

QUICK START GUIDE

SCALAR Platform for real-time Machine Learning Competitions on Data Streams



Authors:

Nedeljko RADULOVIĆ

Dihia BOULEGANE

Albert BIFET

TÉLÉCOM PARIS
Institut Polytechnique de Paris

Abstract

Quick Start Guide and Starter Pack

This document represents the supporting material for **SCALAR** - a novel platform for real-time machine learning competitions on data stream. 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 setup a competition, 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

Abstract	i
List of Abbreviations	iv
1 Accessing and Registration to the platform	1
2 Setting up a competition	1
3 Environment setup	6
3.1 Communication with server	6
3.1.1 gRPC and Protobuf	6
3.1.2 Python	7
3.1.3 Java	8
3.1.4 R programming language	10
4 Tracking results online	13
Tracking results online	13

List of Figures

1	Login dialog	1
2	Registration dialog	1
3	Adding a datastream	1
4	<i>.proto</i> file example	2
5	Adding a competition	3
6	Browsing through competitions	4
7	Subscribing to a competition	5
8	<i>protoc</i> command example	7
9	Python client example	8
10	Java generated files	9
11	Java client class	9
12	Java client example	10
13	R client example	11
14	Live results	14
15	Leaderboard	14

List of Abbreviations

IoT	I nternet o f T hings
gRPC	g oogle R emote P rocedure C all
Protobuf	P rotocol B uffers
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 start the the application, copy the code and follow the instructions on the [Github](#) page of the platform. Once you started the application, you can access the web application from your browser at: <http://localhost:80/>.

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)

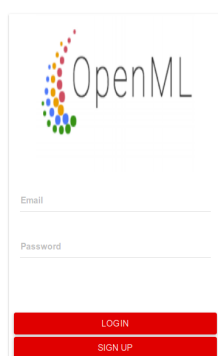
The login dialog features the OpenML logo at the top. Below it are two input fields: 'Email' and 'Password'. At the bottom, there are two red buttons: 'LOGIN' and 'SIGN UP'.

Figure 1: Login dialog

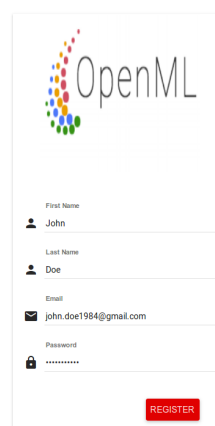
The registration dialog features the OpenML logo at the top. Below it are four input fields: 'First Name' (with 'John' entered), 'Last Name' (with 'Doe' entered), 'Email' (with 'john.doe1984@gmail.com' entered), and 'Password' (with masked characters). A red 'REGISTER' button is located at the bottom right.

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.

You will be registered as regular user, to get the **ADMIN** privileges, connect directly to your **MySQL** database and change the role in the table **USERS**. From terminal, connect to the **MySQL** database. To fetch the ip-address of the 'sql_db' container run:

```
docker inspect sql_db
```

Then connect to the database `sample`, and in the table `USERS`, change your role to `ADMIN` instead of `USER`:

```
mysql -u mysql -pmysql -h sql_db IP_address sample
```

After this change, restart the platform.

Chapter 2

2 Setting up a competition

Once signed in as an user with `ADMIN` privileges, you are able to set up a competition. First step is to upload the data stream. Under the `DATASTREAMS` tab, on the web application, there is an option `ADD DATASTREAM`, which launches simple dialog box where you provide the name, description and the data stream file, as shown in the Figure 3:

