universite
PARIS-SACLAY

Projet Master 2 Informatique
Apprentissage, Information et Contenu (AIC)

Preparing a starting kit and a challenge bundle

Isabelle Guyon
iguyon@lri.fr
** Please preferably communicate through Piazza **
Version 2, Nov. 14, 2018

Getting started:

(1) Install on your computer **Anaconda Python 3.6**. If you already have another version of Python or Anaconda on your computer, use virtual environments:

```
conda update conda
conda create -n python3 python=3.6 anaconda
```

Then you will be able to use the 3.6 version by activating the virtual environment:

```
conda activate python3
```

To get out, use:

```
conda deactivate
```

This installation should include python 3.6, the spyder IDE and jupyter-notebook (verify).

(2) Another possibility will be to use the docker codalab/codalab-legacy:py3. More on dockers at the end of this document. If you are going to use OTHER libraries than those provided in Anaconda, definitely use dockers!

(3) Clone the starting kit template: https://github.com/madclam/m2aic2019/tree/master. If you are new to Github, take a tutorial. Using Git and Github is essential to your project.

(4) **When done creating your starting kit, push it to your own Github repo (your homework will be your Github repo URL). Turn in your homework on Piazza, tag 2.starting-kit before Nov. 21.**

(5) Each group has an email groupname@chalearn.org, which has been used to create accounts on: https://chalab.lri.fr/ and https://codalab.lri.fr/.
The **password** in the same for all accounts: **m2_aic++**

1) Starting kit structure:

A starting kit should imperatively have the following structure, do NOT change it (with exception of the L2RPN group working on reinforcement learning for whom the data is replaced by the simulator):

- **README.md:** Short description of your task and credits.
- **README.ipynb:** Sample code reading data, calling sample code, generating sample submissions.
- **sample_data/:** A SMALL data sub-sample (could be a subset of the training data, but should NOT include any validation or test examples on which the competitors will be tested).
- **sample_result_submission/:** Example prediction files, for the validation and test set.

- **sample_code_submission/:** Self-contained, ready-to-submit example code submission, including the predictive mode class `model.py` and a `metadata` file (with just a comment).
- **ingestion_program/:** Program and libraries needed to run code submissions on the challenge platform and generate prediction results using the `input_data` (labeled training data and unlabeled validation and test data) and `model.py`.
- **scoring_program/:** Program and libraries needed to score the prediction results on the challenge platform using the target values (solution), also called `reference_data`. A library of scoring functions to choose from is provided in libscore.py. All you need to do is to put the name of the chosen function in `metric.txt`. If needed, you can supply the code of your own metric in `my_metric.py`. You can also use a scikit-learn metric.

2) Data preprocessing and formatting:

This year we have a variety of tasks, depending on the project you chose. Those include **classification**, **regression (survival), matrix factorization**, and **reinforcement learning**.

- **Data representation:** For the first (basic) version of the starting kit, all datasets will have to be transformed/preprocessed into a "**feature representation**". This especially important for text, speech, vision, or any datasets whose raw data consist of images or sequences. For example, for image data, you can use features calculated by a pre-trained CNN. The Areal and PersoData groups working on computer vision applications will keep a non-preprocessed version of their dataset, for later (we'll do a second version from raw data for more advanced students).
- **Missing data:** Avoid dataset will a large number of missing values (more than 10%). Represent the missing values with the symbol **NaN.**
- **Binary and categorical variables:** Represent all variables with a **NUMERIC code**. Avoid categorical variables with a large number of values, some of which appear very infrequently. To that end, you may group infrequent categories. Likewise, for binary variables, avoid datasets in which variables are very imbalanced (very sparse data). The target values (solutions) must also be in a numerical format.
- **Number of classes:** Avoid problems with a too large number of classes (>20) and have too few examples per class (less than 1000 in test data).
- **Size dataset:** Do not consider too small datasets of less than 50 000 examples. You must have at least **10 000 test examples** and preferably a large training set and a validation set of the same order of magnitude as the test set. We want enough samples, but the overall preprocessed compressed volume of data should not exceed **300 MB**. Quantizing values to enough precision (but not too much) is recommended, as part of preprocessing. Usually quantizing between 0 and 999 is enough. To further reduce data volume, you may also reduce the number of features by feature selection (this should be the last resort because it is good for the L2 students to struggle with many features). So no limit on the number of features.
- **Data homogeneity:** Avoid data that are non-homogeneous or non-identically-distributed. Shuffle the order of your examples randomly.

