**Parallel implementation of Binary Classification**

**Itai Zeilig**

Final project  
Course 10324, Parallel and Distributed Computation   
2019 Spring Semester

# Problem Definition

Given a set of N points in K-dimensional space. Each point X is marked as belonging to set A or B. Implement a Simplified Binary Classification algorithm to find a Linear Classifier. The result depends on the maximum iteration allowed and a value of the chosen parameter . The purpose of the project is to define a minimal value of  that leads to the Classifier with acceptable value of Quality of Classifier.

The data set contains at least **100000** but not more than **500000** points.

Input data

* **N** - number of points
* **K** – number of coordinates of points
* Coordinates of all points with attached value: 1 for those that belong to set A and -1 for the points that belong to set B.
* **0** – increment value of ****, **max** – maximum value of alpha(a).
* LIMIT – the maximum number of iterations.
* QC – Quality of Classifier to be reached

# **Input File format**

The first line of the file contains N K a0 amax LIMIT QC.

Next lines are coordinates of all points, one per line, with attached value 1 or -1.

For example:

200000 5 0.1 2.5 200 0.025

2.3 4. 5 6. 55 -2.3 13.3 1

76.2 -3.56 50.0 12 -0.7 -1

…

45.23 20 -167.1 98.0 -113.2 -1

# **output File format**

The output file contains following information

* The minimal value of a with q < QC and a value of q. If for every checked value of the value of q is bigger than QC – “Alpha is not found” is printed.
* Values of corresponding weights W.

# Solution

In implementation of Binary Classification, I used **MPI**, **OPEN** **MP** and **CUDA** in order to get maximum speed up that can be achieved.

* The input data set is stored in a text file in one machine, so only one process (the Master process) handles the reading.
* The master divides the dataset and send each process **number of alphas to process.**
* The master sends the input data to all other processes – by MPI.
* All processes have their set of points and know all the necessary information to start the algorithm.
* All processes begin the iterations over the alphas.
  + the iteration continues until there is no alphas left or found the right q that answer the condition: q < QC.
  + all processes calculate the new Quality of Classifier.
* The master collects all the alphas and Qualities of Classifier in order to find the minimum alpha.
* In case quality is not achieved – Proper message will be given.

# Implementation

**Step 1 – Initialization:**

-Master read all the data from the input file text and initializes array that store all the points and all the necessary data.

**Step 2 – Share the data:**

-Master broadcast with MPI\_Bcast and MPI\_send all the other processes the clusters and all the necessary data.

**Step 3- Divide data between all processes:**

-The master divides the alphas between all processes so that each process including himself will get the same number of alphas – **max** / (**0 \*** NumProcs). The process [NumProcs-1] might have an addition alpha to process in some cases.

- All processes receive the points from Master with MPI\_Recv.

**Step 4 – Update points position and start iterations**

-All process performing the same amount of iterations which is LIMIT.

**Step 5 –iterations:**

**5.1** Each process performs an iteration over the alphas.

**5.1.2Each process performs up to** LIMIT **iterations on a specific alpha until:**

* the number of iterations exceed to LIMIT.
* until no points moved in all processes.

**5.1.2.1.** Each iteration will iterate over all the given points until:

The expected result not equal to the current result- in that case the process will "train" the weights with the equation:  
W = W + [\*sign(f(P))] P

**5.1.3** Each process calculates the Quality Of Classifier after each iteration  
 in order to Check if the Quality of Classifier is reached.

* Find the number of points that are wrongly classified by CUDA and OpenMP:
* The GPU calculate and send to CPU the array gasses with size N.
* The CPU compare the gasses to the answers for each point – with openMP for reduction(+: nMis).
* Calculate a Quality of Classifier q according the formula  
  q = Nmis / N
* Check if the Quality of Classifier is reached (q is less than a given value QC).

**Step 6 – Gather all Alphas and Qualities of Classifier**

-Master Gatherthe data from all the processes (Alphas and Qualities of Classifier) by MPI\_Gather.

**Step 7- evaluate quality**

**-**The master process finds the minimum alpha that his q <QC.

**Step 8- Finish**

-If the quality did not achieve the program writes a proper message to output text file.

-if the quality the Master process write the result to output text file.

# Requirements

* The input file input.txt initially is known for one machine only. The results will be written to the file output.txt on the same machine
* The computation time of the parallel program will be faster than sequential solution.
* The set contains at least 100000 but not more than 500000 points, the number of dimensions is less than 20, the number of values of range alpha to check is less than 100, and there will be no more than 1000 iterations.

# Complexity

* **N** - number of points.
* **K** – number of coordinates of points.
* Coordinates of all points with attached value: 1 for those that belong to set A and -1 for the points that belong to set B.
* **0** – increment value of ****, **max** – maximum value of **** .
* LIMIT – the maximum number of iterations.
* QC – Quality of Classifier to be reached.

The heaviest operation in the program is evaluating the Quality of Classifier – **Step 5**.

In this step I have the function: calculate\_Perceptron:

* initWeights function: O (1). (k<20)
* while -> for O(limit)
* while -> for -> for-> O(limit\*n)
* f function: O(limit\*n\*k)
* train function: O (limit\*n\*k + k)
* calculate nMiss is parallel so it is O (1)

because each process calculates this algorithm:

O((limit\*n\*K)/numOfPro))

Assume that limit<1000, k<20 we get O(n).