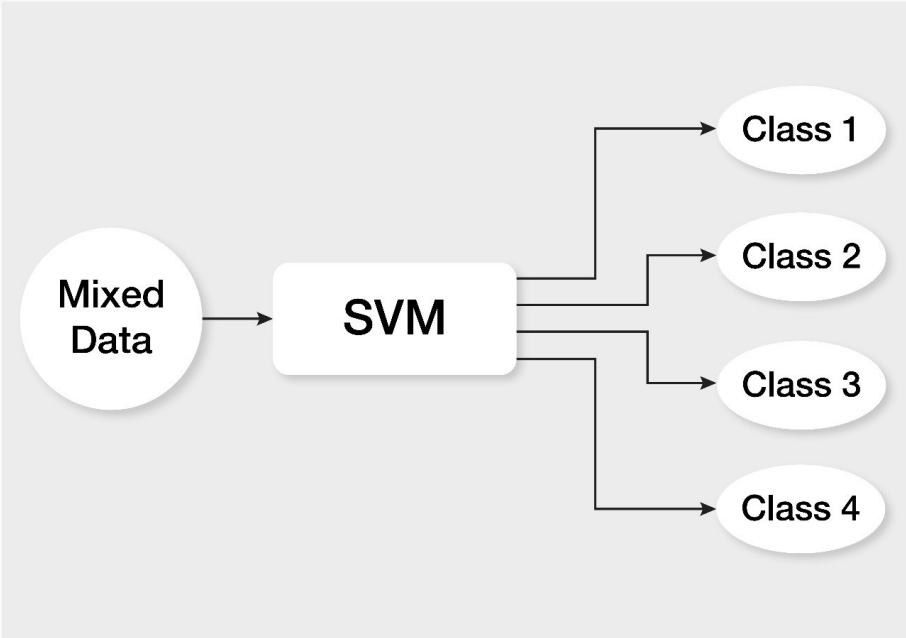
CSE/ECE-471: Statistical Methods in AI **Spring 2020**

Formulation of Support Vector Machine (SVM)

Prepared by: Roll no.s 201599594, 2019201007, 2019201016

1 Introduction



Machine learning involves predicting and classifying data and to do so we employ various machine learning algorithms according to the dataset.

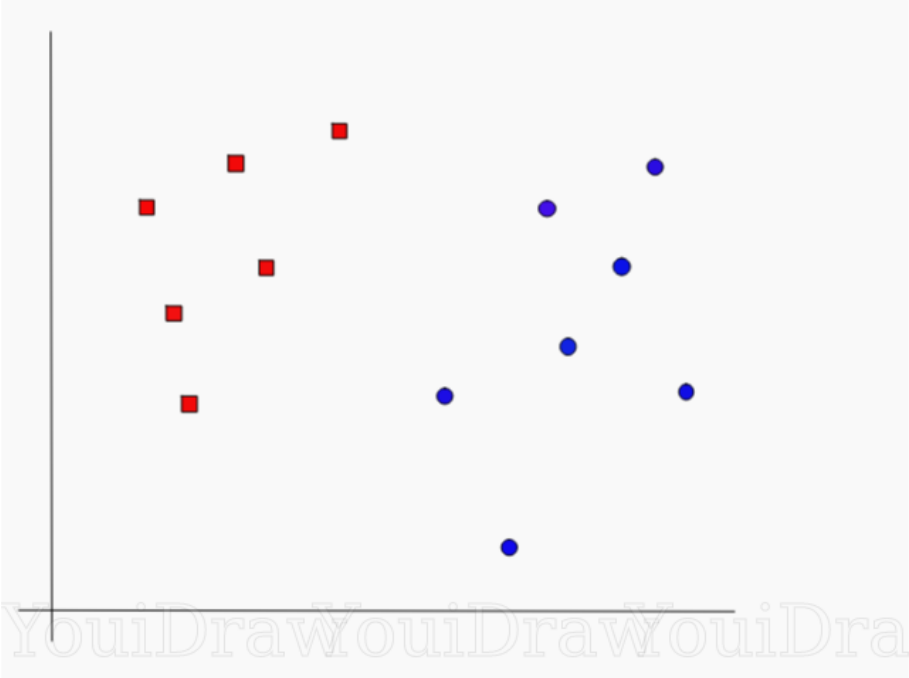
SVM or Support Vector Machine is a linear model for classification and regression problems. It can solve linear and non-linear problems and work well for many practical problems. The idea of SVM is simple: The algorithm creates a line or a hyperplane which separates the data into classes.

