Report for COMP6511A

K-Nearest Neighbors Implementation on Layer Zero

Naiyan Wang, Kaixiang Mo, Zhongqi Lu
{nwangab,kxmo,zluab}@ust.hk

May 14, 2014

1 Introduction

In this project, we implement K-Nearest Neighbors (K-NN) algorithm on Layer Zero machines. This document serves as the program manual and project report. In this document, we would like to first discuss the background of the project by introducing the basic version of K-NN algorithm and the Layer Zero project. Then, we show the experimental results with our implementation of K-NN on Layer Zero machines.

2 Preliminary

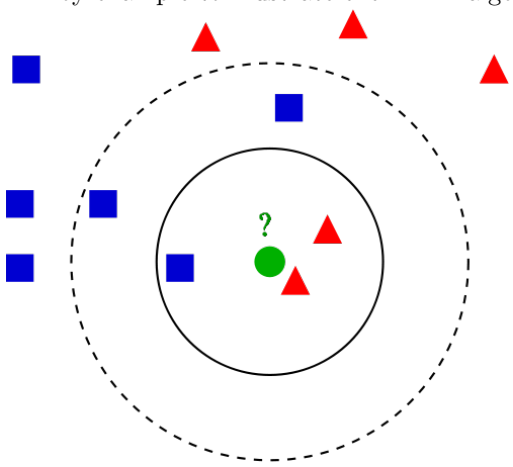
2.1 K-Nearest Neighbors Algorithm

K-Nearest Neighbors algorithm is a non-parametric method used for either regression or classification. In this work, we focus on the classification tasks. A testing sample is classified by a majority vote of its neighbors, with the sample being assigned to the class most common among its k nearest neighbors.

We present an example for K-NN algorithm. As illustrated in Figure 1, the testing samples are circled in Green. AS we discussed before, the testing samples should be classified either to the class of blue squares or to the class of red triangles, depending on the value of K. If $K = 3$ (solid line circle), it is assigned to the red triangles class because there are two triangles and only one square inside the solid line circle. However if $K = 5$ (dashed line circle), it is assigned to the squares class, because we have three squares versus two triangles inside the dashed line circle.

K-NN is a type of instance-based learning, where function is only approximated locally and all computations deferred until classification. K-NN has some nice properties. First, K-NN has some strong consistency results. A the amount of data approaches infinity, the algorithm is guaranteed to yield an error rate

Figure 1: Toy example to illustrate the K-NN algorithm.



no worse than twice the Bayes error rate, which is the minimum achievable error rate given the distribution of the data [1]. Besides, K-NN is guaranteed to approach the Bayes error rate for some value of K. Second, the K-NN algorithm is among the simplest machine learning algorithms. The major advantage of K-NN is that it is easy to implement by computing the distances from the testing sample to all known samples.

However, the easy implementation of K-NN leads to intensive computation for large training sets with many training samples. As we have discussed, in order to train on a dataset of M samples and to test with N samples, the K-NN algorithm should take $O(MN)$ time on a single thread implementation. However, we are expected to see an approximate $\frac{1}{c^2}$ drop of running time, when run the parallel implementation on a c-process machine.

2.2 Layer Zero

Layer Zero (`http://www.lazero.net/`) is a general purpose substrate for cloud computing. It forms many physical hosts in datacenters to a big virtual machine by a new ISA and provides the programmers system software for data center-scale programming [2]. The programming interface simulates single thread programming and it provides C-like API for developers.

We implement K-NN algorithm on the Layer Zero machines.

3 Experiments

3.1 Settings

In order to view the source code and run for the experiments, we first login the virtual machine:

```
ssh lsd@newton.baijia.info -p 22161
```

Our implementation of K-NN algorithm is at the following path:

```
/home/lsds/forest/l0sys/l0sys/apps/knn/L0/
```

The implementation is in file "knn.c". Before running the program, you will have to specify several parameters in "knn.c", which are self-explained by comments:

```
#define N (10000000) // training set size
#define M (100) // testing set size
#define C (3) // number of class
#define K (2) // K in K-NN
```

../code/knn.c

Besides, in this preliminary implementation, we assume each point is represented by four dimensions. The distance between two points is computed by:

```
#define distance_sq(x1, y1, z1, w1, x2, y2, z2, w2) \
(((x1) - (x2)) * ((x1) - (x2))) + \
(((y1) - (y2)) * ((y1) - (y2))) + \
(((z1) - (z2)) * ((z1) - (z2))) + \
(((w1) - (w2)) * ((w1) - (w2)))
```

../code/knn.c

After changing the "knn.c", compile at the source directory and run the Layer Zero program with sample input "knn.input":

```
make
am run -n 1 knn.bin knn.input
```

In the sample input file "knn.input", a line represents a sample and is defined as:

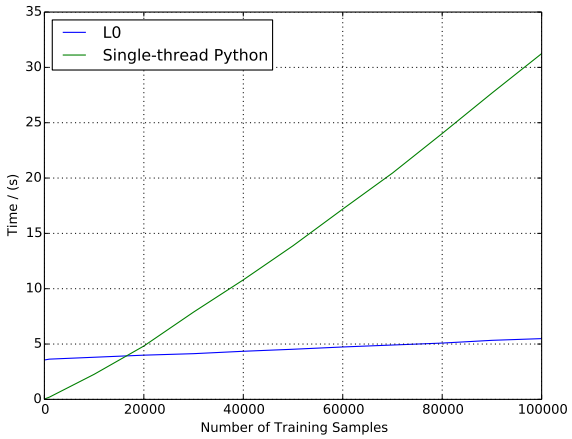
```
Dimension1 Dimension2 Dimension3 Dimension4 ClassLabel
```

where terms in the line are separated by a space, and the line ends with "\n".

3.2 Performance Comparison

In order to check the performance of the parallel K-NN on Layer Zero machines, we preset the number for testing sample to be 100 and vary the number of training samples. Ideally, we expect the running time increases linearly upon

Figure 2: Test the running time of K-NN on Layer Zero machines



the increasing of training samples, and running on larger number of nodes leads to less running time.

However, because the number of assigned virtual machines is 2, the maximum number of nodes is only 2. In our experiments, there are little differences between running the program on one node and running on two. Therefore, we only report the results obtained on 2 nodes.

In our experiments, we compare with the single thread Python implementation of K-NN. The results is shown in Figure 2. Notice that this is not a fair comparison, because the single thread Python implementation does not follow the same logic with the Layer Zero version. But the rates for running time increases should be comparable. There are several observations. First, when the number of training samples increases, the running time increases linearly. This meets our expectation of the performance. Second, when the size of training data is large, the parallel implementation outperforms the single thread Python implementation, and as the number of training samples increases, the rate of increasing running time for the single thread version is larger than the rate for parallel version.

References

[1] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[2] Zhiqiang Ma, Zhonghua Sheng, Lin Gu, Liufei Wen, and Gong Zhang. Dvm: Towards a datacenter-scale virtual machine. *ACM SIGPLAN Notices*, 47(7):39–50, 2012.