**Learning algorithm:**

$$\arg\min_{w\in\mathbb{R}} \left\{ ||Xw - Y||_2^2 + \lambda ||w||_2^2 \right\}$$

$$X \in \mathbb{R}^{N\times d}; w \in \mathbb{R}^d, \lambda \in \mathbb{R}, y_i \in \{-1, 1\}$$

Analytical solution to the problem:

$$w^* = \left(X^T X + \lambda I_d\right)^{-1} X^T Y$$

Generating a prediction in the binary case due to Bayes Decision Rule:

$$\hat{y_i} = \text{sign}(\langle x_i, w^* \rangle)$$

Multi-class: $k$ is the number of classes
For $j \in [1, k]$:
Solve

$$\arg\min_{w_j} \left\{ \frac{1}{N} \sum_{i=1}^{N} (\langle w_j, x_i \rangle - y_i^k)^2 + \lambda ||w||_2^2 \right\}$$

$$\hat{y_i} = \text{index}\left(\arg\max\{\langle x_i, w_1^* \rangle, \cdots, \langle x_i, w_j^* \rangle, \cdots, \langle x_i, w_k^* \rangle\}\right)$$

**Parallelizable computations:**
*Matrix multiplication*:

$$A = (X^T X + \lambda I_d)^{-1}$$

$$n_\lambda(2Nd^2 + d^2 + d^3)$$

$$A(X^T Y)$$

$$2d^2 + 2Nd$$

$$X_v w$$

$$2Vd$$

$$C_p = 2n_\lambda(Nd^2 + kd(d + N + 2V))$$

We note that since $k \ll d$, the $Nd^2$ is going to dominate the computation. We can time the amount of time it takes to compute the parallelizable and overhead computations as

$$t_{node} = t_p + t_{overhead,openmp}$$

In the model-parallel case using MPI, we can break up the total time to train and validate as

$$t_{total} = \gamma \left( \frac{n_\lambda}{n_{ranks}} \right) t_{node} + t_{overhead,mpi}$$

We can empirically derive the speedup due to OpenMP parallelism by running the model in various thread configurations one one node (Figure XX). We time just the parallel computations, and compare to the total time to run the simulation. We can see from this plot that $t_p$ decreases while $t_{overhead,openmp}$ does ....

[Figure]

When we switch to just an MPI framework, we can gauge the total speedup due to MPI by comparing the total time of computation against $t_{node}$ as a ratio of ranks to lambdas (Figure XX), when the number of OpenMP threads is fixed at one.

[Figure]

We see that speedup increases a function of number of nodes....
We can combine the OpenMP and MPI framework to form hybrid parallelism. Below we plot the speedup, scaled speedup, and efficiency of our hybrid configuration (Figure XX). We see that we get the most optimal performance for X nodes, and Y threads.

[Figure]

Data parallelism divides the training set of images into $N/p$ chunks, over which the model fits $p$ weights in each MPI node. Then the weights are combined into an average that is tested on the validation set. Figure (XX) shows the speedup running hybrid data parallelism.