SVM using CUDA

Report

**By: Hardik Thami, 19114035**

# Problem Statement

**Implement ML algorithm SVM (Support Vector Machines) in CUDA.**

# Novelty of work done

Support Vector Machines are supervised learning models for classification and regression problems. They can solve linear and non-linear problems and work well for many practical problems [1]. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points [2].

Through this project, I plan to implement the SVM in CPU and GPU using CUDA. To implement it, the focus will be to calculate a hyperplane that separates sample data set into classes to find the least parameters. Using those parameters, we will be able to find classes for new data. By implementing in GPU, we can obtain higher performance when compared to CPU implementation.

# Evaluation parameters / Data set used

To make the implementation and testing faster, the size of the input and prediction data set has been limited. Throughout the implementation the following data set was used:

* Sample Training Data Points: {(1, 7), (2, 8), (3, 8), (5, 1), (7, 3), (6, -1)}
* Sample Output class: {-1, -1, -1, 1, 1, 1}
* Sample data to be predicted: {(0, 10), (1, 3), (3, 4), (3, 5), (5, 5), (5, 6), (6, -5), (5, 8)}

While adding more data points to training set will help, to make it easier to understand and implemented the data set was not expanded.

**Methodology**

Theory

The idea of SVM is to implement a hyper plane that lies at maximum distance from all data points. The distance of a data point from hyperplane is the vector projection. This projection needs to maximize for all data points. This is calculated as parameter, w for the model. However, some data points may fall on the other side of the hyperplane even if it belongs to other class. To adjust this, we need to add another factor called bias, b. The best model will have low w and low b which means that all data points have large margin from hyper plane and lowest bias to include all data points. Hyper plane is defined as:

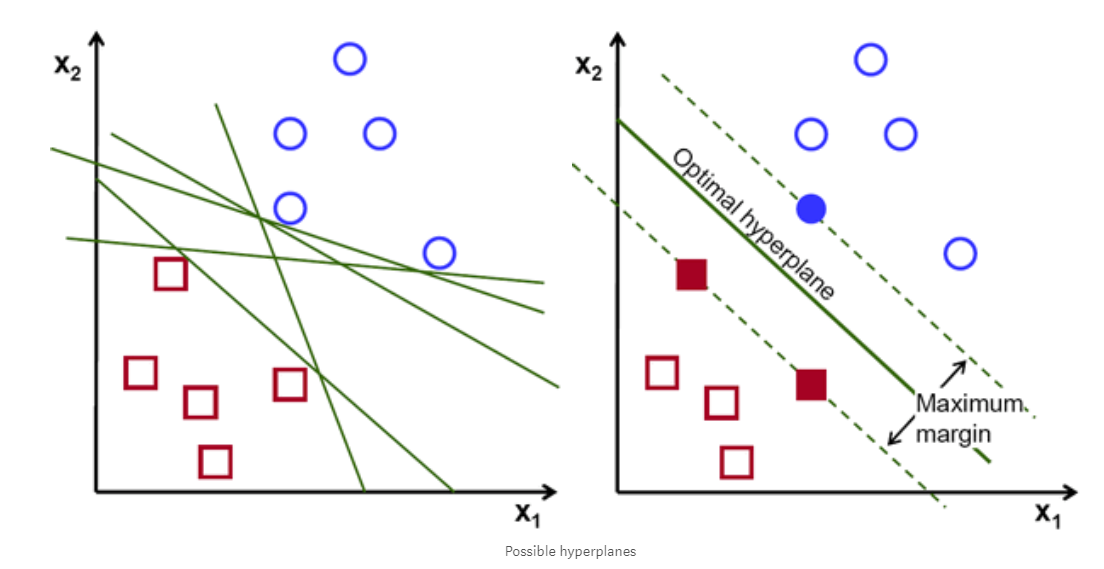
Where - class of a data point

- input data point (features)