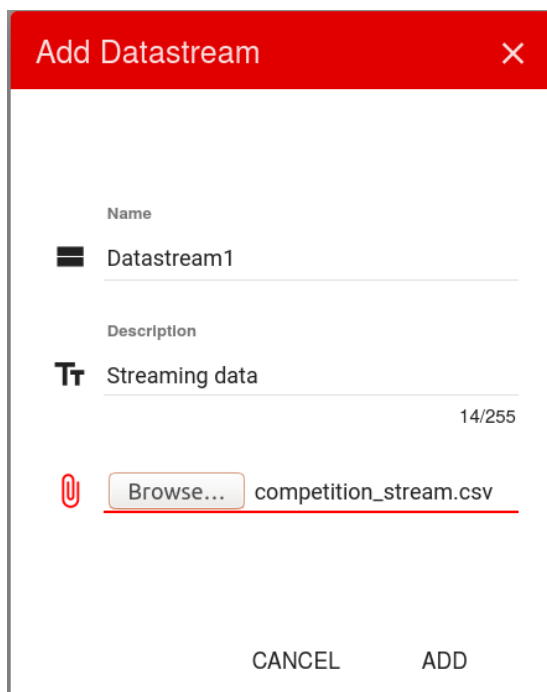


Figure 3: Adding a datastream

After adding a data stream file, you need to create the *.proto* file that corresponds to that datastream. The *.proto* file is used to define the communication between the users and the server and we explain this communication in Section 3.1. Here, we will explain the usage of *.proto* file and its syntax and the

code generated after compiling it. The **.proto** file is specific for every competition and it is available for download on the competition page.

The example of **.proto** file is shown in Figure 4.

```

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

// 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 date = 3;
    float target = 4;
    string Deadline = 5;
    string Released = 6;
    string tag=7;                               ← 3 - Data format definition
}

message Prediction{
    int32 rowID = 1;
    float target = 2;
}

```

Figure 4: **.proto** file example

In the first part of **.proto** file 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`.

The rest of the **.proto** file is dedicated to definition of service and data formats used for communication.

First we define service, which in this case is `DataStreamer` and it is defined as bi-directional streaming service since it (looking from client side) is sending stream of `Predictions` and receives stream of `Messages`. After defining the service, we define the data structures. In this case we define `Message` and `Prediction`. These structures depend on the competition and dataset used and in this way it is defined what type of message will client receive from the server (which are the fields, attributes) and in which format the response message should be sent. When defining the `Message` data structure, one needs to include the names and the types of all features of a given dataset and additional fields for maintaining the competition:

- `rowID` - identifier of each instance is added when the stream is generated
- `Deadline` and `Released` - timestamp which indicates when the instance was released/ when is the deadline to send a prediction

- **tag** - Every instance is tagged depending to which batch it belongs: Test, Train or Initial.

For **Prediction** data structure, it is sufficient to receive the **target** value and the **rowID** of the instance, for which prediction is being sent.

After writing the **.proto** file, you can create a competition using the data stream file that has been uploaded. Go to **COMPETITIONS** tab and click **ADD COMPETITION**. New dialog box will appear, as shown in Figure 5. In the dialog box, as in Figure 5, you setup the name of the competition and the datastream file to be used for it. Then you define the name of the column that represents the value to predict (**target**). You can define several columns. For each of them choose the evaluation metric, selecting it from drop down menu next to the **ADD FIELDS** button, and confirm the selection by clicking the **ADD FIELDS** button. Next section is competition setting, which includes: start and end date of the competition, intervals between two **batches**, interval to send prediction, etc. In the end, you need to upload the **.proto** file and you can add the description for the competition.

Add Competition [X]

Name
Test Competition

Datastream
Datastream1

Has Target
☒

Target
target

ADD FIELDS

Start Date and Time

Start Date: 07 / 09 / 2020 Start Time: 06 : 00 PM

End Date and Time

End Date: 07 / 09 / 2020 End Time: 08 : 00 PM

Initial Settings

Initial Training Time: 20 Initial Batch Size: 100

Regular Settings

Time Interval: 5 Batch Size: 5

Predictions Settings

Predictions Interval: 5

CANCEL ADD

Figure 5: Adding a competition

After creating a competition, you will be able to see it in the **COMPETITIONS** (active, coming and finished) as shown in Figure 6.