The goal of SVM is to identify an optimal separating hyperplane which maximizes the margin between different classes of the training data.

2 Theory

At first approximation what SVMs do is to find a separating line(or hyperplane) between data of two classes. SVM is an algorithm that takes the data as an input and outputs a line that separates those classes if possible.

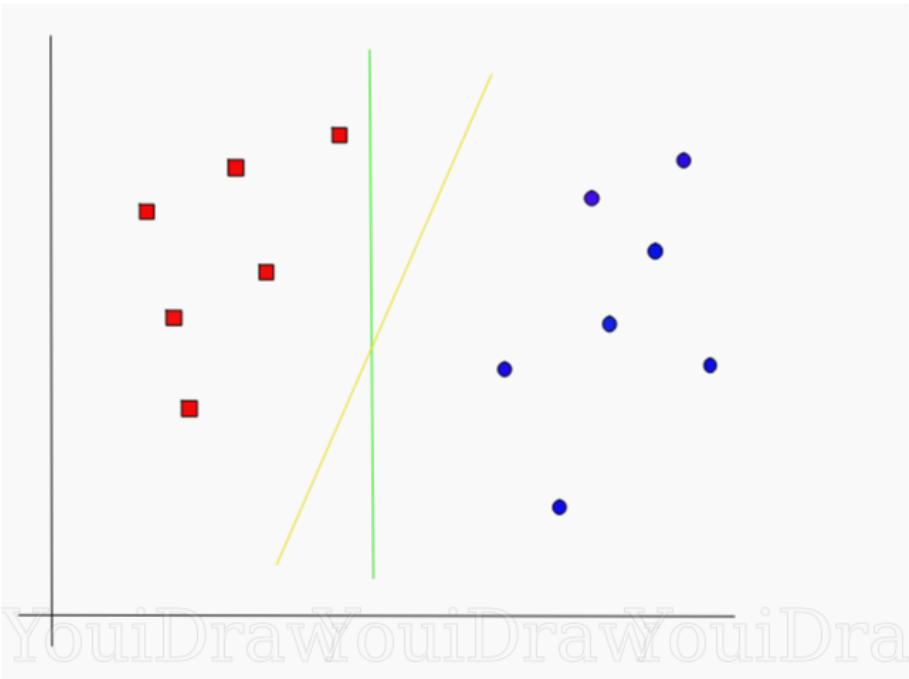
Lets begin with a problem. Suppose you have a dataset as shown below and you need to classify the red rectangles from the blue ellipses(let's say positives from the negatives). So your task is to find an ideal line that separates this dataset in two classes (say red and blue).



Find an ideal line/ hyperplane that separates this dataset into red and blue categories

But, as you notice there isn't a unique line that does the job. In fact, we have an infinite lines that can separate these two classes. So how does SVM find the ideal one???

Let's take some probable candidates and figure it out ourselves.

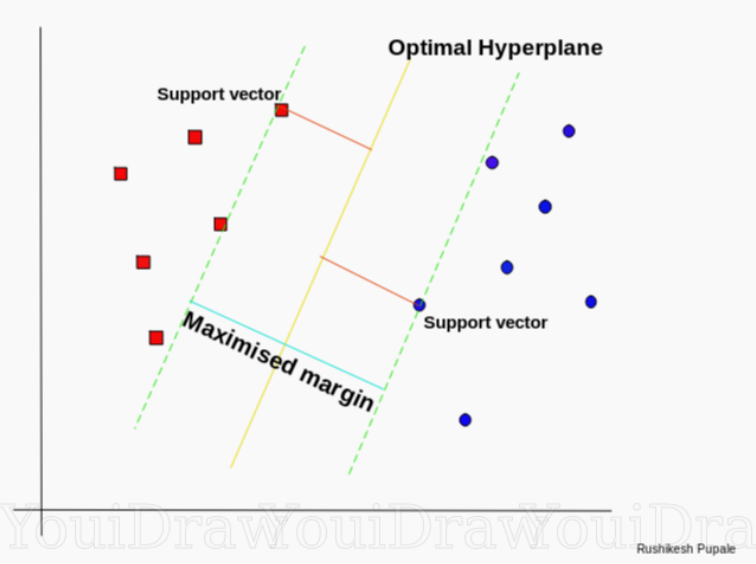


Which line best separates the data??