Data format: All pre-processed data must be produced in **AutoML format:**.

This is imperative because you will all share the same data reader (called by the ingestion program). You will have to split the data into a **"training set",** a development test set a.k.a.**"validation set"** for the challenge phase 1, and a (final) **"test set"** for the challenge phase 2. You should eventually produce ALL the following files, but, for the first version, produce at least those outlined in boldface:

DataName_train.data # This is the training matrix X
DataName_train.solution # This is the training matrix Y (target values)
DataName_valid.data # This is the first test set called "validation set" used during development
DataName_valid.solution # and its target values (hidden to the participants)
DataName_test.data # This is the FINAL test set
DataName_test.solution # and its target values
DataName_feat.name
DataName_label.name
DataName_feat.type
DataName_public.info
DataName_private.info

We ask that you to:
+ Provide all the requested files (even those not in boldface) because this will greatly help the participants and this be useful documentation for future reference.
+ NOT provide "un-split" data in the simpler form: DataName.data and DataName.solution, because **providing your own data-split** is better for several reasons:

- You have more control and you can, for instance, ensure that each class has a balanced mix of examples (in the case of multiple data sources of multiple types of subjects), or, on the contrary, that you have different types of examples in training and test data (for instance if you want to address speaker independent speech recognition, samples from different speakers should be in training and test data).
- You are sure that the examples are well shuffled.
- You are sure that the example order matches your sample result submission.

Data leakage: This refers to features that inform inadvertently about the target values: running a feature selection algorithm often reveals such features. Typically, there may be a column with sample IDs highly correlated with the target. There are other forms of data leakage. For example if you forget to shuffle the examples and all the examples of the same class appear next to one another. Check https://www.cs.umb.edu/~ding/history/470_670_fall_2011/papers/cs670_Tran_PreferredPaper_LeakingInDataMining.pdf for other horror stories.

Sample data and challenge data: You need to prepare 2 datasets in AutoML format:

- **Challenge data:** your actual challenge data. In the challenge platform, it will be split into downloadable **public_data** or **input_data** (everything except the DataName_valid.solution, DataName_test.solution, and DataName_private.info) and **reference_data** not leaving the platform: DataName_valid.solution for phase 1 and DataName_test.solution for phase 2.
- **Sample data:** a small data subset carved out the challenge TRAINING data, for practice purposes only (do not compromise real validation or test data).

3) Sample submissions:

You must prepare two zip files:

1) sample_result_submission.zip: A zip file with NO directory structure containing 2 files:

- **dataName**_valid.predict
- **dataName**_test.predict

corresponding to predictions of the **target values** made by your sample code for the validation set and the test set, one sample per line. For the HADACA project, your participants must predict 2 matrices A and T of dim (N, C) and (C, M). To respect the format, please write matrix A and T' one after the other to get a file with N+M lines and C columns.

Tentatively, you can put random values or constant values (with the right number of lines), but later you may want to create a "real" sample submission with the sample code provided in sample_code_submission. One easy way is to upload a code submission to Codalab, then "Download evaluation output from prediction step", but you need to remove the `metadata` file!

2) sample_code_submission.zip: A zip file with NO directory structure containing a Python class `model.py`. This should be a basic but functional baseline method solving your problem. Using the "ingestion program" (see below), you can generate a sample_result_submission from your sample_code_submission. This zip file should also contain a `metadata` file with just a comment. It may or may not contain a pickle with a trained model (maybe you want to create two versions).

4) Ingestion program:

A sample ingestion program is provided (it should not need to be modified if you respect the AutoML format and the sunmission format). The ingestion program receives the code submission of the participants on the server and runs it. However, you can also run it locally to produce a result submission: python ingestion_program/ingestion.py **public_data** sample_result_submission ingestion_program sample_code_submission

where `public_data/` should contain your real challenge data (but not the validation and test target values/solutions). **Remove the pickle in sample_code_submission, if any** (may cause an error).

5) Scoring program:

A scoring program is provided. It should not need to be modified, however, the scoring metric can be chosen by the organizers (you). Place your code in `my_metric.py,` and indicate the name of your scoring function in `metric.txt.` Instead of writing your own code, you may also select one of the metrics for libscore.py and specify it in `metric.txt`:

Binary classification	**Multi-class classification**	**Multi-label classification**	**Regression**
bac_binary	bac_multiclass	bac_multilabel	abs_regression
auc_binary		auc_multilabel	r2_regression
pac_binary	pac_multiclass	pac_multilabel	
f1_binary		f1_multilabel	

Test your metric with:

python scoring_program/score.py **reference_data** sample_result_submission scoring_output

where `reference_data/` should contain your validation and test target values/solutions: DataName_valid.solution for phase 1 and DataName_test.solution for phase 2, and `scoring_output` is a directory that will be created.