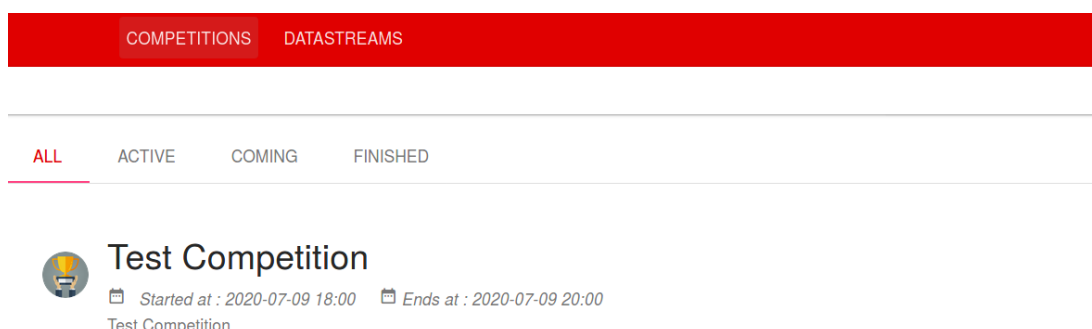


Figure 6: Browsing through competitions

You can click on a specific competition and browse for further details as shown in Figure 7 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.

The screenshot shows the Kaggle 'Test Competition' interface. At the top, there are tabs for 'COMPETITIONS' and 'DATASTREAMS'. Below this is a header image of a galaxy. The main section is titled 'Test Competition' and includes a red 'UNSUBSCRIBE' button. To the right of the button, blue arrows point to annotations: '1- Subscribe/Unsubscribe' and '2- Competition code'. Below the button, the competition details are shown: 'Started at : 2020-07-09 18:00' and 'Ends at : 2020-07-09 20:00'. A 'code: p2' label is also present. Below this, a 'Secret Key' is displayed, with a blue arrow pointing to it labeled '3- Secret key'. At the bottom, there are tabs for 'OVERVIEW', 'DATA', 'EVALUATION', 'LEADERBOARD', and 'RANKING', with a blue arrow pointing to them labeled '4- Information tabs'. Under the 'OVERVIEW' tab, there is an 'Information' section with stream settings: 'Initial Batch : 20', 'Initial Time : 20', 'Batch Size : 5', 'Time Interval : 5', and 'Predictions Interval : 5'. A blue arrow points to these settings labeled '5- Stream settings'. Below this is a 'Source Files' section with a link to 'file.proto', with a blue arrow pointing to it labeled '6- .proto file'.

Figure 7: Subscribing to a competition

Chapter 2

3 Environment setup

This platform was conceived with goal to allow users as much freedom as possible in choosing their resources for building models. Users are free to use any setup they desire as long as they are able to connect to the secure channel and they are sending the predictions on time and in the right format. They 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.

3.1 Communication with server

For communication between server and client, we used **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).

3.1.1 gRPC and Protobuf

In order to be able to communicate with the server, users have to install **gRPC** package and compiler for *.proto* file.

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 it is already mentioned, it is up to user to decide which programming language will he/she use, but the only condition is that there is support for **gRPC** framework. The procedure for installation **gRPC** framework depends on your operating system and the programming language you will be using and is given on this [link](#). If you want to use R as programming language, you will notice that it is not supported by **gRPC**, but we provide special package (in Starter pack) which enables usage of R. In that case follow the instructions for installation of **gRPC** for Python.

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. **.proto** file is provided for every competition because, in that file, we have the definition of communication services and data formats used in communication. 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 compilation, files containing data structures and classes for communication will be generated. We explain details about **.proto** file in next section.

For compiling **.proto** file, a special **protoc** command is used. We will show the usage of this command for different languages in following sections.

