Parallel Data Mining

Team 2 – Flash Coders

Team Research Investigation Final Report
Foundations of Parallel Computing

Umang Gala Robert McCartney Mandeep Shetty

Rochester Institute of Technology

Table of Contents

1. Overview of topic area

Big data is everywhere. Organizations deal with massive amounts of data every single day. Corporate entities, universities, research groups, and even individuals consume and produce these vast amounts of data as an integral part of their business models, research interests, and workflows.

Here are a few use cases for Big Data:

- Walmart has over 200 million POS transactions per day.
- Facebook has a 300 petabyte data warehouse. To this massive store they add nearly 600 terabytes of data daily.
- YouTube gets 4 billion views per day.
- Instagram users upload 40 million photos per day.
- On a daily basis, Google processes over 20,000 terabytes of data.

Companies use data to understand customers' habits and gather insights into how the markets work, allowing these businesses to target consumers better and maximize their revenue. This is a form of prediction from data modeling, which is one of the main motivations for gathering all this data in the first place. Other types of predictions that rely heavily on data include what the weather will be like, how a baseball team will perform, what real estate prices will be like next year, and the list goes on.

These types of questions are the subject matter of machine learning. Simply stated, machine learning is computers processing data to "learn" a model from which predictions or decisions can be made. Once the model is learned, the computer no longer needs explicit instructions like a traditional program or finite state machine, but can instead operate from its learned probabilities and observed patterns to extrapolate into a prediction, similar to the higher order reasoning possessed by every human being.

There are a variety of algorithms that computers can use to learn from data. Regression analysis is one of the major techniques used for machine learning. Regression is a process in which the relationship between input and output variables is estimated from a known dataset. The two major types of regression are called linear regression and logistic regression.

In linear regression, the relationship between dependant and independent variables is estimated as a weighted sum of the input variables without any higher-order terms, hence the 'linear' nomenclature. This is usually used for estimating the values of continuous data given some input vector of features. One example is estimating the price of an apartment based on its various features such as area of the apartment, age, locality, and so on.

Logistic regression is a modification of linear regression for use in probabilistic classification. The probability that an outcome belongs to a certain class given some pieces of input evidence is the question logistic regression attempts to answer, by in effect using the sigmoid function to send the output of linear regression to 0 or 1 continuously.

Both of the aforementioned regression models try to minimize what is known as a cost function by adjusting its parameters. For instance, squared error, or the square of the difference between actual and expected results, is the cost function used in linear regression. It is only this cost function that varies between many slightly different forms of regression.

2. The computational problem

a. Gradient Descent

The problem that this team research investigation aims to tackle is using logistic regression with gradient descent in a parallelized manner on a muti-node cluster. Gradient descent is a 1st order optimization algorithm generally used to maximize or minimize an input function.

The range of values that a function takes can be visualized as a bumpy surface in multi-dimensional space, and in turn gradient descent can be likened to taking small steps on the surface of this space to reach a local minima or maxima. For minimizing a cost function like logistic regression, gradient descent takes small steps 'downhill' until it reaches a local minimum, where a step in any direction leads to a higher cost. This local optima found may or may not be a global optima, depending on the initialization of the parameters to the algorithm.

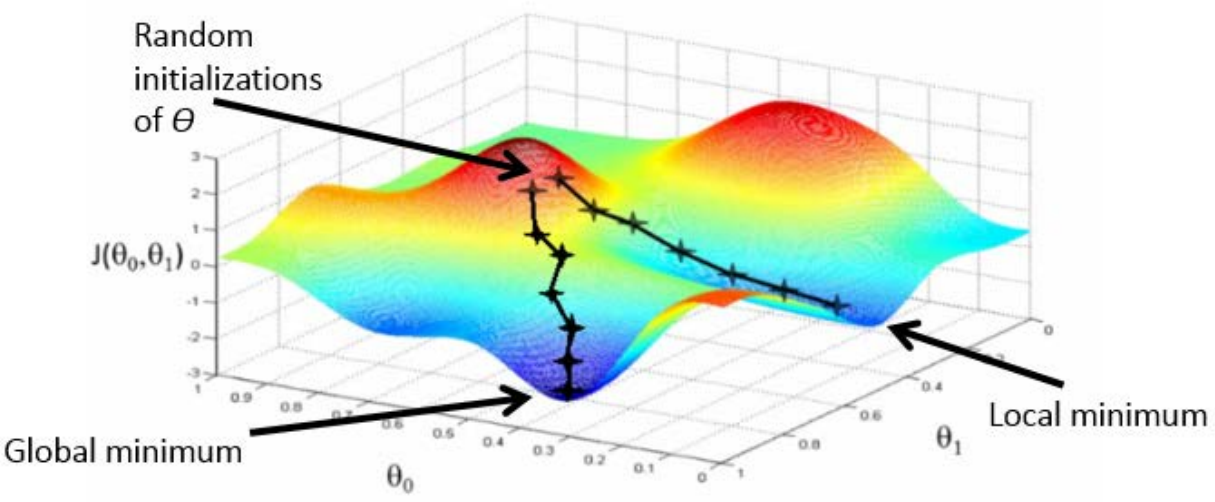


Figure 1. Gradient Descent

The figure above visualizes stochastic gradient descent. The blue areas are the minima on the surface of the function. As we can see, there is both a local and global minima. A small difference in the starting value of gradient descent can make a big difference in the final destination, depending on the contours and shape of the function. To deal with this, usually a series of random restarts are done with each restart from a different random point on the surface. The smallest of the minima returned from these restarts is then used to build the model.

b. Sequential Logistic Regression

As we discussed, logistic regression tries to minimize the cost function, which captures how far from the actual class our estimate of the input features is. The mathematical notation for squared error (there are other forms of the cost function) is:

$$\min_{\Theta} J(\Theta) = {}^{1}/_{2m} * \sum_{i=1}^{m} (h_{\Theta}(x^{i}) - y^{i})^{2}$$

