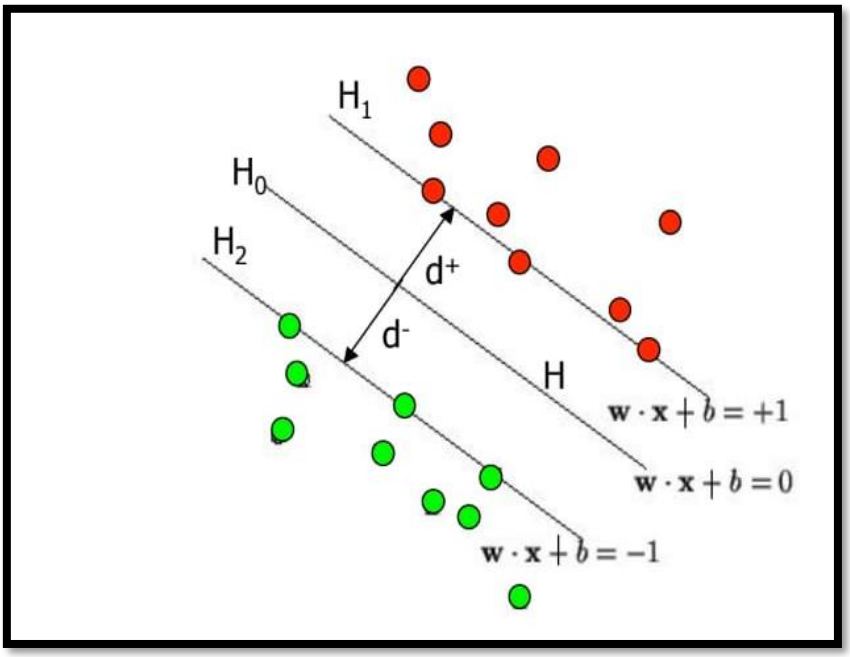
Decoding SVM

- Now we want to explore the maths and logic running behind the SVM which appears like a magic. We will decode it end to end. Let's get started:

  First, we have 'm' number of feature vector in 'n' dimensional plane. So, we plot the m vectors in the n dimensional plane and we would be having m number of vector points.

- Now our objective is to find such a hyperplane that would:

  1) Correctly classifies the classes of data points.
  2) Ensures the maximum distance between the closest data point of each class and the boundary which is also known as 'margin'.



<u>So, we want the equation of the hyperplane ensuring above two conditions.</u>

As, equation of line in 2D is (Slope-Intercept form):

$$\boldsymbol{y = mx + c}$$

Hyperplane is generalization is the analogy of line in 2D. So, its equation will be:

$$\boldsymbol{w1 * x1 + w2 * x2 + w3 * x3 + \cdots + b = 0}$$

$$\boldsymbol{\Rightarrow wx + b = 0}$$

Where,

$w = [w1, w2, w3 \ldots, wn] (Itdeterminestheorientationofthehyperplane)$
$b = \ intercept\ which\ determines\ the\ position\ of\ hyperplane$

The w is always perpendicular to the hyperplane. To prove it:

- We consider two feature points of the hyperplane which can't be traced in the figure (Just Imagine it). So:

$$w \cdot x1 + b = 0$$

and

$$w \cdot x2 + b = 0$$

$$\Rightarrow w \cdot (x2 - x1) = 0$$

Since the vector $(x2 - x1)$ will lie in the hyperplane and its dot product with the w is zero. This proves that they are perpendicular.

Here we are considering that we have Hard-Margin means we strictly considering that there is no point existing between the wall and the boundary and in the wrong side.

Now we assign labels to each class i.e.:

For red= y_r= +1

For green= y_g= -1

- So, as per the **hard margin consideration** the **red points** will have **label= +1** and obey $\boldsymbol{w \cdot x + b \geq 1}$ and for green points will have label= -1 and obey $\boldsymbol{w \cdot x + b <= -1}$.

Therefore, we can write the combined form of the conditions in a single inequality: $\boldsymbol{y_i * (w \cdot x_i + b) \geq 1}$.

Now our interest is to find the hyperplane more specifically equation of the hyperplane achieving my objective.

Here the boundary we have considered are $\boldsymbol{w \cdot x + b = 1}$ and $\boldsymbol{w \cdot x + b = -1}$.