If you selected the yellow line then congrats, because thats the line we are looking for. It's visually quite intuitive in this case that the yellow line classifies better. But, we need something concrete to fix our line. The green line in the image above is quite close to the red class. Though it classifies the current datasets

it is not a generalized line and in machine learning our goal is to get a more **generalized separator** because even a little Perturbation in points will cause **Misclassification**.

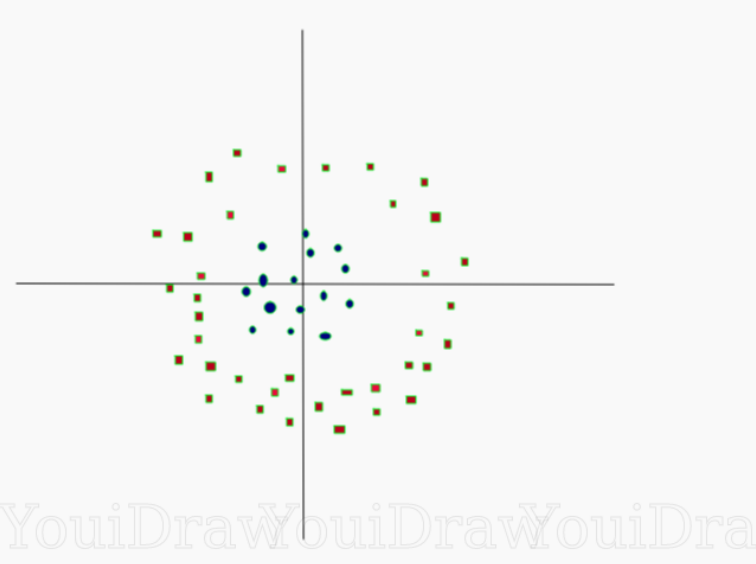
SVM's way to find the best line According to the SVM algorithm we find the points closest to the line from both the classes.These points are called support vectors. Now, we compute the distance between the line and the support vectors. This distance is called the margin. Our goal is to maximize the margin. The hyperplane for which the margin is maximum is the optimal hyperplane.



Optimal Hyperplane using the SVM algorithm

Thus SVM tries to make a decision boundary in such a way that the separation between the two classes(that street) is as wide as possible.

Let's consider a bit complex dataset, which is not linearly separable.

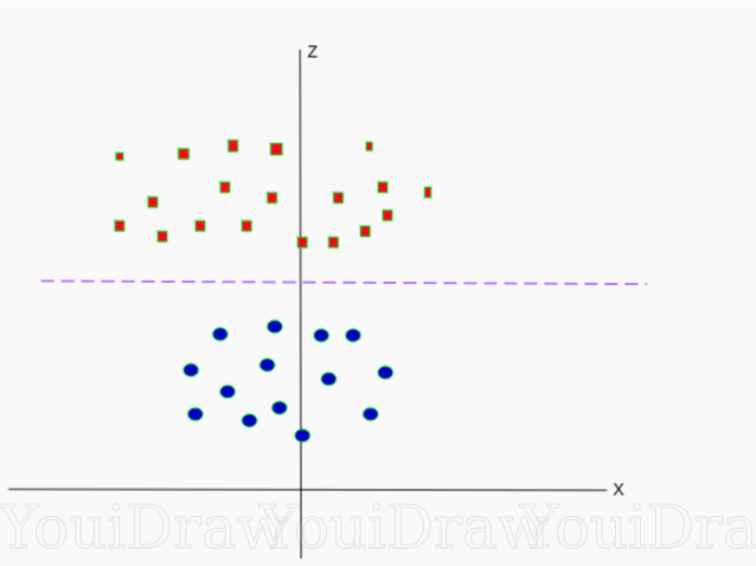


Non-linearly separable data

This data is clearly not linearly separable. We cannot draw a straight line that can classify this data. But, this data can be converted to linearly separable data in higher dimension. Lets add one more dimension and call it z-axis. Let the co-ordinates on z-axis be governed by the constraint,