6) Jupyter-notebook:

We provide a sample Jupyter notebook to help the participants getting started. Keep the basic structure of the template but replace the yellow place holders with relevant graphics and explanations: jupyter-notebook README.ipynb.

7) From the starting kit to the challenge bundle:

The challenge data and starting kit are the basis for the Codalab challenge. Here is your **check list**:

- ✓ LOGO: Prepare a **logo** (a small square jpg file) that we will refer to as **DataName_logo.jpg.**
- ✓ DATA: The challenge **data** should have this structure:

DataName/	**DataName_train.data**
	DataName_train.solution
	DataName_valid.data
	DataName_valid.solution
	DataName_test.data
	DataName_test.solution
	DataName_feat.name
	DataName_label.name
	DataName_feat.type
	DataName_public.info
	DataName_private.info

Zip it **with** the top directory structure and call it **DataName_data.zip.**

At this stage it is fine to use a small sample dataset as a place holder for debug purposes.

- ✓ STARTING KIT: The **starting kit** should have this structure:

README.md.
README.ipynb
sample_data/
sample_result_submission/
sample_code_submission/
ingestion_program/
scoring_program/

Zip it and call it **DataName_starting_kit.zip.**

- ✓ PROBLEM: The contents of `ingestion_program/` should be un-changed:

metadata
ingestion.py
data_converter.py
data_io.py
data_manager.py

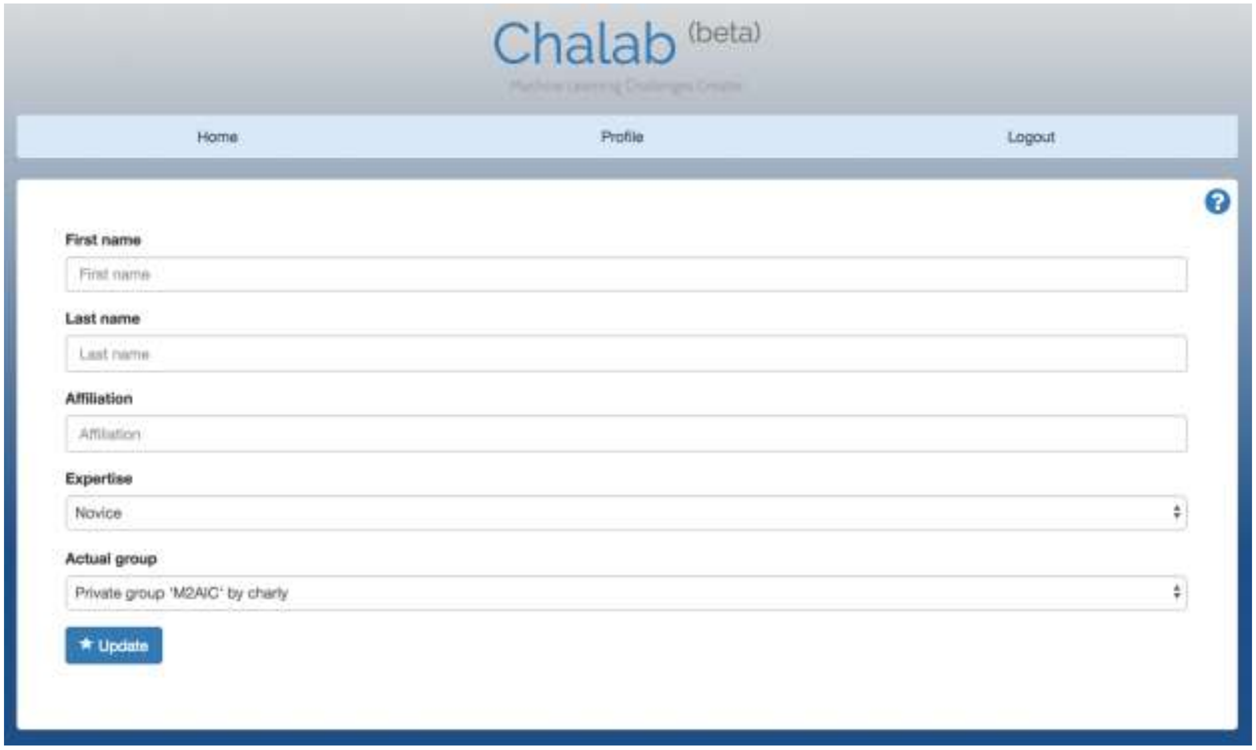
Zip it without directory structure and call it **DataName_ingestion.zip.**

