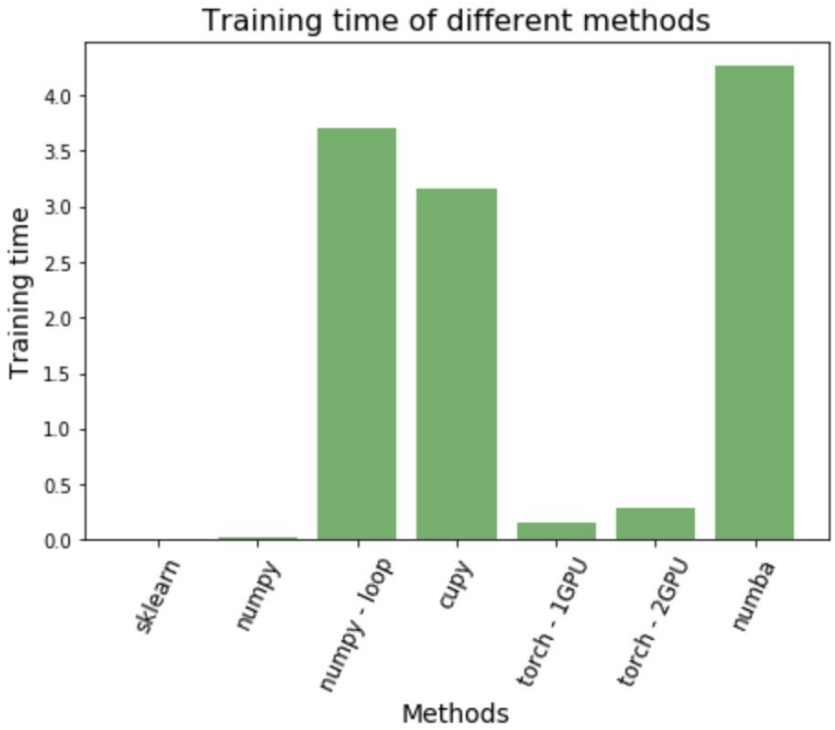
Logistic Regression via Different Computational Methods

Student: Elizaveta Lazareva

Goals of the project

- Implement Logistic Regression from scratch using:
  - NumPy (with matrix and loops operations);
  - CuPy;
  - PyTorch (1 GPU and 2 GPUs);
  - Numba.
- Compare training time on 2 datasets:
  - Heart Disease Dataset (toy data);
  - Titanic Dataset
    - 1309 rows with 10 features;
    - 13090 rows with 10 features;
    - 130900 rows with 10 features.

