Alper Canberk Balcı

2017400087

27.01.2021

**Cmpe300 Fall 2020 MPI Project**

**Introduction**

In supervised machine learning, we have labeled input data which consist of feature values and class label for every instance. Class label is for classification of the instances, and in this project we have two values for it, 0 and 1. However, there may be several features for each instance. In the case of that there are too many features for instances, most likely there are irrelevant features. These irrelevant features decrease the performance of our machine learning algorithm, therefore we have to get rid of them. I implemented a parallel algorithm for feature selection using Relief in this project. We use parallel algorithms in case of there are too many instances, and we may use the benefits of parallel algorithms to reduce time complexity.

**Program Execution**

I wrote the program in C++14.

To compile our code:

mpic++ -o cmpe300\_mpi\_2017400087 ./cmpe300\_mpi\_2017400087.cpp

To run our code:

mpirun -oversubscribe -np <P> ./cmpe300\_mpi\_2017400087 <Input>

P is number of processes: 0 is the master, 1 to P-1 are the slaves.

Input is the path of the input file which consist of the input data in the form of:

P

N A M T

N lines of data…

P is the number of processes, N is the number of instances, A is the number of features for each instance, M is the number of iterations of the Relief Algorithm, T is the number of top features we select among the features, N lines of data are instances and each line consist of A features and the class label. They are white-space-seperated.

Output of our program when we run our code with

mpirun -oversubscribe -np 3 ./cmpe300\_mpi\_2017400087 ./input.txt

and if the input.txt consist of the data:

3

10 4 2 2

6.0 7.0 0.0 7.0 0

16.57 0.83 19.90 13.53 1

0.0 0.0 9.0 5.0 0

11.07 0.44 18.24 15.52 1

5.0 5.0 5.0 7.0 0

16.55 0.25 10.68 17.12 1

7.0 0.0 1.0 8.0 0

17.44 0.01 18.55 17.52 1

5.0 3.0 4.0 5.0 0

16.80 0.72 10.55 13.62 1

Output should be:

Slave P1 : 2 3

Slave P2 : 0 2

Master P0 : 0 2 3

Order of the slave input lines may be different sometimes due to parallelism.

**Program Structure**

In this project parallel programming is the essential part, and we were expected to implement our code in a way that master reads all the inputs, handle the inputs, and then distribute the inputs to the slaves by using MPI\_Send method. By using MPI\_Send in a for loop, I sent the instance data lines in a 2d array to each slave. I also sent the first 2 lines of input to the slaves in order to obtain M and T to use these in the Relief Algorithm.

Then, in the slaves I received these data by using MPI\_Recv method. I collected the data into an 2d array named arr. So, I was able to use the data in a convenient way. The program, then, select the most important T many features among the features. Then prints them in an ascending order to the console and imply in which slave process the features are found. Then using MPI\_Send, send the resulting top\_features array to the master who is going to receive the data by using MPI\_Recv in a for loop in order to obtain all the resulting features lists found by the slaves.

Then the master process handles these lists to eliminate duplicate features, and print them in ascending order to the console.

**Difficulties Encountered**

I was able to write the code easily, but in the end I found an extremely unfortunate bug in the end of a 3-hour search in my code. Debugging the program was hard because we use parallelism, and the data and algorithm are complex. The bug was that I did not saw the absolute value on diff function in the Relief Algorithm in the part of weight vector calculation.

**Conclusion**

All in all, the program works expectedly. Our goal in writing this program is that reducing the time complexity of the Relief algorithm by taking advantage of parallel programming. By doing so, not only we improve our machine learning algorithm performance by using Relief algorithm, but also we finish the feature selection in much more less time. By using P-1 processors, and dividing N lines of information among these, we improve our speed of algorithm in factor of N/(P-1). And it is impressive in the case of very large size of data.