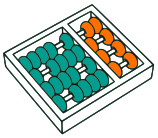

MO435 — Probabilistic Machine Learning
INSTITUTO DE COMPUTAÇÃO — UNICAMP
1st Semester 2020

UNICAMP

Project 2

Prof. Adín Ramírez Rivera
adin@ic.unicamp.br

1 Description

In this project you will implement a Bayesian Optimization procedure using Gaussian Processes (GPs). The general goal is that you encapsulate the behavior of the GPs in a handy class that can be called and used, e.g.,

```
gp = GP(kernel, sigma)
gp.addData(xtrain, ytrain)
mu, s2 = gp.posterior(xtest)
```

where `GP` is the class you will create in this project, `kernel` is any kernel function, and `sigma` is the prior of the noise's standard deviation. As you see, this implementation encapsulates the Bayesian Optimization process in a single class.

2 Implementation

Your task is, given an (n,d)-array of data `xtrain` and an n-vector `ytrain`, to predict the distribution of the (m,d)-array of test data `xtest`. To do such thing you need to compute the predictive posterior parameters μ and Σ of the function distribution.

Your default kernel function will be the squared-exponential kernel. For instance, the one-dimensional case is

$$k_\alpha(x,x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2l^2}\right), \tag{1}$$

where $\alpha = (\sigma, l)$ is the vector of parameters, and x and x' are the data points used in the kernel, where the covariance matrix entries are $K_{ij} = k_\alpha(x_i, x_j)$. In general, the kernel receives two arrays of size (n,d) and (m,d) and returns an (n,m)-array of all the pairwise evaluations. Your kernel signature must be `kernel(x1, x2, ell=1.0, s=1.0)`, where the parameters are optional.

Now, for the `posterior`, you need to explain what is form of the parameters of the Gaussian that defines it. That is, you need to derive and explain how you arrive to the posterior. (Review MLAPP [1, Ch. 15] for additional details.) Afterwards, implement the posterior function.

3 Experiments

For your experiments, you need to create a dataset for training and testing. For your tests you can use sine and cosine functions. Define a function, and sample 5, 10, 50 points from it, these will be your train points. Repeat the process for your test data. Setup a seed such that you can obtain the same data if the process is executed again. (This is needed to get the same data on the pipeline for execution, see § 6.)

Explain what happens when you train and test your data for the different amounts of data. Moreover, repeat your experiments for at least **three** functions.

To observe the behavior of the learned function, you can plot the mean plus minus two standard deviations and see the variation of your predicted functions. Explain what you observe.

Your implementation should be general enough to evaluate higher dimensional data. You should do it and see what it predicts. At least try with 2D data.

Also, see the effect of the kernel parameters, and the kernel itself, on your predictions. Change its values and the kernel, and explain what you see.

4 Evaluation

Your grade will be defined by the following aspects:

1. Explanation of the derivation of the GP 20%
2. Implementation of your class 20%
3. Exp. of functions approx. 20%
4. Exp. of different kernels 20%
5. Discussion of experiments 20%

Each item corresponds to the questions and requirements defined in the previous sections. The overall report, will be evaluated on the aspects regarding writing and requested plots or figures in the text. **Your language usage won't be graded**, but your ability to present your results, ideas, and how they are supported will be. Each point will be evaluated according to the completeness and correctness of the requested items.

Moreover, **the evaluation includes the contribution of each team member to the solution through the commits history and level of engagement of each team member through the repository**. Hence, do not commit all the work using one single student account. Each member should commit (using the email from the university that contains their student ID[1]) and handle their work on the repository.

5 Submission

Your submission must be through your assigned group on Gitlab. You must work with the same group. In case, you need to change from group or incorporate new members, let me know ASAP. The deadline for this is **06/04**. You must create a repository named `project-2` and commit all the code and report there.

Your submission must have the following subfolders:

- `input`: a directory containing the input assets (images, videos or other data) needed to execute your experiments. Your generated data should be placed here.
- `output`: a directory where your application should produce all the generated files (otherwise stated in the problem). This folder and all its contents must be added to the `artifacts` path on the `.gitlab-ci.yml` setup.

  Name your outputs according to a convention that reflects the experimental setup.
- `src`: a directory containing all your source code. You only need to submit files that are not derived from other files or through compilation. In case some processing is needed, prefer to submit a script that does that instead of submitting the files.
- `Makefile`: a makefile (or equivalent) that executes your code through the docker image. You must provide the image for the project. It is strongly recommended that you use a vanilla version of your language of choice, for example for python use `python:latest`. The code **must** be executed through a standard call to `make` (or equivalent). Moreover, note that your code **must** run on the Gitlab executor (pipeline) on the server itself. Hence, you need to use the `.gitlab-ci.yml` configuration. You can base your work on the demo that already provides an example of such functionality.
- `report`: a directory containing the source files that produce your report. This directory must be only on its branch called `report`.

  Your report **must** be written using LaTeX (and friends) and compiled within the pipeline of the repository in a stage named `report`. Consequently, **the PDF must not be committed**. You can use the images from `adnrv/texlive` to build your PDFs. **Only the PDF of the report** must be added to the `artifacts` path on

[1] Note that the repository information is extracted through scripts that use your student ID as an identifier for the committer. If you do not have the repository set correctly it won't find you. No effort will be made to find the missing information if you do not have your git set up correctly.

the `.gitlab-ci.yml` setup, and not other intermediary files of your report (e.g., images, log files, auxiliary files).

Your report must show all your work for the given project, including images (labeled appropriately, that is, following the convention given) and other outputs needed to explain and convey your work. When needed include explanations to the questions given in the project. **Your report should be constrained up to 4 pages in content (excluding images, tables, and references).**

The last commit that triggered the build must be before the deadline of the submission. Do not worry about the time executing the pipeline. However, **you must ensure that your code works as no attempt will be made to patch or run broken code**.

6 Notes

- All commits **must** come from the student's email (the one that includes the student ID). That is, you **must** configure your local git credentials in **every** machine that you will code with

```
git config --global user.email 123@unicamp.br
```

- **You are not allowed to use libraries that implement the GPs.** You may use, however, a library to do the sampling from the Gaussian, in case you want to show different functions from the prior. But even that can be achieved by re-parameterizing the Gaussian and then generating random numbers.
- All the submissions must be self-contained and must be executable in a Linux environment. Specifically, your code must execute in the docker image within the Gitlab pipeline.
- Your report must be written in English.
- While there are several images available for LaTeX, I recommend you to use one of the docker images `adnrv/texlive`, available at docker hub (`https://hub.docker.com/r/adnrv/texlive/`). In particular the `basic` and `full` images are standard versions of `texlive`. However, their sizes vary considerably (consequently, the execution times on the pipelines). If you need particular packages it may be easier to take a `basic` image and just install the packages you need using `tlmgr`.
- It is your responsibility to make sure your code compiles and executes correctly. No effort will be made to run your code besides executing the pipeline within the Gitlab repository.
- The proposal for the execution is to use a `Makefile`—§ 5. However, you may use another scripting language for running and executing the code from the stages of the pipeline. No support will be given in this case regarding your decisions on other tools.
- To test your code locally, I recommend a two-fold approach. Execute the code on the docker image by binding your working directory and the docker using volumes. Once your code is running, test the pipelines with a local Gitlab runner. That way, you won't be waiting for the runners to compile, run, and then report errors back to you.

References

[1] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.