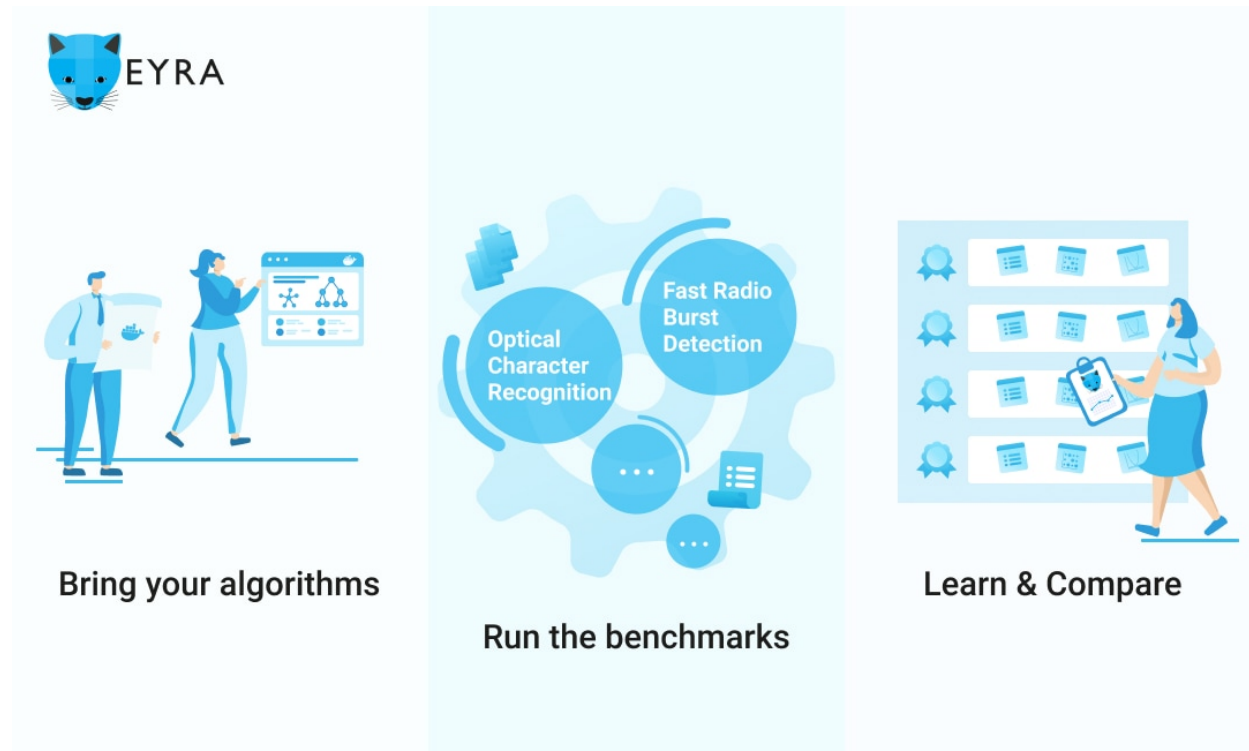

EYRA Tools

Jul 14, 2020

Contents

1	Links	3
1.1	Installation	3
1.2	User Manual	4
1.3	What are all the files?	10
1.4	Scripts	11
1.5	Best Practices	13
1.6	More examples	14
1.7	Demo benchmark: the Iris data	14
1.8	Building these docs	28

EYRA Tools is a Python package that helps you generate Docker containers containing a submission or evaluation for the EYRA Benchmark Platform.



Benchmark participants should implement a submission container, while benchmark organizers should implement an evaluation container. The Docker containers can be generated by typing:

```
eyra-generate [submission|evaluation] <container_name> [-d <docker hub account>]
```


CHAPTER 1

Links

- [EYRA Benchmark Platform](#)
- [EYRA Tools on Github](#)
- [EYRA Tools documentation](#)

Warning: EYRA Tools and the documentation are works in progress. Not everything already works as advertised. If you are interested in organizing a Benchmark on the EYRA Benchmark Platform, please contact us via info@eyrabenchmark.net.

1.1 Installation

1.1.1 Requirements

Make sure you have the following tools installed:

- Docker
- Python 3.5+
- virtualenv/pip

1.1.2 Prepare virtual environment

With virtualenv/pip:

```
git clone git@github.com:EYRA-Benchmark/eyra-tools.git
cd eyra-tools
virtualenv eyra_venv
source eyra_venv/bin/activate
```

(continues on next page)

(continued from previous page)

```
pip install -r requirements.txt
pip install -e .
```

1.2 User Manual

If you want to participate in a benchmark on the EYRA Benchmark Platform, you need to create a model based on participant data, that can predict outcomes (e.g., class labels or numeric values) given test data. What you submit to the benchmark platform is a Docker container that does the predictions, given your model or algorithm. Benchmark organisers need to provide a Docker container for evaluating the results produced by the models/algorithms of participants. The EYRA tools can be used to generate boilerplate containers that set up as much as possible, so you can focus on implementing the prediction algorithm or evaluation metrics.

To be able to use the Docker container as an submission or evaluation, you need to publish it on [Docker Hub](#). If you (or your organization) do(es) not yet have a Docker Hub account, you need to [sign up](#) for one.

1.2.1 Quickstart

1. Generate a boilerplate container by running: `eyra-generate [submission|evaluation] <name>`. A directory called `<name>` is created.
2. Put the input data in the `<name>/data/input/` directory.
 - Benchmark participants should download the public test data and put it in this directory. The file should be called `data/input/test_data`.
 - Benchmark organisers should create a ground truth file as `data/input/ground_truth` and create an example output file (as if created by a participants container), and store it as `data/input/implementation_output`.
3. For submissions, implement the prediction code in `src/submission.py`. For evaluations, implement evaluation metrics in `src/evaluation.py`. You can test your code by running `python src/submission.py` or `python src/evaluation.py` from the `<name>` directory. Output is written to `<name>/data/output`.

Important: Please do not change the file paths in the part of the code after `if __name__ == "__main__":`. Running submissions and evaluations depends on default file names. When running the container locally (see step 7 of this quickstart), these paths are mapped to your local copy of the input and output directories. When running on the benchmark platform, they are mapped to the challenge input and output directories.

4. Add any file you need to the `<name>/src` directory. This can be (Python) code, but also (binary) files containing a model (see [the tutorial](#) for an example). For evaluations, you might not need additional files.
5. Add your code's dependencies to the `<name>/requirements.txt` file.
6. Update the `run()` method of the `Submission` or `Evaluation` object to call a function or functions from `src/submission.py` or `src/evaluation.py`.

Tip: The generated code contains a complete, albeit very simple, example of a submission or evaluation. For a more realistic example have a look at the [demo benchmark tutorial](#).

7. Test your container by running `test.sh`. This will build the container and run the prediction or evaluation code on your local copy of the data. Output is written to `<name>/data/output`.
8. If you are done developing, run the `push.sh` script to tag your Docker container with a version number and push it to [Docker Hub](#).

```
./push.sh [version]
```

If you omit the version number, the Docker image is tagged with `latest`.

