Tutorial

Platform for Machine Learning Competitions on Data Streams

Authors:
Albert Bifet
Dihia Boulegane
Nedeljko Radulović

IEEE Big Data Cup Challenges 2019
"Real-time Machine Learning Competition on Data Streams"

Télécom Paris
Institut Polytechnique de Paris

August 4, 2019

Abstract

Tutorial and Starter Pack

This document represents the supporting material for IEEE Big Data Cup Challenge 2019 - Real-time Machine Learning Competition on Data Streams. In this document we give instructions on how to use the dedicated platform for this novel type of the machine learning competition. We explain how to register on the platform, how to subscribe to a competition, how to setup your computer to connect to secure channel in order to receive the data and as a part of the Starter Pack we provide code examples to make your participation much easier.

If you have any additional questions which are beyond this tutorial don't hesitate to write to us on streaming.challenge@gmail.com.

Contents

List of Figures

List of Abbeviations

IoT	**I**nternet **o**f **T**hings
gRPC	**g**oogle **R**emote **P**rocedure **C**all
Protobuf	**Proto**col **Buf**fers
JSON	**J**ava**S**cript **O**bject **N**otation
CSV	**C**omma **S**eparated **V**alues
API	**A**pplication **P**rogramming **I**nterface

Chapter 1

1 Accessing and Registration to the platform

The platform for Machine Learning Competitions on Data Streams provides a user friendly web application. Users can register and browse through organized competitions and subscribe to the ones they might be interested in. Also, once the competition starts, it provides online leader board and real-time evaluation.

To access the web application, from your browser go to: Platform for Machine Learning Competitions on Data Streams starting from the 5th of August, 2019.

Once you access the web application you will be offered two options:

- Sign in - if you already have an account (Figure 1)
- Register - if you don't have an account yet (Figure 2)

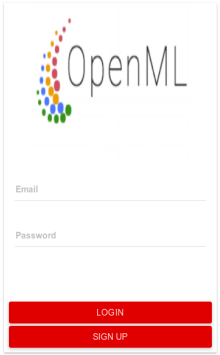


Figure 1: Login dialog

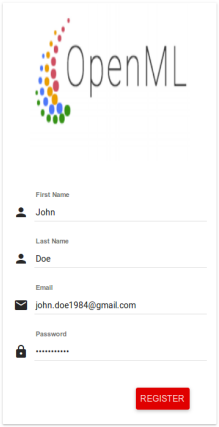


Figure 2: Registration dialog

You will be asked to provide common registration information (name, email, ...). Afterwards, you will receive the confirmation e-mail with authentication token, to confirm your account.

Once signed in, you will be able to browse the available competitions (active, coming and finished) as shown in Figure 3.