Let the margin to be 'd'.

Now the distance of a point $(x1, y1)$ from a line $ax + by + c = 0$ is:

$$\text{Distance} = mod\left(\frac{ax1+by1+c}{\sqrt{a^2+b^2}}\right)$$

Similarly, the distance between the planes H_1 and H_0 is:

$d = \frac{w \cdot x1+b}{\sqrt{w^2}} = \frac{1}{mod(w)}$ as the point lies on H_1 and satisfies $w \cdot x1 + b = 1$.

So, the total distance between the H_0 and H_1 is:

$2d = \frac{2}{mod(w)}$ = distance between the two hyperplanes.

Now we need to calculate w and b such that the 2d distance is maximum as well as it is in between the hyperplanes to correctly classify the classes.

So, we have to maximize $\frac{2}{mod(w)}$.

$\Rightarrow$ We need to minimize $mod(w)$ and for mathematical convenience we would be minimizing the $f(w) = \frac{1}{2} * mod(w)^2$ (as its derivative will be just w!)

But the w is subjected to the following constraint:

$$\boldsymbol{y_i * (w \cdot x_i + b) \geq 1}$$

So, we can't consider any value of the w. It must be such a value which will minimize the f(w) as well as satisfy the constraint.

It's a messy problem to solve with inequality. To remove inequality, we go with Lagrange Multiplier Method.

$$L = \frac{1}{2}w^2 - \sum_1^m a_i * [y_i * (w \cdot x_i + b) - 1]$$

Now we need to minimize the L wrt w and b and maximize wrt a_i. As:

1) We want L to be minimum and the a_i are positive so the L must be maximized wrt a_i.
2) The L must be minimized wrt w and b as we want the maximum value of the margin.

So, implementing the minimization of the L wrt w and b:

$$\frac{\partial L}{\partial \vec{w}} = \frac{1}{2} * \frac{\partial(\vec{w} \cdot \vec{w})}{\partial \vec{w}} - \sum_1^m a_i * y_i * \left[\frac{\partial(\vec{w} \cdot \vec{x_i})}{\partial \vec{w}}\right]$$

$$= \vec{w} - \sum_1^m a_i * y_i * \vec{x_i}$$

Now computing the $\frac{\partial L}{\partial \vec{w}} = 0$, we get:

$$\vec{w} = \sum_1^m a_i * y_i * \vec{x_i}$$

As, for above step we can extend justification as:

$$\vec{w} = [w1, w2, \ldots, w_n]$$

$$\vec{w} \cdot \vec{w} = [w1^2, w2^2, w3^2, \ldots, w_n{}^2]$$

So, $$\frac{\partial L}{\partial \vec{w}} = \left[\frac{\partial(w1)^2}{\partial w1}, \frac{\partial(w2)^2}{\partial w2}, \frac{\partial(w3)^2}{\partial w3}, \ldots, \frac{\partial(w_n)^2}{\partial w_n}\right]$$

$$= \frac{1}{2} * [w1, w2, \ldots, w_n]$$

$$= \frac{1}{2}\vec{w}$$

Similarly, for partial derivative of x_i dot product with w vector wrt w vector will give x_i.

Now, minimizing wrt b, we get:

$$\frac{\partial L}{\partial b} = 0 - \sum_1^m a_i y_i$$

On equating $\frac{\partial L}{\partial b} = 0, then\ \sum_1^m a_i y_i = 0.$

So, we got a constraint here that $\sum_1^m a_i y_i = 0.$

Reconstructing the original L:

$$L = \frac{1}{2} * \left(\sum_1^m a_i * y_i * \overrightarrow{x_i} \cdot \sum_1^m a_j * y_j * \overrightarrow{x_j} \right) - \sum_1^m a_i * \left[y_i * \left(\sum_1^m (a_j * y_j * \overrightarrow{x_j}) \cdot x_i + b \right) - 1 \right]$$

$$\Rightarrow L = \sum_1^m a_i - \sum_1^m a_i * y_i * b - \frac{1}{2} \sum_1^m a_i a_j y_i y_j \overrightarrow{x_i} \cdot \overrightarrow{x_j}$$

$$\Rightarrow L = \sum_1^m a_i - \frac{1}{2} \sum_1^m a_i a_j y_i y_j \overrightarrow{x_i} \cdot \overrightarrow{x_j}$$

