Distributed Logistic Regression

Wu Yidi
Department of Computer Science and Engineering
the Chinese University of Hong Kong
Email: 1155047054@link.cuhk.edu.hk

***Abstract*—The Husky platform is developed mainly for in-memory large scale data mining, and also serves as a general research platform for designing efficient distributed algorithms. In this project, I developed an efficient and scalable logistic regression algorithm on Husky. This implementation outperforms the MLLib on the state of art data processing platform Spark in speed and maintains the same quality, which demonstrates good potential in practical industry usage.**

***Index Terms*—Husky, machine learning, logistic regression**

I. Introduction

The core of Husky [1] platform is written in C++. Due to relatively low cost in the programming language compared with Scala employed in Spark and careful architecture design, Husky attains a good balance in high performance and low development cost. Logistic regression can be viewed as a special case of generalized linear model. The relationship between the categorical dependent variable and one or more independent variables is estimated by computing the probabilities through a logistic function, where the linear combination of independent variables appears in the exponent term. It is widely used in binary or multinomial classification.

A. the Husky platform

The programming model of Husky includes master, workers, global and local objects, list of objects. Workers can be viewed as an abstraction of threads on computing clusters. Master is responsible for coordinating workers while workers perform the actual computations and message passing. A basic work flow of an application on Husky has the following procedure:

1) creating basic objects,
2) dividing objects to lists,
3) repeating following until condition satisfied:
    a) performing simultaneous computation,
    b) passing messages,
4) outputting result.

The task scheduling, message passing and fault handling are hidden in the core of Husky and can be used through simple interface thus enable programmers to focus on the algorithm design. Basic API of Husky is summarised by lists below

```
class BaseWorker:
    def load(url, format)
    def create_list(name)
    def globalize_list(obj_list)
    def localize_list(obj_list)
    def add_object(obj_list, obj)
    def list_execute{obj_list, mode}
```

Listing 1: Husky Worker API

```
class BaseObject:
    def partition() # return partition id
    def execute()
    # The following are used inside execute(,
    def get_msgs()  # get incoming messages
    def push(msg, id)
    def pull(id)
    def migrate(worker_id)
    def broadcast(msg, worker_id)
```

Listing 2: Husky Object API

B. Logistic regression

The logistic regression discussed in this report classifies the input into binary categories. Multiclass problem can be handled directly by generalizing the binary case.Given a set of labeled data, where every data is described as N feature-value pairs and a label valued 0 or 1, logistic regression tries to estimates the parameters as precisely as possible based on the given data. The coming new data can be labeled according to its feature-value pairs and the estimated parameters. A cost function is used to describe the precision of the parameters. More formally, the probability of label taking value 1 or 0 is given by

$$P\left(y=1|x;\theta\right)=h_\theta\left(x\right) \tag{1}$$

$$P\left(y=0|x;\theta\right)=1-h_\theta\left(x\right) \tag{2}$$

θ and x are both a vector of size N. $h_\theta\left(x\right)$ is the sigmoid function

$$h_\theta\left(x\right)=\frac{1}{1+e^{-\theta x}} \tag{3}$$

The log likelihood function thus is

$$l\left(\vec{Y}\middle|X;\theta\right)=\sum_{i=1}^{m}y^{(i)}\log h_\theta\left(x^{(i)}\right)+(1-y^{(i)})\log(1-h_\theta\left(x^{(i)}\right)) \tag{4}$$

Optimize the log likelihood with respect to θ, we can then classify new data x according to

$$y=\begin{cases}0 & \text{if } h_\theta\left(x\right)>=0.5\\ 1 & \text{else}\end{cases}$$

II. Implementation

In order to optimize log likelihood function $l\left(\vec{Y}\middle|X;\theta\right)$, a distributed gradient descent optimizer is implemented.

A. *Gradient descent*

Gradient descent works by repeatedly stepping towards the opposite direction of the gradient of the function at current point. When current point is at the local minimum, the gradient descent algorithm will not move since the gradient at local minimum is 0. The log likelihood function $l\left(\vec{Y}\middle|X;\theta\right)$ turns out to have only one local minimum as global minimum thus is suitable for using gradient descent. The updating equation for the logistic regression is given by

```
Loop {
    for i=1 to m, {
        $\theta_j := \theta_j + \alpha\left(y^{(i)} - h_\theta(x^{(i)})\right)x_j^{(i)}$    (for every $j$).
    }
}
```

B. *Distribution strategy*

Every worker first stores a stale copy of the whole θ which is used to compute the gradient. Then the parameter θ are divided to equally K parts, where K is the number of workers. After all the workers finished computing the gradient of the part that they are allocated, they first send their parts to all the other workers then using the receiving part to update the stale version of parameters.