- ✓ METRIC: Make a note of the name of the **metric** you used (e.g. **chosen_metric**), declared in `scoring_function/metric.txt,` if you used one of the metrics provided in `libscores.py`, OR save the code of your metric in a file **my_metric.py.**
- ✓ SAMPLE SUBMISSIONS: You should also have prepared 2 **samples submissions**:

sample_code_submission.zip [containing an empty metadata file and model.py]
sample_result_submission.zip [this one should have been generated from the sample code submission using the challenge data (that you will put on the challenge platform) **NOT the sample data, which is just for debug purposes**; it should contain DataName_valid.predict and DataName_test.predict].

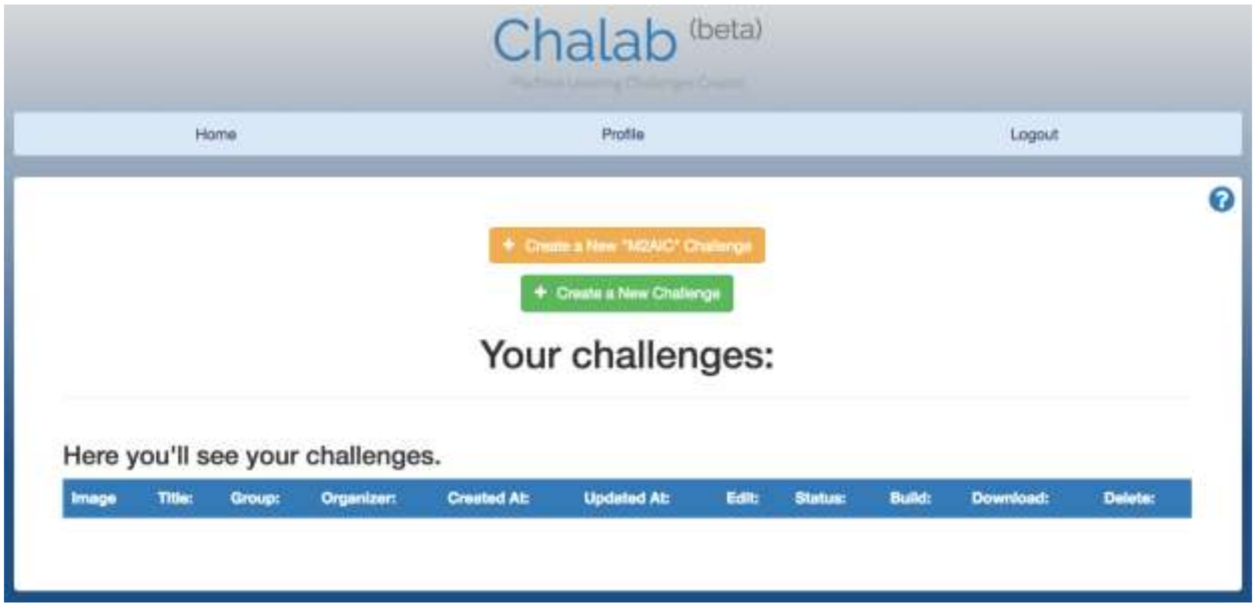
Upload your challenge to ChaLab

Go to http://chalab.lri.fr/. We created accounts with your group name ({vision, friend, ecolo, credit, biomed}@chalearn.org, please use them. You are part of a class group that can use a pre-filled challenge template. To activate it, go to "Profile" and select "Private group M2AIC":



First name: First name
Last name: Last name
Affiliation: Affiliation
Expertise: Novice
Actual group: Private group 'M2AIC' by charly

The create a "New m2aic challenge":

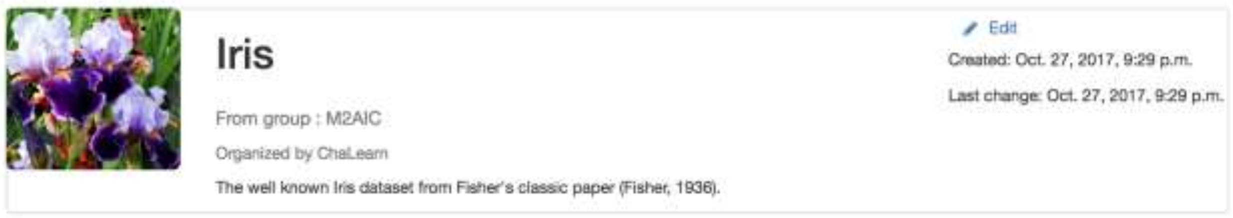


Your challenges:

Here you'll see your challenges.