3.1.2 Python

One of the most used languages for Machine Learning applications is Python. It has dedicated libraries to handle Machine Learning tasks: **scikit-learn**¹, **scikit-multiflow**². That is why we dedicate this section to show how to use Python for participating the competition. Alongside with this tutorial we have provided you the example code in Python which can help you to involve in competition much easier and faster. We strongly encourage you to use the sample code as a starting point 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 following Python packages:

- **grpcio**
- **grpcio-tools**

This can be done using **pip** command (See the **ReadMe.txt** file in Python folder). Now, it is necessary to compile the **.proto** file to generate classes for communication for Python. For this we use **protoc** command. We show the typical usage of this command for Python:

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

Figure 8: protoc command example

¹<https://scikit-learn.org/stable/>

²<https://scikit-multiflow.github.io/>

In the figure 8 we set the `python_out` parameter and in this way, we define for which language we are compiling the *.proto* file. After running this command files, with 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 9

```
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.Queue()
    self.channel = grpc.insecure_channel('52.143.143.143:50051')
    self.stub = file_pb2_grpc.DataStreamStub(self.channel)
    self.user_email = 'john.doe1984@gmail.com'
    self.competition_code = 'w8'
    self.token = 'ew2hbc101Juz11N1IsInR5CCI61kpXVC39_ey21b21yZXRpdGlvd19pZCI6MTMsInVzZXJfaRTl0JmZuZWRlbnR5S3YwR1BjC92aWM4OEbnWFRtCjB2b01fQy7Tps_w8-T1qR69M3NnQ8pgc7Ivdy831fsSyom8fZQ'
    self.predictions_to_send.put(file_pb2.Prediction(rowID=1000000, Target=333))
    self.metadata = self.create_metadata(user_id=self.user_email, code=self.competition_code, token=self.token)
```

Figure 9: Python client example

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

3.1.3 Java

Java is also often used for Machine Learning applications since it also provides several libraries that are dedicated for those tasks: WEKA³, MOA⁴. 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 3.1.1.

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 10:

³<https://www.cs.waikato.ac.nz/ml/weka/>

⁴<https://moa.cms.waikato.ac.nz/>

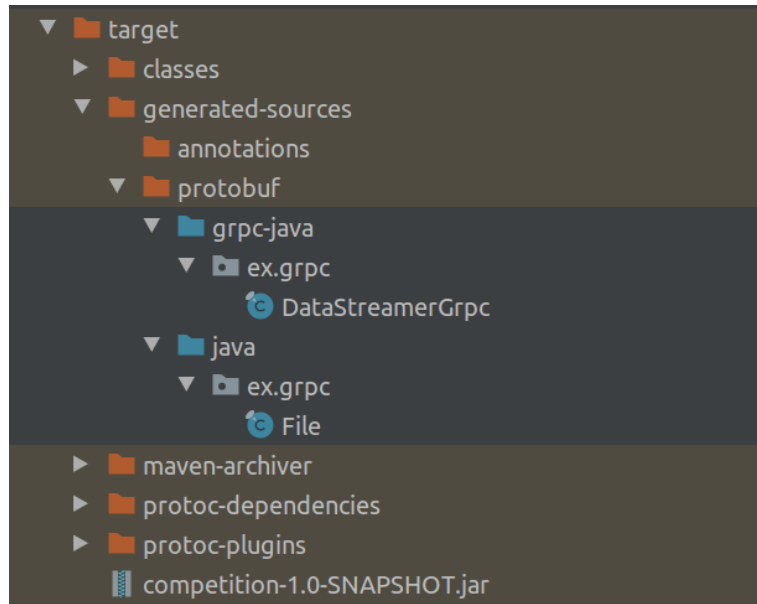


Figure 10: Java generated files

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 11 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 DataStreamGrpc.DataStreamStub 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 = DataStreamGrpc.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 11: Java client class

In figure 12 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.eyJWc3sub9eIZkZhoSlKc270CBNV8QtrDCPjNxdGdvA");

    try {

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

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

Figure 12: Java client example

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

3.1.4 R programming language

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 analysis⁵. R is interpreted language and usually is used from command-line but also has several front-ends of which most important is RStudio⁶. R has a variety of libraries and packages that provide statistical and graphical techniques. Those libraries provide classification, clustering, regression techniques, linear and non-linear modeling, time series analysis and many others. Its source code is written in C, Fortran and R. It runs and compiles under many UNIX platforms (including Linux), Windows and MacOS. 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 order to overcome this we provide a specific package for R. This package enables communication with **gRPC** server using Python. It also provides connection between Python and R so the calculations can be done in R and sent to Python to be forwarded to **gRPC** server.