III. Experiment result

The scalability of the distributed algorithm can be roughly measured by the linear relationship between the number of machines and the running time for the same dataset as shown in the following figure

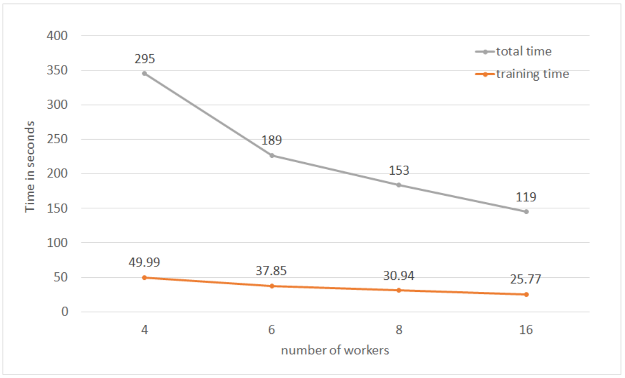

figure 1 The scalability of logistic regression.

As indicated by the figure, training time and total time both decrease nearly propotional to the increase in number of workers.
Besides of runing time, another important indicator is the convergence graph of the error rate, which is presented below. The error rate is computed by the number of records whose predicted label is different from its original label over the total number of records.

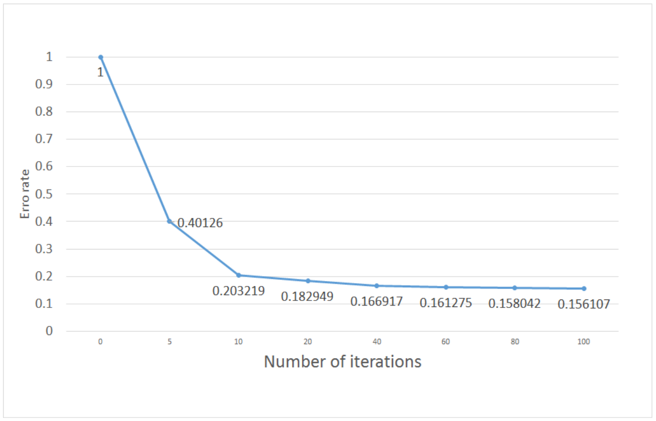

figure 2. Error rate converging as the number of iterations increases

It is assumed that the error rate at initial state is 1. After many iterations of the gradient descent algorithm, the curve drops slower and slower due to the convergence of the estimated parameters and the theoretical parameters.
Comparing the performance of logistic regression implemented on Husky with the LogisticRegressionWithSGD implemented on Spark, conclution can be drawn that the convergence time of the former is significantly shorter than the latter one. The webspam dataset contains 350000 records.Each record consists of 16609143 features. The a9 dataset is relatively small containing 32561 records and each with 123 features.

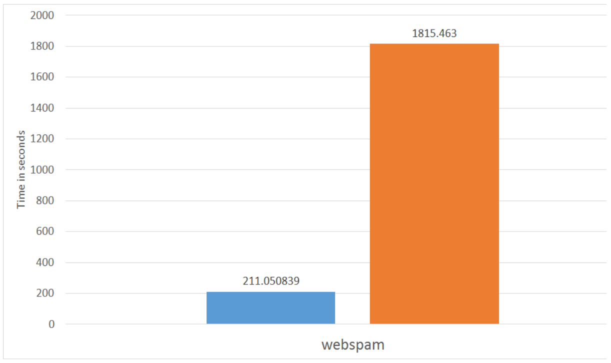

figure 3. Comparison of convergence time for webspam

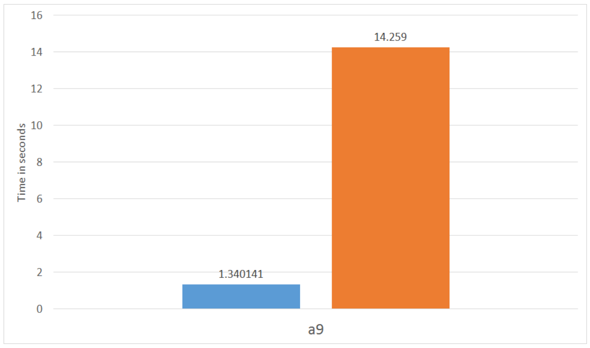

figure 4. Comparison of convergence time for a9

IV. Conclusion

In this project, a distributed logistic regression algorithm is developed on Husky platform. In terms of speed and quality, this implementation outperforms the state of art MLlib on Spark platform.

Acknowledgment

I would like to express my special thanks to Fan Yang, Yunjian, Jinfeng, Legend, Kelvin and others in lab for their guidance and generous help. I would also like to thank professor James Cheng for giving me such great chance to do the wonderful project on machine learning.

References

[1] F. Yang, J. Li, and J. Cheng. Husky: Towards a more efficient and expressive distributed computing framework. *PVLDB*, 9(5):420–431, 2016.