Parallel Implementation of Logistic Regression

Aditya Joglekar 201401086
Rajdeep Pinge 201401103

What is Logistic Regression?

- Logistic regression is a technique to build a classifier.
- The input is a labelled dataset i.e. a set of training samples each having some input features and a class label (class to which the sample belongs).
- The features are typically values which encode some information. These features help us classify the sample into one of the available classes.
- Classification Types:
  1. Binary Classifier - 0/1, yes/no classification
  2. Multi-class Classifier - one vs all classification, extension of binary classification.

Basic Details of Algorithm (Examples)

We assume real valued weights such that the linear combination of weights and features through a nonlinear function will give us an estimate of the conditional probability of whether the training sample is positive (1) given the learnt parameters. Use this to classify unseen samples.

E.g - Given a patient's medical diagnosis predict if the patient is healthy or sick (yes/no problem).

This is a clever extension of the same binary classification idea called one vs all classification. If we have k classes, we train k binary classifier by considering the input labels to be 1 if they are of the type being considered (say i^{th}) else 0.

E.g - Classify MNIST handwritten images of digits 0 to 9 into 0-9 classes (standard ML problem). Input: Image parameters. Output: Number to which the image resembles.

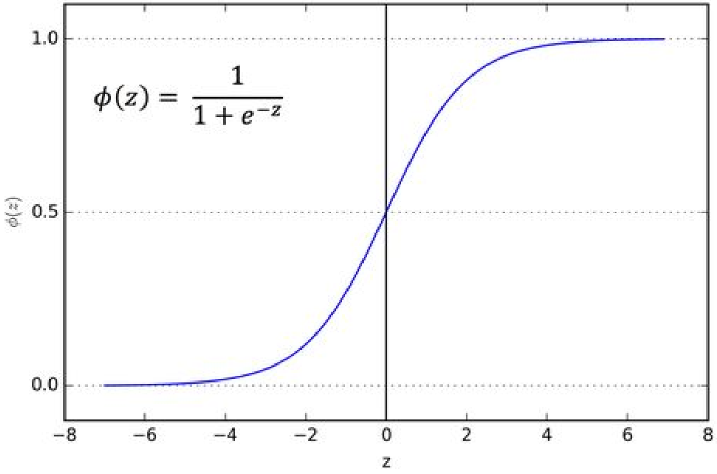
Basic Details of Algorithm (Terms, Strategies)

- Parameters/Weights - used to measure the response of a feature
- Cost- A measure to gauge the quality of the weights
- Learning - Finding **optimal** weights such that cost is minimized
- Gradient descent - **Iterative** method to gradually learn the above weights
- Prediction - To use learnt weights to classify **new, unseen** training examples
- Sigmoid function - Non linear function, helps in learning and prediction based on input features.

$$\rightarrow J(\theta) = \frac{1}{m} \underline{\sum_{i=1}^{m}} \underline{\mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})}$$

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Note: $y = 0$ or 1 always



Complexity of Algorithm

Method 1: OLF (Outer Loop over features)

Serial: O(tn^2m). t: number of sequential iterations, m: number of samples used for training, n: number of features for each sample.

Parallel: O((tn^2m)/p) where p is the number of processors.

Theoretically we should get the maximum possible speedup of p. However, we will soon see that this reasoning is specious as this does not take into account the overhead for maintaining the threads

Method 2: OLT (Outer Loop over Training Samples)

Serial complexity: O(t(mn+2n)) ~ O(tmn)

The outer loop over training samples has been parallelized. Which takes O(mn) time

Hence the parallel complexity: O(t((mn/p)+2n)). Thus if we take m to be sufficiently larger than n, then we can expect speedup close to p.

Figure 4: Formula for gradient descent to adjust weights

Repeat until convergence

{

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

}

Profiling

Code	Percentage TIme Spent in sigmoid()	Number of sigmoid() calls
Normal Serial Code for binary classification OLF	51.68%	60,10,990
Serial Code for binary classification with OLT	0.01%	14,990
Multiclass classification with OLT	0.31%	54,00,000

- Two important function train() and sigmoid().
- Calls to sigmoid depend on the implementation and complexity of train()
- Difference in Sigmoid calls in binary code
- Compare with number of calls in multi-class code
    1. Time taken
    2. Multi-class faster than binary!

