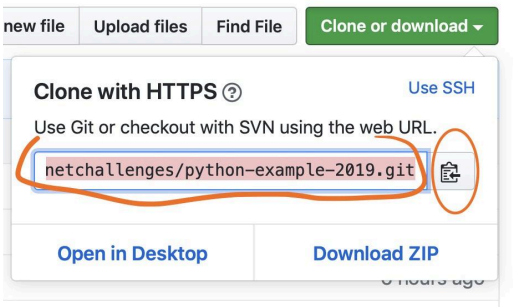
PhysioNet/CinC Challenge 2019 - Cloud Submission Instructions

Table of Contents

Preparation and submission instructions

1. Create a GitHub repository for your code. Add `physionetbuddy` as a collaborator if your repository is private.
2. Add your prediction code to your repository. Your code must be in the root directory of the master branch.
3. Do not add training data or anything else that is not needed to run your prediction code.
4. Follow the instructions for the language of your submission.
5. Use Google Forms to submit your entry:
    a. **Please submit your entry here.**
    b. Use the email address that you used to register for PhysioNet.
    c. We will clone your GitHub repository using the HTTPS URL that **ends in .git** (see figure on right). You can get this URL by clicking on "Clone or download" on GitHub and copying and pasting the URL. Please **DO NOT** enter the URL at the top of your web browser because it does not end in .git. Please see here for more details.



6. We will put the scores for successful entries on the leaderboard. The leaderboard will publicly show your team name, run time, and score.

MATLAB-specific instructions

1. Confirm that your MATLAB code compiles and runs in MATLAB 2019a.
2. Using our sample MATLAB prediction code (link) as a template, format your code in the following way. Consider downloading this repository, replacing our code with your code, and adding the updated files to your repository.
3. `AUTHORS.txt`, `LICENSE.txt`, `README.md`: Update as needed. Unfortunately, our submission system will be unable to read your README.

4. `load_sepsis_model.m`: Update this script to load your model weights and any parameters from files in your submission. It takes no input (place any filenames, etc. in the body of the function itself) and returns any output that you choose. **You must implement this function in the load_sepsis_model.m script.**
5. `get_sepis_score.m`: Update this script to run your model. It takes a matrix of clinical data (columns are clinical variables and rows are time windows) and the output from `load_sepsis_model` as input and returns a **single** risk score and a **single** binary prediction as output. **You must implement this function in the get_sepsis_score.m script.**
6. `driver.m`: **Do not change this script.** It calls your `load_sepsis_model` function only once and your `get_sepis_score` function for the first t = 1, 2, … time windows (first row, first and second rows, etc.) of each patient. It also performs all file input and output.
7. Add your code to the root/base directory of the master branch of your GitHub repository.
8. We will download your code, compile it using `mcc`, and run it on Google Cloud.
9. Here is a sample repo that you can use as a template: https://github.com/physionetchallenges/matlab-example-2019.git

Python-specific instructions

1. Using our sample Python prediction code (link) as a template, format your code in the following way. Consider downloading this repository, replacing our code with your code, and adding the updated files to your repository.
2. `Dockerfile`: Update to specify the version of Python that you are using on your machine. Add any additional packages that you need. Do not change the name or location of this file. The structure of this file is important, especially the 3 lines that are marked as Do Not Delete.
3. `requirements.txt`: Add Python packages to be installed with pip. Specify the versions of these packages that you are using on your machine. Remove unnecessary packages, such as Matplotlib, that your prediction code does not need.
4. `AUTHORS.txt`, `LICENSE.txt`, `README.md`: Update as needed. Unfortunately, our submission system will be unable to read your README.
5. `get_sepis_score.py`: Update this script to load and run your model using the following functions.
    a. `load_sepsis_model`: Update this function to load your model weights and any parameters from files in your submission. It takes no input (place any filenames, etc. in the body of the function itself) and returns any output that you choose. **You must implement this function in the get_sepsis_score.py script.**
    b. `get_sepsis_score`: Update this function to run your model. It takes a matrix of clinical data (columns are clinical variables and rows are time windows) and the output from `load_sepsis_model` as input and returns a **single** risk score a **single** and binary prediction as output. **You must implement this function in the get_sepsis_score.py script.**
6. `driver.py`: **Do not change this script.** It calls your `load_sepsis_model` function only once and your `get_sepis_score` function for the first t = 1, 2, … time windows (first row, first and second rows, etc.) of each patient. It also performs all file input and output.
7. Add your code to the root/base directory of the master branch of your GitHub repository.