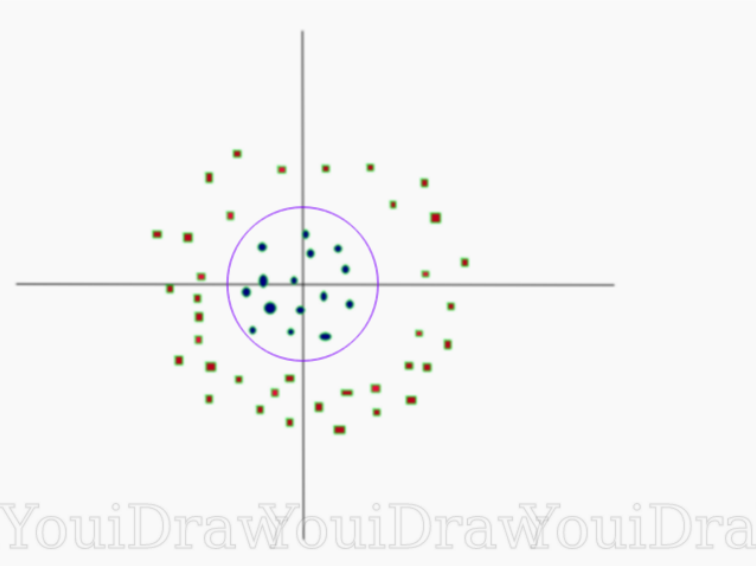
$$z = \left(x^2 + y^2\right)$$

So, basically z co-ordinate is the square of distance of the point from origin. Let's plot the data on z-axis.



Dataset on higher dimension

Now the data is clearly linearly separable. Let the purple line separating the data in higher dimension be z=k, where k is a constant. Since, $z = \left(x^2 + y^2\right)$ we get $k = \left(x^2 + y^2\right)$ which is an equation of a circle. So, we can project this linear separator in higher dimension back in original dimensions using this transformation.



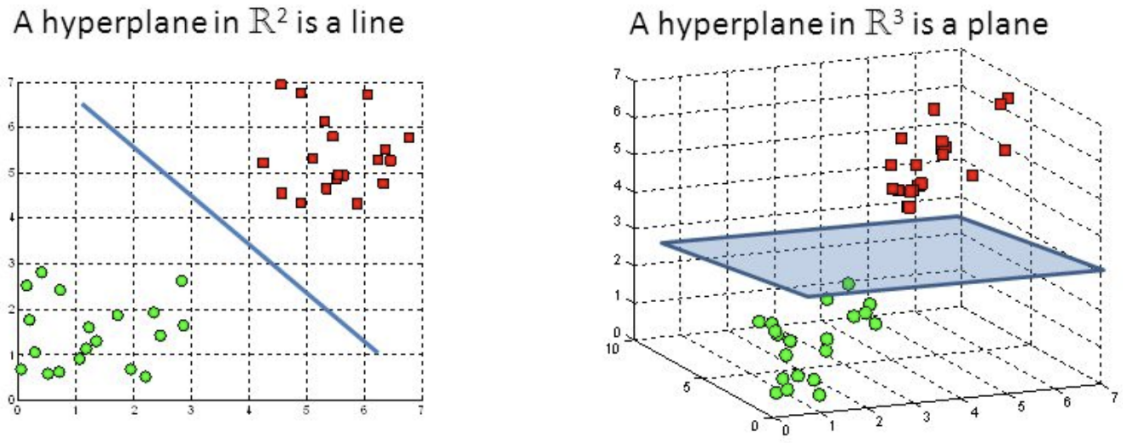
Decision boundary in original dimensions

Thus we can classify data by adding an extra dimension to it so that it becomes linearly separable and then projecting the decision boundary back to original dimensions using mathematical transformation. But finding the correct transformation for any given dataset isn't that easy. That's where comes the concept of Kernel's.

3 What is a Hyperplane

In mathematics, a hyperplane H is a linear subspace of a vector space V such that the basis of H has cardinality one less than the cardinality of the basis for V. In other words, if V is an n-dimensional vector space than H is an (n-1)-dimensional subspace. Examples of hyperplanes in 2 dimensions are any straight line through the origin. In 3 dimensions, any plane containing the origin. In higher dimensions, it is useful to think of a hyperplane as member of an affine family of (n-1)-dimensional subspaces (affine spaces look and behavior very similar to linear spaces but they are not required to contain the origin), such that the entire space is partitioned into these affine subspaces. This family will be stacked along the unique vector (up to sign) that is perpendicular to the original hyperplane. This "visualization" allows one to easily understand that a hyperplane always divides the parent vector space into two regions.