Parallelization Strategy, Most important Optimizations used

OLF:

- Straight-forward method. Easier to implement in C.
- Simple parallelization of the inner loop over training samples.
- Parallelizing the outer loop over the feature columns. The effectiveness of this method depends on the problem size
- The parallelizable work should be able to cover the overhead of maintaining the threads.

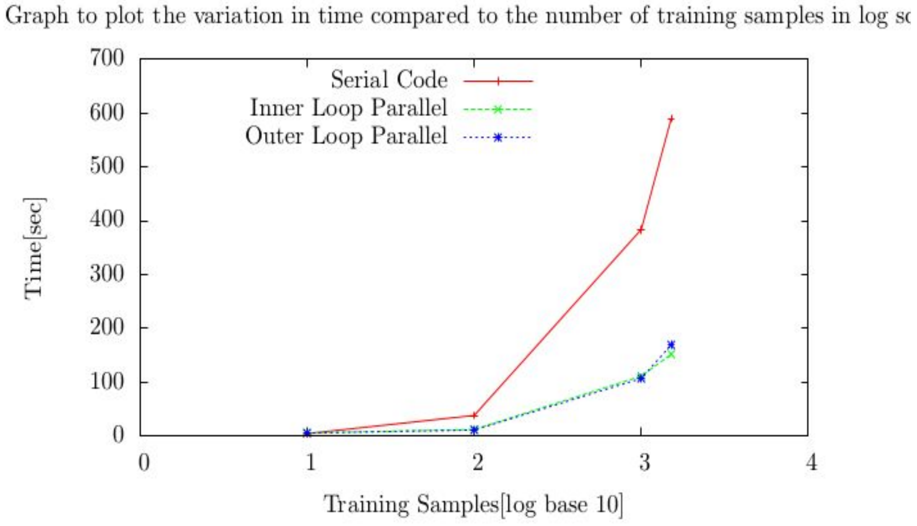
OLT:

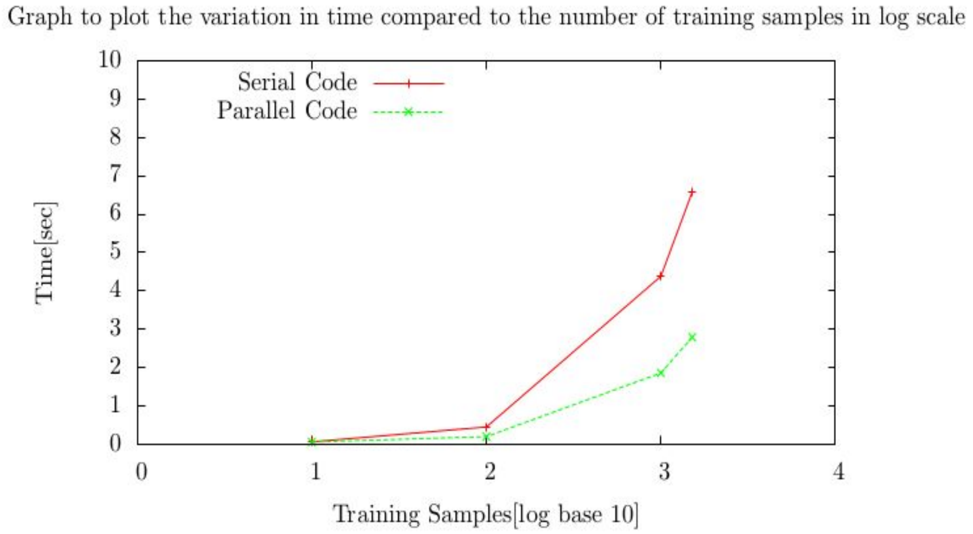
- Flip the loop and parallelize the loop over training samples.
- Decreases work complexity by a factor of n.
- Much faster than OLF because of reduced complexity.
- Better parallelization possible as done over training samples.
- Best speedup for large enough datasets.