8. We will download your code, build a Docker container from your Dockerfile, and run it on Google Cloud.
9. Here is a sample repo that you can use as a template: https://github.com/physionetchallenges/python-example-2019.git

R-specific instructions

1. Using our sample R prediction code (link) as a template, format your code in the following way. Consider downloading this repository, replacing our code with your code, and adding the updated files to your repository.
2. `Dockerfile`: Update to specify the version of R that you are using on your machine. Add any additional packages that you need. Do not change the name or the location of this file. The structure of this file is important, and especially the 3 lines that are marked as Do Not Delete.
3. `AUTHORS.txt`, `LICENSE.txt`, `README.md`: Update as needed. Unfortunately, our submission system will be unable to read your README.
4. `get_sepis_score.R`: Update this script to load and run your model using the following functions.
    a. `load_sepsis_model`: Update this function to load your model weights and any parameters from files in your submission. It takes no input (place any filenames, etc. in the body of the function itself) and returns any output that you choose. **You must implement this function in the get_sepsis_score.R script.**
    b. `get_sepsis_score`: Update this function to run your model. It takes a matrix of clinical data (columns are clinical variables and rows are time windows) and the output from `load_sepsis_model` as input and returns a **single** risk score and a **single** binary prediction as output. **You must implement this function in the get_sepsis_score.R script.**
5. `driver.R`: **Do not change this script.** It calls your `load_sepsis_model` function only once and your `get_sepis_score` function for the first $t = 1, 2, \ldots$ time windows (first row, first and second rows, etc.) of each patient. It also performs all file input and output.
6. Add your code to the root/base directory of the master branch of your GitHub repository.
7. We will download your code, build a Docker container from your Dockerfile, and run it on Google Cloud.
8. Here is a sample repo that you can use as a template: https://github.com/physionetchallenges/r-example-2019.git

Julia-specific instructions

9. Using our sample Julia prediction code (link) as a template, format your code in the following way. Consider downloading this repository, replacing our code with your code, and adding the updated files to your repository.
10. `Dockerfile`: Update to specify the version of Julia that you are using on your machine. Add any additional packages that you need. Do not change the name or the location of this file. The structure of this file is important, and especially the 3 lines that are marked as Do Not Delete.
11. `AUTHORS.txt`, `LICENSE.txt`, `README.md`: Update as needed. Unfortunately, our submission system will be unable to read your README.

12. `get_sepis_score.jl`: Update this script to load and run your model using the following functions.
    a. `load_sepsis_model`: Update this function to load your model weights and any parameters from files in your submission. It takes no input (place any filenames, etc. in the body of the function itself) and returns any output that you choose. **You must implement this function in the get_sepsis_score.jl script.**
    b. `get_sepsis_score`: Update this function to run your model. It takes an array of clinical data (columns are clinical variables and rows are time windows) and the output from `load_sepsis_model` as input and returns a **single** risk score and a **single** binary prediction as output. **You must implement this function in the get_sepsis_score.jl script.**
13. `driver.jl`: **Do not change this script.** It calls your `load_sepsis_model` function only once and your `get_sepis_score` function for the first *t* = 1, 2, … time windows (first row, first and second rows, etc.) of each patient. It also performs all file input and output.
14. Add your code to the root/base directory of the master branch of your GitHub repository.
15. We will download your code, build a Docker container from your Dockerfile, and run it on Google Cloud.
16. Here is a sample repo that you can use as a template: https://github.com/physionetchallenges/julia-example-2019.git