Here, the function $h_\theta(x)$ takes an input vector x and returns a continuous prediction of the output classification. This difference of the estimate from the actual class is squared to give the cost of this instance. Often, a regularization term is also added to the cost to prevent over-fitting the parameters, but that term is left off here for brevity. For batch training, this cost is calculated on every example in the dataset and added together for the final cost of one iteration of training. To start training, the algorithm randomly initializes the values of θ, equivalent to choosing the random starting point in Figure 1. The weights θ are then updated each iteration according to the partial derivatives of the cost function for that parameter and a learning rate alpha, for instance as:

$$\theta j = \theta j - {}^{\alpha}/_{m} * \sum_{i=1}^{m} \left(h_\Theta\left(x^i\right) - y^i\right) * x^i$$

In Figure 1, the above equation is equivalent to taking one of the small steps towards the minima in an attempt to minimize the cost function. The learning rate α is analogous to the size of the step we take or how fast we descend down the function to the minima. If α is too low, then the descent might take a long time to converge to correct value. But if α is too high, then we might continually take excessively large steps around the minima or step over it without ever finding the optimal point.

This whole process is repeated till convergence to a point of local minimum cost. As we can see, this process of summing over the errors of the entire dataset can take a long time when done sequentially, especially if the number of instances m is in the millions. As mentioned previously, datasets this large have become commonplace, so it is not infeasible to run up against this problem in industry or academia. The power of parallel computing can help us perform logistic regression over such extremely large data sets.

After learning all the parameters θ of the model from our dataset using regression, the final step is to classify unseen instances of data based off of their input values. This is the prediction step, the purpose for performing such a machine learning algorithm in the first place. With a trained model, we can classify unseen instances using a weighted sum in linear regression (hence its nomenclature) to get a continuous prediction on the dependent variable:

$$h_\Theta\left(x^j\right) = \sum_{j=1}^{n} \theta j * x^j$$

Logistic regression takes an extra step, applying the sigmoid function to this weighted sum in order to send its output towards 0 and 1 as the classification decision on this previously unseen instance. Thus, setting the above summation equal to z, logistic regression gets an output value of:

$$h_\Theta\left(x^j\right) = {}^{1}/_{1 + e^{-z}}$$

In this way, we have discovered useful patterns and knowledge in the data without explicitly programming the decision conditions or relationships between variables. We can say the machine has learned.

c. Parallel Logistic Regression

The sequential algorithm can be greatly help through judicious use of parallel computing. As we noted above, there is a summation in batch training taking place over all instances of the data that is

independent of other data instances. Thus, there are two clear ways this algorithm can be parallelized in a cluster computer. In one node of the cluster, we can use multiple cores of the node to do the summation over instances with a parallel reduction. Second, we can do such core-parallelized summations in parallel over worker nodes, each with a separate subset of the data, before aggregating such summations from every node in the cluster on a master node. The master node can then make updates to the parameters before sending the results back to the workers for the next iteration of learning. A third parallel approach would be to dedicate different random initializations to different nodes, but that approach is not discussed here.

3. The URL dataset

The dataset we used for this project is the URL dataset the University of California, San Diego. It is a dataset that contains classifications for URLs as malicious or benign based on their lexical and host-based features. The authors decided to use these features as a way to learn about the safety of a URL without exposing themselves to the risk of downloading the page, and as a computationally efficient classification technique compared to techniques used by antivirus software or other computationally expensive methods. The dataset itself consists of 2.4 million instances of URLs with 3.2 million features, most of which are binary as part of a bag of words. The dataset is labelled with +1 corresponding to a malicious URL and -1 to a benign one.

4. Paper 1: Evaluating parallel logistic regression models

Problems:

- Provide design guidelines to choose most effective combination of parameters for logistic regression on large data sets
- Compare and evaluate different distributed platforms, parallel algorithms, and sub linear approximation techniques.

Approaches:

- Platforms: Hadoop & Spark
- Algorithms:
  - Sequential optimization with stochastic gradient descent using LIBLINEAR (machine learning library)
  - Sub linear algorithms: use sampling of the dataset

Results:

- LIBLINEAR was the most precise and also the fastest
  - Ran on single machine
  - No inter-machine communication overhead and uses memory fully
  - Scalable, but limited by memory size
- LIBLINEAR was best if data set fits in memory, but couldn’t fit entire URL dataset
- Spark performs better than Hadoop
- Most machine learning algos iterate over the dataset
- Hadoop works off of disk while SPARK works off memory using RDDs
- Hadoop has non-negligible overhead for setting up tasks and passing parameters, especially for small datasets
- To get general sense of running time, we should consider both volume and the sparsity of data
- Observed drop in speedup with increased number of nodes in the cluster due to communication overhead (poor strong scaling)

Uses:

- LIBLINEAR was the best logistic regression technique

- Our dataset, however, is too big to fit in memory. Has about 3 million features, 2 million instances, and is very sparse
- Authors recommend parallel gradient descent for this situation, which is exactly what we will do using the tardis cluster
- Measure speedups with different number of workers

5. Paper 2: Efficient Mini-batch Training for Stochastic Optimization

Problems:

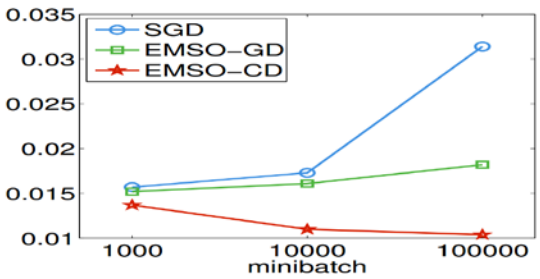
- Stochastic gradient descent for large-scale optimization problems has significant communication overhead
- To reduce the communication cost in parallelized SGD, mini-batch training needs to be employed
- But, when mini-batch size increases then rate of convergence decreases (the problem in Paper 1)
  - Uses the same URL data here from Paper 1
- This paper discusses a technique that regularizes the objective function within each mini-batch iteration to prevent the convergence rate from decreasing with increased mini-batch sizes

Modified minimization equation:

- Prevents divergence from previous consensus found
- Limits dramatic changes of parameter w over batch iterations
- Batching the data reduces the synchronization cost.