9. Specify the Docker container using `<docker hub account>/<name>:<version>` on the EYRA Benchmark Platform.

1.2.2 Complete example

After *installing the EYRA Tools*, and acquiring a [Docker Hub](#) account, generate a boilerplate container by running:

```
eyra-generate [submission|evaluation] <name> [-d <docker hub account>]
```

For submissions, this will create a directory with the following structure:

```
<name>
├── .gitignore
├── Dockerfile
├── README.md
├── build.sh
├── data
│   ├── .gitignore
│   ├── input
│   │   ├── .gitignore
│   │   └── test_data
│   └── output_data_appears_here.txt
├── export.sh
├── push.sh
├── requirements.txt
├── src
│   ├── run_submission.py
│   └── submission.py
└── test.sh
```

For evaluations, the file names are slightly different:

```
<name>
├── .gitignore
├── Dockerfile
├── README.md
├── build.sh
├── data
│   ├── .gitignore
│   ├── input
│   │   ├── .gitignore
│   │   ├── ground_truth
│   │   └── implementation_output
│   └── output_data_appears_here.txt
├── export.sh
├── push.sh
└── requirements.txt
```

(continues on next page)

(continued from previous page)

```
├── src
│   ├── evaluation.py
│   └── run_evaluation.py
└── test.sh
```

For more information on what the files and directories are used for, have a look at [What are all the files?](#)

Tip: It is good practice to use version control when writing code. Now is a good time to do so. If you are using git, run:

```
git init
git add -A
git commit -m "Initial commit"
```

Note: It is also good practice to separate code and data. However, because the boilerplate container contains a bash script for running the code inside the Docker container, it is convenient to have the data in the directory generated by running `eyra-generate` (otherwise you wouldn't be able to run `test.sh` on another computer). To prevent the data from being uploaded to github or another Git repository hosting service, the `data` directory is ignored by git.

Data preparation

The boilerplate code comes with example data. The data files can be found in the `data/input` directory. For submissions, there is a single data file called `test_data` and for evaluations there are two files, `ground_truth` and `implementation_output`. Both submissions and evaluations should produce a single file called `output`, that is written to the `data` directory.

Important: As the EYRA Benchmark Platform uses these default file names, do not change them when working on your own submission or evaluation!

For a submission, you can get the data files from the benchmark page on the EYRA Benchmark Platform. For evaluations, it is the responsibility of the benchmark organizers to prepare the data files.

Implementation

All files related to your algorithm should be put in the `src` directory. Because for development and debugging it is easier to run the code on your computer instead of inside the Docker container, the code is divided over two files: `submission.py` and `run_submission.py` for submissions and `evaluation.py` and `run_evaluation.py` for evaluations.

`<container_type>.py` contains the functionality for running the submission or evaluation and `run-<container_type>.py` contains the code for running the submission or evaluation inside the Docker container. `submission.py` looks like:

```
1 from pathlib import Path
2
3 # Add your imports here; numpy is only used as an example
4 import numpy as np
5
```

(continues on next page)

(continued from previous page)

```

6
7 def my_submission(test_file, out_file):
8     with open(test_file, 'r') as test:
9         data = np.loadtxt(test)
10
11     with open(out_file, 'w') as f:
12         for sample in data:
13             f.write(str(int(sample % 2)))
14             f.write('\n')
15
16
17 if __name__ == "__main__":
18     # Run the algorithm on your local copy of the data by typing:
19     # python src/submission.py
20
21     # These are the default file paths (names) for input and output, so don't
22     # change them.
23     test_file = str(Path('data')/'input'/'test_data')
24     out_file = str(Path('data')/'output')
25
26     my_submission(test_file, out_file)

```

The code contains a single function `my_submission()` that takes two arguments: a path to the input file (i.e., `data/input/test_data`) and a path to the output file (i.e., `data/output`) (line 7). On line 8 and 9, the test data is read and put into a numpy array. Next, the output file is opened for writing (line 11) and we loop over the values in the test data. The example prediction algorithm is very simple: for every value in the test data, we write a zero to the output file if the number is even and a one if the number is odd.

The code can be run by typing `python src/submission.py`. If you do that, everything after line 17 is executed. First, we create file paths for the input and output file (lines 23 and 24). Then `my_submission()` is called.

After running the code, the data directory contains a new file called `output`.

The boilerplate code for evaluations (`evaluation.py`) is very similar:

```

1 import json
2 from pathlib import Path
3
4 # Add your imports here; numpy is only used as an example
5 import numpy as np
6
7
8 def my_evaluation(submission_file, test_gt_file, out_file):
9     with open(submission_file, 'r') as subm:
10         submission_labels = np.loadtxt(subm)
11     with open(test_gt_file, 'r') as gt:
12         gt_labels = np.loadtxt(gt)
13
14     num_correct = 0
15     for subm_label, gt_label in zip(submission_labels, gt_labels):
16         if subm_label == gt_label:
17             num_correct += 1
18
19     output = {'metrics': {'accuracy': float(num_correct)/len(gt_labels)}}
20
21     with open(out_file, 'w') as f:

```

(continues on next page)

(continued from previous page)

```

22         json.dump(output, f)
23
24
25
26 if __name__ == "__main__":
27     # Run the algorithm on your local copy of the data by typing:
28     # python src/evaluation.py
29
30     # These are the default file paths (names) for input and output, so don't
31     # change them.
32     submission_file = str(Path('data')/'input'/'implementation_output')
33     test_gt_file = str(Path('data')/'input'/'ground_truth')
34     out_file = str(Path('data')/'output')
35
36     my_evaluation(submission_file, test_gt_file, out_file)

```

The main function is called `my_evaluation()` and requires three arguments: the submission file (i.e., `data/input/implementation_output`), the ground truth (i.e., `data/input/ground_truth`), and the output file (i.e., `data/output`). On lines 11 and 12 and 13 and 14, the input files are read into numpy arrays. For the evaluation, we are going to count how often the numbers in both arrays are the same. We set a counter to zero (line 16), and loop over the numbers in both arrays simultaneously (line 17). If the numbers are equal (line 18) we add one to the counter (line 19). Next, we prepare the output. The output should be a json file containing a single object (dictionary) with a `metrics` key. The value of `metrics` is an object (dictionary) listing the names of the metrics and their value. In this case, we have a single value called `accuracy` for which we calculate the value by taking the number of samples for which the predicted label was equal to the gold standard and dividing by the total number of samples. On lines 23 and 24, this data is written to `data/output`.

If you are done implementing your submission or evaluation code, it is time to make sure it can be run inside the Docker container. In order to do so, you need to update `run_submission.py` or `run_evaluation.py`.

`run_submission.py` looks like:

```

1  from pathlib import Path
2
3  from submission import my_submission
4
5  class Submission(object):
6      def run(self, test_file, out_file):
7          """This is boilerplate. Delete the contents of this method and put your
8          own code here. Please do not change the class name (Submission),
9          the method name (run), or the arguments.
10         """
11         my_submission(test_file, out_file)
12
13
14 # Please do not change anything below
15 if __name__ == "__main__":
16     # These are the default file paths (names) for input and output
17     test_file = Path('/')/'data'/'input'/'test_data'
18     out_file = Path('/')/'data'/'output'
19
20     Submission().run(test_file, out_file)
21

```

On line 3, we import the `my_submission()` function from `submission.py`, which is called inside the `Submission` object's (line 5) `run()` method (line 11) on line 11. For your submission, change the code to import the function(s) needed to create the output, and call them in the `run()` method.