In machine learning, it may be useful to employ techniques such as support vector machines to learn hyperplanes to separates the data space for classification. The most common example of hyperplanes in practice is with support vector machines. In this case, learning a hyperplane amounts to learning a linear (often after transforming the space using a nonlinear kernel to lend a linear analysis) subspace that divides the data set into two regions for binary classification. If the dimensionality of the data set is greater than 2, this may be performed multiple times to achieve a multi-way classification.



An image of a Hyperplane

4 Functional and Geometric margins

The functional margin in SVM, $\hat{\gamma}$, is defined as:

$$\hat{\gamma} = d_i \left(w^T x_i + b\right)$$

d_i is the output label associated with the input vector x_i;

$d_i = 1$ if $x_i \in$ positive class, and
$d_i =$ -1 if $x_i \in$ negative class

As seen earlier,

$$g(x) = \left(w^T x_i + b\right)$$

where w is a weight vector that is orthogonal to the hyperplane, i.e., w tells us the direction of the hyperplane, and b is the bias value, which indicates the distance of the hyperplane from the origin.
Here, if we were to replace w by 5w and b by 5b, (choosing an arbitrary scalar constant) then $g(x) = \left(5w^T x_i + 5b\right)$, and this doesn't make a difference to the value of g(x), because we simply need to satisfy this condition:

$g(x) \geq \gamma$, if the vector x $\in$ positive class, and
$g(x) \leq -\gamma$, if the vector x $\in$ negative class.

That is, we are only bothered about the sign of the output to determine which class the training example belongs to. This implies that we can scale w and b to any values, without making any difference in a relative sense.
The geometric margin in SVM, γ, is defined as:

$$\gamma = d_i \left(\left(\frac{w}{\|w\|} \right)^T x_i + \frac{b}{\|w\|} \right)$$

Here, $\left(\frac{w}{\|w\|} \right)^T$ is simply a unit vector corresponding to the weight vector w^T.
The geometric margin is also invariant to scaling of the parameters w and b; i.e., if we replace w with 2w and b with 2b, then the geometric margin does not change.
So, we can impose an arbitrary scaling constraint on w without changing anything important; for instance, we can demand that $\|w\| = 1$, or $|w_1| = 5$, and any of these can be satisfied simply by rescaling w and b.

5 Finding the optimal margin (Optimal Hyperplane)

We assume that we are given a training set of data that is linearly separable; i.e., it is possible to separate the positive and negative training examples using some separating hyperplane.

Definition 5.1. *A hyperplane is called an optimal hyperplane, if it maximizes the geometric margin between itself and the closest data points from both classes. These data points, which are the closest to the hyperplane, are called support vectors.*

Why do we need to find the optimal hyperplane, i.e., the one which gives the maximum margin? Because this will reduce the chances of misclassification, i.e., a positive training example being predicted as negative, or vice-versa. How can we find the optimal hyperplane? For this purpose, we pose the following optimization problem:

$$\begin{aligned} &\max_{\gamma,w,b} \gamma \\ &\text{subject to } d_i\left(w^T x_i + b\right) \geq \gamma, \quad i = 1, \ldots, N \\ &\|w\| = 1 \end{aligned}$$

The first set of constraints ($d_i\left(w^T x_i + b\right) \geq \gamma$) means that for each training example (feature vector), the functional margin $d_i\left(w^T x_i + b\right)$ must be at least γ.
The second constraint, $\|w\| = 1$, makes the functional margin equal to the geometric margin, so this guarantees that the geometric margin is at least γ.
However, $\|w\| = 1$, i.e., $\sqrt{(w_1^2 + w_2^2 + \cdots w_n^2)} = 1$ is a non-convex constraint, so it is not computationally efficient to solve it directly. So, we use the idea that $\gamma = \frac{\hat{\gamma}}{\|w\|}$, (the geometric margin is obtained on dividing the functional margin by the magnitude of the weight vector w) in order to convert the problem into:

$$\begin{aligned} &\max_{\gamma,w,b} \frac{\hat{\gamma}}{\|w\|} \\ &\text{subject to } d_i\left(w^T x_i + b\right) \geq \gamma, \quad i = 1, \ldots, N \end{aligned}$$