Results:

- Objective value versus mini-batch size after 10^7 examples are processed
- Note SGD diverges but efficient mini-batch training does not



Uses:

- Use the technique described for efficient mini-batch training without subsequent derease in the convergence rate

6. Paper 3: Parallel Large Scale Feature Selection for Logistic Regression

Problems:

- Need to try all 2^N feature combinations to find the best model
  - In sparse, high-dimensional data this is infeasible
- Logistic regression works well in high-dimensional data since it is likely linearly separable (this is the premise of kernels)

Theory:

- Approximate the best feature to add to an existing model by building a new model in parallel for each feature NOT already in the model

    - Dth iteration, need to train N – D models
- Add the best feature found to the model and re-run full logistic regression, to make sure approximation error does not propagate
    - Have to run full regression N times, or stop short

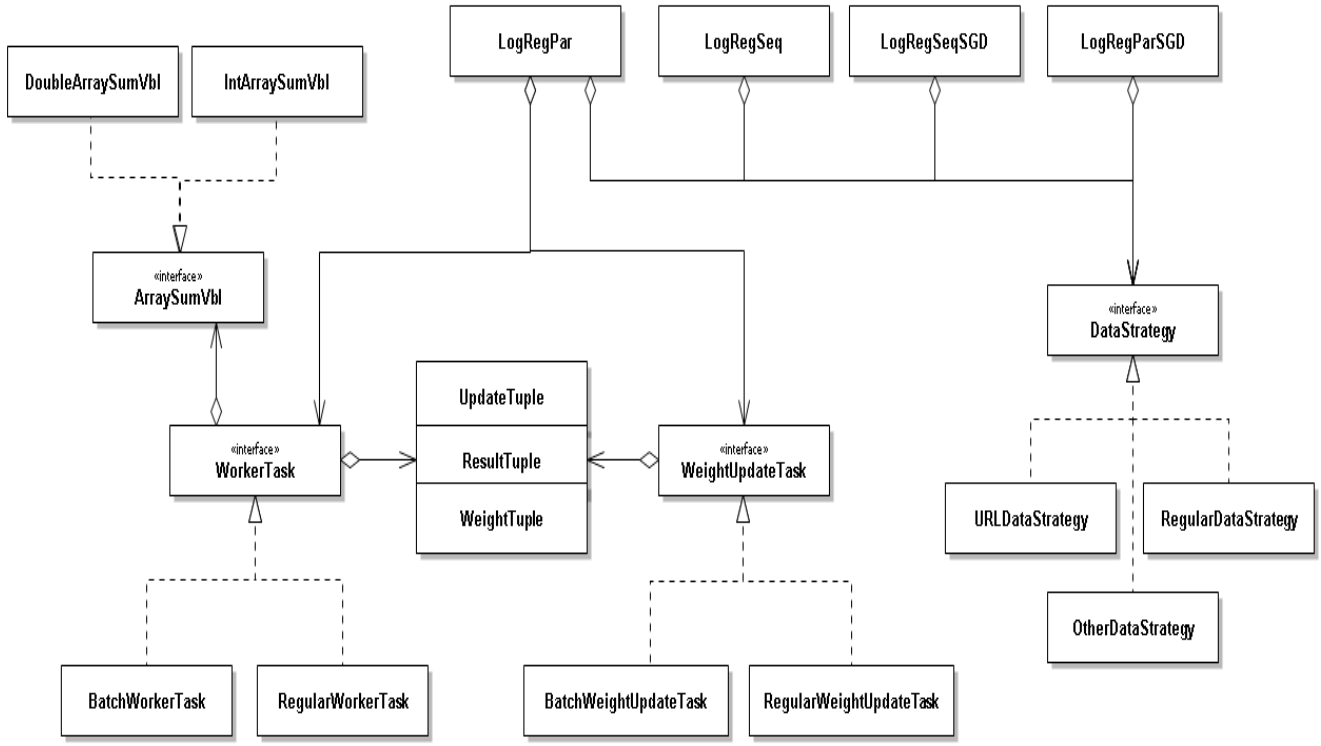
Results:

- Accurately found the true optimal feature to add on many iterations of feature selection
- Accuracy decreased as the number of irrelevant features in the dataset increased
    - Larger search space to find the true optimal
- Accuracy decreased as the number of base features increased
    - Each added feature has smaller marginal effect, making optimal one harder to find
- Feature rankings stable after looking at only 28% of the data

Uses:

- To improve our model, can use feature selection to grab only those relevant features of the 3 million inside the URL data
    - Good performance and comparable to results found in re-learning the whole model

7. Sequential program



The design of the sequential program is identical to the above equations, but implemented as a Task in the pj2 framework with required cores set to 1. It consists of nested for loops over training examples, with the inner loop calculating the summation of the error terms over the data and the outer loop updating the weight vector to take a small step in the direction of minimum cost. In the program, a regularization term is used to prevent over-fitting. This outer loop continues either until convergence, using a command-lined value for epsilon as the minimum allowed change in cost to be considered at a steady-state, or until reaching a user-defined limit on the number of iterations. Following training, the entire dataset is classified and printed as a confusion matrix, printing metrics such as time taken, accuracy, recall, and precision to the user as the results from training. Alternatively, the user can specify to use a testing set, in which case some user-defined percentage

of the data is withheld from training and used exclusively in the above procedure for accuracy calculations. This gives a more accurate depiction of how the model will perform on unseen instances, as it did not have a chance to learn the same data it is tested on. Note that there is also a batch version of the sequential algorithm, which implements the procedures outlined in Paper 2. Further, there is a sequential stochaistic gradient descent version of the algorithm that performs a weight update on each instance of data in the dataset, rather than summing over training examples for the weight update decision.