If you need to split up your code into multiple Python files or if you need additional files for predicting outcomes, put them in the `src` directory, and load them inside the `run()` method (see [Demo benchmark: the Iris data](#) for an example).

Warning: If you change the code below line 14, you run the risk that it will not work on the EYRA Benchmark Platform.

Note: This is the code for submissions. The code for evaluations looks slightly different. For evaluations line 5 is `class Evaluation(object):` and the `run()` method has three arguments (i.e., the submission file, the ground truth file, and the output file) instead of two.

Important: When running the Docker container locally, the input and output directory on your hard drive are mapped to `/data/input/` and `/data/` using docker. When running it on the benchmark platform, these directories are mapped to the benchmark input and output directories.

Output formats

For submissions, the benchmark page should specify the output format. For evaluations, the output should be a json file containing:

```
{
  "metrics": {
    "metric1": <numeric value>
    ...
  }
}
```

Dependencies

All dependencies (like `numpy` (line 5) in the example submission), should be listed in `requirements.txt`, so they are installed inside the container.

Building the Docker container

To build the Docker container, run `build.sh`.

```
$ ./build.sh
Sending build context to Docker daemon 31.74kB
Step 1/7 : FROM python:3.7-slim
--> 783362c5ef81
Step 2/7 : RUN mkdir -p /opt/src /input /output
--> Using cache
--> 5cf2874fe9d2
Step 3/7 : WORKDIR /opt/src
--> Using cache
--> 4876a0b73b86
Step 4/7 : COPY requirements.txt /opt/src/
--> Using cache
```

(continues on next page)

(continued from previous page)

```
---> d397b4c2f203
Step 5/7 : RUN python -m pip install -r requirements.txt
---> Using cache
---> b7815db6c39e
Step 6/7 : ADD src /opt/src/
---> Using cache
---> e6ea2d81fb51
Step 7/7 : ENTRYPOINT "python" "-m" "run_submission"
---> Using cache
---> 013e85d0112d
Successfully built 013e85d0112d
Successfully tagged <name>:latest
```

If you want to build the Docker container and test your code, run `test.sh`.

Have a look at the [scripts page](#) for an overview of the scripts available for manipulating the Docker container.

Tagging and pushing

If you are done developing, run the `push.sh` script to tag your Docker container with a version number and push it to [Docker Hub](#).

```
./push.sh [version]
```

If you omit the version number, the Docker image is tagged with `latest`.

Warning: Please use [semantic versioning](#) rather than the `latest` tag. If you submit `latest` Docker images, the EYRA Benchmark Platform might not use the latest version.

Tip: If you get the following message: `denied: requested access to the resource is denied` `unauthorized: authentication required`. You need to login to Docker Hub using `docker login`.

You can also manually tag and push your image:

```
docker tag <name/id> <docker hub account>/<name>:<version>
docker push <docker hub account>/<name>
```

Submitting

To submit a submission or evaluation to the EYRA Benchmark Platform, put `<docker hub account>/<name>:<version>` in the designated form field.

1.3 What are all the files?

1.3.1 Implementation

- `src/`: directory for storing code your algorithm needs.

- `src/submission.py` or `src/evaluation.py`: Python file for developing the submission or evaluation code. Can be run locally on a copy of the benchmark data.
- `src/run_submission.py` or `src/run_evaluation.py`: Python file containing the code for running a submission or evaluation inside the Docker container.
- `requirements.txt`: file for listing the Python packages your algorithm depends on.

1.3.2 Local data handling

- `data/input/`: local directory used to read the input data from.
- `data/input/test_data`, `data/input/ground_truth`, and/or `data/input/implementation_output`: example data for the boilerplate submission or evaluation. These files need to be replaced with data from the benchmark.
- `data/`: local directory for storing the output of the algorithm.
- `data/output_data_appears_here.txt`: example file to show where the local output is stored (feel free to delete this file).
- `data/.gitignore` and `data/input/.gitignore`: configuration files to make sure the data directories are created on your computer. If you are using git for version control, these files also ensure that data files are not stored in your repository.

1.3.3 Scripts

- `build.sh`: script for building the container (without running the code).
- `test.sh`: script for building the container and running the code.
- `push.sh`: script for submitting the algorithm to the EYRA Benchmark Platform.
- `export.sh`: script for exporting the image of your container to tar file.

Note: `export.sh` is currently not used. Alternative ways of submitting Docker containers will be added to the EYRA Benchmark Platform later.

1.3.4 Miscellaneous

- `Dockerfile`: specifies the Docker container. If your algorithm only uses Python packages, there is no need to change this file.

1.4 Scripts

1.4.1 `test.sh`

If you want to test your container, run `test.sh`. This bash script builds the container and then runs the code.

For the example container, the following output is produced:

```
$ ./test.sh
Sending build context to Docker daemon 78.85kB
Step 1/7 : FROM python:3.7-slim
---> f96c28b7013f
Step 2/7 : RUN mkdir -p /opt/src /input /output
---> Using cache
---> 6bfe8fb274ca
Step 3/7 : WORKDIR /opt/src
---> Using cache
---> 8e0b820ab7b7
Step 4/7 : COPY requirements.txt /opt/src/
---> 02a8f551c536
Step 5/7 : RUN python -m pip install -r requirements.txt
---> Running in 105f5db285be
Collecting numpy==1.16.0 (from -r requirements.txt (line 1))
Downloading https://files.pythonhosted.org/packages/3d/10/
62224c551acfd3a3583ad16d1e0f1c9e9c333e74479dc51977c31836119c/numpy-1.16.0-cp37-
cp37m-manylinux1_x86_64.whl (17.3MB)
Installing collected packages: numpy
Successfully installed numpy-1.16.0
Removing intermediate container 105f5db285be
---> 1a0307108b9a
Step 6/7 : ADD src /opt/src/
---> 2339ef697553
Step 7/7 : ENTRYPOINT "python" "-m" "run_submission"
---> Running in a01619750047
Removing intermediate container a01619750047
---> 49effc50d7a4
Successfully built 49effc50d7a4
Successfully tagged <name>:latest
```

1.4.2 build.sh

If you want to test whether your Docker container can be build, but you don't want to run the code, use the the `build.sh` script.

1.4.3 push.sh

If you are done developing, run the `push.sh` script to tag your Docker container with a version number and push it to [Docker Hub](#).

```
./push.sh [version]
```

If you omit the version number, the Docker image is tagged with `latest`.

Warning: Please use [semantic versioning](#) rather than the `latest` tag. If you submit `latest` Docker images, the EYRA Benchmark Platform might not use the latest version.

Tip: If you get the following message: `denied: requested access to the resource is denied` `unauthorized: authentication required`. You need to login to Docker Hub using `docker login`.

You can also manually tag and push your image:

```
docker tag <name/id> <docker hub account>/<name>:<version>
docker push <docker hub account>/<name>
```

Todo: Specify how benchmark organizers can submit their evaluation algorithms.

1.4.4 `export.sh`

You can export the Docker image to a tar file by running the `export.sh` script.