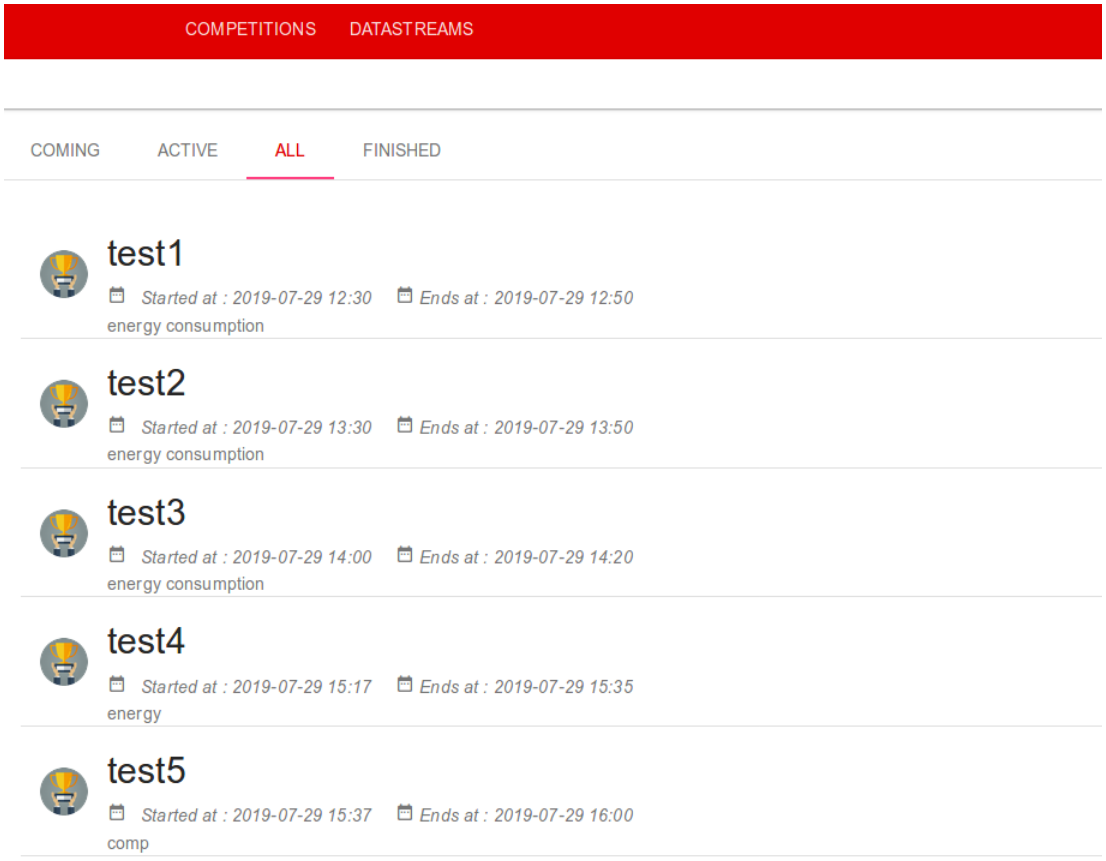


Figure 3: Browsing through competitions

You can click on a specific competition and browse for further details as shown in Figure 4 as follows:

1. **Subscribe and Unsubscribe:** You can subscribe to a competition before it has ended, but it is highly recommended to subscribe before the beginning in order to have time to prepare your model because if you are late with sending predictions you will be penalized and it will significantly impact your score.

2. **Competition code**: A unique code is assigned to every competition

3. **Secret key**: When you are subscribed to a competition a secret code will be provided to you that will play the role of an authentication token. You will need to copy the key in the client code and provide it alongside your predictions for authentication purposes.

4. **Information tabs**: You will be able to navigate through detailed information about the competition such as data description and evaluation metrics. You will also be able to follow in real time your performance and global leaderboard and ranking compared to other participants.

5. **Stream setting**: This section lays down the criteria according to which data will be provided in the stream as follows :
   - Initial batch: Number of records including true labels provided at first to avoid cold start of learning models.
   - Initial time: This is the time dedicated to the initial training phase using the initial batch.
   - Batch Size: This the the size of the regular batches that will be provided in the stream at each data release in terms of number of instances.
   - Time Interval: A batch of data will be released every time interval (unit is seconds)
   - Predictions interval: This determines the due data to submit predictions once the batch received.
6. ***.proto* file**: There is a file describing the structure of the data records that will be sent through the stream for every competition. You will be asked to download and compile it to be able to read and send data in the appropriate format.