Now, we need to maximize it wrt a_i and the equation is subjected to following two constraints:

1) $\sum_1^m a_i y_i = 0$
2) $0 \leq a_i$

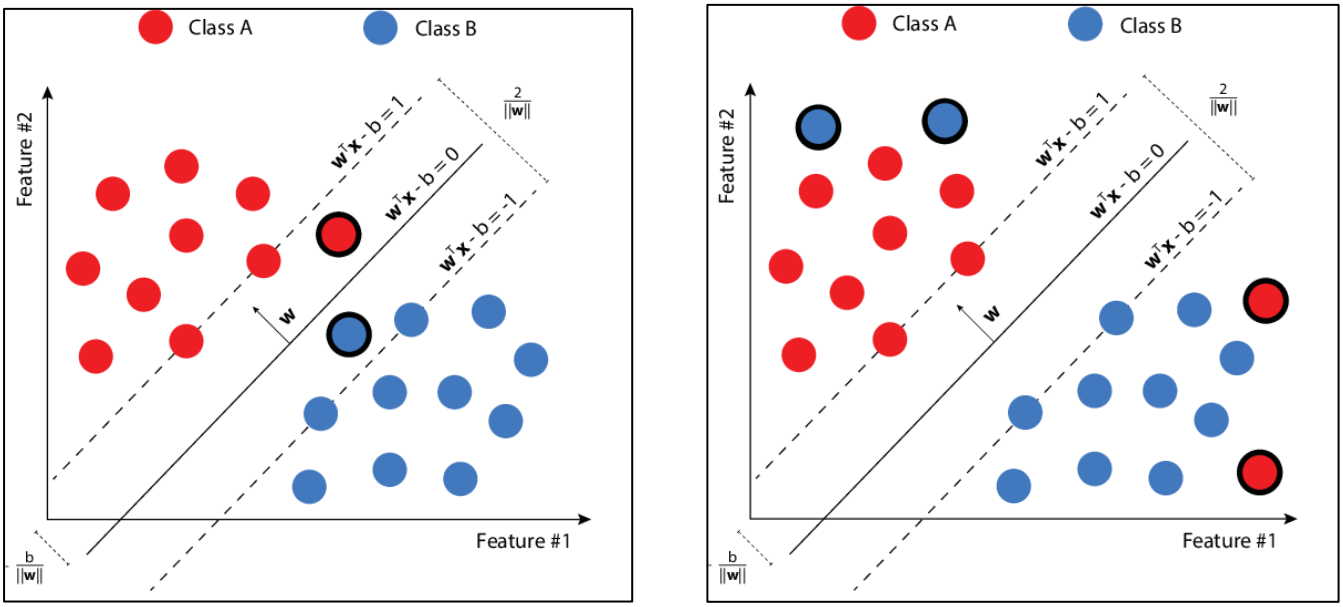
- The above dual optimization problem is solved by the optimizer to obtain the best possible values of a_i and a_j until it finds the best values that maximize the Lagrangian under constraints.

  So, once the a_i and a_j are calculated for the dual optimization problem then w can be calculated very much and then the hyperplane equation can be easily obtained which we want very much.

- Now, the question arises is <u>will we get the data points in our real-life dataset possessing a hard margin</u>?

Certainly not as real-world datasets are very messy. Its very rear to have.

<u>In real world we certainly have these situations:</u>



Now, we need to accommodate this situation:

- The equation of the plane will remain the same very much.
- We need to identify all the points even which are within the margin from hyperplane and on the wrong side. For it, we construct the inequality:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

where, the values of:

ξ=0 for points present in the right location.
0<ξ<1 for points present within the margin and correctly classified.
ξ >1 for the points on the wrong side of the plane.

Now, we want to punish all the points who are not present in the right location. So, our objective function becomes:

$$Objective\_Function = \frac{1}{2}w^2 + C_i * \sum_{1}^{m} \xi_i$$

Now, we want two things as our objective:

1) To minimize the penalty for the losses.
2) To maximize the margin.