Docker-specific FAQs

1. **Why containers?**
   Containers allow you to define the environment that you think is best suited for your algorithm. For example, if you think your algorithm needs a specific version of CentOS, a certain version of a library, and specific frameworks, then you can use the containers to specify this. Here are two links with good, data science-centric introductions to Docker:
   https://towardsdatascience.com/how-docker-can-help-you-become-a-more-effective-data-scientist-7fc048ef91d5
   https://link.medium.com/G87RxYuQIV

2. **Quickly, how can I test my submission locally?**
   Install Docker → clone your repo → build an image → run it on a single patient.

3. **Less quickly, how can I test my submission locally? Tell me more-or-less exactly what to do.**
   Here are instructions for testing the Python example code in Linux. You can test the non-Python example code in a Mac, for example, in a similar way. If you have trouble testing your code, then make sure that you can test the example code, which is known to work.

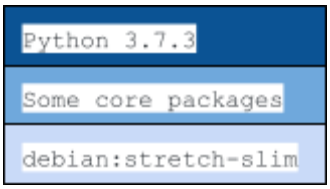
   First, create a folder, `docker_test`, in your home directory. Then, put the example code from GitHub in `docker_test/python-example-2019`, some of the training data in `docker_test/input_directory`, and an empty folder for the predictions in `docker_test/output_directory`. Finally, build a Docker image and ran the example code using the following steps:

```
user@computer:~/docker_test$ ls
input_directory  output_directory  python-example-2019

user@computer:~/docker_test$ ls input_directory/
p000001.psv  p000002.psv  p000003.psv  p000004.psv
p000005.psv

user@computer:~/docker_test$ cd python-example-2019/

user@computer:~/docker_test/python-example-2019$ docker
build -t my-image .

Sending build context to Docker daemon  95.23kB
[...]
Successfully tagged my-image:latest

user@computer:~/docker_test/python-example-2019$ docker
run -it -v
~/docker_test/input_directory:/physionet2019/input_directory -v
~/docker_test/output_directory:/physionet2019/output_directory
my-image bash

root@[...]:/physionet2019# ls
AUTHORS.txt  Dockerfile  LICENSE.txt  README.md
__pycache__  driver.py  get_sepsis_score.py  input_directory
output_directory  requirements.txt

root@[...]:/physionet2019# python driver.py
input_directory/ output_directory/

root@[...]:/physionet2019# exit
Exit

user@computer:~/docker_test$ cd ..

user@computer:~/docker_test$ ls output_directory/
p000001.psv  p000002.psv  p000003.psv  p000004.psv
p000005.psv
```

4. **How do I install Docker?**
   Go here: https://docs.docker.com/install/ and install the Docker Community Edition.

5. **Do I have to use your Dockerfile?**
   NO. The only part of the Dockerfile we care about are the three lines marked as “DO NOT EDIT”. These three lines help ensure that, during the build process of the container, your code is copied into a folder called `physionet2019` so that our cloud based

pipelines can find your code and run it. Please do not change those three lines. You are free to change your base image, and at times you should (see next question).

6. **What's the base image in Docker**?
   

   Think of Docker as a series of images that are layered on top of each other (see figure). This shows that our image is built on top of a very lightweight Ubuntu OS with Python 3.7.3. We get it from the official Docker Hub (think of it as a GitHub for Docker) for Python. Then install our requirements (NumPy and SciPy) on it. If you need the latest version of TensorFlow, then search for it on hub.docker.com and edit your file so that the first line of your Dockerfile now reads as: `FROM tensorflow` If you need a specific version (say 1.11), then lookup the tags and change it accordingly to `FROM tensorflow:1.11.0`

7. **sklearn or scikit-learn**?
   The single most common error we noticed in the requirements.txt file for Python submissions was the sklearn package. If your entry uses scikit-learn, then you need to install via pip using the package name `scikit-learn` instead of `sklearn` in your requirements.txt file: See here.