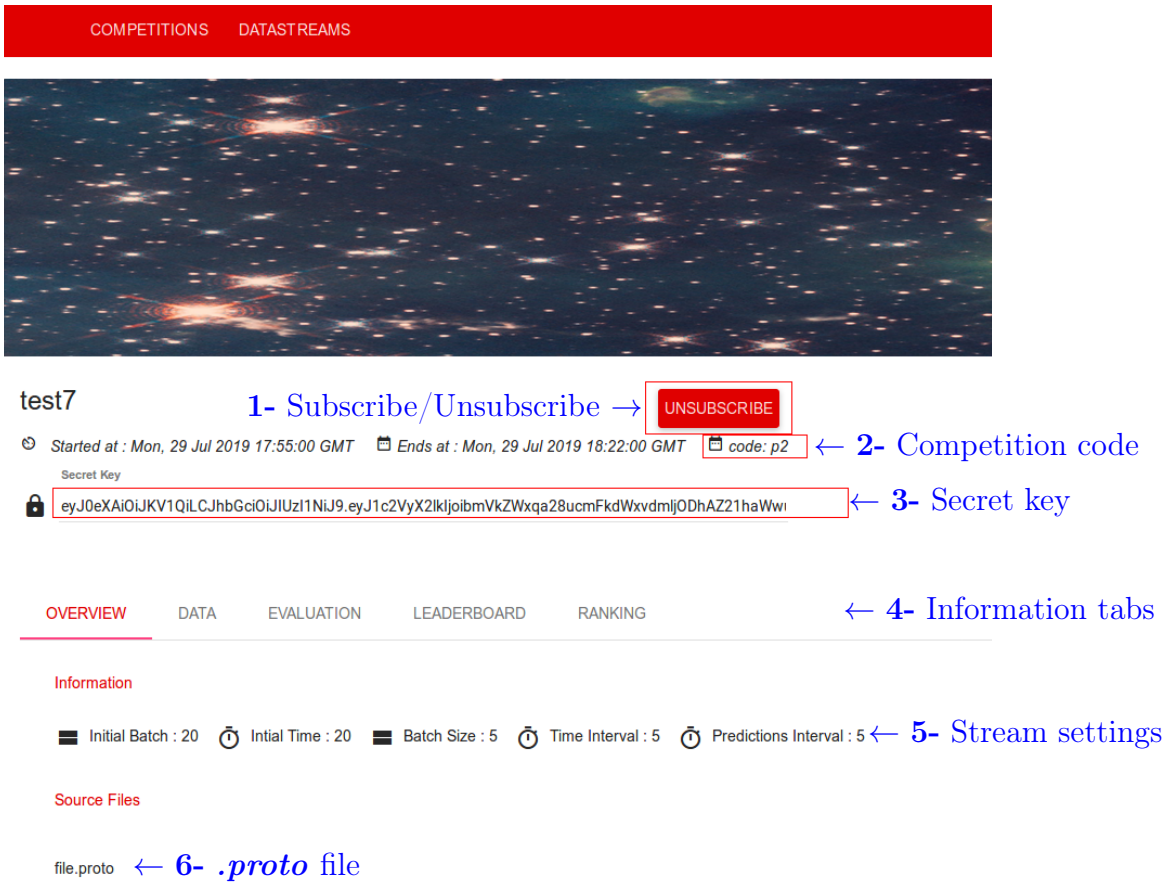


Figure 4: Subscribing to a competition

Chapter 2

2 Environment setup

The platform was conceived to provide users as much freedom as possible in choosing their own resources for building models. They can use any setup they desire as long as they are able to connect to the secure channel and send the predictions on time and in the right format. Users are also allowed to use external data sources if they think that can help improve their models. In this chapter we will state all the software requirements needed to be satisfied in order to be able to participate in the competitions.

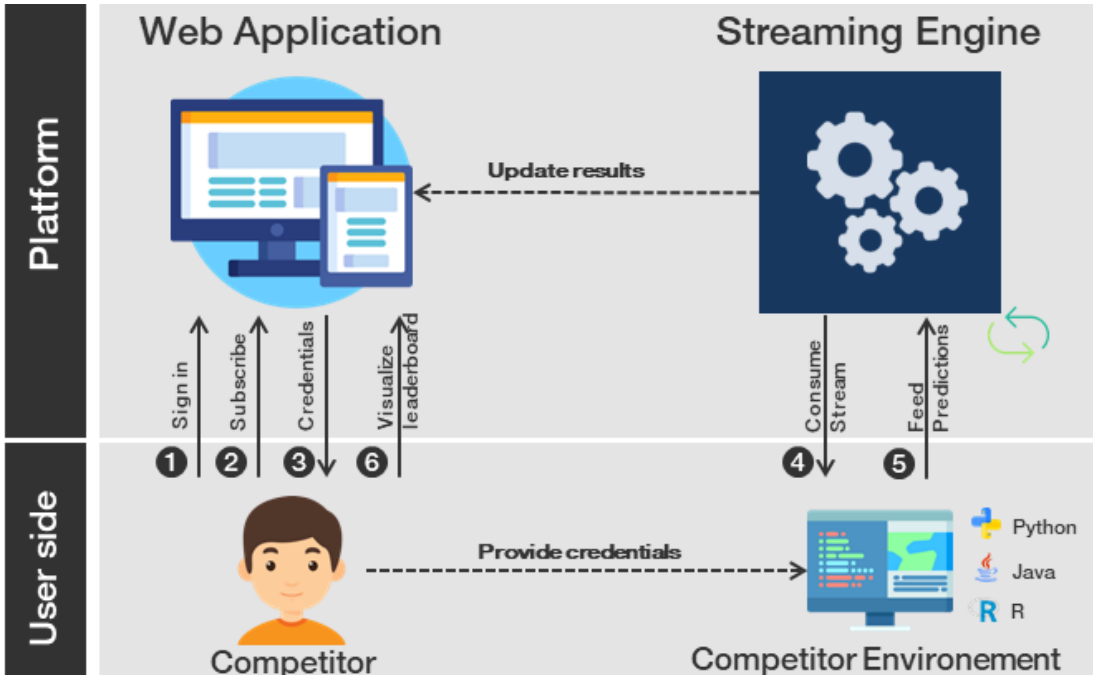


Figure 5: Platform architectures and services

Figure 5[1] gives an overview of the platform's architecture and user interaction with different services. There are two principal components: Streaming engine and Web application and participants will be prompted to interact with the both. The streaming engine is responsible for ensuring bi-directional streams to provide test and training data and receive predictions from every participant. This component will mainly inter-operate with the developing environment (Python, Java