Image	Title:	Group:	Organizer:	Created At:	Updated At:	Edit:	Status:	Build:	Download:	Delete:

On the next page, click "Edit" at the top to replace the logo by **DataName_logo.jpg** and the challenge information:



Iris

From group : M2AIC

Organized by ChaLearn

The well known Iris dataset from Fisher's classic paper (Fisher, 1936).

Created: Oct. 27, 2017, 9:29 p.m.

Last change: Oct. 27, 2017, 9:29 p.m.

If you are using the Iris template, you only have to perform the green steps:

1) **Data:** Upload **DataName_data.zip**. Note: **NO file larger than 300MB**. If your dataset is >50MB, we recommend to upload only the sample data to ChaLab. Then follow instructions in the next section to upload the large dataset to Codalab directly.
2) **Split:** Your data should be pre-split if you followed the instructions; nothing to do.
3) **Problem**: The program **DataName_ingestion.zip** should already be there: nothing to do.
4) **Metric:** choose **chosen_metric** in the list of default metrics or upload **my_metric.py.**
5) **Protocol:** we impose a 2-phase challenge. Keep the date of the template. The beginning of phase 2 shoud be **APRIL 30, 2018:** nothing to do.
6) **Baseline:** Upload **DataName_starting_kit.zip**. No large sample data should be bundled with the starting kits. Files <50 MB please.
7) **Documentation:** Fill up the documentation pages appropriately. **Do not change the terms/rules.**

Then **Package & Publish:** Build your competition bundle and download the zip (challenge bundle): **DataName_challenge_bundle.zip**

Upload the challenge bundle to Codalab

1) Upload **DataName_challenge_bundle.zip** to https://codalab.lri.fr/.

To do this, use your account **groupname@chalearn.org**.

2) Test your challenge with your sample submissions: **sample_code_submission.zip** and **sample_result_submission.zip.**

To do this you will need to go to **Participate>View/Submit results.**

WARNING: you will get errors if the correct docker is not specified. See page 11.

Large datasets

If you have a very large dataset and only uploaded sample data instead of your real challenge dataset on ChaLab, follow these instructions to upload data directly to Codalab.

1) Separate the target values from the rest and create 3 archives:

- **DataName_input_data.zip**, containing:

DataName_train.data
DataName_train.solution
DataName_valid.data
DataName_test.data
DataName_feat.name
DataName_label.name
DataName_feat.type
DataName_public.info

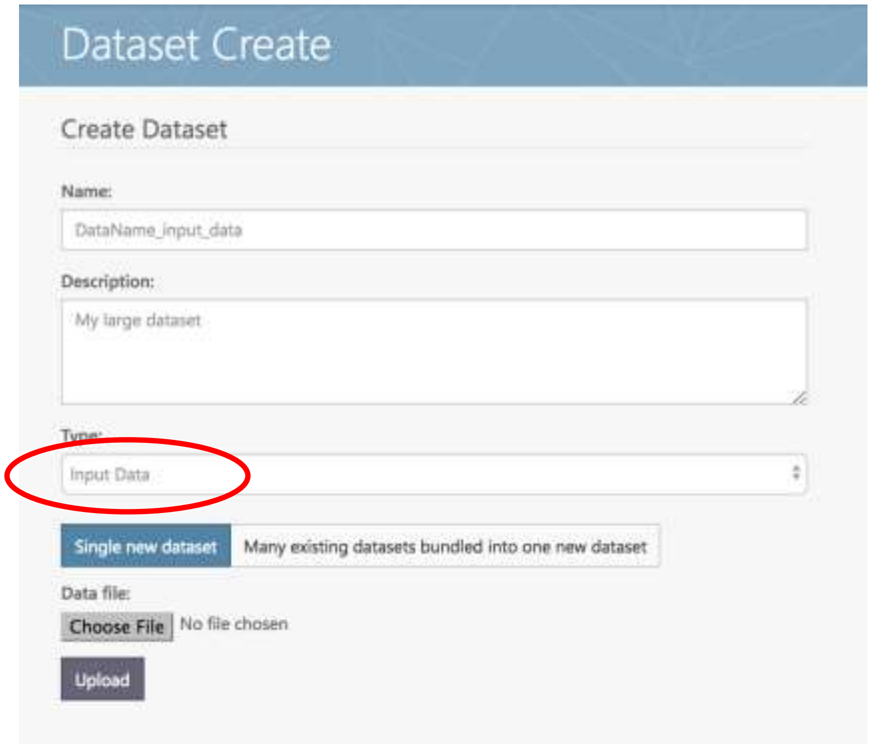
- **DataName_reference_validation_data.zip**, containing:

DataName_valid.solution
DataName_public.info
DataName_private.info

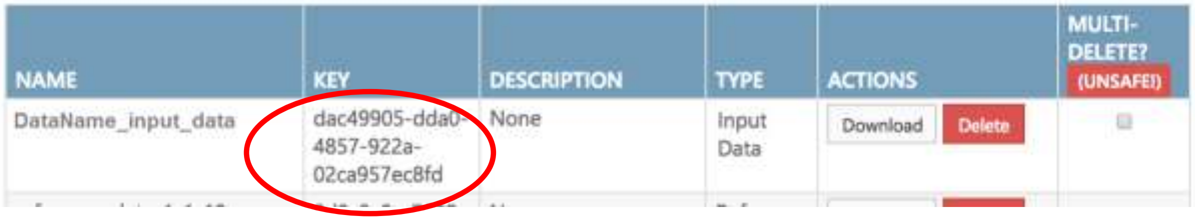
- **DataName_reference_test_data.zip**, containing:

DataName_test.solution
DataName_public.info
DataName_private.info

go to **My Competitions>My Datasets>Create Dataset.** Fill out the form:



Upload the file **DataName_input_data.zip**. After you upload, you should see your dataset in the data table. The KEY can be used to refer to it from the YAML file, both for public_data and input_data.
input_data: dac49905-dda0-4857-922a-02ca957ec8fd
public_data: dac49905-dda0-4857-922a-02ca957ec8fd



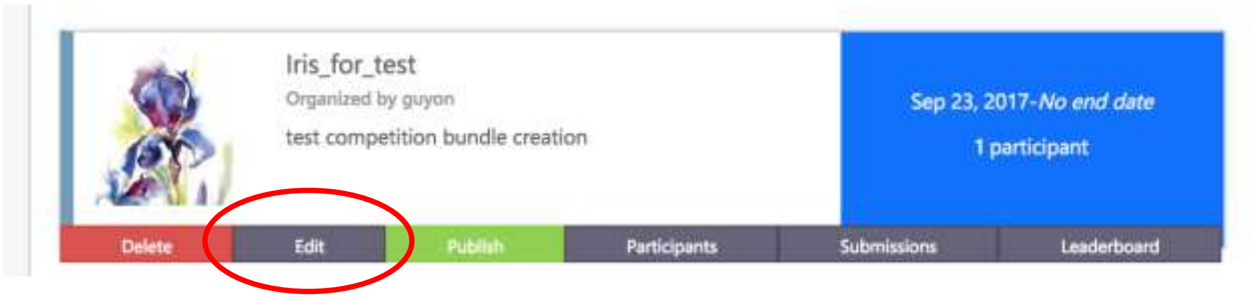
NAME	KEY	DESCRIPTION	TYPE	ACTIONS	MULTI-DELETE? (UNSAFE!)
DataName_input_data	dac49905-dda0-4857-922a-02ca957ec8fd	None	Input Data	Download Delete	☐

Do the same thing for **DataName_reference_data.zip**. IMPORTANT, use the correct data Type **(input_data or reference_data)** in the form when you create the dataset. You can then use the key obtained in the YAML file:

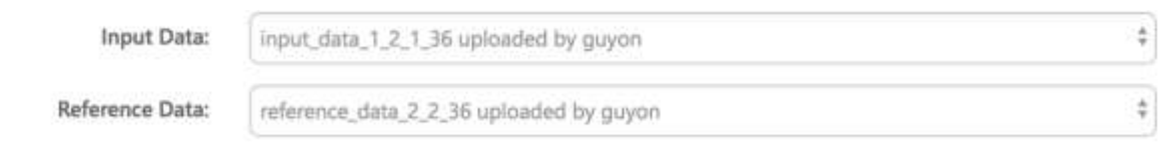
reference_data: 0d8c9c0e-7609-4314-bf1c-3154cdee3462

You need only to specify public_data in phase 1. Use the same input_data and reference_data keys in both phases.