8. Parallel program

The parallel version of the program uses a Master-Slave architecture to perform the iterations of gradient descent. The worker tasks are started in cluster nodes and they initialize their weight vectors according to a shared seed value, sending the first weight vector to the Master node. The workers then sum over training examples, aggregating the error seen. This error is returned to the Master node, who sums over all nodes in the cluster to make an adjustment to the weight vector. The Master then sends this updated vector to the worker nodes for the next iteration of training. When stopping criteria is met, the same way as in the sequential version, the Master sends a special message to the workers to stop executing, at which point they perform the confusion matrix and metric steps individually. These individual results are aggregated at the Master node and printed to the command line for the user. All internode communication occurs in tuple space, with three different types of Tuples for different phases of program execution (WeightTuple from Worker to Master on each iteration, UpdateTuple from Master to Worker on each iteration, and ResultsTuple from Worker to Master upon the completion of training. Like the sequential verison, there is a mini-batch version of the algorithm that implements the ideas in Paper 2. There is also a parallel version of stochaistic gradient descent, but it is not able to run across multiple nodes. It parallelizes the weight update step of SGD with a parallel-for loop over the weight vector using all cores on a single machine, but does not allow for inter-node communication in tuple space.

9. Developer's and user's manual

1. Make sure all path and environment variables are set, to include the PJ2 distribution and your JDK 1.7 installation. As an example:

   ```
   $ export CLASSPATH=.:/var/tmp/parajava/pj2/pj2.jar
   $ export PATH=/usr/local/dcs/versions/jdk1.7.0_11_x64/bin:$PATH
   ```

2. Compile all java files with the command:

   ```
   $ javac *.java
   ```

3. If running on a cluster (such as *tardis*), create a jar file with the command:

   ```
   $ jar cf someName.jar *.class
   ```

4. Run the software using the Parallel Java 2 launcher:

   ```
   $ java pj2 [pj2 parameters] [jar=<name_of_jar>] <class> URL [program parameters]
   ```

The *pj2* parameters include debug options, the number of workers/threads to use, etc.
The optional parameters for the regression programs are different for different versions of the code

(sequential vs. parallel vs stochaistic gradient descent). They can be viewed by running the command in step 4 without any program parameters and viewing the output message on the command line.

The different main classes are:

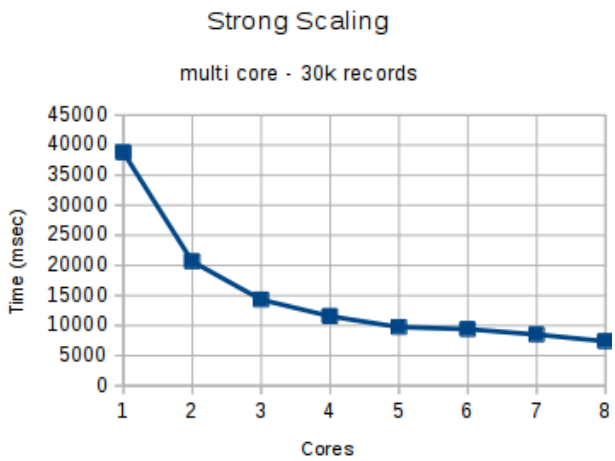
1. LogRegSeq : Logistic regression sequential version, which can be used with or without mini-batch training.
2. LogRegPar : Logistic regression parallel version, which can be used with or without mini-batch training.
3. LogRegSeqSGD: The sequential version of training using stochastic gradient descent.
4. LogRegParSGD: Stochastic gradient descent with the weight update step parallelized over cores of a single node.