[1]https://www.flaticon.com/

or R) as described in Section 2.1. The web application offers a plethora of options allowing users to browse through competitions' information and real-time results and rankings that will be detailed in Chapter 3. The two components communicate with each other in order to update users' performance based on the predictions received.

2.1 Communication with Streaming engine

Streaming engine and client communication is based on **gRPC** framework with **Protobuf**. This combination provides secure communication, full-duplex bidirectional streaming and an easy way to describe services. Also, this framework is language- and platform-neutral and supports several programming languages (Java, Python, Go, C#, Ruby).

2.1.1 gRPC and Protobuf

Users must install **gRPC** package and compiler for ***.proto*** file In order to be able to communicate with the streaming engine.
gRPC is an open Remote Procedure Call(RPC) framework that can run on different platforms(more than 10 supported languages). It enables client and server applications to communication and facilitates data exchange in micro-services architectures. Google has used **gRPC** to build highly scalable, low latency and distributed systems that connect services, mobile applications, real time communication and IoT while at the same time ensuring efficiency in CPU use and bandwidth.
As mentioned before, the choice of the programming language is left to the user to decide. The only condition is that there is support for **gRPC** framework. **gRPC** framework installation depends on your operating system and the programming language you will be using and is given on this link.

However, you may notice that R is not supported by **gRPC** therefor, we provide a special package (in Starter pack) which enables the usage of R. Instructions for installation are provided in the following sections.

Second requirement for successful communication is **Protobuf**. **Protobuf** stands for **Protocol buffers** and is a language-neutral, platform-neutral, extensible way of serializing structured data for use in communications protocols or data storage. **Protobuf** is thought in the same way as XML but smaller, faster and simpler. We provide a ***.proto*** file for every competition, containing the definition of communication services and data formats used. Instructions for **Protobuf** installations are given on their GitHub repository.

After installing **Protobuf**, the user needs to download the ***.proto*** file from competition page and compile it using `protoc` command. After that, files containing data structures and classes for communication will be generated. We explain in more details about ***.proto*** file in next Section 2.1.2.

2.1.2 *.proto* file

In this section, we will explain the usage of ***.proto*** file, its syntax and the code generated after compiling it. The ***.proto*** file is specific for every competition and is available for download on the competition page.
The example of ***.proto*** file is shown in figure 6.

```
syntax = "proto3";
option java_package = "ex.grpc";
package file;
```
← **1** - Syntax

```
// The data service definition.
service DataStreamer {
// Sends multiple greetings
rpc sendData (stream Prediction) returns (stream Message) {}
}
```
← **2** - Service definition

```
message Message{
int32  rowID =1;
string Day = 2;
string Period = 3;
int32  Target = 4;
string Deadline = 5;
string Released = 6;
string tag=7;
}

message Prediction{
int32 rowID = 1;
int32 Target = 2;
}
```
← **3** - Data format definition

Figure 6: ***.proto*** file example

An example of ***.proto*** file is shown in figure 6 and its parts explained as follows:

1. We define its syntax and some parameters that are related for the languages that we will be using. In this case, we show the example of ***.proto*** file that can be used to generate code for Python and Java. For Python there are no special requirements in this part but we define Java package name with parameter `java_package`.

2. We define the service to be called, which in this case is `DataStreamer`. It is defined as bi-directional streaming service since it (looking from client side) is sending stream of `Predictions` and receives stream of `Messages`.

3. We define the data formats used for communication. In this case we define `Message` and `Prediction`. These structures depend on the competition and dataset used and define what type of message the client will receive from the server (which are the fields, attributes) and in which format the response message should be sent. Any message not corresponding to one of the formats will not be taken into account.

