Course Project (EE694)

Topic: Parallelize SVM Training algorithm

Group No: 33

Group Members:

1. Vatsal Sinha (180108053)
2. Shishir Mishra (180108054)
3. Chandrakanta Mohapatra (180108055)

SVM Algorithm:

1. For all xi in training Data:

```
xi.w + b <= -1   if yi = -1 (belongs to -ve class)
xi.w + b >= +1   if yi = +1 (belongs to +ve class)
                           or
                  yi(xi.w+b) >= 1
```

2. For all support vectors(SV) (data points which decides margin)

```
xi.w+b = -1    here xi is -ve SV and yi is -1
xi.w+b = +1    here xi is +ve SV and yi is +1
```

3. For decision Boundary `yi(xi.w+b)=0` here xi belongs to point in decision boundary.

4. Our Objective is to maximize Width W `W = ((X+ - X-).w)/|w|` or we can say minimize |w|.

5. Once we have found optimized w and b using algorithm

    `x.w+b = 1` is line passing through +ve support vectors

    `x.w+b = -1` is line passing through -ve support vectors

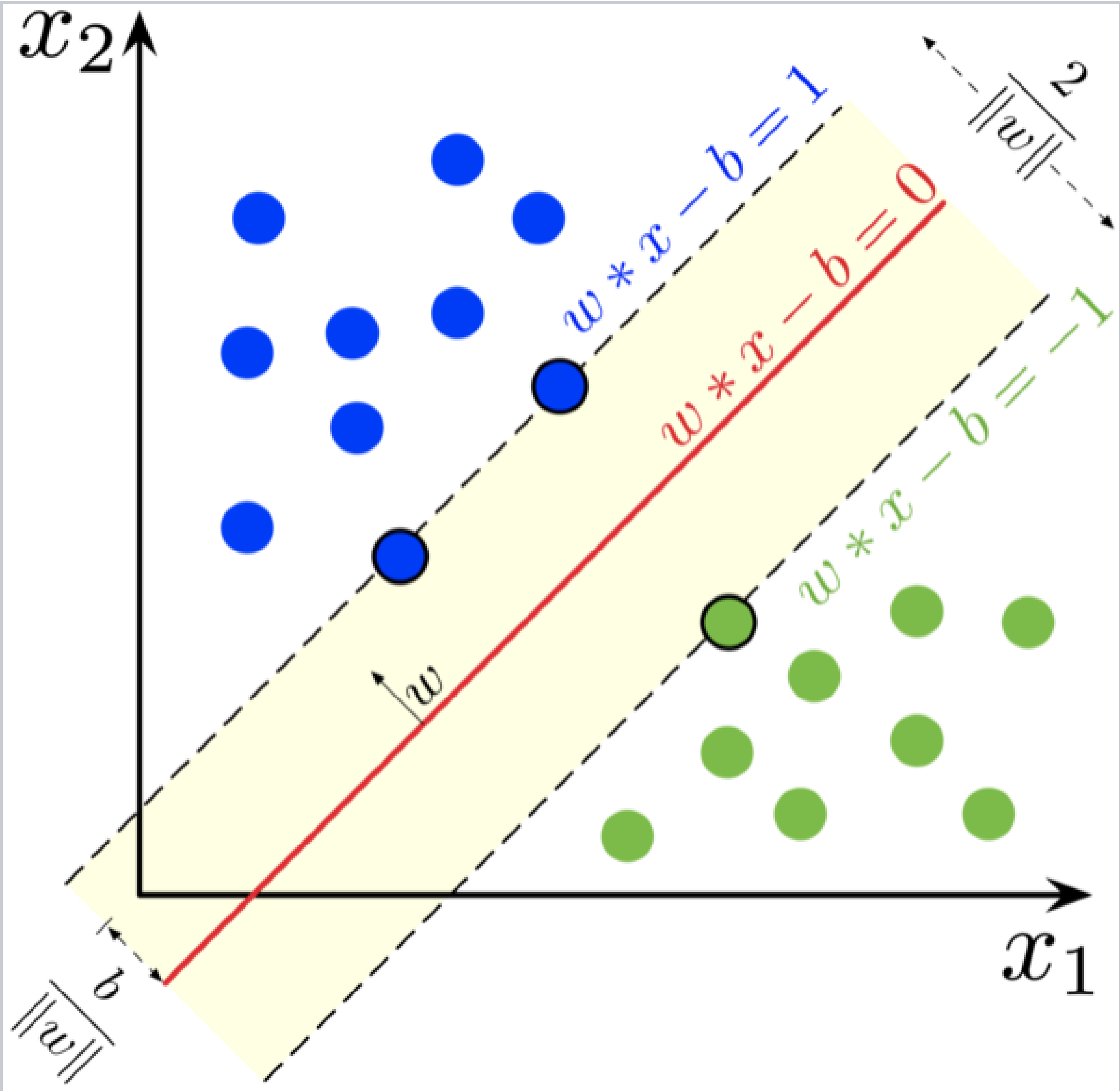
    `x.w+b = 0` is decision boundary

6. It is not necessary that support vector lines always pass through support vectors.

7. It is a Convex Optimization problem and will always lead to a global minimum.

8. This is Linear SVM means kernel is linear.

<u>Graphical representation:</u>



Algorithm in Code:

We try to fit a linear line that can separate the two classes and make the distances from either class is maximized.

The line is represented with w*x + b = 0

For label 1, w*x + b ≥ 1

For label -1, w*x + b ≤ -1

The distances of the gap in-between two classes is 2/|w|

We want to maximize this distance.

If we want to find a line that perfectly separates the two classes, we call this type of SVM cost function as Hard-Margin cost function.

If we allow some of the outliers to be misclassified, we can use Hinge-Loss when designing the cost function.

$$\max\left(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b)\right)$$

Hinge Loss cost function

Result:

1. Serial implementation of SVM

```
y = -0.411807*x1 + -0.143162*x2 + -0.176387
Time elapsed to sequentially train a SVM is = 2.391562 seconds.
```

2. Parallel implementation of SVM (using OpenMP)

```
y = -0.417335*x1 + -0.154361*x2 + -0.165331
Time elapsed to parallely train a SVM is = 1.502451 seconds.
```

3. Parallel implementation of SVM (using Pthreads)

```
y = -0.411807*x1 + -0.143162*x2 + -0.176387
Time elapsed to train a SVM is = 0.923741 seconds with pthreads.
```

Conclusion:

From the results we infer that, least time is taken by Pthread's implementation, and most time is taken by serial implementation.