Problem Size vs Time

Normal Serial code OLF

Code with Swapped Loops OLT



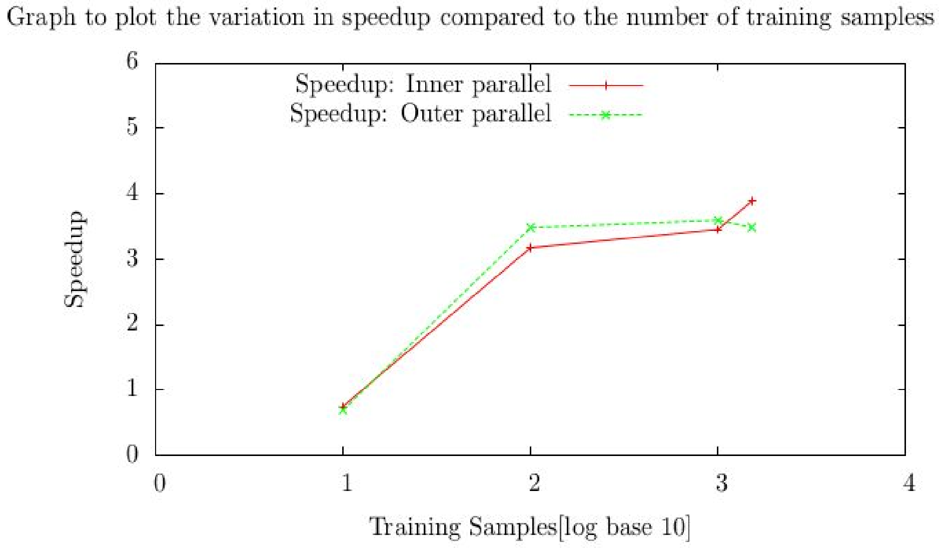


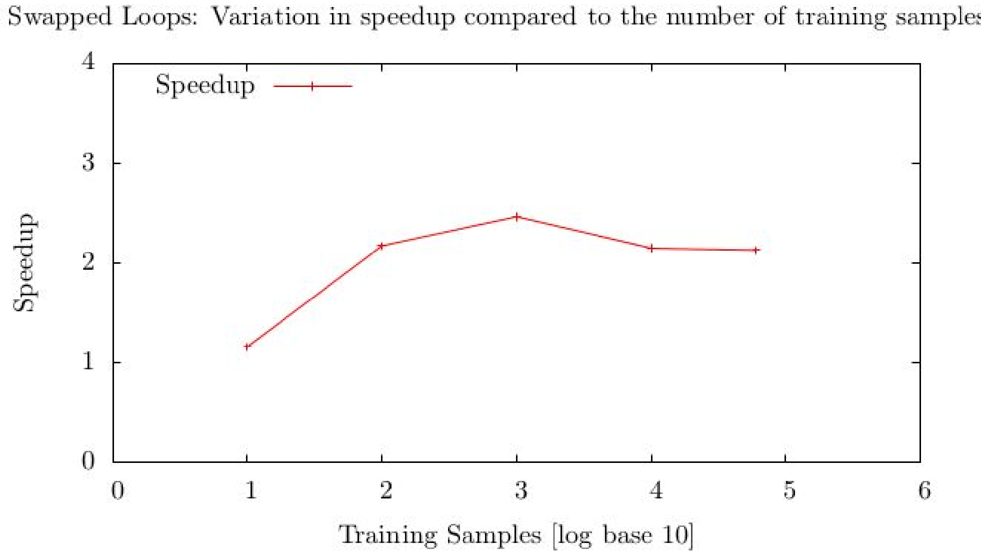
Time Difference

Problem Size vs Speedup

Normal Serial Code OLF

Code with Swapped Loop OLT



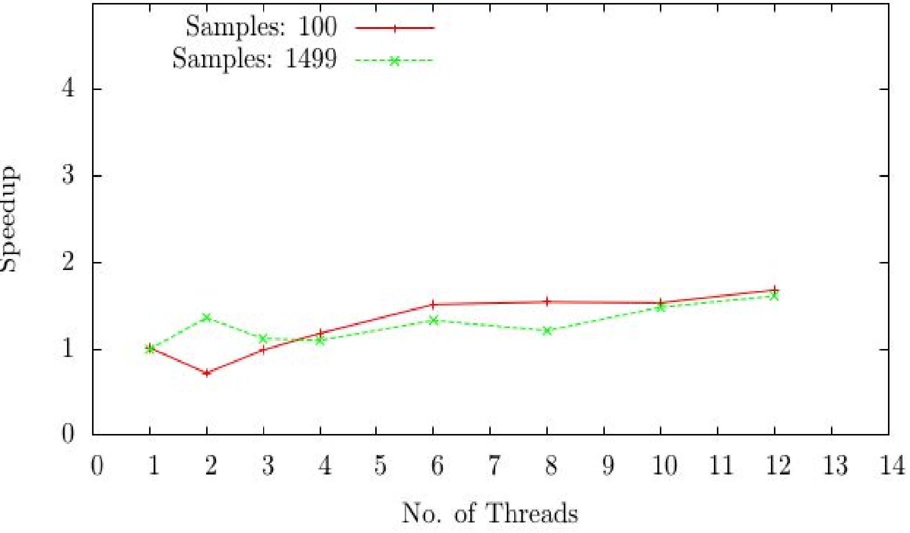


Difference in Speedup obtained

No. of Cores vs Speedup (Cluster)