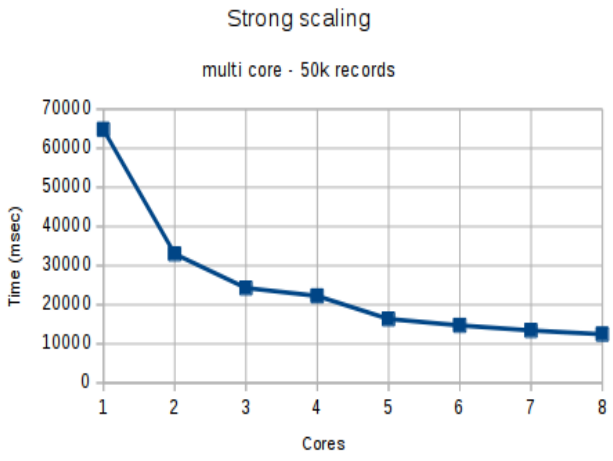
10. Strong scaling performance

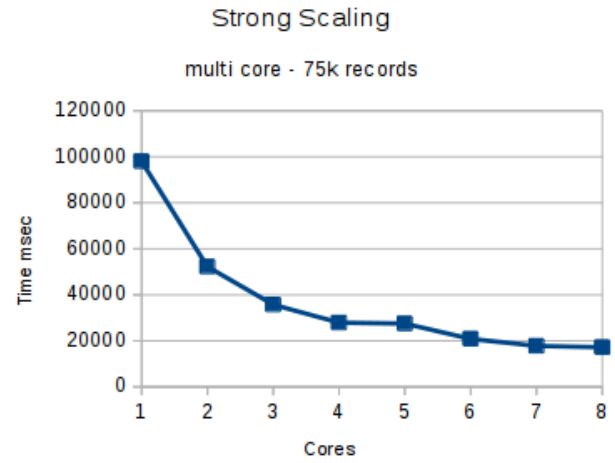
a. Measurement process

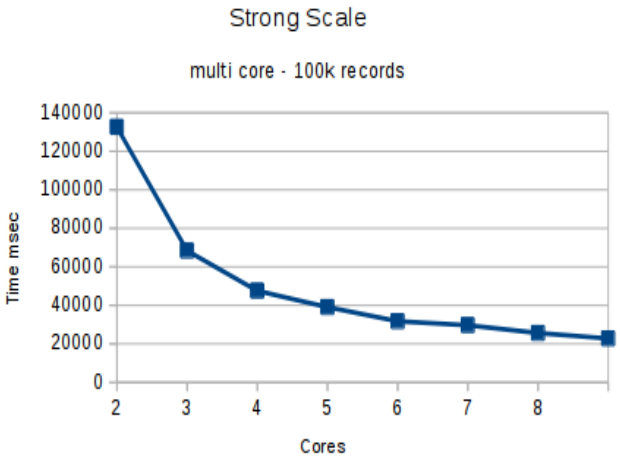
Strong scaling performance is when the amount of data being processed or the amount of computation stays the same, but the number of cores or workers is scaled up. For our experiments of logistic regression over a dataset of records, we decided that the number of records processed would be the constant factor. We measured the performance of the program for the sequential and parallel versions using the same number of data instances. The parallel version went through workers 1 through 8, with a constant number of records split between the number of nodes used. We repeated this process with a total of 5 different record counts. The graphs below show the scaling performance for five problem sizes for workers 1 through 8.

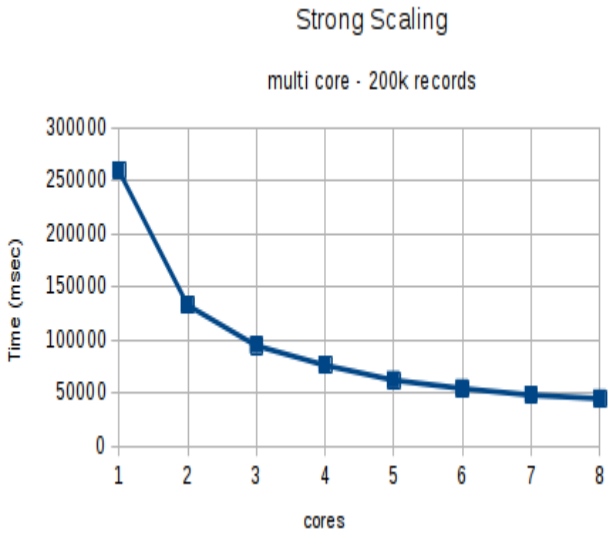
b. Graphs



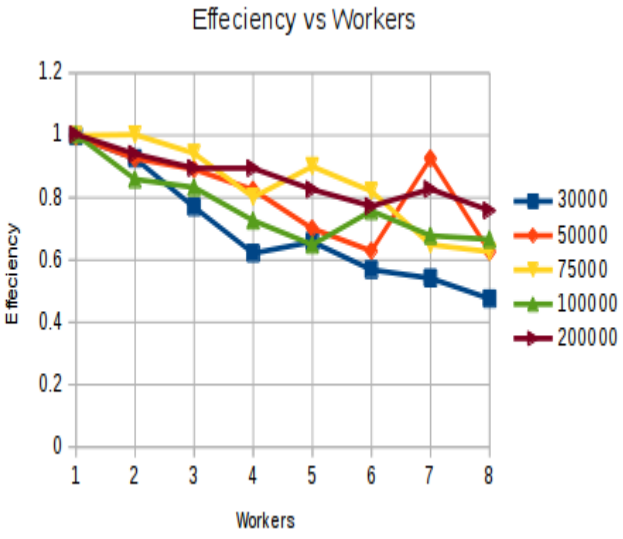


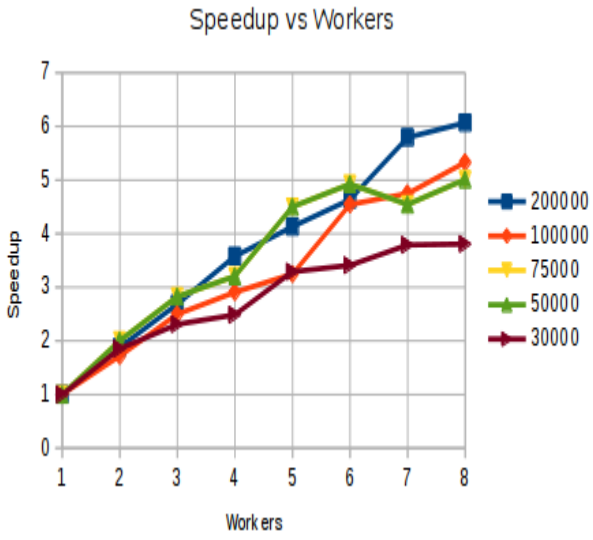






From these, we can plot the calculated speedup and efficiency as the number of workers increase:





c. Tables

Data for strong scaling over multiple cores:

Records	Cores	T msec	Accuracy	Speedup	Efficiency
30,000	Seq	38612	93.584		
	1	38786	93.584	0.9955	0.9955
	2	20662	93.584	1.8687	0.9344
	3	14321	93.584	2.6962	0.8987
	4	11548	93.584	3.3436	0.8359
	5	9758	93.584	3.9570	0.7914
	6	9395	93.584	4.1098	0.6850
	7	8513	93.584	4.5357	0.6480
	8	7392	93.584	5.2235	0.6529
50,000	Seq	64603	94.199		
	1	64784	94.199	0.9972	0.9972
	2	33006	94.199	1.9573	0.9787
	3	24278	94.199	2.6610	0.8870
	4	22240	94.199	2.9048	0.7262
	5	16363	94.199	3.9481	0.7896
	6	14727	94.199	4.3867	0.7311
	7	13427	94.199	4.8114	0.6873
	8	12459	94.199	5.1852	0.6482
75,000	Seq	98276	93.864		
	1	98015	93.864	1.0027	1.0027
	2	52262	93.864	1.8804	0.9402
	3	35736	93.864	2.7501	0.9167
	4	27894	93.864	3.5232	0.8808
	5	27482	93.864	3.5760	0.7152
	6	20835	93.864	4.7169	0.7861
	7	17785	93.864	5.5258	0.7894
	8	17219	93.864	5.7074	0.7134
100,000	Seq	132983	93.913		
	1	132542	93.913	1.0033	1.0033
	2	68394	93.913	1.9444	0.9722
	3	47581	93.913	2.7949	0.9316
	4	39115	93.913	3.3998	0.8499
	5	31743	93.913	4.1894	0.8379
	6	29698	93.913	4.4778	0.7463
	7	25653	93.913	5.1839	0.7406
	8	22831	93.913	5.8247	0.7281