8. **xgboost?**
   For Python, replace `python:3.7.3-slim` with `python:3.7.3-stretch` in the first line of your Dockerfile. This image includes additional packages, such as GCC, that xgboost needs. Additionally, include xgboost in your `requirements.txt` file. Specify the version of xgboost that you are using in your `requirements.txt` file.

   For R, add `RUN R -e 'install.packages("xgboost")'` to your Dockerfile.

9. **Pandas?**
   Replace `python:3.7.3-slim` with `python:3.7.3-stretch` in the first line of your Dockerfile.

10. **Why can't I install a common Python or R package using Python or R's package manager?**
    Some packages have dependencies, such as GCC, that language package managers do not install. Try replacing `python:3.7.3-slim` with `python:3.7.3-stretch`.

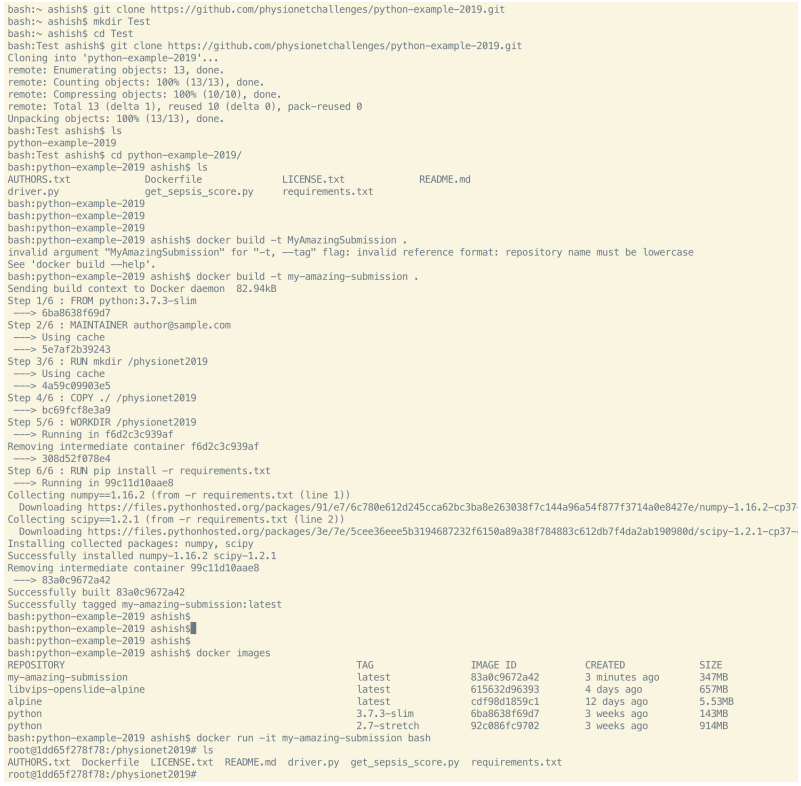
    If the first line of your Dockerfile is `FROM python:3.7.3-slim`, then you are building a Docker image with the Debian Linux distribution, so you can install GCC and other related libraries that many Python and R packages use by adding the line `RUN apt install build-essential` to your Dockerfile before installing your Python or R packages.

11. **How do I build my image?**
    ```
    git clone <<Your URL that ends in .git>>
    cd <<your repo name etc.>>
    ls
    ```
    You should see a Dockerfile and other relevant files here.

```
docker build -t <<some image name – it has to be all lowercase>>
.
docker images
docker run -it <<image name from above>> bash
```

This will take you into your container and you should see your code.

See below for a full walkthrough.



FAQ

1. **Should I submit your example code to test the submission system?**
   No, please only submit your code to the submission system.

2. **Should I submit an empty repository to test the submission system?**
   No, please only submit an entry after you have finished and tested your code.