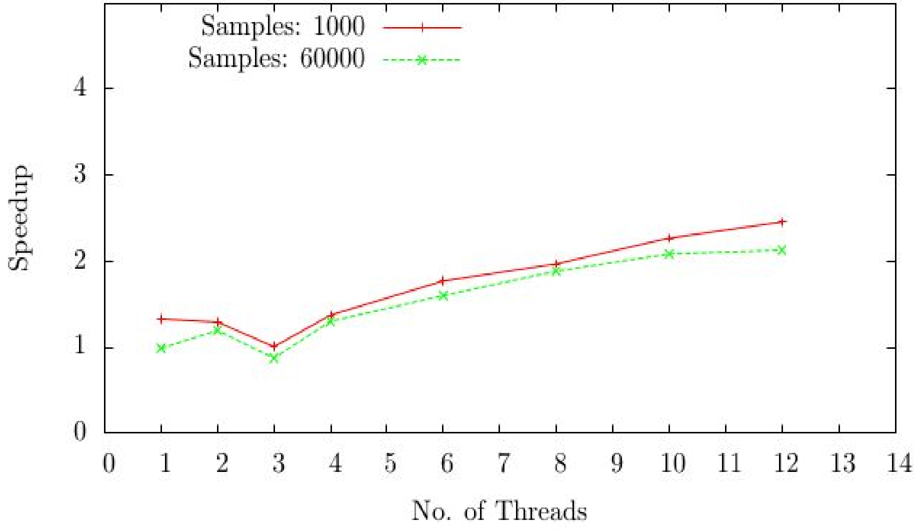
Binary

Binary Swapped Loops: Variation in speedup compared to the number of thre



Multi-class

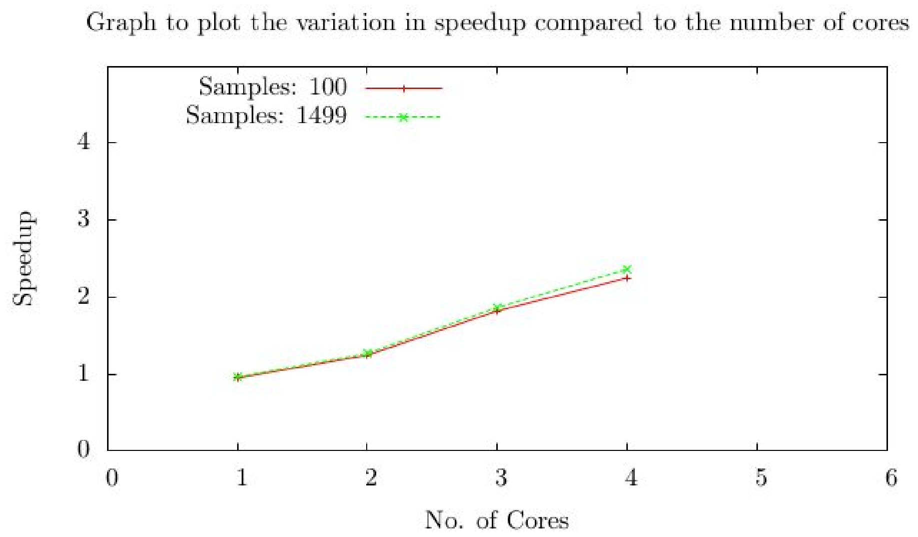
Multiclass Swapped Loops: Variation in speedup compared to the number of threads