However, now the objective function $\frac{\hat{\gamma}}{\|w\|}$ is a non-convex function, so it is still not computationally efficient to solve it directly. So, we now use the idea that we can scale the values of w and b, without changing anything. We impose the scaling constraint that the functional margin of w, b with respect to the training set must be 1, i.e., $\hat{\gamma} = 1$. We can ensure that $\hat{\gamma} = 1$ by scaling the values of w and b appropriately (that is how the idea of scaling w and b is used here).

So now, our objective function is to maximize $\frac{1}{\|w\|}$, which is the same as minimizing $\|w\|$. Why is that so? Intuitively, $\|w\|$ is always positive (because calculating magnitude involves squaring the vector's components, then adding them up and taking square root), and as we increase the value of the denominator, the value of $\frac{1}{\|w\|}$ will decrease. Since we want to maximize (increase) the value of $\frac{1}{\|w\|}$, we need to decrease the value of $\|w\|$.

Similarly, minimizing $\|w\|$ is the same as minimizing $\|w\|^2$. So we can re-write the optimization problem as:

$$\begin{aligned} &\min_{\gamma,w,b} \phi\left(w\right) = \frac{1}{2}\|w\|^2 \\ &\text{subject to } d_i\left(w^T x_i + b\right) \geq 1, \quad i = 1, \ldots, N \end{aligned}$$

We've multiplied the objective function by $\frac{1}{2}$ just for mathematical convenience, to make calculations easier while differentiating later. Why is it ok to multiply by $\frac{1}{2}$? Consider an example – the minimum value of x^2 is the same as the minimum value of $\frac{1}{2}x^2$, i.e., $x = 0$.
So finally, the problem has been transformed to a form that can be solved efficiently. It is now a quadratic, convex optimization problem, which has only linear constraints.

6 VC dimension and Vapnik's equation

VC (Vapnik–Chervonenkis) dimension signifies the maximum no. of classes or collections into which a set of points can be classified. If we can find a set of 'n' points in such a way that it can be **shattered** by a classifier and we can not find any set of 'n+1' points that can be also shattered then the VC dimension of the set is said to be 'n'. Putting in another way, if a classifier can correctly classify all the 2^n possible labeling of a set of 'n' points and for a set of 'n+1' points there is at least one labeling order in which the classifier can not separate all the points correctly then the VC dimension of the set is said to be 'n'. **Shatter** is a concept used in VC-theory. If **A** is a set and **C** is a class or collection of sets then **C** is said to shatter the set **A** if for each subset 'S' of **A**, there is some element 'c' of **C** such that $S = c \cap \mathbf{A}$. Equivalently we can say that C shatter **A** when their intersection is equal to the power set of **A**,i.e., $P(\mathbf{A}) = \{c \cap \mathbf{A} | c \in \mathbf{C}\}$. A set **A** is assumed to be finite because, normally we are interested in shattering of finite sets of data points. If arbitrarily large subsets can be shattered, the VC dimension will be ∞. VC dimension is a concept from computational learning theory that formally quantifies the power of a classification algorithm. The VC dimension of a classifier as defined by Vapnik and Chervonenkis is the cardinality (size) of the largest set of points that the classification algorithm can shatter.

In classification tasks a discriminant machine learning technique aims at finding a discriminant function that can correctly predict predict labels for newly acquired instances. A discriminant classification function takes a data pint 'x' and assigns it to one of the different classes that are part of the classification task. Discriminant approaches require fewer computational resources and less training data, especially for multidimensional feature space and when only posterior probabilities are needed. From a geometric perspective, learning a classifier is equivalent to finding the equation for a multidimensional surface that best separates the different classes in the feature space. SVM is a discriminant technique, and, because it solves convex optimization problem analytically, it always returns the same optimal hyperplane parameters. If many hyperplanes can be learnt during the training phase, only the optimal one is retained. This is because the training is practically performed on samples of the population even though the test data may not exhibit the same distribution as the training set.When trained with data that are not representative of the overall data population, hyperplanes are prone to poor generalization.