- weights

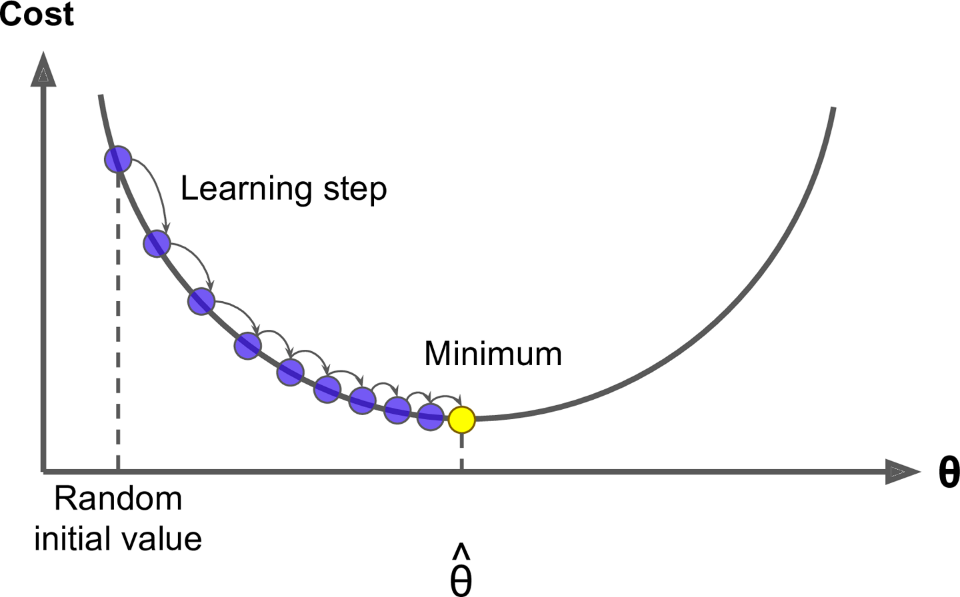
– bias

There are a lot of possible hyperplanes that satisfy the above equation as shown below. Out of these number of hyperplanes, one of them will have least magnitude for w and b.

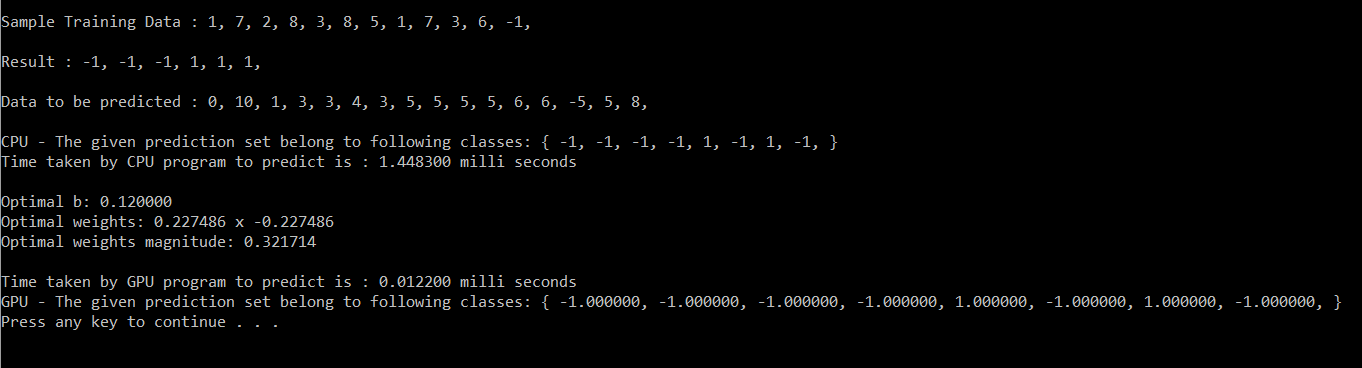


The idea for finding optimal w and b is an optimization problem which is implemented through this project in both CPU and GPU.

The idea of optimization is as shown below where x axis shows difference of from 0 and y axis shows magnitude of w. As shown below first jump will be using one learning step value and then step value will be decreased as it gets closer and closer to median value.



Once the value is calculated we can classify other data points using the value of . If this is greater than 0 then it belongs to class 1. Otherwise it belongs to class -1.

**Results/Screenshots**

**Conclusions**

GPU implementation is so much faster than CPU. GPU completes simulation in micro seconds while CPU completes in milliseconds. This project also gave an overview of how fast this algorithm works on GPU and how it can be used for regression as well as multi class classification problem.

**References**

[1] Medium - <https://medium.com/@LSchultebraucks/introduction-to-support-vector-machines-9f8161ae2fcb>

[2] towardsdatascience.com -<https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>