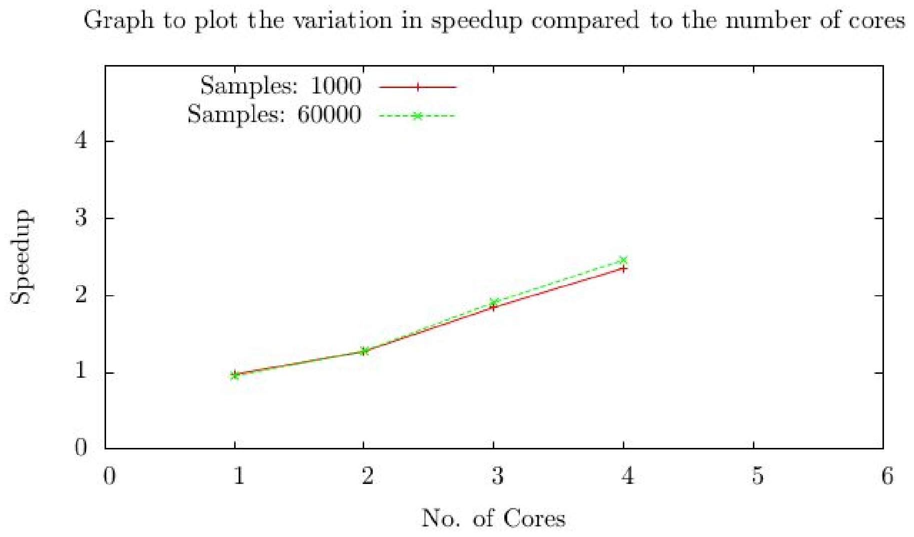
Higher Speedup for smaller problem size

No. of Cores vs Speedup (PC)

Binary



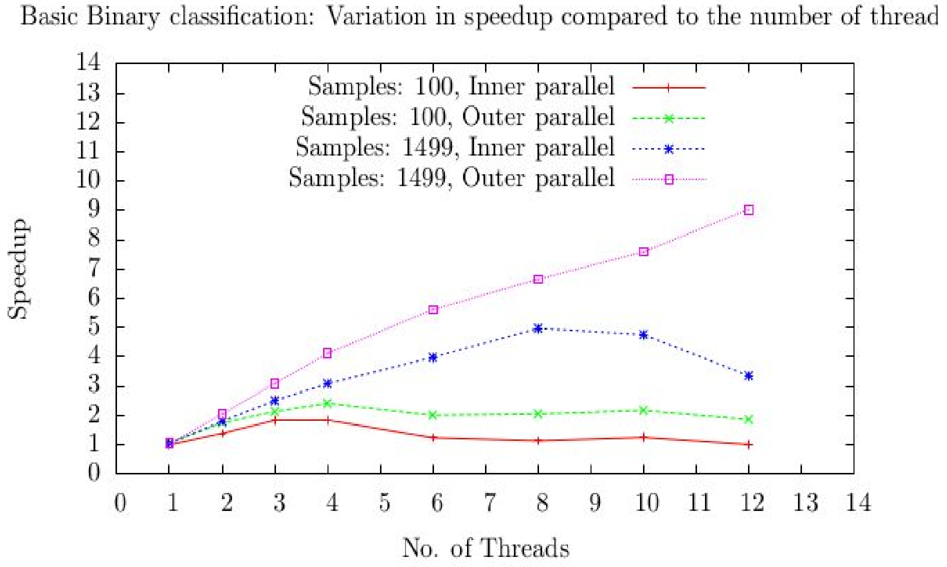
Multi-class



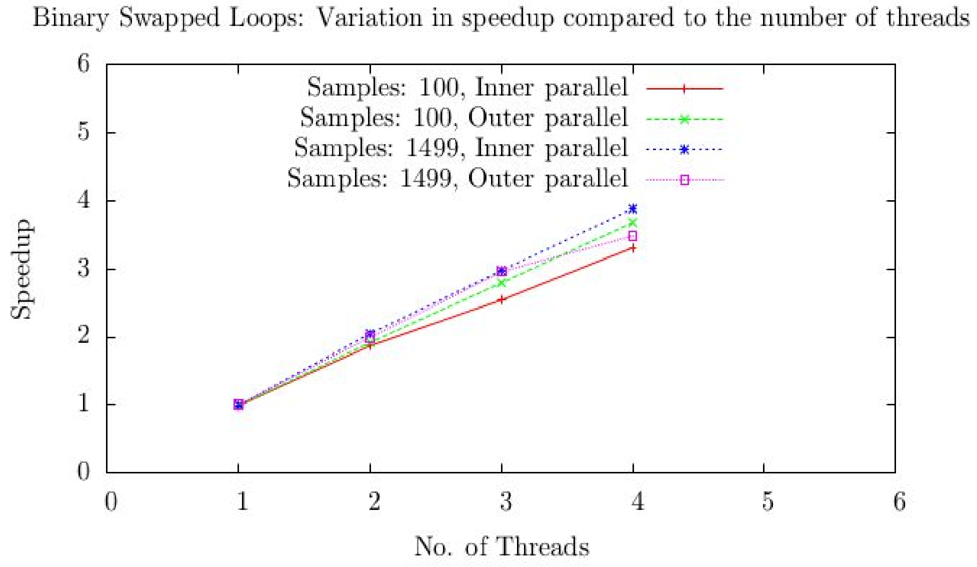
Speedup trend w.r.t Number of Threads

No. of Cores vs Speedup (Basic Binary)

