Multiclass Logistic Regression using Harp

Chao-Hong Chen
Indiana University
chen464@indiana.edu

Qiuwei Shou
Indiana University
qiuwshou@umail.iu.edu

1. INTRODUCTION

MLR (Multiclass Logistic Regression) is a very common problem in machine learning, and in practice we usually need to deal with very huge data sets so exploit parallelism to achieve speed up is important.

In this project, we will try to apply Harp[2] on MLR. The algorithm we use will be based on is Stochastic Gradient descent (SGD). The reference implementations are [1] and scikit-learn[3] which is a python library, we will try to use Harp to parallelize the algorithm to handle large datasets.

In section 2, we will beiefly describe the MLR problems and SGD, in section 3 we will describe our Harp version of SGD for MLR, in section 4 we will evaluate our algorithm using RCV1v2[5] and we conclude this project in section 5.

2. PROBLEM DESCRIPTION

The logistic regression is a classification method used to predict binary class labels with given a set n variables. So what we want is for given input $x_1, \dots, x_n$ and a class C,

$$h(x_1, \dots, x_n) = \begin{cases} 1, & (x_1, \dots, x_n) \in C \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

In this project we will use linear model:

$$f(\overline{x}) = \theta_0 + \sum_{i=1}^{n} x_i \cdot \theta_i = \begin{pmatrix} 1 & x_1 & \dots & x_n \end{pmatrix} \cdot \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix} \tag{2}$$

We will denote $\overline{x} = \begin{pmatrix} 1 & x_1 & \dots & x_n \end{pmatrix}$ and $\overline{\theta} = \begin{pmatrix} \theta_0 & \theta_1 & \dots & \theta_n \end{pmatrix}$ in the rest of this article.
Given $\overline{\theta}$ we can use the sigmoid function to define our predictor:

$$h_{\overline{\theta}}(\overline{x}) = \frac{1}{1 + e^{-f(\overline{x})}} = \frac{1}{1 + e^{-\overline{x} \cdot \overline{\theta}^T}} \tag{3}$$

A set of training samples containts $\overline{x_1}, \dots \overline{x_m}$ and the classified label $y_i = \begin{cases} 1, & \overline{x_i} \in C \\ 0, & \text{otherwise.} \end{cases}$, for a set of training samples we can define the cost function $J(\overline{\theta})$ as:

$$J(\overline{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} y_i \log h_{\overline{\theta}}(\overline{x_i}) + (1 - y_i) \log(1 - h_{\overline{\theta}}(\overline{x_i})) \tag{4}$$

It is obvious that if $J(\overline{\theta}) = 0$ then our predictor $h_{\overline{\theta}}(\overline{x})$ correctly predict every sample in the training set, so our goal is to find $\overline{\theta}$ to minimize $J(\overline{\theta})$.

There are multiple ways to update the parameter for minimizing the cost function, in this project we choose Stochastic Gradient descent (SGD). In SGD the $\overline{\theta}$ will be initialize randomly and then SGD will randomly choose a training sample $\overline{x_i}$ and update the parameters $\overline{\theta}$ with given learning rate α by calculating the partial derivatives:

$$\overline{\theta} := \overline{\theta} - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_{\overline{\theta}}(\overline{x_i}) - y_i) \cdot \overline{x_i} \tag{5}$$

$\overline{\theta}$ will be updated for given amount of iterations or until converge.

The MLR generalizes the logistic regression by applying it to multi-class classification. So instead of just single class C, in MLR we have $C_1, \dots, C_k$ class and any $\overline{x}$ can belongs to multiple classes. We can still use SGD to solve MLR by running SGD seperately for each class.

3. PROPOSED METHOD

We made some modification to SGD, instead of randomly choose a sample we choose it in order to guarantee deterministic result. So given the number of iterations $ITER$ the SGD becomes:

```
1: initialize $\overline{\theta}$
2: for $j = 1$ to $ITER$ do
3:     for $i = 1$ to $m$ do
4:         $\overline{\theta} := \overline{\theta} - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_{\overline{\theta}}(\overline{x_i}) - y_i) \cdot \overline{x_i}$
5:     end for
6: end for
```

Note, this can be easily modified to the original SGD by choosing i randomly for m times.

The whole flow of harpMLR is given in 1.

In our parallel SGD, we distribute the training data across each mapper, each mapper loads assigned part of the dataset at first, see figure 2.

Then we create a table which contain $\overline{\theta}$ for each class (see figure 3), and initialize the scheduler which will manage threads that perform SGD (see figure 4).