Records	Cores	T msec	Accuracy	Speedup	Efficiency
200,000	Seq	259891	93.744		
	1	259533	93.744	1.0014	1.0014
	2	132759	93.744	1.9576	0.9788
	3	94492	93.744	2.7504	0.9168
	4	76079	93.744	3.4161	0.8540
	5	61960	93.744	4.1945	0.8389
	6	54107	93.744	4.8033	0.8005
	7	47914	93.744	5.4241	0.7749
	8	44620	93.744	5.8245	0.7281

Data for strong scaling over multiple nodes:

Records	Workers	T msec	Accuracy	Speedup	Efficiency
30,000	Seq	28972	94.278		
	1	29039	94.278	0.9977	0.9977
	2	15642	94.086	1.8522	0.9261
	3	12547	92.100	2.3091	0.7697
	4	11655	94.031	2.4858	0.6215
	5	8803	93.211	3.2912	0.6582
	6	8494	93.472	3.4109	0.5685
	7	7645	93.887	3.7897	0.5414
	8	7611	92.925	3.8066	0.4758
50,000	Seq	48882	93.507		
	1	49058	93.507	0.9964	0.9964
	2	26406	93.650	1.8512	0.9256
	3	18282	92.293	2.6738	0.8913
	4	14811	93.982	3.3004	0.8251
	5	13950	93.644	3.5041	0.7008
	6	12954	93.450	3.7735	0.6289
	7	7545	93.344	6.4787	0.9255
	8	9755	93.170	5.0110	0.6264
75,000	Seq	75692	91.827		
	1	75746	91.827	0.9993	0.9993
	2	37761	92.621	2.0045	1.0023
	3	26769	93.152	2.8276	0.9425
	4	23610	93.874	3.2059	0.8015

Records	Workers	T msec	Accuracy	Speedup	Efficiency
	5	16816	93.111	4.5012	0.9002
	6	15370	93.308	4.9247	0.8208
	7	16654	93.717	4.5450	0.6493
	8	15105	93.079	5.0111	0.6264
100,000	Seq	88290	93.345		
	1	88037	93.345	1.0029	1.0029
	2	51384	92.962	1.7182	0.8591
	3	35328	93.294	2.4992	0.8331
	4	30375	93.834	2.9067	0.7267
	5	27204	93.348	3.2455	0.6491
	6	19429	93.065	4.5442	0.7574
	7	18598	93.729	4.7473	0.6782
	8	16565	93.337	5.3299	0.6662
200,000	Seq	179828	92.799		
	1	179243	92.799	1.0033	1.0033
	2	95628	92.674	1.8805	0.9402
	3	66983	91.938	2.6847	0.8949
	4	50253	93.322	3.5785	0.8946
	5	43535	92.802	4.1307	0.8261
	6	38787	93.318	4.6363	0.7727
	7	31036	93.158	5.7942	0.8277
	8	29632	93.516	6.0687	0.7586

d. Non-ideal strong scaling

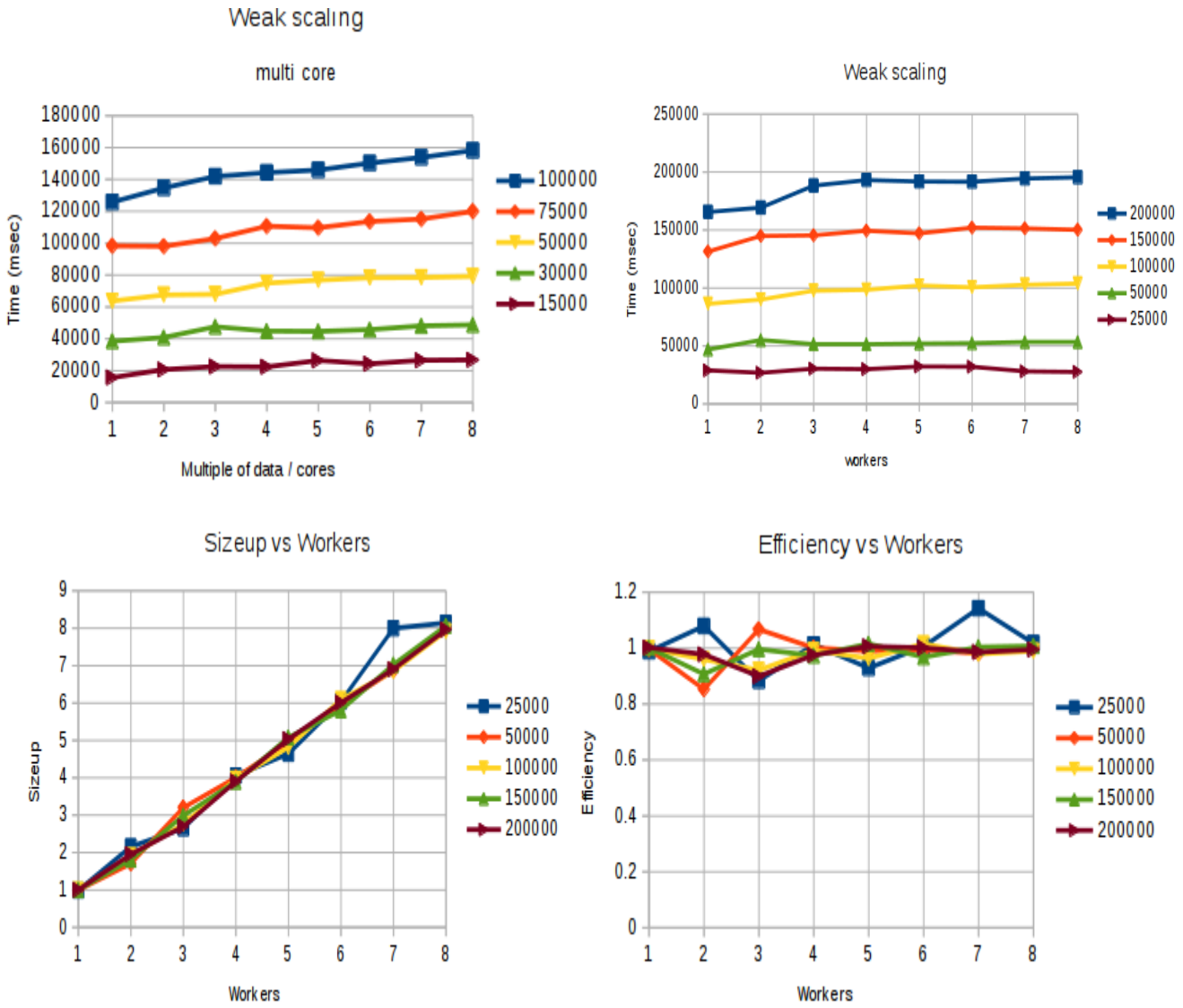
As the number of workers increased, the problem size stayed the same in strong scaling. As a consequence, each worker has a smaller chunk of the problem to work on, since each worker is given (N/W) instances to process. Despite a smaller number of instances, the amount of communication between the workers remains relatively constant (equal to the weight vectors put into tuple space). This decreases the ratio of computation to communication, which results in the non-ideal performance for strong scaling. As nodes increase, this corresponding increase in relative communication results in diminishing returns as the number of workers goes from 4 to 8. If we continued to increase the number of workers, we would expect the computation time to actually increase as inter-node communication becomes unmanageable, especially given the number of double values communicated on each iteration (equal to the number of features used).