We will show in the following sections the usage of the special `protoc` command to compile ***.proto*** file. We recall that this tutorial addresses Python, Java and R Languages

2.1.3 Python

One of the most used languages for Machine Learning applications is Python. It has dedicated libraries to handle Machine Learning tasks: `scikit-learn`[2], `scikit-multiflow`[3]. Therefor, this section will focus on how to use Python for participating the competition. Alongside with this tutorial, e provide the sample code in Python that can help in joining the competition. We strongly encourage you to use the provided code as a baseline when starting the competition. First step is to download the ***.proto*** file from the competition page and copy it in the same folder with the code example that is provided with this tutorial (`client.py`). In order to be able to run (`client.py`) properly, it is necessary to install the following Python packages:

- `grpcio`
- `grpcio-tools`

This can be done using `pip` command (See the `ReadMe.txt` file in Python folder). Second, it is necessary to compile the ***.proto*** file to generate classes for communication for Python. For this we use `protoc` command.

```
python -m grpc_tools.protoc -I=$SRC_DIR - -python_out=$DST_DIR -
              -grpc_python_out=$DST_DIR file.proto
```

Figure 7: `protoc` command example

We show the typical usage of this command for Python in Figure 7 where we set we set the `python_out` parameter, which defines in which directory compiled files for chosen language will be placed. Afterwards, the appropriate classes and data structures will be created, in our case:

- `file_pb2.py`
- `file_pb2_grpc.py`

In the `client.py` we give the example of the code which shows how to connect to the server and receive messages and send predictions. Every user needs to put his credentials and the address of the server with whom the communication will be established. We highlight those parameters in the figure 8

[2]https://scikit-learn.org/stable/
[3]https://scikit-multiflow.github.io/

```
def __init__(self, batch_size):
    """

    :param batch_size: Integer value, defined by the competition and available at competition page
    :param server_port: Connection string ('IP:port')
    :param user_email: String, e-mail used for registering to competition
    :param token: String, received after subscription to a competition
    :param competition_code: String, received after subscription to a competition
    :param first_prediction: Prediction, class generated from .proto file. Used to initiate communication with the
    server. Not influencing the results. Should contain appropriate fields from .proto file.
    """

    self.batch_size = batch_size
    self.stop_thread = False
    self.predictions_to_send = Queue()
    self.channel = grpc.insecure_channel('52.143.143.143:50051')
    self.stub = file_pb2_grpc.DataStreamerStub(self.channel)
    self.user_email = 'john.doe1984@gmail.com'
    self.competition_code = 'w8'
    self.token = 'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1N20ifQ.HTvQUNLRVR72NFGgOFrqxBVH9ID2GjtktltmToPHFro'
    self.predictions_to_send.put(file_pb2.Prediction(rowID=1000, Target=333))
    self.metadata = self.create_metadata(user_id=self.user_email, code=self.competition_code, token=self.token)
```

Figure 8: Python client example

After that, the main task during the competition is to edit the `loop_messages` method and build the model.

2.1.4 Java

Java is also often used for Machine Learning applications since it also provides several libraries that are dedicated for those tasks: WEKA[4], MOA [5]. For Java we also provide the code example and it can be found in `Java` folder next to this tutorial. We assume that you have followed the steps in section 2.1.1.
In this case you can immediately approach the provided code. We advise you to create Maven project with root directory `competition/`. Go to competition page and download the ***.proto*** file. Copy ***.proto*** to `competition/src/main/proto`. Now in the project root directory `competition/` run command: `mvn clean package`. That command will build the project and create the classes that will be used for communication with server. The files will be generated in `competition/target/generated_sources/protobuf`, as shown in figure 9:

[4]https://www.cs.waikato.ac.nz/ml/weka/
[5]https://moa.cms.waikato.ac.nz/

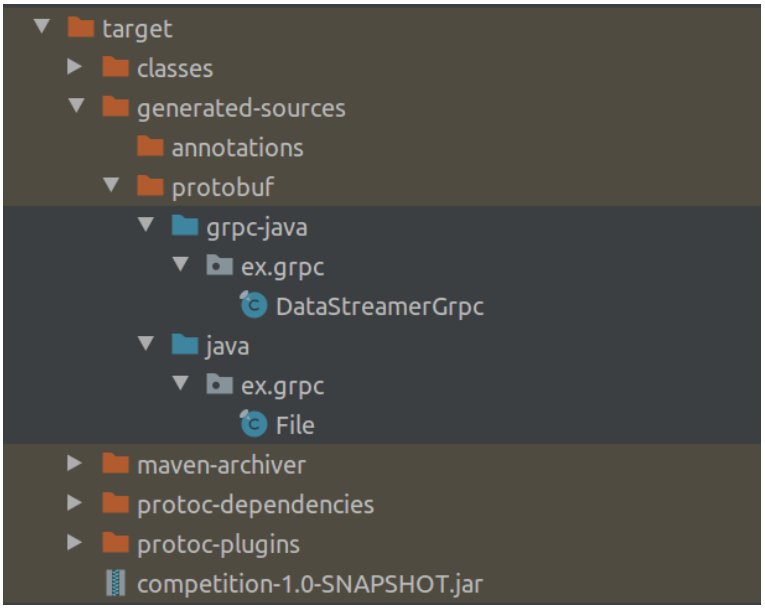


Figure 9: Java generated files

In the `Client.java` we give the example of the code which shows how to connect to the server and receive messages and send predictions. Every user needs to put his credentials and the address of the server with whom the communication will be established. In figure 10 we show the Java class for communication.

```
public class Client implements Runnable {
    /**
     * gRPC client class for Streaming competition.
     * Bi-directional streaming.
     */
    private String topicTitle;
    private ManagedChannel channel;
    private Integer batch_size;
    private File.Prediction first_prediction = Prediction.newBuilder().setRowID(100000).setTarget(3333).build();
    private ArrayList<Prediction> predictions = new ArrayList<>();
    private DataStreamerGrpc.DataStreamerStub stub;
    private String userID, compCode, token;
    private Metadata metadata;
    private StreamObserver<Prediction> requestObserver;

    Client(String topic, String server, Integer port, Integer batch_size, String user, String code, String user_token) {
        /**
         * Client class Constructor
         */
        this.topicTitle = topic;
        this.batch_size = batch_size;
        this.channel = ManagedChannelBuilder.forAddress(server, port).usePlaintext().build();
        this.stub = DataStreamerGrpc.newStub(channel);
        this.predictions.add(first_prediction);
        this.userID = user;
        this.compCode = code;
        this.token = user_token;
        this.metadata = CreateMetadata(this.userID, this.token, this.compCode);
        this.stub = MetadataUtils.attachHeaders(this.stub, this.metadata);
    }
```

Figure 10: Java client class

In figure 11 we give the example of one instantiation of that Java class:

```
public static void main(String[] args) {
    /**
     *
     */
    Client client = new Client( compName: "rtc6",  server: "52.143.143.195",  port: 50051,  batch_size: 5,
            user: "john.doe1984@gmail.com",
            code: "w8",
            user_token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9._WVcsub9eIZkJhoSlKc27OCBNv8QtrDCPjNxgdvA");
    try {


        (new Thread(client)).start();

    } catch (Exception e) {
        System.out.println("Exception");

    }
}
```

Figure 11: Java client example

After that, the main task during the competition is to edit the `predict` method and build the model.

2.1.5 R

Except Java and Python, there is another programming language that is used more and more for data science and data analysis tasks, that is R. R is a programming language and software environment for statistical computing and graphical analysis[6]. It is an interpreted language and is used from command-line but also has several front-ends of which most important is RStudio[7]. R has a variety of libraries and packages that provide statistical and graphical techniques. There exists a variety of libraries and packages dedicated to R that provide classification, clustering, regression techniques, linear and nonlinear modeling, time series analysis. It is widely used by statisticians and data scientists so it was natural to include R as one of the programming languages supported on our platform.

There is one important difference when it comes to using R for our platform. As explained earlier, users need to connect to the server through **gRPC** channel and communicate over **Protobuf** protocol. Both of these are not supported for R programming language. In order to enable R for our platform, we developed a package/wrapper that offers extended capabilities for R and allows users to connect to **gRPC** server. In this way, we are able to use all power that R possesses in terms of statistical techniques that are supported.

Since we will be using Python for communication, the first part of the setup is the same as in the section 2.1.3. Once the ***.proto*** file has been compiled one can switch to R environment. An important thing to mention is to copy the files that have been generated by the `protoc` command to the same directory as the wrapper provided for interconnection to Python. You will be required to

