redis_key_output = 'outputs' + redis_key

    # Redis connection to connect the python program to Redis
    redisConn = configuringRedis()
    if(len(sys.argv)>3):
        redisConn = configuringRedis(host=sys.argv[2], port=int(sys.argv[3]))

    #read the testing data from redis
    list_data = redisConn.lrange(redis_key_data, 0, -1)
    if not list_data:
        print("Please check if the testing data is stored properly in Redis!")
        return

    # read the labels from redis for the testing data
    list_output = redisConn.lrange(redis_key_target, 0, -1)
    if not list_output:
        print("Please check if the labels of the testing data are stored properly in Redis!")
        return
        .
        .
        .
    # loading the existing model from redis
    model = None
    try:
        model = pickle.loads(redisConn.get(redis_key_model))
    except Exception as ex:
        pass
    # if model doesn't exist then exit (model is not ready yet or something wrong in training phase
    if not model:
        return
        .
        .
        .

   # push the outputs in redis (as a list)
    for v in list_outputs:
        redisConn.rpush(redis_key_output, str(v))
```

6.2 PHASE 2: INTEGRATE OR CREATE YOUR USE CASE APPLICATION

1) Add a folder with the project name
2) Add two configuration files to this folder:
    a. Algs.props
    b. Streaming.props



```
setup
  all_twizzys
  captor1
  captor2
  kdd-cup-99_23class
    algs.props
    streaming.props
  leak-detec_sec1
  leak-detec_sec2
```

<u>Algs.props:</u>

```
# The algorithm used for this application is the KDD CUP 99 Cybersecurity dataset
algorithm = HoeffdingTreeClassifier
# Online training will be performed every 20 seconds
training.interval = 20000
# The maximum allowed data for training
training.maxdata = 5000000
#10000000
# This source is used for Offline training
#training.source = ../data/data_kdd-cup-99/binary_classification_kdd_cup_99.csv
# configurations for evaluation metrics
evaluation.metrics=
AccuracyMultiClassClassificationMetric,SensitivityMultiClassClassificationMetric,PrecisionMultiClassCla
ssificationMetric,F1scoreMultiClassClassificationMetric
#this should be optional
classes.number=23
```

Note: The user can choose metrics for evaluating the model by simply adding the name of the metric class. It is also possible for the user to extend the existing metrics with new ones.

<u>Streaming.props:</u>

```
bootstrap.servers=localhost:9092

visualization = true
```

```
####Producer:
acks=all
retries=0
batch.size=16384
auto.commit.interval.ms=1000
linger.ms=0
key.serializer=org.apache.kafka.common.serialization.StringSerializer
value.serializer=org.apache.kafka.common.serialization.StringSerializer
block.on.buffer.full=true

containsHeader = true

# The path of the data used in the kdd cup 99 cybersecurity project (online classification algorithm)
datafile = ../data/data_kdd-cup-99/multiclass_classification_kdd_cup_99.csv

### Production Type (TIMESTAMP /BLOCK)
producerType = BLOCK

## Production TIMESTAMP (by timestamp)
epsilonne = -3000
scale = 1800

## Production BLOCK (by blocks)
maxBlocks = 1000
recordsPerBlock = 50
producerTimeInterval = 10000


####Launcher:
group.id=test
application.id=test
#enable.auto.commit=true
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
# fast session timeout makes it more fun to play with failover
session.timeout.ms=10000
# These buffer sizes seem to be needed to avoid consumer switching to
# a mode where it processes one buffer full every 5 seconds with multiple
# timeouts along the way.  No idea why this happens.
fetch.min.bytes=50000
receive.buffer.bytes=262144
max.partition.fetch.bytes=2097152

readingTimeInterval = 100
problem.type = KDDClassification
#consumer.prepostprocessor = PrePostProcessorKDDClassification (to be added if needed)
outputTopic = OutputPlatformKDDB

#### Common Params
mainTopic = topic_kddmulticlass2
```

The attribute problem.type = KDDClassification means that we should create a class named KDDClassificationRecord that extends the main TimeRecord class.

3) Create the record class:
   a. It is essential to name the record class in this exact way:

      problem.type + “Record” + “.java”

   b. Specify the separator used in the original data source file (by default: semicolon).
   c. The separator to be used in Redis for separating the different values can also be determined from this class using the method setSeparatorRedis() (by default: space).
   d. You can precise the headers of the dataset

e. Implement the fill method to indicate how the records should be filled with the correct values.
f. In case the dataset does not contain timestamp, use sleep and generate a new timestamp.

```
package cea.streamer.core;

public class KDDClassificationRecord extends TimeRecord {


                public KDDClassificationRecord() {

                super();

                setSeparatorRawData(",");

                List<String> headers_list = Arrays.asList({"duration",

                                  "src_bytes",

                                  "dst_bytes",

                                  "land",

                                  "wrong_fragment",

                                  "urgent", ...

                                  "flag_SH",

                                  "label");

                headers.addAll(headers_list);

        }


        @Override

        public void fill(String data) {

                data = data.replace(" ","").toLowerCase();

                if(data != "") { //if record is not empty

                        try {

                                //This sleep is added for the reason that I am assigning the timestamps
to the used dataset and thus we should have a little gap in time to obtain different microseconds

                                TimeUnit.MICROSECONDS.sleep(10);

                        } catch (InterruptedException e) {

                                e.printStackTrace();

                        }

                        setSourceFromKafkaKey(key);

                        timestamp        =        new        SimpleDateFormat("dd-MMM-yyHH:mm:ss.SSS",
Locale.ENGLISH).format(new Date());


                        String[] features = data.split(getSeparatorRawData());


                        for(int i=0;i<features.length - 1;i++) {

                                values.put(headers.get(i), features[i]);
                                extractors.put(headers.get(i), new NumericalExtractor());
                        }
                        setTarget(features[features.length-1]);
                        extractors.put("classification", new NumericalExtractor());
                }
}
```

As you arrived this far, it is time to run the application:

```
cmd1> java -cp dsplatform-1.0.0-test-jar-with-dependencies.jar cea.streamer.LauncherMain kdd-cup-99
cmd2> java -cp dsplatform-1.0.0-test-jar-with-dependencies.jar cea.streamer.ProducerMain kdd-cup-99
```