Note: `export.sh` is currently not used. Alternative ways of submitting Docker containers will be added to the EYRA Benchmark Platform later.

1.5 Best Practices

Some best practices for creating submissions and evaluation containers.

1.5.1 Put your code under version control

After you generated a submission or evaluation project, put it under version control! If you are using `git`, you can so by typing:

```
git init
git add -A
git commit -m "Initial commit"
```

And put it on [GitHub](#).

1.5.2 Put the complicated parts of your algorithm in a separate Python package

Although all code and other files that are put inside the `src` directory are copied to the container, once your code gets complicated, it might be a better idea to create a separate Python package for your code, install it inside the container and call the required functionality from `src/run_submission.py` or `run_evaluation.py`.

Have a look at the [section on building and packaging Python code](#) in the Netherlands eScience Center Software Development guide to get pointers on how to create a Python package.

1.5.3 Use semantic versioning for your Docker tags

Using [semantic versioned](#) Docker tags helps you and others to keep track of what you did. If you use the `latest` tag for your submission or evaluation, the EYRA Benchmark Platform might fail to use the latest version.

Start with version 0.1.0.

1.5.4 Add an Open Source license to your code

Having a license on your code allows others to inspect and (re-)use your code. This is essential to reproducibility, peer-review, and the ability to build upon others' work. Do you need more information about software licenses and/or advice on what license to choose? Have a look at these blog posts:

- [A license to science](#) by Lourens Veen
- [A Data Scientist's Guide to Open Source Licensing](#) by mattymecks

1.6 More examples

Todo: Replace with examples that used EYRA Tools. We need to have both submissions and evaluations.

1.6.1 Heimdall

Heimdall is an algorithm for Fast Radioburst Detection (FRD).

- Submission Docker container: [frb-heimdall](#)

1.6.2 Amber

Amber is an algorithm for Fast Radioburst Detection (FRD).

- Submission Docker container: [frb-amber](#)

1.6.3 frb-evaluation

Tools for analyzing the output of the FRB search software challenge under EYRA.

- Evaluation Docker container: [frb-evaluation](#)

Do you want us to include your submission or evaluation container? Send an email to info@eyrabenchmark.net.

1.7 Demo benchmark: the Iris data

This is a demo benchmark set up to show potential benchmark organizers what needs to be done to set up a benchmark on the EYRA Benchmark Platform and to allow potential benchmark participants to try submitting an algorithm. All data and code used for this demo can be found [on github](#).

The figure above shows how things work on the EYRA Benchmark Platform. It shows the data sets involved (participant data, which can or cannot be ground truth data, public test data, and optionally private test data). The figure also shows which parts of the benchmark are done by the platform and the parts that should be done by benchmark organizers and participants.

Benchmark organizers have to prepare different data sets, and make (some of them) make available (i.e., participant data and public test data). The EYRA Benchmark Platform is used for distributing the data, but data preparation should be done by the benchmark organizers themselves. To give a complete overview of what organizing a benchmark entails, the demo also explains how the different data sets were made.

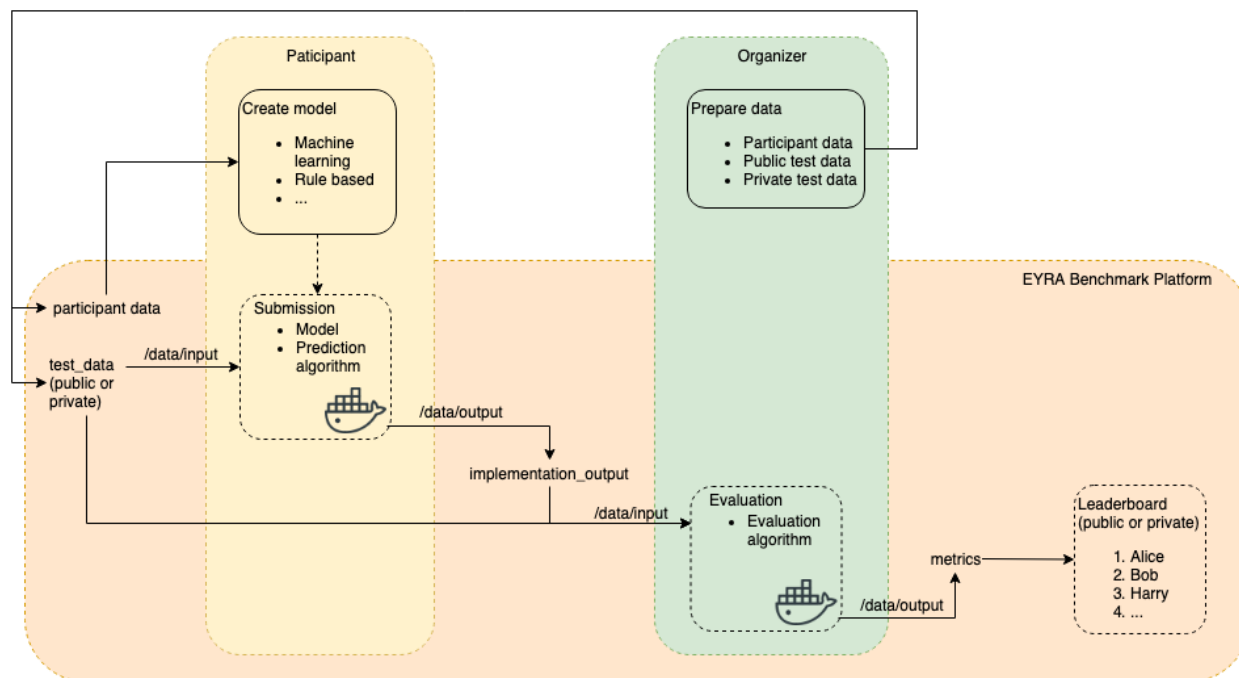


Fig. 1: Overview of the EYRA Benchmark Platform.

Benchmark participants use the data provided by the benchmark organizers to create a model. As shown in the figure, model creation is the responsibility of benchmark participants, and is not done on the platform. Because we need a model to show how you can submit to the Benchmark Platform, the demo specifies the model creation phase. For the demo, a machine learning model is created, but it is possible to create other types of models (WOULD BE NICE TO INSERT A REFERENCE HERE WITH EXAMPLES). The demo model creation code is in the `model_creation` directory. Usually, benchmarks have available public test data that can be used to test the performance of the algorithm.

The demo benchmark uses the Iris dataset. The Iris dataset is a multivariate dataset containing measurements of the petals and sepals of different types of irises. The task is to predict the species of Iris (Setosa, Virginica, or Versicolor) based on the length and width of petals and sepals. The data set was randomly divided into three sets containing 50 samples each; one set is used as participant data, one set as public test data and one set as private test data. See under **‘For organizers: creating a benchmark’** for more details about the creation of the datasets.

1.7.1 For participants: create a model

Get the data

Download the participant data and public test data from the [demo benchmark](#).

- Go to the [benchmark page](#).
- Click Data.
- Read the data description.
- Download the files.