[6]https://cran.r-project.org/
[7]https://www.rstudio.com/products/rstudio/

install the package `reticulate`[8]. Then open the `Rclient.R` file, which contains the example code for participating in the competition. User needs to provide the path to the Python used and import three packages `reticulate`, `jsonlite` and `wRapper2`(which is provided in Starter pack). First part is the same as when using Python. Example of R code is given in figure 12:

```r
library(reticulate)

# provide path to python
use_python("~/anaconda3/bin/python3.6")
source("wRapper2.R")
library(jsonlite)
#######################
#     User data:      #
#######################

# E-mail:
e_mail <- "john.doe1984@gmail.com"
# Token: Obtain token when subscribing to a competition
user_token <- "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ.l8At1FaQIePJsTqwTjNkUyFV5fkUjIfjKt4i0Vl8"
# Competition code: Obtain competition code when subscribing to a competition
comp_code <- "w8"
# Batch size: Check competition configuration
batch_Size <- 5
# set first message to initialize communication
first_message <- toJSON(list(rowID = as.integer(100000), Target = as.integer(3333)))
# Set up server and port of the web application
port <- '52.143.143.195:50051'
# Creating gRPC client using wRapper2.R
gRPC_client <- create_client(batchSize = batch_Size, port = port, user_token = user_token,
                             e_mail = e_mail, comp_code = comp_code, first_message = first_message)
# Starting the client
run_client(gRPC_client)
```

Figure 12: R client example

Our wrapper offers several functions that are used to communicate with Python objects:

- `create_client` - this function creates an object of Python class `Client` which contains methods to ensure communication through **gRPC** and **Protobuf**. This function is called with user information arguments shown in code snippet in Figure 12.
- `run_client` - this function starts a thread in Python that will enter a `While` loop for receiving and sending messages to the server. The `Client` class uses the `Queue` to store the messages that are received from the server and takes the predictions from a list to send to the server.
- `get_messages` - this function will retrieve the messages from the `Queue` so they can be processed in R.

[8] https://rstudio.github.io/reticulate/

- `send_predictions` - after processing the messages and making the predictions using the model in R, these predictions are appended to a list for sending to the server.

The user code in R should call the functions to create and run the object of the `Client` class. After, same as in the case for Python, in a `While` loop user calls functions to retrieve the messages (new instances) from the server, train the model or make the prediction and then call the function to append the prediction to a list for sending.

Chapter 3

3 Tracking results online

If you have followed all the steps described in previous sections, you should be ready to participate in the competition. Visit competition page and check the starting time of the competition. It is very important to start the competition on time, so you can use all the training time to prepare your model. Once the competition has started you will run your program and you will receive first, training batch. This batch will contain many instances in order to give you the opportunity to train your model. Also, this phase will last long enough so you can tweak your model for better results. After the training phase is finished you will start receiving regular batches, and you will be testing your model by predicting and submitting those predictions. It is very important that you have finished tuning your model before this part starts, otherwise you will be penalized for every prediction that you don't submit. Once the regular batches start arriving and you start testing your model, you will be able to track your score and progress on competition page. On competition page there are two sections:

- Leaderboard - graphical representation of contestants scores
- Ranking - ranking list of the contestants on selected evaluation metrics

When you are on `Leaderboard` section of your competition, you need to choose the name of the target(field that you are predicting) and the name of the evaluation metric used in that competition in order to see the live results, as shown in Figure 13.