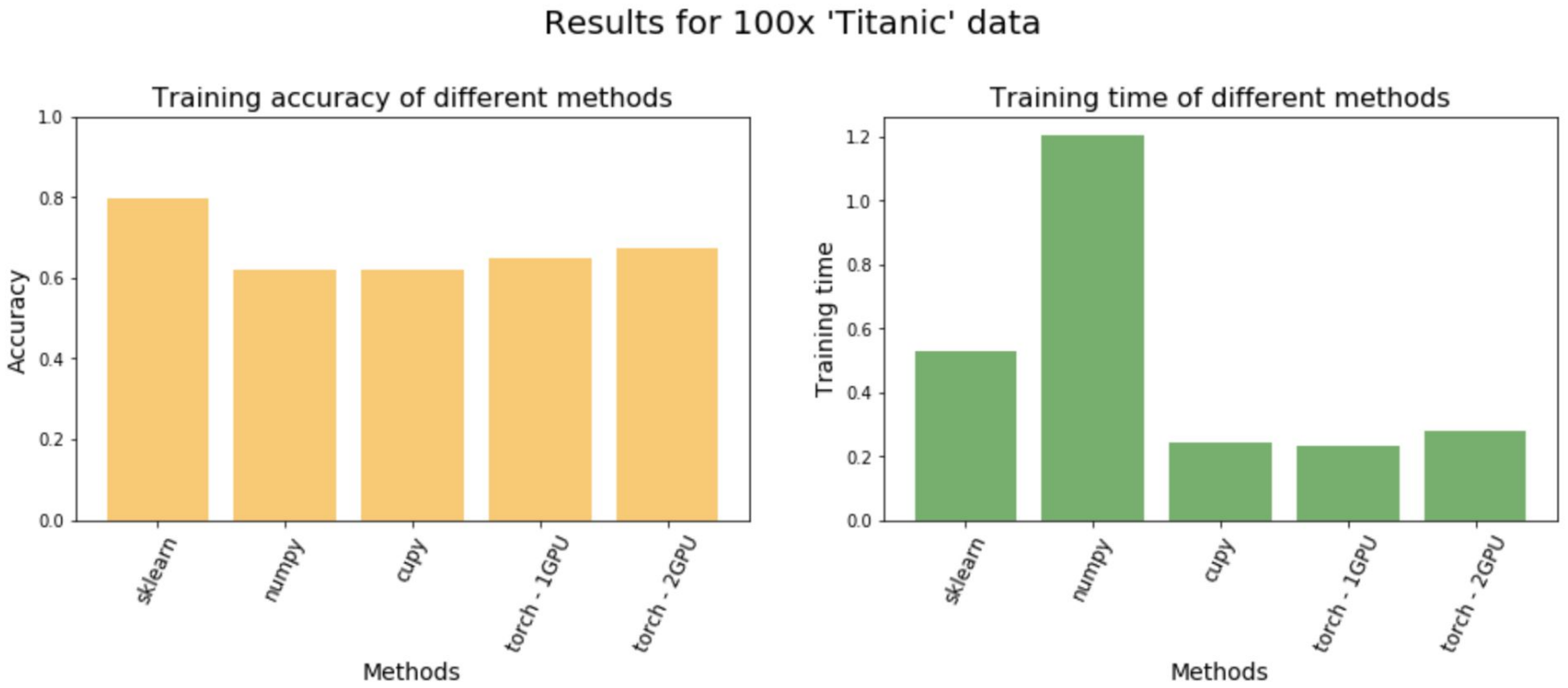
Results: Heart Disease Dataset



Results: Heart Disease Dataset

- The **fastest** methods are **Sklearn** and **NumPy** implementation with NumPy matrix multiplication.
- Computation using **PyTorch** takes a little more time for transferring data to GPUs: we can see that **time of training on 2 GPUs is twice bigger then for one GPU**.
- CuPy implementation takes much more time to transfer the data to GPU then PyTorch.
- The code with NumPy (with loops) implementation using njit failed in some reason. The time of use jit is presented in the plot, which is bigger that just using a NumPy “with loops” implementation. It is because of the time spent to transform the code into machine code and attempts to compile it.

Results: Titanic Dataset



Results: Heart Disease Dataset

- The training times for **Numba** and **NumPy** (with loops) implementations is **100 times bigger** than for other methods, which is very expected result.
- For the big number of data (100x ‘Titanic’ data) **NumPy** implementation is the **slowes**t one.
- The **CuPy** and **PyTorch** implementations performs the same and they are **the best**.
- We can see that there is **not enough data to PyTorch parallelism** become useful for this task: the cost for transferring data between different GPUs is not compensated by speed-up in computation.

Conclusions

- For small data (~1.000 rows x 10 features) it is more efficient to use Sklearn rather then other implementation because the cost of transferring the data to GPU is too big.
- Starting from approximately 50.000 rows x 10 features data it is more efficient to use GPU computation, because time(transfering to GPU) + time(computation on GPU) < time(computation on CPU).
- The transferring the small number data to GPU using CuPy, probably, takes more time then using GPU.
- More data than 100.000 rows x 10 features is needed to make computation and transfering the data to 2 GPUs more efficient than computation on one GPU.