You can also use the editor. In Competitions I'm running, find your competition and click "Edit":



Find the menus for Input Data, Reference Data, and Public Data. Select the right dataset, as don't forget to **SAVE YOUR CHANGES.**

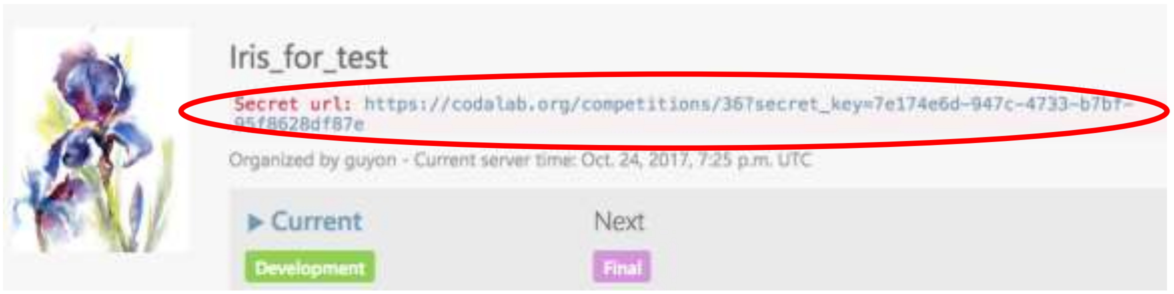


Input Data: input_data_1_2_1_36 uploaded by guyon

Reference Data: reference_data_2_2_36 uploaded by guyon

Editing your challenge on Codalab

Once your challenge is uploaded to Codalab, your whole team can contribute to improving it. Before you make your challenge public (by clicking "Publish") you can share the secret URL with others.

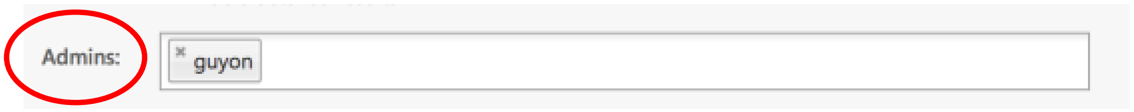


Iris_for_test

Secret url: https://codalab.org/competitions/36?secret_key=7e174e6d-947c-4733-b7bf-95f8628df87e

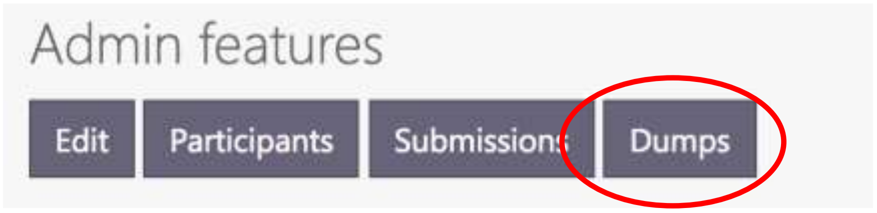
Organized by guyon - Current server time: Oct. 24, 2017, 7:25 p.m. UTC

But this will not allow them to edit the challenge. To give them this privilege, go to the editor and add their Codalab ID to the list of admins:

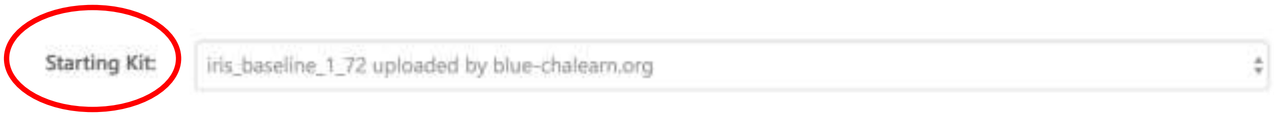


Admins: guyon

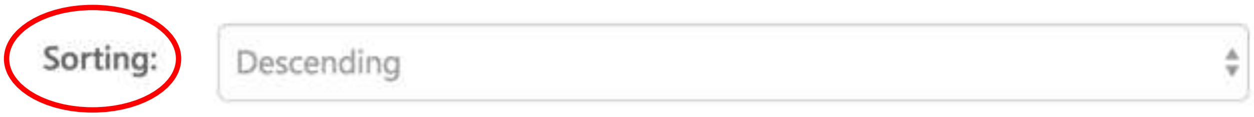
The Codalab editor can be used to edit your challenge. However, try to minimize to avoid winding up with a broken site. Save your intermediate challenge bundles by using the **Dump** button.