But the Objective_Function is subjected to the following constraints:

1) $y_i(w \cdot x_i + b) \geq 1 - \xi_i$
2) $0 \leq \xi \leq 1$

So, the Lagrangian will be:

$$L = \frac{1}{2}w^2 + C_i \sum_{1}^{m} \xi_i - \sum_{1}^{m} a_i * (y_i(w \cdot x_i + b) + \xi_i - 1) - \sum_{1}^{m} a_j * \xi_i$$

Now, we need to:

1) Minimize the Lagrangian wrt w, b, ξ_i.
2) Maximize the Lagrangian wrt a_i and a_j.

So,

$$\frac{\partial L}{\partial w} = \vec{w} - \sum_{1}^{m} a_i y_i \vec{x_i}$$

$$\Longrightarrow \vec{w} = \sum_{1}^{m} a_i y_i \vec{x_i} \ when \ \frac{\partial L}{\partial w} \ is \ equated \ with \ 0$$

Also,

$$\frac{\partial L}{\partial b} = -\sum_{1}^{m} a_i y_i$$

$$\Rightarrow \sum_{1}^{m} a_i y_i = 0 \ when \ \frac{\partial L}{\partial b} \ is \ equated \ with \ 0$$

And,

$$\frac{\partial L}{\partial \xi} = \sum_{1}^{m} C_i - a_j - a_i$$

$$\Rightarrow \sum_{1}^{m} C_i = \sum_{1}^{m} a_i + a_j \ when \ \frac{\partial L}{\partial \xi} \ is \ equated \ with \ 0$$

As, ai and aj are greater than zero so we can easily conclude that:

$0 <= a_i <= C_i$ and $0 <= a_j <= C_j$

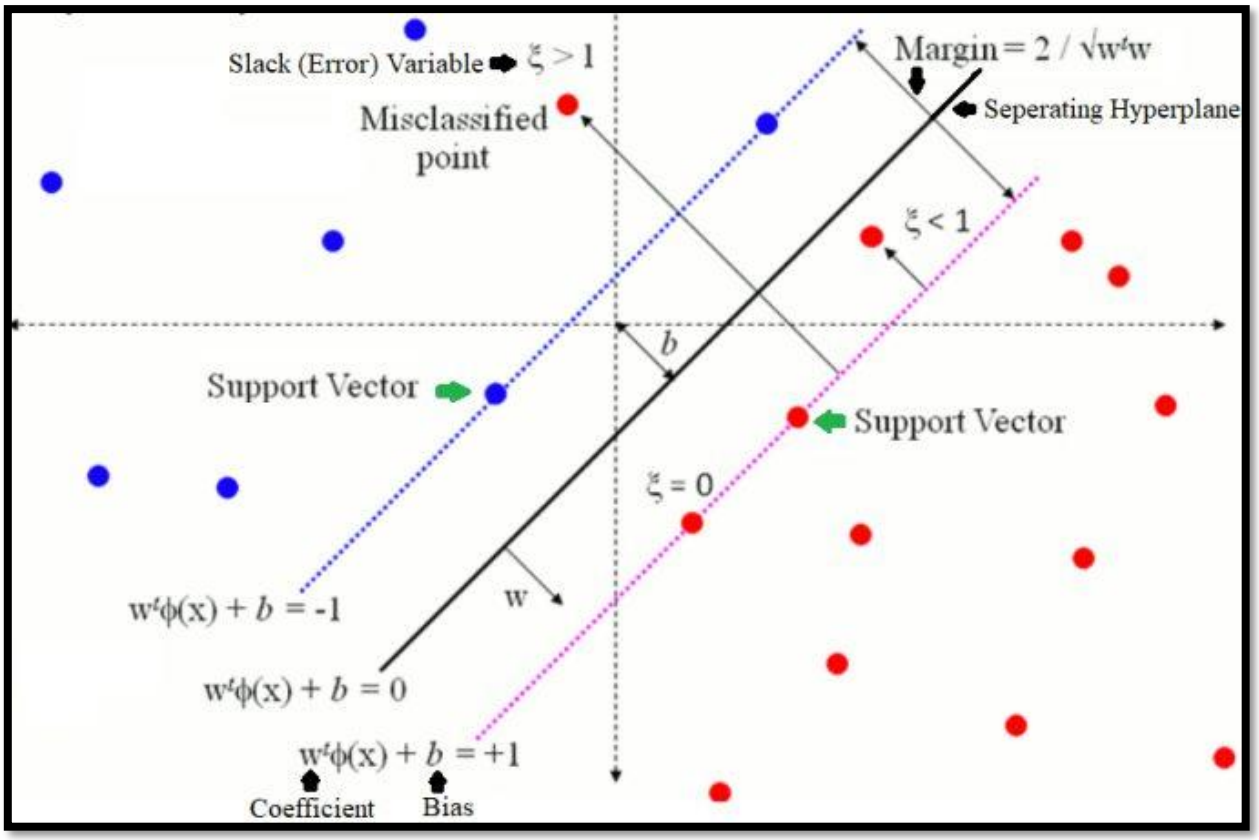
- Now, we are only remained with the multiplier finding which is too computed by the optimizer in the same way as done in Hard-Margin as the dual form is the very much same for them.
- And using support vectors we would be able to compute the value of b as datapoints are known to us so we would be easily able to do it.