3. **What can I do to make sure that my submission is successful?**
   You can avoid most submission errors with the four following steps:
   a. Do not change the driver script. We will only use the driver scripts (`driver.m`, `driver.py`, and `driver.R`) in the MATLAB, Python, and R example repositories (https://github.com/physionetchallenges), so any changes that you make will not be used.
   b. Do build your Docker image. The above FAQ provides advice for common Docker-related issues.
   c. Do test your Docker code on at least one file from the training dataset.
   d. Do try to reduce the run time of your code by moving code from the `get_sepsis_score` function to the `load_sepsis_model` function for

repeated tasks. Most submissions run in a couple of hours on the test data.

4. **Can I still use the old submission system? I don't want to rewrite my code.**
   No. We are using a cloud submission system for the official phase. Our previous submission system will not be available. The changes needed to run your code on the cloud submission system should be minimal.

5. **What do I need to change in my code for the official phase?**
   We are asking participants to report a risk score and binary sepsis prediction using the first *k* hours of a patient's clinical record. You no longer need to load data, save results, or think about file formats or extensions. See the example prediction code in https://github.com/physionetchallenges/ for details.

6. **Do I need to upload the training data? What about the code for evaluating my algorithm?**
   No, but please train your model on the training data before submitting it.

7. **Do you run the code that was in my GitHub repository at the time of submission?**
   No, not yet. If you change your code after submitting, then we may or may not run the updated version of your code. If you want to update your code but do not want us to run the updates (yet), then please make changes in a subdirectory or in another branch of your repository.

8. **Why is my entry unsuccessful on your submission system? It works on my computer.**
   There are several common reasons for unexpected errors:
   a. You may have changed the driver script. For consistency across submissions from different participants, we will use the driver scripts available on https://github.com/physionetchallenges/.
   b. You may have unmet dependencies. Note that packages in the requirements.txt file for Python submissions may have dependencies, such as gcc, that pip is unable to install. You can often identify such issues by trying to build a Docker image from your Dockerfile.
   c. You may have used a specific version of a Python or R package on your computer, but you didn't specify the version of the package in your Dockerfile or your requirements.txt file, so we installed the latest available version of the package. These versions may be incompatible. For example, if you train your data using one version of a machine learning package and we test it with another version of the package, then your entry may fail.

9. **Why does my code take so long to run on your submission system? It runs quickly on my computer.**
   We use a Google Cloud virtual machine with 2 CPU cores and 13 GB RAM. Our sample prediction code in https://github.com/physionetchallenges/ runs in approximately 1 minute on the test data; other Docker-related and cloud-related steps require several more minutes. If your prediction code takes significantly longer, then you may be able to significantly reduce your run time with one or more of the following changes:
   a. Train your model before submission.

b. Omit unnecessary packages, files, etc. from your entry. For example, unless your prediction code uses Matplotlib, remove it from your requirements.txt file.
c. Use the load_sepsis_model function to load model weights and perform other tasks that you can reuse across patients. We call the load_sepsis_model function once and the get_sepsis_score function many times, so you can use the load_sepsis_model function to avoid repeated tasks.
d. Profile your code. For example, it should take roughly twice as much time to make predictions for 200 patients as it does for 100 patients. If it takes significantly longer, then there is likely room for improvement.
e. Look into best practices for any machine learning packages that you are using in your entry. For example, loading model weights in TensorFlow for each patient in the get_sepsis_score function instead of once in the load_sepsis_model function will make your code run much more slowly.

10. **My entry had some kind of error. Did I lose one of my ten entries?**
    No, only scored entries (submitted entries that receive a score) count against the total number of allowed entries.

11. **Why is my utility score from the official phase lower than my utility score from the unofficial phase?**
    We are asking participants to write algorithms that make predictions in a causal manner using past and current but not future information. Some participants submitted non-causal algorithms for the unofficial phase, and we did not require algorithms to run in a causal manner for the unofficial phase. We are now enforcing causality, so algorithms that use future data for their predictions may receive lower utility scores on the same test data.