Create a model

For the demo, we are keeping things simple and train Support Vector Machines using scikit-learn. The code can be found in `model_creation/train.py` and looks like:

```
1 import os
2
3 import pandas as pd
4
5 from pathlib import Path
6 from joblib import dump
7
8 from sklearn import svm
9
10
11 def train_svm(in_file):
12     """Train Support Vector Machines for the EYRA Demo Benchmark.
13     """
14
15     # Read the training file
16     train = pd.read_csv(in_file)
17
18     train_data = train[['sepal_length', 'sepal_width', 'petal_length',
19                        'petal_width']].values
20     train_targets = list(train['class'])
21
22     # Train the classifier
23     clf = svm.SVC(gamma=0.001, C=100.)
24     clf.fit(train_data, train_targets)
25
26     return clf
27
28
29 def save_model(clf, path):
30     dump(clf, path)
31
32
33 def predict(clf, test_data_file):
34     test = pd.read_csv(test_data_file)
35
36     return clf.predict(test.values)
37
38 if __name__ == "__main__":
39     # Create the model using a local copy of the data by typing:
40     # python model_creation/train.py
41     root = Path(__file__).absolute().parent.parent
42
43     participant_data = Path(root)/'data'/'iris_participant_data.csv'
44     out_file = Path(root)/'model_creation'/'iris_svm_model'
45
46     clf = train_svm(str(participant_data))
47     save_model(clf, str(out_file))
48
49     # Test whether the prediction mechanics work
50     test_file = root/'data'/'iris_public_test_data.csv'
51     result = predict(clf, str(test_file))
52     print(result)
```

In line 1-8 we import the Python libraries we need. The model creation code consist of two functions. Generally speaking, it is a good idea to divide your code into multiple functions, e.g., one for data preprocessing, one for training, and one for making predictions. The function for training the model starts on line 11. It has as input parameter the participant data file. The function reads the participant data (line 16) and puts it into the format that is required by machine learning algorithms in scikit-learn (lines 18-20). Next, a Support Vector classifier is trained (lines 23-24) and returned (line 26).

The `save_model()` function on line 29 takes as input a classifier and path and then saves the model to that path, so it can be saved in the submission docker container and used for predicting iris labels.

The `predict()` function on line 33 can be used to test whether predicting labels given a test file works as expected. It has two input parameters, a trained classifier and the path to a test data file. The function reads the test data (line 34) and returns the predictions (line 36).

Note: When creating a model, feel free to use more functions and Python files if needed. For complicated code and if you want to be able to re-use for example data preprocessing functions, it might be a good idea to create your own Python package.

Todo: Add link to good tutorial about creating installable packages.

During implementation, you can test your model by running `python train.py` in the `model_creation` directory. Line 38 makes sure the code below runs only if you type that command. First we do some administrative work to determine where the input data can be found and output should be written (lines 41-44). Next, we train the classifier (line 46) and save the model to a file (line 47).

To test whether the model can be used for prediction, a variable containing the path to a test data file is created (line 50). Next, the `predict()` function is called and the result of that is printed to the screen. The output produced by running the train script is:

```
$ python model_creation/train.py
['Iris-versicolor' 'Iris-virginica' 'Iris-virginica' 'Iris-virginica'
'Iris-setosa' 'Iris-virginica' 'Iris-virginica' 'Iris-setosa'
'Iris-setosa' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor'
'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'
'Iris-virginica' 'Iris-setosa' 'Iris-versicolor' 'Iris-versicolor'
'Iris-setosa' 'Iris-versicolor' 'Iris-setosa' 'Iris-versicolor'
'Iris-virginica' 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica'
'Iris-virginica' 'Iris-setosa' 'Iris-virginica' 'Iris-virginica'
'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-virginica'
'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor'
'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa'
'Iris-setosa']
```

This result tells us that the prediction mechanics work, but it doesn't say anything about performance. If you would like to estimate performance of your model using participant data only, have a look at [cross-validation](#).

1.7.2 For participants: create a submission

Every benchmark on the EYRA Benchmark Platform has a leaderboard that shows the performance of models according to the metric(s) proposed by the benchmark organizers. To have the benchmark platform calculate these metrics for our model, we need to create a submission Docker container that contains a prediction algorithm.

Prepare the Docker container and data

A boilerplate submission container can be generated using the [EYRA Tools](#). Run `eyra-generate submission iris_svm` (after *installing the EYRA Tools*), `cd` into the directory that is generated (`iris_svm`), and copy the data into the boilerplate container:

```
eyra-generate submission iris_svm
cd iris_svm
cp ~/Downloads/iris_public_test_data.csv data/input/test_data
mkdir src/model
cp ~/code/eyra-iris-demo/model_creation/iris_svm_model src/model/iris_svm_model
```

Please note that the file paths from which files are copied need to be changed to the correct paths on your computer.

Todo: Resolve file naming issues (the submission container expects input files with specific names and should produce output files with specific names. These names are different from what the files are called now).

Implement the prediction algorithm

Implement the prediction algorithm in `src/submission.py`. For the iris SVM this file looks like:

```
1 import pandas as pd
2
3 from pathlib import Path
4 from joblib import load
5
6 from sklearn import svm
7
8
9 def iris_svm_predict(model_file, test_file, out_file):
10     # Load classifier
11     clf = load(model_file)
12
13     # Predict
14     pred = predict(clf, test_file)
15
16     # Write the output to file
17     output = pd.DataFrame()
18     output['class'] = pred
19     output.to_csv(out_file, index=None)
20
21
22 def predict(clf, test_data_file):
23     test = pd.read_csv(test_data_file)
24
25     return clf.predict(test.values)
26
27
28 if __name__ == "__main__":
29     # Run the algorithm on your local copy of the data by typing:
30     # python algorithm_scr/algorithm.py
31
32     model_file = Path(__file__).absolute().parent/'model'/'iris_svm_model'
33
34     # These are the default file paths (names) for input and output, so don't
```

(continues on next page)

(continued from previous page)

```

35     # change them.
36     test_file = Path('data')/'input'/'test_data'
37     out_file = Path('data')/'output'
38
39     iris_svm_predict(model_file, test_file, out_file)

```

The prediction algorithm has two functions, `iris_svm_predict()` (line 9) and `predict()` (line 22), which is the same as the `predict()` function we used during model creation. The `iris_svm_predict()` function first loads a model from file (line 11). It then predicts class labels given the model and the test data file (line 14). On line 17-19 a pandas DataFrame containing a single column called 'class' is created and written to a csv file. Starting from line 28, we see how these functions are called to generate the implementation output used to calculate performance. Line 32 defines a variable containing the path to the model file. Lines 36 and 37 define variables that contain the paths to the test data and output respectively. Finally, on line 39, the `iris_svm_predict()` function is called using the paths as arguments.

You can test the prediction algorithm by running `python src/submission.py` from the `iris_svm` directory. If everything works as expected, we can start with preparing the Docker container.

Specify the dependencies

Add all Python libraries needed to run the prediction algorithm to `requirements.txt`. For the demo we add `pandas` and `scikit-learn`.