So, this is how the SVM actually works.

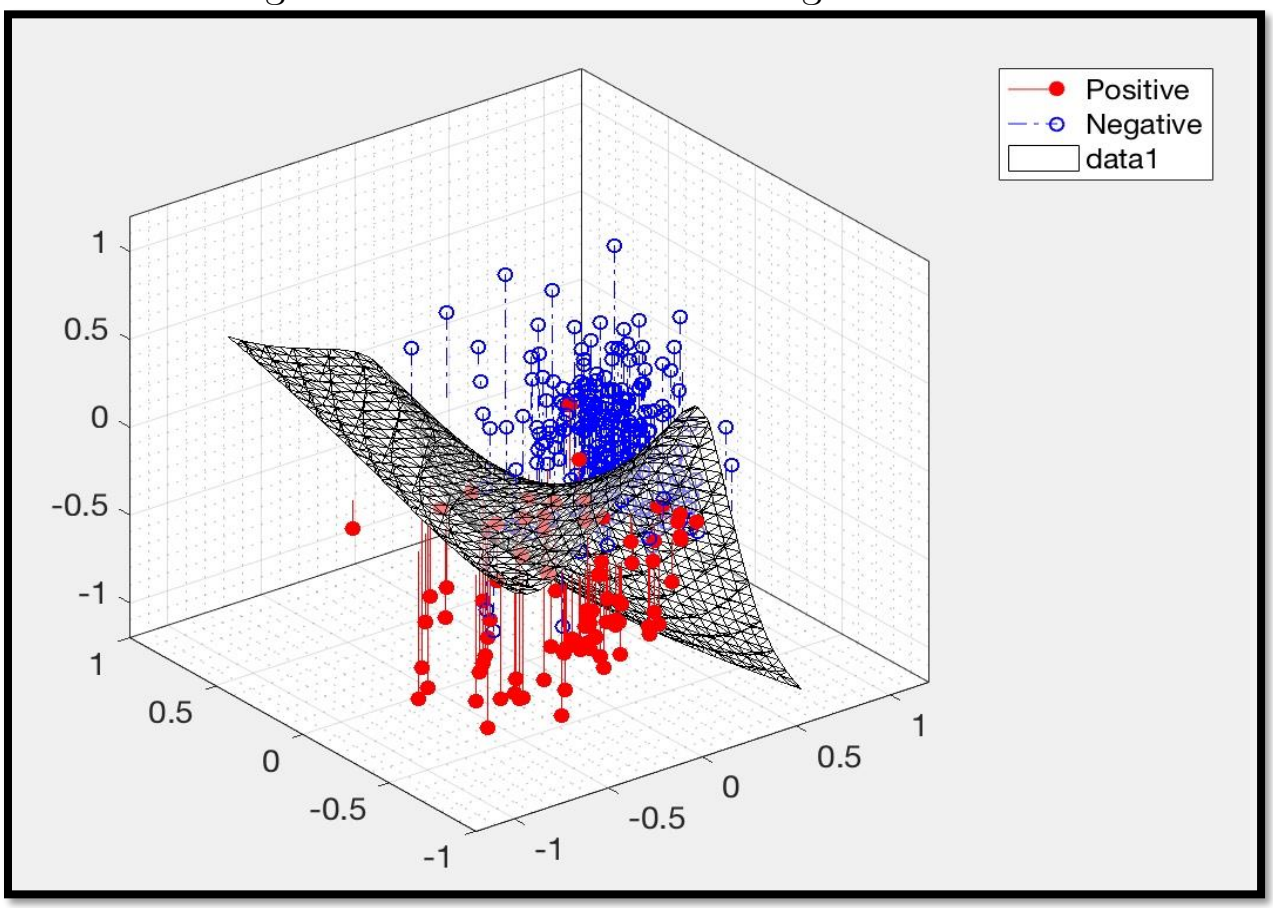
Further we will look for its implementation to solve real world problems and with time we will also explore the types of kernel and many more.