Next we distribute all $\overline{\theta}$s across map use regroup provided by Harp, after this each mapper get a set of class $C_{i_1}, \dots, C_{i_n}$ and the corresponding $\overline{\theta_{i_j}}$. For each iteration, every mapper submit job to thread scheduler to compute SGD for every the given set of class, the scheduler will assign thread to compute SGD using the training data the mapper have loaded, and update each $\overline{\theta_{i_j}}$ (see figure 5), we use ratate to passing the updated $\overline{\theta}$s to other mapper and receiving another set of

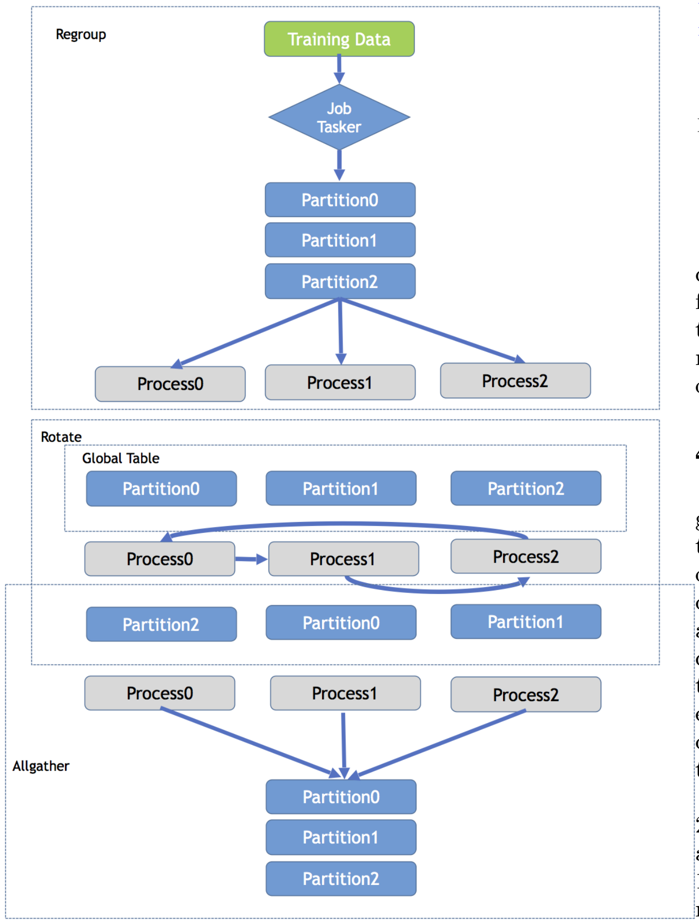

Figure 1: The whole flow of harpMLR

```
private void LoadAll(KeyValReader reader) throws
    IOException, InterruptedException {
  topics = Util.LoadTopicList(topicPath, conf);
  qrels = Util.LoadQrels(qrelsPath, conf);
  data = new ArrayList<Instance>();

  while (reader.nextKeyValue()) {
    String value = reader.getCurrentValue();
    Util.LoadData(value, conf, data);
  }
}
```

Figure 2: Load trainging data

```
private Table<DoubleArray> wTable;

private void initTable() {
  wTable = new Table(0, new DoubleArrPlus());
  for (int i = 0; i < topics.size(); ++i) {
    wTable.addPartition(new Partition(i, DoubleArray.
        create(TERM + 1, false)));
  }
}
```

Figure 3: Create tables

```
private List<GDtask> GDthread;
private DynamicScheduler<Partition, Object, GDtask> GDsch;

private void initThread() {
  GDthread = new LinkedList<>();
  for (int i = 0; i < numThread; i++) {
    GDthread.add(new GDtask(alpha, data, topics, qrels));
  }
  GDsch = new DynamicScheduler<>(GDthread);
}
```

Figure 4: Initialize scheduler

class and corresponding $\overline{\theta}$s from other thread. Repeat this for number of mapper times is equal to update all $\overline{\theta}$s for the whole training data set (see figure 6). After the given number of iterations, we use allgather to collect all $\overline{\theta}$s and output the final $\overline{\theta}$s.

4. EVALUATION

In this section we will conducting evaluation of our algorithm using RCV1v2[5]. RCV1v2 is a text categorization test collection, we will use our algorithm to classify the topic of documents. There are 103 topics in RCV1v2, and for each document there are 47236 terms. So we have 103 classes, and every instance is a 47237 dimension vector (add 1 for constant). The original training set in [5] contains 23149 training vector which is very small, since the point of this evaluation is to test the capibility of our algorithm to handle large data set, so we use the original test data as our training data which contains 781265 vectors.

We use two machines each has Intel(R) Xeon(R) CPU E5-2670 v3 with 128GB ram. Figure 7 shows the runtime of our algorithm with 2 and 4 mappers for 100 iterations with 1 to 16 local thread, figure 8 shows the speedup with different number of local thread.

According to the results, increase local thread number gives almost linear speedup, and increase local thread number is slightly faster than increase number of mappers.

We also use the training set in [5] to caculate the effectiveness of the output results, the macroaveraged $F_{1.0}$ (defined in [4]) is 0.62. Note since the training data we use is a lot larger than the test data, the purpose of this is not mean to show that our algorithm is very effective, but merely to verify that the output of our algorithm is what we expected.

5. CONCLUSION

In this project we propose a parallel version of SGD to solve MLR using Harp, and we evaluate our algorithm using RCV1v2. According to the results, we achive expected speedup from increase number of mappers and increase number of local threads.