Make sure the code can run inside the Docker container

Update `src/run_submission.py`, so it calls the `iris_svm_predict()` function:

```

1  import os
2  from pathlib import Path
3
4  from submission import iris_svm_predict
5
6
7  class Submission(object):
8      def run(self, test_file, out_file):
9          # Additional data required by the prediction algorithm
10         model_file = Path(__file__).absolute().parent/'model'/'iris_svm_model'
11
12         iris_svm_predict(model_file, test_file, out_file)
13
14
15     # Please do not change anything below
16     if __name__ == "__main__":
17         # These are the default file paths (names) for input and output
18         test_file = Path('/')/'data'/'input'/'test_data'
19         out_file = Path('/')/'data'/'output'
20
21         Submission().run(test_file, out_file)

```

Because the code for the prediction algorithm is already done, we only need to make a few changes to the boilerplate code. Lines 1-4 contain the imports. On line 4 we import the `iris_svm_predict()` function from `src/submission.py`. On line 5, a class `Submission` is defined that has a single method `run()`. On the EYRA Benchmark Platform, this function is executed, with specific input arguments for `test_file` and `out_file`, as specified in lines 16-21. Lines 16-21 should not be changed. It specifies the paths to which the input (`test_file` on

line 18) and the output (out_file on line 19) are mapped inside the Docker container. On line 21, a Submission object is created and its run() method is called with the correct arguments. To make it work with our particular prediction algorithm, we need to update the run() method. On line 9, we specify the path to the model file. We copied this file into the src/model directory in a previous step. Everything that is inside the src directory is copied to the Docker container. So that is where you can put data and code required by the prediction algorithm. On line 12, we call iris_svm_predict() as before.

Test the Docker container

To build the Docker container and run the algorithm inside it, type ./test.sh. You get the following output:

```
$ ./test.sh
Sending build context to Docker daemon   38.4kB
Step 1/7 : FROM python:3.7-slim
3.7-slim: Pulling from library/python
1ab2bdf9778: Pull complete
b5d689d9c40c: Pull complete
5b13ee99f0ea: Pull complete
d617973d7fa5: Pull complete
abfef9fe6f0b: Pull complete
Digest: sha256:5f83c6d40f9e9696d965785991e9b85e4baef189c7ad1078483d15a8657d6cc0
Status: Downloaded newer image for python:3.7-slim
--> f96c28b7013f
Step 2/7 : RUN mkdir -p /opt/src /input /output
--> Running in 0e4aelaa1317
Removing intermediate container 0e4aelaa1317
--> 20dd0c44733e
Step 3/7 : WORKDIR /opt/src
--> Running in ff5c0b8dbbb3
Removing intermediate container ff5c0b8dbbb3
--> ea18e90edb31
Step 4/7 : COPY requirements.txt /opt/src/
--> 7e097b17cd59
Step 5/7 : RUN python -m pip install -r requirements.txt
--> Running in 700020ca752d
Collecting pandas (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/7e/ab/
  ↪ea76361f9d3e732e114adcd801d2820d5319c23d0ac5482fa3b412db217e/pandas-0.25.1-cp37-
  ↪cp37m-manylinux1_x86_64.whl (10.4MB)
Collecting scikit-learn (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/9f/c5/
  ↪e5267eb84994e9a92a2c6a6ee768514f255d036f3c8378acfa694e9f2c99/scikit_learn-0.21.3-
  ↪cp37-cp37m-manylinux1_x86_64.whl (6.7MB)
Collecting numpy>=1.13.3 (from pandas->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/25/eb/
  ↪4ecf6b13897391cb07a4231e9d9c671b55dfbbf6f4a514a1a0c594f2d8d9/numpy-1.17.1-cp37-
  ↪cp37m-manylinux1_x86_64.whl (20.3MB)
Collecting python-dateutil>=2.6.1 (from pandas->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/41/17/
  ↪c62facbfbfd163c7f57f3844689e3a78baelf403648a6afb1d0866d87fbb/python_dateutil-2.8.0-
  ↪py2.py3-none-any.whl (226kB)
Collecting pytz>=2017.2 (from pandas->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/87/76/
  ↪46d697698a143e05f77bec5a526bf4e56a0be61d63425b68f4ba553b51f2/pytz-2019.2-py2.py3-
  ↪none-any.whl (508kB)
Collecting scipy>=0.17.0 (from scikit-learn->-r requirements.txt (line 2))
```

(continues on next page)

(continued from previous page)

```

    Downloading https://files.pythonhosted.org/packages/94/7f/
    ↳b535ec711cbcc3246abead4385d17e1b325d4c3404dd86f15fc4f3dba1dbb/scipy-1.3.1-cp37-cp37m-
    ↳manylinux1_x86_64.whl (25.2MB)
Collecting joblib>=0.11 (from scikit-learn->-r requirements.txt (line 2))
    Downloading https://files.pythonhosted.org/packages/cd/c1/
    ↳50a758e8247561e58cb87305b1e90b171b8c767b15b12a1734001f41d356/joblib-0.13.2-py2.py3-
    ↳none-any.whl (278kB)
Collecting six>=1.5 (from python-dateutil>=2.6.1->pandas->-r requirements.txt (line
    ↳1))
    Downloading https://files.pythonhosted.org/packages/73/fb/
    ↳00a976f728d0d1fecfe898238ce23f502a721c0ac0ecfedb80e0d88c64e9/six-1.12.0-py2.py3-
    ↳none-any.whl
Installing collected packages: numpy, six, python-dateutil, pytz, pandas, scipy,
    ↳joblib, scikit-learn
Successfully installed joblib-0.13.2 numpy-1.17.1 pandas-0.25.1 python-dateutil-2.8.0
    ↳pytz-2019.2 scikit-learn-0.21.3 scipy-1.3.1 six-1.12.0
Removing intermediate container 700020ca752d
---> 624723a5788f
Step 6/7 : ADD src /opt/src/
---> a46bb5006326
Step 7/7 : ENTRYPOINT "python" "-m" "run_submission"
---> Running in 12b22dbb283f
Removing intermediate container 12b22dbb283f
---> e119001e4684
Successfully built e119001e4684
Successfully tagged iris_svm:latest

```

Push the container to Docker Hub

If everything works as expected, you can push your container to Docker Hub:

```
./push.sh [version]
```

The output looks like:

```

$ ./push.sh 0.1.0
Sending build context to Docker daemon   38.4kB
Step 1/7 : FROM python:3.7-slim
---> f96c28b7013f
Step 2/7 : RUN mkdir -p /opt/src /input /output
---> Using cache
---> 20dd0c44733e
Step 3/7 : WORKDIR /opt/src
---> Using cache
---> ea18e90edb31
Step 4/7 : COPY requirements.txt /opt/src/
---> Using cache
---> 7e097b17cd59
Step 5/7 : RUN python -m pip install -r requirements.txt
---> Using cache
---> 624723a5788f
Step 6/7 : ADD src /opt/src/
---> Using cache
---> a46bb5006326
Step 7/7 : ENTRYPOINT "python" "-m" "run_submission"

```

(continues on next page)

(continued from previous page)

```
---> Using cache
---> e119001e4684
Successfully built e119001e4684
Successfully tagged iris_svm:latest
Using tag: 0.1.0.
The push refers to repository [docker.io/eyra/iris_svm]
94c7819de05e: Pushed
3e64f1dd2a39: Pushed
764a69f7b78b: Pushed
56edc5355daa: Pushed
1808163506f4: Mounted from library/python
acd5d8dd0a11: Layer already exists
e18984c86e86: Layer already exists
523a99e3c88d: Layer already exists
1c95c77433e8: Layer already exists
0.1.0: digest: 2205
sha256:6f128fa8e62abecd98a36cea12a69071f4569b6cea031b84649f273afaa01fd3 size: 2205
```

Tip: You need to login to Docker Hub (`docker login`) before you can push the container.