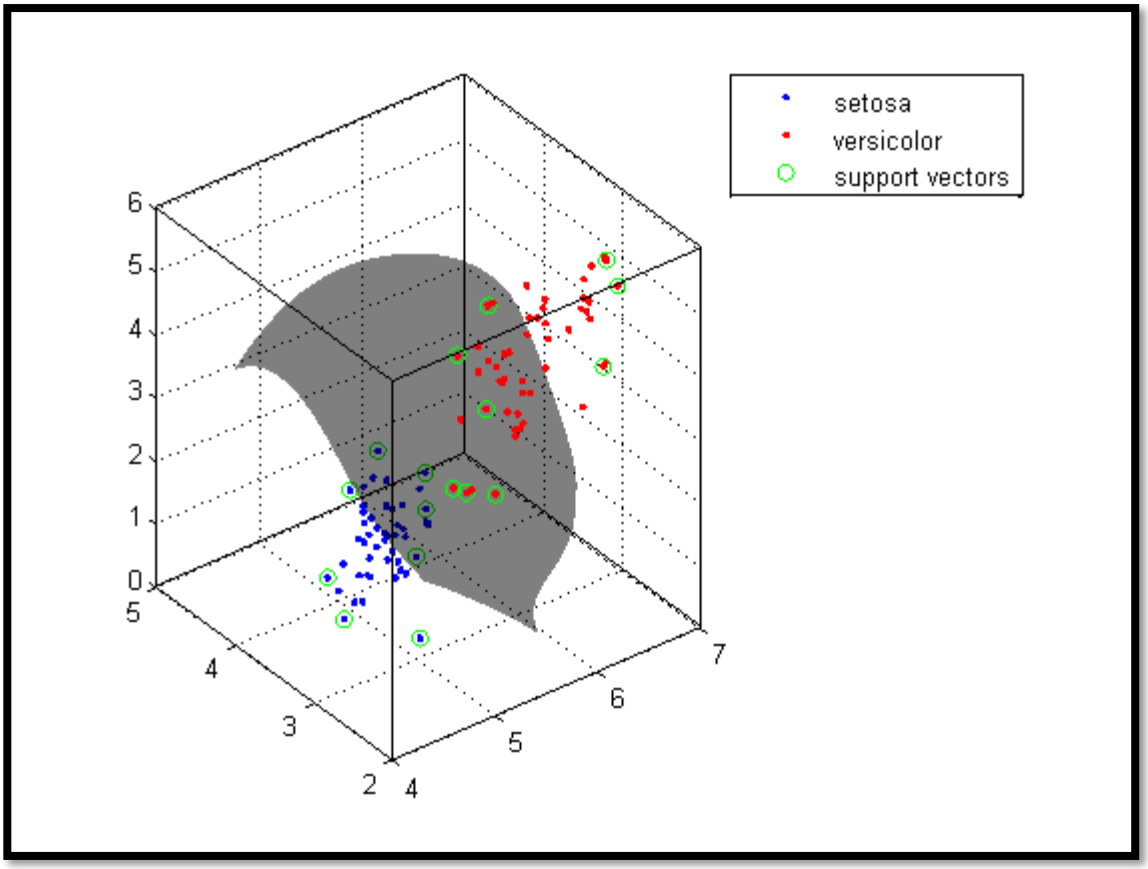
Visuals for summarizing

- What we have looked so far is:



- Some 3d insights to make stuffs interesting:





- Finally, the formulas:

$\sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i x_j$	$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i x_j$
Dot product distance or similarity measurement formula.	Kernel looks version
$\sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j f(x_i x_j)$	$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i x_j)$
Constraints from derivatives	
$w = \sum_{i=1}^{n} \alpha_i y_i x_i$ $\sum_{i=1}^{n} \alpha_i y_i = 0$	