6. REFERENCES

[1] https://github.com/tpeng/logistic-regression.
[2] Harp. http://salsaproj.indiana.edu/harp/index.html.
[3] scikit-learn. http://scikit-learn.org/.
[4] D. D. Lewis. Evaluating and optimizing autonomous text classification systems. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '95, pages 246–254, New York, NY, USA, 1995. ACM.

```java
public class GDtask implements Task<Partition, Object> {
  private double alpha;
  private HashMap<Integer, ArrayList<String>> qrels;
  private ArrayList<Instance> data;
  private ArrayList<String> topics;

  public GDtask(double A, ArrayList<Instance> D, ArrayList
      <String> T,
                HashMap<Integer, ArrayList<String>> Q) {
    qrels = Q;
    alpha = A;
    data = D;
    topics = T;
  }

  @Override
  public Object run(Partition par) throws Exception {
    String cat = topics.get(par.id());
    double[] W = ((DoubleArray)par.get()).get();
    double p;
    double label;
    Instance inst;

    for (int i = 0; i < data.size(); ++i) {
      inst = data.get(i);
      p = predict(W, inst.term);
      if (qrels.get(inst.id).contains(cat))
        label = 1.0;
      else
        label = 0.0;

     W[0] += alpha * (label - p); // constant
      for(Map.Entry<Integer, Double> entry : inst.term.
          entrySet()) {
        int key = entry.getKey();
        double value = entry.getValue();

       W[key] += alpha * (label - p) * value;
      }
    }
    return null;
  }

  private static double sigmoid(double z) {
    return 1.0 / (1.0 + Math.exp(-z));
  }

  private static double predict(double W[], HashMap<
      Integer, Double> x) {
    double res = W[0]; // constant

    for(Map.Entry<Integer, Double> entry : x.entrySet()) {
      int key = entry.getKey();
      double value = entry.getValue();

      res += W[key] * value;
    }

    return sigmoid(res);
  }
}
```

Figure 5: Gradient decent task

```java
protected void mapCollective(KeyValReader reader, Context
     context) throws IOException, InterruptedException {
  LoadAll(reader);
  initTable();
  initThread();
  regroup("MLR", "regroup_wTable",wTable,
          new Partitioner(getNumWorkers()));

  GDsch.start();
  for (int iter = 0; iter < ITER * numMapTask; ++iter) {
    // submit job
    for (Partition par : wTable.getPartitions()) {
      GDsch.submit(par);
    }
    // wait until all job completed
    while (GDsch.hasOutput()) {
      GDsch.waitForOutput();
    }

    rotate("MLR", "rotate_" + iter, wTable, null);

    context.progress();
  }
  GDsch.stop();
  allgather("MLR", "allgather_wTable", wTable);

  if (isMaster()) {
    Util.outputData(outputPath, topics, wTable, conf);
  }

  wTable.release();
}
```

Figure 6: Harp SGD

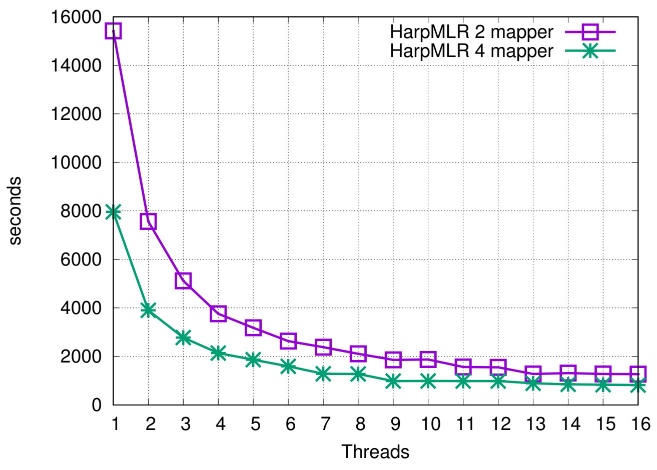

Figure 7: Runtime of HarpMLR with 2 mappers

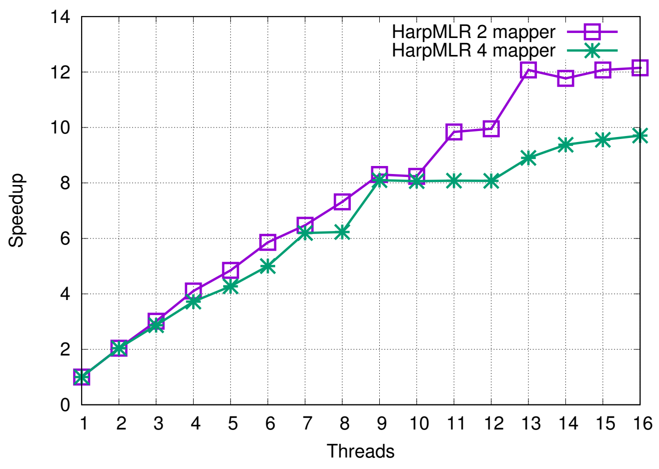

Figure 8: Speedup of HarpMLR with 2 mappers

[5] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397, Dec. 2004.

Job assignment:
Chao-Hong Chen:

1. implement harpMLR, sequential MLR
2. running experiments
3. writing report

Qiuwei Shou:

1. set up hadoop and harp
2. implement code proceess RCV1v2 (Util.java)
3. writing report
4. poster