Cluster



PC



Optimum threads

Important Observations

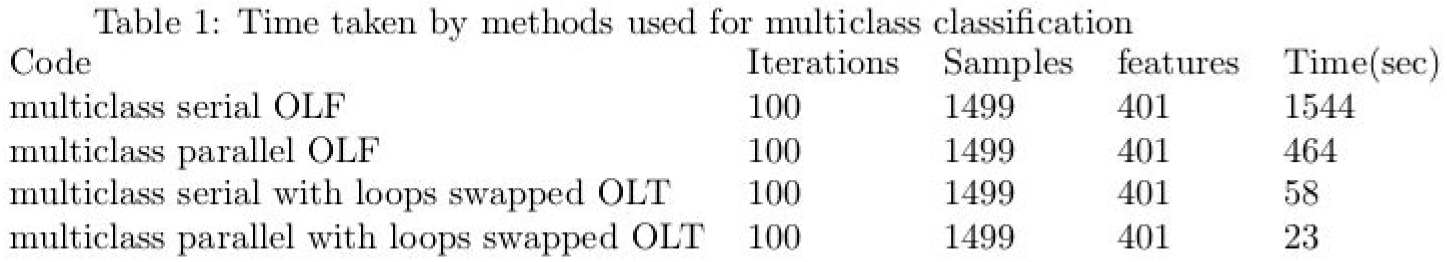


Table 1: Time taken by methods used for multiclass classification

Code	Iterations	Samples	features	Time(sec)
multiclass serial OLF	100	1499	401	1544
multiclass parallel OLF	100	1499	401	464
multiclass serial with loops swapped OLT	100	1499	401	58
multiclass parallel with loops swapped OLT	100	1499	401	23

- Time difference between different implementations

- Sequential number of iterations, constant speedup for change in number of iterations for training.

Performance Analysis, Limitation in speedup

- Training accuracy close to 100%
- Testing accuracy ~ 90% (multi-class)

  ~ 95% (binary)

- Best speedup ~ x2.5 (4 threads)

  ~ x2.45 (12 threads)

- What are the reasons for suboptimal speedup?

.

Limitations:

1. A major problem, the adjust weights function is iterative. This cannot be parallelized. So, there is a significant portion of the code which is serial in nature.
2. Data access is huge. Of the order of 10^9 or 10^{10}, Because we go over all training samples and update each weight for a number of iterations till the least cost is obtained.
3. The inner loop has expensive functions like sigmoid(). We can't parallelize them without drastically increasing the overhead.

Conclusion

- The standard problems in this domain have little communication, high granularity.
- The learning procedure is iterative. Also Thread management overhead and heavy amount of data access due to vector operations.
- Step dependency while training.
- The speedup obtained in OLT is less than OLF but so is the absolute time in OLT. The OLT serial code runs faster than OLF parallel code.
- Prominent sequential part and data access main problems for speedup.
- Huge difference in time and speedup curves for two different methods used.
- The speedup stays nearly constant even for large datasets. Scalable in this sense. Unlike usual problems, the non-parallelisable part also increases with increase in input size.
- Speedup of x2.5 pretty significant with respect to typical machine learning timeline (days, weeks, months).

Problem Idea and Possible Extensions

- Standard machine learning problem.
- Reference for the basic theory part of the problem: Online course conducted by Prof. Andrew Ng, Stanford University.
- The complete code has been written by us with the help of this theory material.
- We have used GNU Scientific Library for linear algebra support.
- The standard MNIST data-set for ML has been used for testing purpose.
- Extension to make the code suitable for distributed computing systems so that more number of parallel computing units can be used.
- Improvements to the algorithms to increase prediction accuracy (obtained ~90%, expected 94%).
- The state of the art classification model are neural networks which build on the logistic regression idea. Parallel implementation of neural networks will be a challenging extension of this problem.