Tip: If you want to use the `push.sh` script in the [EYRA Iris Demo github repository](#), you need to replace `eyra` with your own Docker Hub account.


Submit the Docker container to the benchmark

On the EYRA Benchmark Platform, log in and go to the [EYRA Iris Demo page](#) and click ‘create submission’. Now you see a form:

Algorithm	<div>Select an algorithm</div>	or <input type="checkbox"/> Create New Algorithm
Name	<div>(e.g. Amber v3)</div>	
Version	<div>(e.g. 3)</div>	
Docker image name	<div>(e.g. eyra/amber:3)</div>	
<div>SUBMIT</div>		

First you need to specify the algorithm you are submitting. If your algorithm isn’t in the dropdown list, you can create a new one by checking the ‘Create New Algorithm’ box. The algorithm name should refer to the model you created. For this demo we choose ‘Iris SVM’. Under ‘Name’ we put ‘Iris SVM 0.1.0’, ‘Version’ is ‘0.1.0’, and as ‘Docker image name’ we write `eyra/iris_svm:0.1.0`.

After clicking ‘Submit’ a job that will run the submission is created. This can take a little while. The submission is added to the ‘Results’ tab of the benchmark page. If the submission job is finished, you can see the performance of the algorithm:

Name	Version	F1	Recall	Precision	Visualization	Date	Logs
Iris SVM 0.1.0	-	0.96	0.96	0.96	-	Aug 28, 2019, 1:09:35 PM	

1.7.3 For organizers: create a benchmark

To set up a benchmark on the EYRA Benchmark Platform you should [contact the EYRA Benchmark team](#), and we will get back to you with more specific instructions. In addition to thorough and concise descriptions of the task, the data, the ground truth, and the evaluation metrics ([see the benchmarks on the platform for examples](#)), organizers need to provide:

- Participant data
- Public test data + public test ground truth: this is the data used for creating the public leaderboard.
- Private test data + private test ground truth: this is hold-out data, that is used for creating the private leaderboard.
- A Docker container containing the evaluation algorithm. Have a look at the tutorial to see how this is done.

Note: For demonstration purposes, the demo benchmark's data is available on [github](#). For a real benchmark, the public test ground truth, and the private test data and ground truth should not be shared with participants. This helps to [make sure participants' submissions focus on understanding the problem and advancing science rather than incrementally improving metrics](#).

A [notebook](#) specifying how the demo datasets were created is available in the [demo benchmark repository on github](#).

1.7.4 For organizers: create an evaluation

Benchmark organizers must submit a Docker container that compares the output of a submission to gold standard data and calculates performance.

Prepare the Docker container and data

We start with generating a boilerplate evaluation Docker container by running:

```
eyra-generate evaluation iris_eval -d <docker hub account>
```

Next, we copy data to the `data/input` directory:

```
cd iris_eval
cp ~/code/eyra-iris-demo/iris_public_test_gt.csv data/input/ground_truth
cp ~/code/eyra-iris-demo/iris_public_test_gt.csv data/input/implementation_output
```

Note: For convenience we are using the public test ground truth as implementation output. For real benchmarks it might be a good idea to (manually) create an implementation output file that is not the same as the ground truth.

Please note that the file paths from which files are copied need to be changed to the correct paths on your computer.

Todo: Resolve file naming issues (the submission container expects input files with specific names and should produce output files with specific names. These names are different from what the files are called now).

Implement the evaluation algorithm

Implement the evaluation algorithm in `src/valuation.py`. For the demo benchmark this file looks like:

```

1  import json
2  from pathlib import Path
3
4  import pandas as pd
5
6  from sklearn import metrics
7
8
9  def evaluate_iris(submission_file, test_gt_file, out_file):
10     df = pd.read_csv(test_gt_file)
11     test_gt = df['class'].to_list()
12
13     df = pd.read_csv(submission_file)
14     pred = df['class'].to_list()
15
16     prec = metrics.precision_score(test_gt, pred, average='weighted')
17     recall = metrics.recall_score(test_gt, pred, average='weighted')
18     f = metrics.f1_score(test_gt, pred, average='weighted')
19
20     out = {'metrics': {'Precision': prec,
21                       'Recall': recall,
22                       'F1': f}}
23     with open(out_file, 'w') as f:
24         json.dump(out, f)
25
26
27  if __name__ == "__main__":
28     # Run the algorithm on your local copy of the data by typing:
29     # python src/evaluation.py
30
31     # These are the default file paths (names) for input and output, so don't
32     # change them.
33     submission_file = str(Path('data')/'input'/'implementation_output')
34     test_gt_file = str(Path('data')/'input'/'ground_truth')
35     out_file = str(Path('data')/'output')
36
37     evaluate_iris(submission_file, test_gt_file, out_file)

```

As usual, we first import the required Python libraries (line 1-6). On line 9 the main evaluation function (`evaluate_iris()`) is defined. Inside this function, the ground truth data is read (line 10) and converted to a list of class labels (line 11). Next, the same is done for the implementation output (lines 13 and 14). For this evaluation, we calculate precision, recall and F1-measure using `sklearn` (lines 16-18). The metrics are put inside the data structure required by EYRA Benchmark leaderboards:

```

{
  'metrics': {
    'Precision': 1.0,

```

(continues on next page)

(continued from previous page)

```

    'Recall': 1.0,
    'F1': 1.0
}
}

```

And written to a json file (lines 23 and 24).

When the `src/evaluation.py` script is run, we define paths to the input and output data (lines 33-35). Finally, the `evaluate_iris()` function is called.

You can check that everything works as expected by inspecting the output file:

```

$ cat data/output
{"metrics": {"Precision": 1.0, "Recall": 1.0, "F1": 1.0}}

```

Specify the dependencies

Add all Python libraries needed to run the evaluation algorithm to `requirements.txt`. For the demo we add `pandas` and `scikit-learn`.

Make sure the evaluation runs inside the Docker container

To actually evaluate performance of submissions, we need to run the evaluation code inside a Docker container. In order to be able to do this, we change `src/run_evaluation.py` to:

```

1  from pathlib import Path
2
3  from evaluation import evaluate_iris
4
5
6  class Evaluation(object):
7      def run(self, submission_file, test_gt_file, out_file):
8          """This is boilerplate. Delete the contents of this method and put your
9             own code here. Please do not change the class name (Evaluation),
10             the method name (run), or the arguments.
11             """
12             evaluate_iris(submission_file, test_gt_file, out_file)
13
14
15  # Please do not change anything below
16  if __name__ == "__main__":
17      # These are the default file paths (names) for input and output
18      submission_file = Path('/').joinpath('data', 'input', 'implementation_output')
19      test_gt_file = Path('/').joinpath('data', 'input', 'ground_truth')
20      out_file = Path('/').joinpath('data', 'output')
21
22      Evaluation().run(submission_file, test_gt_file, out_file)

```

The only lines we have to change in the boilerplate code are line 3 and line 12. On line 3 we import the `evaluate_iris()` function from `src/evaluation.py`. And on line 12, we call this function with the predefined arguments.