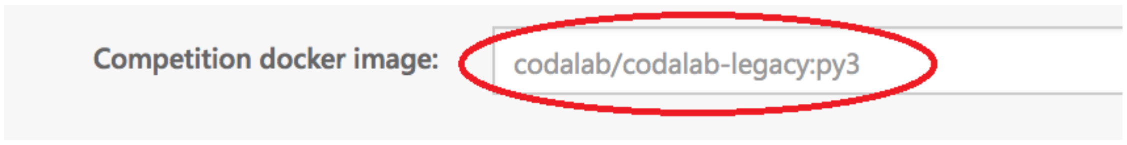
Editing your challenge allows you to update your starting kit. You can upload it in my datasets, then point to it from the editor:


Starting Kit: iris_baseline_1_72 uploaded by blue-chalearn.org

If your score is an “accuracy” (better is higher), you want the leaderboard sorted in descending order, but if it is an “error” (better is lower), you want it in ascending order. You can change that:


Sorting: Descending

We want to run the code into Python 3 dockers. Make sure the correct docker is indicated:


Competition docker image: codalab/codalab-legacy:py3

Using dockers

Codalab allows you to specify a docker in which you code will be running. Thus it is convenient to test your code in this docker ahead of time.

When you submit code to the Codalab platform, your code is executed inside a docker container. This environment can be exactly reproduced on your local machine by downloading the corresponding docker image. The codalab/codalab-legacy:py3 docker, which can be pulled from https://hub.docker.com/r/codalab/codalab-legacy/tags/ contains a large number of pre-loaded programs, including Python 3 and many libraries including scikit-learn. See https://github.com/codalab/codalab-dockers/tree/master/legacy-py3 for details.

To run the starting kit inside the Codalab docker from a terminal:

Step 1) Create a temporary folder to put everything you will need (data and starting kit):

```
mkdir ~/aux
cp starting_kit.zip ~/aux
```

Copy any other data/code you want to that folder, as we will instruct you to map such local folder to be visible when you are within the docker. If you don’t do such mapping procedure, you will not be able to see what you need from within the docker.

Step 2) Run the docker (and map such folder, described above, to /home/aux, inside the docker):

```
docker run -it -v ~/aux:/home/aux codalab/codalab-legacy:py3
```

The first time around it takes a while because docker must download the entire docker image. if everything goes well, you will be inside the docker and the prompt will be like below:

```
root@c74a8b1ccaf7:/#
```

Step 3) Run the starting kit. Inside the docker, go to "/home/aux" and unzip and run the starting kit.

```
# cd /home/aux
# unzip starting_kit.zip
# cd starting_kit
# python3 ingestion_program/ingestion.py sample_data sample_result_submission ingestion_program sample_code_submission
# python3 scoring_program/score.py sample_data sample_result_submission scoring_output
# exit
```

WARNING: inside the docker, python 3.6 is called python3.

Step 4) Run the jupyter notebook.

Same thing as before, but add : **-p 8888:8888**

```
docker run -it -p 8888:8888 -v ~/aux:/home/aux codalab/codalab-legacy:py3
```

Go to a web browser and check that the notebook is running at http://localhost:8888/.

Then open README.pynb. WARNING: the default notebook kernel is Python 2, you'll have to switch to Python 3. If you do not have a Python 3 kernel… you are in trouble. Fixing it is complicated.

Modifying your docker

The docker we recommend contains a lot of libraries, but you may need more for your specific competition. You can then make changes to it following these instructions. But AVOID THIS if you can because you will then have another docker as everyone else, this may confused the L2 students. Also, you will need to test that it works fine on Codalab. If you make a new docker, it must be posted on Docker Hub. Preferable use a docker file to specify your changes and put them under revision control on Github.

How to use Git and Github

There are several good tutorials and cheat sheets on the Internet. You should start by cloning the repo of the starting kit, this is better than downloading the zip:

```
git clone https://github.com/madclam/m2aic2019.git
```

If you want to create another repo, follow these instructions.