Data Lab 4: Parallelization

ACMS 40876/60876 (Spring 2026)

Due Friday, March 27, 2026 11:59 PM via GitHub

Background and Goals

As the volume of data and complexity of models continue to grow, so too must the computational skills of modern data scientists. In particular, whether you are working at a university or in industry, data scientists must be comfortable with writing efficient, parallelizable code and working on remote servers. Without these tools, the computational cost of fitting state-of-the-art models on even moderately-sized datasets will be prohibitively time-consuming.

In this lab, you will practice parallelizing your code and submitting jobs on Notre Dame's high-performance computing system (CRC).

Instructions

As discussed in class, rather than performing the training-test split once, we can instead do many training-test splits in order to obtain an estimate of the uncertainty or *variance* of the test (generalization) error. One way to do this repeated training-test split is to use a cross-validation scheme so that each sample appears in the test split exactly once. In what follows, you will be asked to code up a cross-validation scheme to obtain this variance estimate[1] and to compare the time it takes to run the cross-validation procedure with and without parallelization. Specifically:

1. Choose one of your models f and evaluation metrics m from Lab 3. Also, choose the number of folds K to use in cross-validation and an appropriate data splitting scheme for the cloud satellite data from Lab 3. In the `lab4/README.md` file, document your choice of model f, evaluation metric m, number of folds K, and how you split your data into the various folds.

2. For your chosen data splitting scheme, model f, and evaluation metric m, run the following K-fold cross-validation algorithm twice on the CRC using the cloud satellite image data from Lab 3. Do this once without parallelization and once where the for loop has been parallelized across K cores. You may write `.py` and `.R` scripts instead of `.qmd` notebooks for this lab.

    - If you choose a model with hyperparameters, you may either (i) use an internal round of CV to tune those hyperparameters in line 2 of Algorithm 1 or (ii) fit the model with a fixed set of hyperparameters for simplicity.

[1] Note: Cross-validation is a data splitting procedure, which can be used for multiple purposes. One purpose is to tune hyperparameters within a model. Another purpose is to obtain an estimate of the variance of the generalization error.

Algorithm 1: K-fold Cross-Validation

Input: cloud data $\{X, Y\}$, K data folds $\{X^{(1)}, Y^{(1)}\}, \ldots, \{X^{(K)}, Y^{(K)}\}$, prediction model f, evaluation metric m

1 **for** $k = 1, \ldots, K$ **do**

2 Train the model f using the $k-1$ training folds $\{X^{(\setminus k)}, Y^{(\setminus k)}\}$. Call this trained model $\hat{f}$.

3 Use $\hat{f}$ to make predictions on the validation fold $X^{(k)}$: $\hat{f}(X^{(k)})$.

4 Evaluate the validation fold predictions using the evaluation metric m:

$$err_k = m(\hat{f}(X^{(k)}), Y^{(k)}).$$

5 **end**

6 Compute the mean of the cross-validation errors via $\overline{err} = \frac{1}{K}\sum_{k=1}^{K} err_k$.

7 Compute the variance of the cross-validation errors via $\frac{1}{K-1}\sum_{k=1}^{K}(err_k - \overline{err})^2$.

- There are ways to parallelize K-fold cross-validation using built-in `sklearn` and `caret` functions. See the `n_jobs` argument in `sklearn`'s `GridSearchCV()` and the `allowParallel` argument in `caret`'s `trainControl()`. However, for the purposes of this lab, DO NOT use these built-in methods. Please code up your own K-fold cross-validation according to Algorithm 1.

3. For each run (parallelized and unparallelized) of the cross-validation algorithm, print out (i) the mean of the cross-validation errors (from line 6), (ii) the variance of the cross-validation errors (from line 7), and (iii) the time taken to run the code. Print these outputs to the outputted `.o*` file. In your `lab4/README.md` file, record which `.o*` file corresponds to the parallelized run and which corresponds to the sequential (unparallelized) run.

Submission Details

Please push a folder named `lab3/` to your `dsip` GitHub repository **by 11:59 PM on Monday, March 16, 2026**. I will run an *automated* script that pulls from each of your GitHub repositories promptly at 12:00 AM and attempts to reproduce your report so please follow the file structure and names *exactly* with the exception that you may *add* folders as you wish.

The structure of your `lab4/` folder should follow the project structure discussed in class:

```
lab4/
├── data/
├── R/ (for R users)
├── python/ (for python users)
├── scripts/
├── job_scripts/
├── renv/ (for R users)
└── renv.lock (for R users)
```

```
├── environment.yml (for python users)
├── conda-lock.yml (for python users)
└── REAMDE.md
```

The `R/` and/or `python/` folders should contain all functions (and only functions, no scripts) necessary to reproduce your report. The `data/` folder should contain the raw data files, but **do not** push the `data/` folder to GitHub. In general, it is not good practice to store data on GitHub due to their restrictions on maximum file size (max: 100MB). The `scripts/` folder should contain all scripts, and the `job_scripts/` folder should contain all job submission scripts and the outputted `.o*` files.

In your `lab4/` folder, be sure to include all code necessary to reproduce your results including the job submission scripts. Please also include the two output (i.e., `.o*`) files with the printed outputs from problem (3) above.

A Note on Grading + Rubric

You will be graded on both the quality and reproducibility of your analysis.

A detailed rubric can be found on Canvas.

Please recall the course policy regarding collaboration: Collaboration *of ideas* with the instructor and with classmates is encouraged throughout this course, with the following caveats:

- You must write up the final code and text by yourself.
- If you collaborate or use any resources other than course texts, you must explicitly acknowledge your collaborators (e.g., in writing at the end of your report) and cite the resources you used.