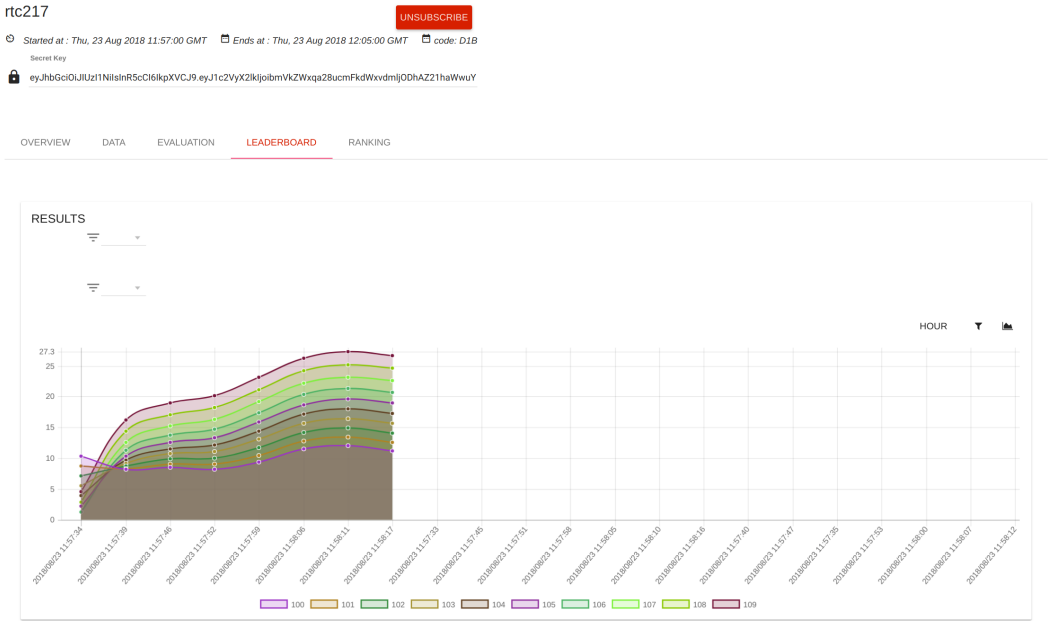
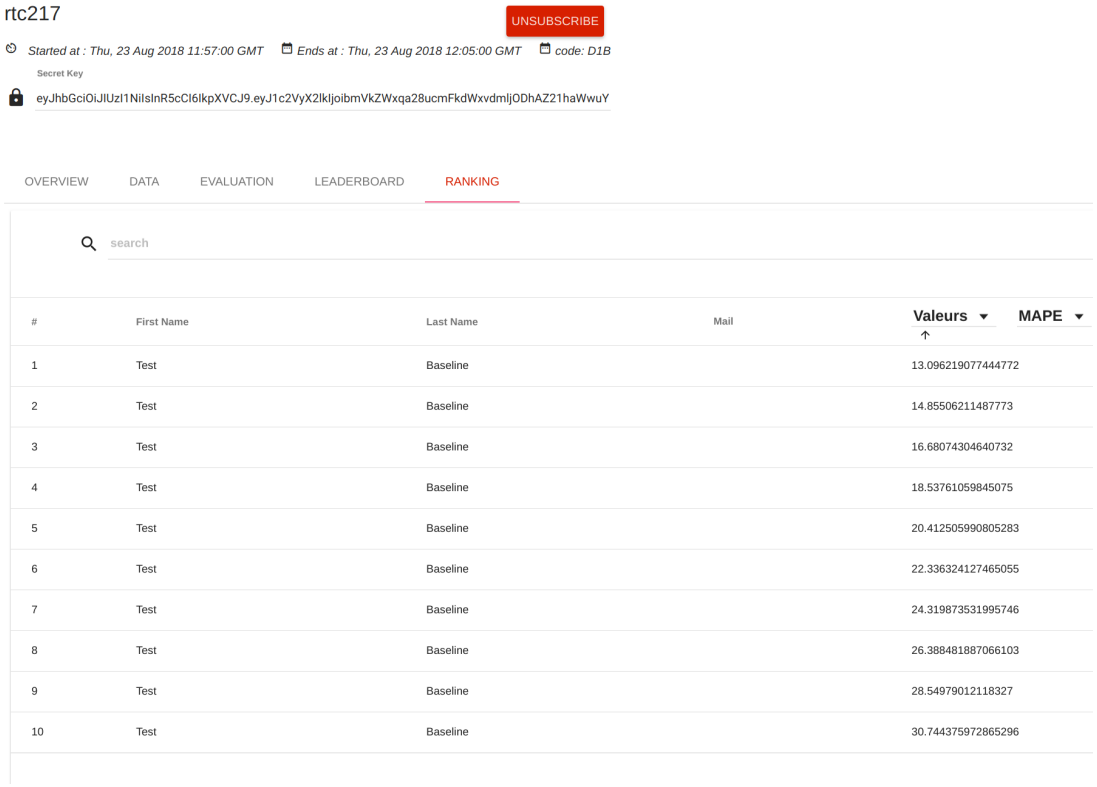


Figure 13: Live results

When you visit the `Ranking` section of your competition, you have to do the same thing, choose the target and evaluation metric in order to see the ranking, like in the figure 14



#	First Name	Last Name	Mail	Valeurs	MAPE
1	Test	Baseline		13.096219077444772	
2	Test	Baseline		14.85506211487773	
3	Test	Baseline		16.68074304640732	
4	Test	Baseline		18.53761059845075	
5	Test	Baseline		20.412505990805283	
6	Test	Baseline		22.336324127465055	
7	Test	Baseline		24.319873531995746	
8	Test	Baseline		26.388481887066103	
9	Test	Baseline		28.54979012118327	
10	Test	Baseline		30.744375972865296	

Figure 14: Participants ranking