11. Weak scaling performance

a. Measurement process

In weak scaling, the problem size increases in step with the number of workers. For our measurements, we multiplied the sequential problem size, which is the number of records, by the number of cores or workers in order to calculate the new amount of records to process. That is, if *w* workers work on *r* records, then *kw* workers will work on *kr* records. We measured the running times and efficiencies for 5 problem sizes with *k* going from 1 to 8.

b. Graphs



c. Tables

Data for weak scaling over multiple cores:

Records	Amount of Data	Cores	T msec	Accuracy	Sizeup	Efficiency
15,000		Seq	15402	94.033		
	x1	1	15675	94.033	0.9826	0.9826
	x2	2	20690	93.584	1.4888	0.7444

Records	Amount of Data	Cores	T msec	Accuracy	Sizeup	Efficiency
	x3	3	22615	93.855	2.0432	0.6811
	x4	4	22402	94.540	4.0380	1.0095
	x5	5	26394	93.864	2.9177	0.5835
	x6	6	24294	93.855	3.8039	0.6340
	x7	7	26533	93.898	4.0634	0.5805
	x8	8	26731	93.641	4.6095	0.5762
30,000		Seq	38324	93.584		
	x1	1	38484	93.584	0.9958	0.9958
	x2	2	40856	94.309	1.8761	0.9380
	x3	3	47583	93.641	2.4162	0.8054
	x4	4	44862	94.309	4.2426	1.0607
	x5	5	44676	93.754	4.2891	0.8578
	x6	6	45770	93.591	5.0239	0.8373
	x7	7	48150	93.750	5.5715	0.7959
	x8	8	48716	93.754	6.2935	0.7867
50,000		Seq	63804	94.199		
	x1	1	63719	94.199	1.0013	1.0013
	x2	2	67521	93.913	1.8899	0.9450
	x3	3	67951	93.754	2.8169	0.9390
	x4	4	74940	93.744	3.6270	0.9067
	x5	5	76833	93.323	4.1521	0.8304
	x6	6	78302	92.968	4.8891	0.8148
	x7	7	78485	93.356	5.6906	0.8129
	x8	8	79173	93.272	6.4470	0.8059
75,000		Seq	98438	93.864		
	x1	1	98305	93.864	1.0014	1.0014
	x2	2	98041	93.754	2.0081	1.0040
	x3	3	102926	93.785	2.8692	0.9564
	x4	4	110679	92.968	3.7198	0.9300
	x5	5	109692	93.270	4.4870	0.8974
	x6	6	113612	93.218	5.1986	0.8664
	x7	7	115171	93.169	5.9830	0.8547
	x8	8	120002	93.168	6.5624	0.8203
100,000		Seq	125990	93.913		
	x1	1	125807	93.913	1.0015	1.0015
	x2	2	134653	93.744	1.8713	0.9357

Records	Amount of Data	Cores	T msec	Accuracy	Sizeup	Efficiency
	x3	3	141912	92.968	2.6634	0.8878
	x4	4	144272	93.272	3.9346	0.9836
	x5	5	145942	93.174	4.3164	0.8633
	x6	6	150268	93.168	5.0306	0.8384
	x7	7	153804	93.009	5.7341	0.8192
	x8	8	158211	93.002	6.3707	0.7963

Data for weak scaling over multiple nodes:

Records	Amount of Data	Workers	T msec	Accuracy	Sizeup	Efficiency
25,000		Seq	28568	93.387		
	x1	1	28906	93.387	0.9883	0.9883
	x2	2	26793	93.419	2.1577	1.0789
	x3	3	30348	93.584	2.6486	0.8829
	x4	4	29952	93.403	4.0529	1.0132
	x5	5	32246	93.275	4.6443	0.9289
	x6	6	32058	93.466	6.0352	1.0059
	x7	7	28062	93.209	7.9968	1.1424
	x8	8	27570	93.261	8.1428	1.0178
50,000		Seq	46793	93.670		
	x1	1	46964	93.670	0.9964	0.9964
	x2	2	55050	92.942	1.7062	0.8531
	x3	3	51582	92.588	3.2017	1.0672
	x4	4	51522	92.881	4.0047	1.0012
	x5	5	52049	93.148	4.9494	0.9899
	x6	6	52391	92.600	5.9608	0.9935
	x7	7	53376	92.976	6.8708	0.9815
	x8	8	53342	92.939	8.0051	1.0006
100,000		Seq	86291	94.134		
	x1	1	86348	94.134	0.9993	0.9993
	x2	2	89994	93.673	1.9190	0.9595
	x3	3	97665	93.153	2.7644	0.9215
	x4	4	98436	92.629	3.9687	0.9922
	x5	5	102178	92.965	4.8169	0.9634
	x6	6	100589	93.209	6.0948	1.0158

Records	Amount of Data	Workers	T msec	Accuracy	Sizeup	Efficiency
	x7	7	102744	93.298	6.8532	0.9790
	x8	8	103819	93.090	7.9172	0.9896
150,000		Seq	131592	93.334		
	x1	1	131428	93.334	1.0012	1.0012
	x2	2	144903	92.788	1.8140	0.9070
	x3	3	145440	93.355	2.9889	0.9963
	x4	4	149426	92.427	3.8933	0.9733
	x5	5	147138	93.058	5.0778	1.0156
	x6	6	152049	93.043	5.8062	0.9677
	x7	7	151443	93.043	7.0280	1.0040
	x8	8	150240	92.815	8.0641	1.0080
200,000		Seq	165862	93.323		
	x1	1	165546	93.323	1.0019	1.0019
	x2	2	169329	93.830	1.9553	0.9777
	x3	3	188334	93.075	2.6973	0.8991
	x4	4	193105	92.660	3.9012	0.9753
	x5	5	191880	92.949	5.0319	1.0064
	x6	6	191667	92.546	6.0067	1.0011
	x7	7	194444	92.951	6.9000	0.9857
	x8	8	195472	92.807	7.9579	0.9947

d. Weak scaling results

As opposed to in strong scaling, in weak scaling the ratio of computation to communication remains relatively constant as workers increase. There will be slightly more Weight tuples for the Master to process on each iteration, but the amount of computation for each worker to perform remains constant so the ratio of communication to computation is relatively unchanged. We see here that the small increase in tuple communication his had negligible impact on the scaling results, and the weak scaling results are quite good for this problem domain.

11. Future work

There are a lot of parameters that can be tuned to get different results in machine learning. For instance, here we have the seed to perform random restarts in gradient descent, allowing us to chose the results with lowest cost. We can tune the learning rate to control how big a step we take, or epsilon to determine when we have reached convergence. Further paramters include lambda, or the weight given to the regularization term, the number of features and records to use, the number of iterations to train, the threshold to decide the final classification of either -1 or 1, and the percentage of testing data to withhold from training. If performing mini-batch training, then we have further parameters for the number of batches to use, the gamma cost parameter, and the number of

iterations to perform. Both gamma and alpha can be made to change over time, resulting in possibly improved convergence rates.

Another future application includes implementing Paper 3 on this dataset, which is an ideal candidate for parallel feature selection. Given that the data has about 3 million features, it becomes very difficult to use all of them in training. Some add a lot of important information to the model in helping the classification decision, while others are clearly noise. Thus, implementing a way to efficiently select the best features from among this large corpus will likely lead to large improvements in classification rates and convergence times.

12. What we learned

Tuple space does not like ~3 million features being transferred twice per iteration. For example, when using 50 features logistic regression took 13.346 seconds in 100 iterations with final results of 73% accuracy. In contrast, when using all 3,231,962 features of the dataset it took 438.815 seconds to perform 100 iterations and ended with 66% accuracy. This is almost a 3,000% increase in time required for the same number of instances.

13. Who did what?

- Robert came up with the design and wrote most of the code.
- Mandeep performed the tests, recorded results, and made the plots
- Mandeep and Umang wrote most of the report. Umang made most of the presentations.
- Robert reviewed and made improvements to the presentations and report.

14. References

- Sameer Singh, Jeremy Kubica, Scott Larsen and Daria Sorokina. 2009. Parallel Large Scale Feature Selection for Logistic Regression. In *Proceedings of 2009 Society for Industrial and Applied Mathematics* (SIAM) Data Mining, SIAM, Philadelphia, PA, USA, 1172-1183. **URL**:http://epubs.siam.org/doi/pdf/10.1137/1.9781611972795.100
- Haoruo Peng, Ding Liang, and C. Choi. Evaluating parallel logistic regression models. In *Proceedings of the 2013 IEEE International Conference on Big Data*, pp 119-126, 6-9 Oct 2013. **URL**:http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6691743&isnumber=6690588
- Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. 2014. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (KDD '14). ACM, New York, NY, USA, 661-670. **URL**:http://dl.acm.org/citation.cfm?id=2623330.2623612&coll=DL&dl=ACM&CFID=415399891&CFTOKEN=69514427
- Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. 2009. Identifying suspicious URLs: an application of large-scale online learning. In *Proceedings of the 26th Annual International Conference on Machine Learning* (ICML '09). ACM, New York, NY, USA, 681-688. **URL**:http://cseweb.ucsd.edu/~savage/papers/ICML09.pdf
- Michele Banko and Eric Brill. 2001. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics* (ACL '01). Association for Computational Linguistics, Stroudsburg, PA, USA, 26-

33.
URL: http://dl.acm.org/citation.cfm?id=1073017

- Mohammed Javeed Zaki. 1999. Parallel and Distributed Data Mining: An Introduction. In *Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD*, Mohammed Javeed Zaki and Ching-Tien Ho (Eds.). Springer-Verlag, London, UK, UK, 1-23.
  URL: http://dl.acm.org/citation.cfm?id=744383
- Andrew Ng. Machine Learning course materials, Coursera.
  URL: https://www.coursera.org/course/ml. Accessed September 10, 2014.