# Technical Document – Introduction to Support Vector Machine

## Abstract

This document serves as the consolidation of the research done on the mathematics and algorithms behind training a Support Vector Machine, and guides the implementations in the Jacobi Library.

Firstly, the main idea and the mathematics behind SVMs are explored. Then it would introduce the Sequential Minimal Optimization algorithm for training the SVM, and several implementation details. Next the text would look at the stochastic approximation using PEGASOS algorithm. Finally, it would include some original research on which method to use and attempt to mix the algorithms for better accuracy and efficiency.

## Problem Formulation

Given a set of data, the support machine is a tuple that it follows the classification rule:

Furthermore, also minimizes the Hinge-Loss function with maximal margin:

The exact definition may varies from text to text by some scaling factors. This is the form with least complication.

In essence, it is to separate the data by the linear hyperplane. The margin of the hyperplane is given by, thus with the norm also minimized a maximal margin is achieved. Some text may generalize to a Kernel function, but it is equivalent to use the kernel trick to include non-linear terms. This derivation only works for simple dot product.

Since the function is non-differentiable, slack variables are introduced to formulate it as an optimization problem:

When, therefore it would have the same effect with the function. C is a control parameter to balance between maximizing the margin and minimizing the error. A smaller C leads to a greater margin.

Applying the Lagrange Multipliers, the Lagrangian function can be obtained:

For to be a stationary point,

Together with other KKT Conditions:

* Primal feasibility
* Complementarity
* Dual feasibility

These forms the necessary and sufficient conditions for a global optima.

Substitute into the Lagrangian,

When,

When,

When,

Combining the above equations yields:

If a set of Lagrange multipliers satisfies the above conditions, a solution is found.

## Sequential Minimal Optimization

The Sequential Minimal Optimization (SMO) algorithm is an algorithm to solve for the solution of the Lagrangian optimization problem stated above without involving Quadratic Programming algorithms, which are notoriously difficult to comprehend, let alone implement.

The main idea is optimize the Lagrangian with a pair of at each step, and iterates until all satisfies the KKT conditions.

Recall

In each step only is updated, since,

Therefore

Let,

, where

Consider the quadratic form

When,

Since L is a quadratic function on, let be the linear coefficient,

Therefore, setting,

, where is the current value of that yields the constant.

Since,

### Update bounds

Recall

Let be the sequence of updates.

If

Therefore.

If

Therefore.

WLOG, both are bounded by the limit. If the update causes to go beyond the limits, the values are simply truncated.

### Incremental update

Let be the changes of the term

, where

Recall

Let

The bias term should be adjusted such that the KKT is satisfied for.

Recall

If, then.

If both, the average of calculated using the above formula is used.

### Implementation details

The pseudocode of the algorithm is as follows:

|  |
| --- |
| 1. Initialize all as zeroes. 2. Select an that violates the KKT conditions 3. Select an 4. Update and using the above rules 5. Iterates until all satisfies the KKT conditions |

Recall

In plain language, this means for instances within the same class, change in must result in opposite change in. In the beginning of the algorithm, all are zeroes. Since cannot be negative, no can be update with an that is of the same class. In general, if, it cannot be updated with an instances that is of the same class.

Similar rule also applies to instances that are of different classes, however the suitability depends on and, in which if computed, one would get the delta value very easily. Since it doesn’t save much resources, it is not a good heuristics in picking which multiplier to optimize.

### Heuristics

A direct observation on the SMO algorithm is that it might goes through all pairs of instances, assuming it would not revisit, it still takes time, which for medium to large datasets it would be prohibitively slow. This section discuss optimizing SMO within the algorithm itself.

One may notice since, when, it could only move in 1 direction (positive or negative). Also it is easier to satisfy the KKT condition when, since it would need the instance to lie exactly on the margin otherwise. Instances that are within bound, i.e., are more likely to give greater optimization. Thus these instances are favoured to be chosen.

## Primal Estimated sub-GrAdient SOlver algorithm for SVM

The SMO algorithm can find a global optima with relative ease (comparing to Quadratic Programming), but it is still far from efficient. It guarantees to converge but does not guarantee the converge rate. It makes no promise that a pair of multipliers would not be re-visited after an iteration, thus it may be worse the quadratic time.

Here it is represented another algorithm to find an approximate solution to the SVM problem.

Recall the problem in the primal form:

Notice the formulation is slightly different from the one in SMO in which the regulation strength coefficient is moved to the norm term, and the error is averaged. This is to follow the original paper and the two forms are equivalent with designed C and λ.

H is not differentiable but if one ignores this fact and assign a zero value as derivative, its sub-gradient can be obtained.

And the problem can be solved using Stochastic Gradient Descent.

PEGASOS algorithm uses a learning rate that relates to λ and decreases with time, which is the number of iteration taken, and uses a fixed number of iteration. For each iteration, an instance is randomly picked and update the normal vector as follows:

, where

### General Kernel

Unlike deriving the Lagrangian, stochastic gradient descent doesn’t required the inner product to be the vector dot product. Thus it can be used with any general inner product function:

Thus the sub-gradient becomes

Similarly,

### Independent Bias

The *b* term in the formulation is commonly known as the bias term. Here it is formulated to be inherently negative to resemble the hyperplane equation:

, where is a point that lies on the hyperplane.

If one inspect the derivative w.r.t. the bias term:

, which is completely independent of.

In practice, this causes *b* to fluctuate in a pace far different than. This symptom is not stated in the original paper, only the remedies:

1. Treat *b* as the same as one of the components of. This is equivalent to using the Kernel trick and add a -1 to the feature vectors. However, it slightly changes the problem formulation that the value of *b* is also included in the norm term. But it seems to work well in practice.
2. Incorporate the bias term inherently, which is what was done in the above. The problem still stands.
3. Execute the iterations in batches (called mini-batches in the original paper), and solve for the optimal bias term after batch update.
4. Search for the bias term globally using binary search.

All remedies other than 1 poses extra complexity to the original simple algorithm, while not being clever or providing strong mathematical proof on how and why it works. (Binary search the bias term seems particularly a brute-force and inefficient method, considering training with a single chosen bias requires many iteration already. Also, other methods doesn’t provide how would this work or why would the bias term requires such special treatment.) In this text, it is recommended that the 1st approach is to be adapted.