Note: For a real benchmark it is a good idea to replace the boilerplate documentation on lines 8-11 with something

that helps users understand what is going on.

Test the Docker container

To build the Docker container and run the algorithm inside it, type `./test.sh`. You get the following output:

```
$ ./test.sh
Sending build context to Docker daemon 37.38kB
Step 1/7 : FROM python:3.7-slim
3.7-slim: Pulling from library/python
1ab2bdfe9778: Pull complete
b5d689d9c40c: Pull complete
5b13ee99f0ea: Pull complete
d617973d7fa5: Pull complete
abfef9fe6f0b: Pull complete
Digest: sha256:5f83c6d40f9e9696d965785991e9b85e4baef189c7ad1078483d15a8657d6cc0
Status: Downloaded newer image for python:3.7-slim
--> f96c28b7013f
Step 2/7 : RUN mkdir -p /opt/src /input /output
--> Running in 162d645f6ab4
Removing intermediate container 162d645f6ab4
--> 6bfe8fb274ca
Step 3/7 : WORKDIR /opt/src
--> Running in 728e00c886c8
Removing intermediate container 728e00c886c8
--> 8e0b820ab7b7
Step 4/7 : COPY requirements.txt /opt/src/
--> 86c6dd868d39
Step 5/7 : RUN python -m pip install -r requirements.txt
--> Running in 5b6aa5bc84c6
Collecting pandas (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/7e/ab/
  ↪ea76361f9d3e732e114adcd801d2820d5319c23d0ac5482fa3b412db217e/pandas-0.25.1-cp37-
  ↪cp37m-manylinux1_x86_64.whl (10.4MB)
Collecting scikit-learn (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/9f/c5/
  ↪e5267eb84994e9a92a2c6a6ee768514f255d036f3c8378acfa694e9f2c99/scikit_learn-0.21.3-
  ↪cp37-cp37m-manylinux1_x86_64.whl (6.7MB)
Collecting python-dateutil>=2.6.1 (from pandas->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/41/17/
  ↪c62facbfbdb163c7f57f3844689e3a78baelf403648a6afb1d0866d87fbb/python_dateutil-2.8.0-
  ↪py2.py3-none-any.whl (226kB)
Collecting numpy>=1.13.3 (from pandas->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/25/eb/
  ↪4ecf6b13897391cb07a4231e9d9c671b55dfbbf6f4a514a1a0c594f2d8d9/numpy-1.17.1-cp37-
  ↪cp37m-manylinux1_x86_64.whl (20.3MB)
Collecting pytz>=2017.2 (from pandas->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/87/76/
  ↪46d697698a143e05f77bec5a526bf4e56a0be61d63425b68f4ba553b51f2/pytz-2019.2-py2.py3-
  ↪none-any.whl (508kB)
Collecting joblib>=0.11 (from scikit-learn->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/cd/c1/
  ↪50a758e8247561e58cb87305b1e90b171b8c767b15b12a1734001f41d356/joblib-0.13.2-py2.py3-
  ↪none-any.whl (278kB)
Collecting scipy>=0.17.0 (from scikit-learn->-r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/94/7f/
  ↪b535ec711cbcc3246abea4385d17e1b325d4c3404dd86f15fc4f3dba1dbb/scipy-1.3.1-cp37-cp37m-
  ↪manylinux1_x86_64.whl (25.2MB)
```

(continued from previous page)

```

Collecting six>=1.5 (from python-dateutil>=2.6.1->pandas->-r requirements.txt (line
↳1))
  Downloading https://files.pythonhosted.org/packages/73/fb/
↳00a976f728d0dlfecfe898238ce23f502a721c0ac0ecfedb80e0d88c64e9/six-1.12.0-py2.py3-
↳none-any.whl
Installing collected packages: six, python-dateutil, numpy, pytz, pandas, joblib,
↳scipy, scikit-learn
Successfully installed joblib-0.13.2 numpy-1.17.1 pandas-0.25.1 python-dateutil-2.8.0
↳pytz-2019.2 scikit-learn-0.21.3 scipy-1.3.1 six-1.12.0
Removing intermediate container 5b6aa5bc84c6
---> 99373d908df0
Step 6/7 : ADD src /opt/src/
---> d842ba9ce114
Step 7/7 : ENTRYPOINT "python" "-m" "run_evaluation"
---> Running in 54d46f982d81
Removing intermediate container 54d46f982d81
---> 4e928f9dc2fd
Successfully built 4e928f9dc2fd
Successfully tagged iris_eval:latest

```

Push the container to Docker Hub

If everything works as expected, you can push your container to Docker Hub:

```
./push.sh [version]
```

The output looks like:

```

$ ./push.sh 0.1.0
Sending build context to Docker daemon 37.38kB
Step 1/7 : FROM python:3.7-slim
---> f96c28b7013f
Step 2/7 : RUN mkdir -p /opt/src /input /output
---> Using cache
---> 6bfe8fb274ca
Step 3/7 : WORKDIR /opt/src
---> Using cache
---> 8e0b820ab7b7
Step 4/7 : COPY requirements.txt /opt/src/
---> Using cache
---> 86c6dd868d39
Step 5/7 : RUN python -m pip install -r requirements.txt
---> Using cache
---> 99373d908df0
Step 6/7 : ADD src /opt/src/
---> Using cache
---> d842ba9ce114
Step 7/7 : ENTRYPOINT "python" "-m" "run_evaluation"
---> Using cache
---> 4e928f9dc2fd
Successfully built 4e928f9dc2fd
Successfully tagged iris_eval:latest
Using tag: 0.1.0.
The push refers to repository [docker.io/eyra/iris_eval]
2a0fb79103de: Pushed

```

(continues on next page)

(continued from previous page)

```
f25642385f8b: Pushed
27f89623cc9c: Pushed
21bb16c9b065: Pushed
1808163506f4: Mounted from eyra/iris_svm
acd5d8dd0a11: Mounted from eyra/iris_svm
e18984c86e86: Mounted from eyra/iris_svm
523a99e3c88d: Mounted from eyra/iris_svm
1c95c77433e8: Mounted from eyra/iris_svm
0.1.0: digest:
↪sha256:492fbb2791b0613817150521b2e2597e6ca02ac07d0157f5f253765568f63a91 size: 2204
```

Tip: You need to login to Docker Hub (`docker login`) before you can push the container.

Tip: If you want to use the `push.sh` script in the [EYRA Iris Demo github repository](#), you need to replace `eyra` with your own Docker Hub account.

Set the benchmark evaluation container

Set the benchmark's evaluation container to `eyra/iris_eval:0.1.0`.

Todo: There is no form yet for benchmark organizers. The tutorial should be updated once it is clear how benchmark organizers should specify their evaluation container.

1.8 Building these docs

Install the EYRA Tools package.

Run `make html` from the `docs/` folder. This will generate an html version of the documentation in `docs/_build/html`.

```
# Build docs to docs/_build/html
cd docs
make html
```

The EYRA Iris Demo scripts have to be copied to `docs/code/demo` by hand. You only have to do this if scripts from [the demo](#) are updated.