⁵<https://cran.r-project.org/>

⁶<https://www.rstudio.com/products/rstudio/>

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 3.1.2. Once the *.proto* file has been compiled one can go 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 install the package `reticulate`⁷. Then open the `Rclient.R` file, which contains the example code for participating in 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 13:

```
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 <- "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJiOiJ8At1FaQIePJsTqwTjNkUyFV5fkUjIfjKt4i0VL8"
# 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(333)))
# 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 13: R client example

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

- `create_client` - this function creates object of Python class `Client` which contains methods for communication through **gRPC** and **Protobuf**. This function is called with user information arguments mentioned in code snippet 3.4.

⁷<https://rstudio.github.io/reticulate/>

- `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
- `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

4 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 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 tweaking your model before this part starts, because 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, like in figure 14:

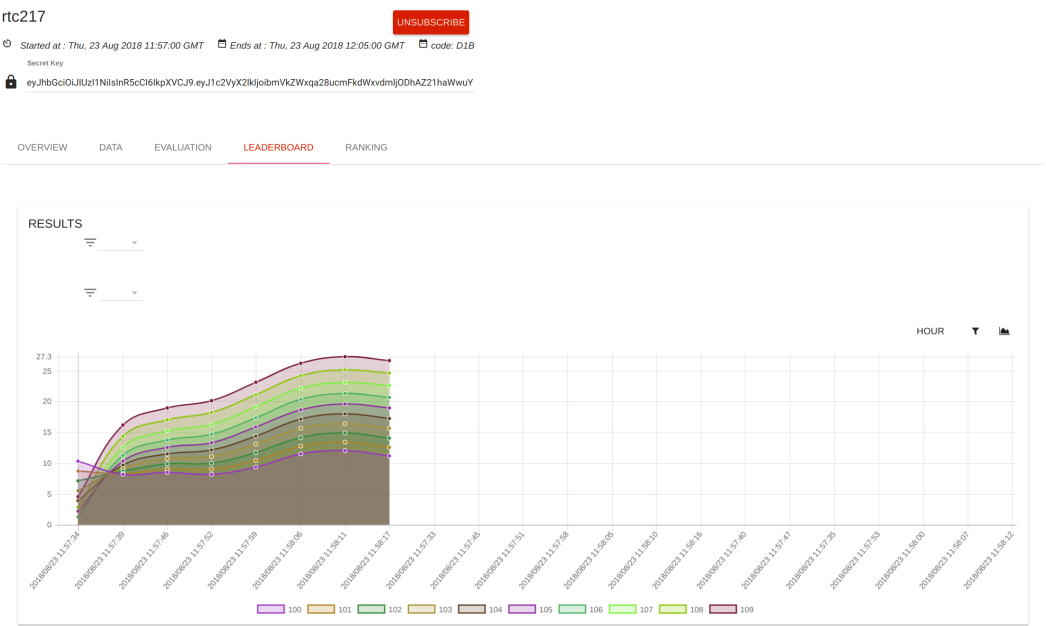


Figure 14: Live results

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 15

rtc217

Started at : Thu, 23 Aug 2018 11:57:00 GMT

Ends at : Thu, 23 Aug 2018 12:05:00 GMT

code: D1B

Secret Key

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2kiOiJ0bmVrZWxqa28ucmFkdWxvdmJ0dHAZ21haWwuyY

OVERVIEW

DATA

EVALUATION

LEADERBOARD

RANKING

Q search

#	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 15: Leaderboard