SVM is deeply rooted in the principles of statistics, optimization, and machine learning.It relies on the complexity of the hypothesis space and emperical error which is a measure of how well the model fits the training data.

Vapnik-Chervonenkis (VC) theory proves that a VC bound on the risk exists. VC is a measure of the complexity of the hypothesis space. The VC dimension of a hypothesis H relates to the maximum number of points that can be shattered by H. H shatters N points if it can correctly separates all the positive instances from the negative ones. So, the VC capacity is equal to the number of training points N that the model can separate into 2^N different labels. This capacity is related to the amount of training data available. The VC dimension, h, affects the generalization error, as it is bounded by $||w||$, where w is the weight vector of the separating hyperplane and the radius of the smallest sphere R that contains all the training data points.

The bound on expected loss is given by:

$$\mathrm{R}(\alpha) \leq R_{train}(\alpha) + \sqrt{\frac{f(h)}{N}}$$

Here h is the VC dimension, $\alpha is some parameter or a vector of adjustable parameters on which training is done$,R($\alpha$) is the test loss or error, $R_{train}(\alpha)$ is training error or loss, N is the no. of training data points. The

function f(h) is given by:

$$\text{f(h)}=h+\text{hlog(2N)}-\text{hlog(h)}-\text{C}$$

where C is some constant.

The test error is given by:

$$R(\alpha) = E[\frac{1}{2}|y - f(x, \alpha)]$$

The train error is given by:

$$R_{train}(\alpha) = \frac{1}{R}\sum_{k=1}^{R}\frac{1}{2}|y_k - f(x_k, \alpha)|$$

To reduce the test error, the training error should be kept as low as possble, say 0 (zero), and then we should minimise the VC dimension h.

Considering the relative margin $\frac{\rho}{D}$, where ρ is the margin, D is the Data Diameter, the relation for h is given by:

$$h \leq min\{d, \lceil\frac{D^2}{\rho^2}\rceil\} + 1$$

7 Solving optimization problems using Lagrange's method

We're given a problem of this form:

$$\begin{aligned} &\min_{w} f(w) \\ &\text{subject to } h_i(w) = 0, \quad i = 1, \ldots, m \end{aligned}$$

Then, we define an equation called the **Lagrangian**, as:

$$\mathcal{L}(w, \alpha) = f(w) + \sum_{i=1}^{m} \alpha_i h_i(w)$$

So, the Lagrangian is the objective function $f(w)$ added with the sum of each constraint multiplied by a scalar called **Lagrangian multiplier**, denoted by α_i here. The next step is to calculate partial derivatives of the function L w.r.t w and α, equate them to 0, and find the values of w and α.
Note: The constraints need not always be equations, they can represent inequalities too. In this case, we would use a different notation for the Lagrangian multiplier, say, β, and multiply each inequation by β.
We can consider this as the **primal form** of the optimization problem, where we need to minimize w.r.t w.

According to Lagrange's theory, we can convert a problem in the primal form into its **dual form**, where the objective function would be a function of the Lagrangian multipliers α *and* β (β is needed in case there are any inequations), and we would have to maximize the objective function w.r.t α *and* β. (For more details, please read reference [2]).

Under certain conditions, the optimal solution to the primal form leads to an optimal solution for the dual form of the problem and vice-versa, so in this case, if the dual form is easier to solve, we can solve the dual form instead of the primal form.

Let w^* be the solution to the primal form of the problem, and let α^* *and* β^* be the solution to the dual form of the problem. Then, w^*, α^* *and* β^* satisfy the **Karush-Kuhn-Tucker (KKT) conditions**, which are:

$$\begin{aligned} \frac{\partial}{\partial w_i} \mathcal{L}(w^*, \alpha^*, \beta^*) &= 0, \quad i = 1, \ldots, n \\ \frac{\partial}{\partial \beta_i} \mathcal{L}(w^*, \alpha^*, \beta^*) &= 0, \quad i = 1, \ldots, l \\ \alpha_i^* g_i(w^*) &= 0, \quad i = 1, \ldots, k \\ g_i(w^*) &\leq 0, \quad i = 1, \ldots, k \\ \alpha^* &\geq 0, \quad i = 1, \ldots, k \end{aligned}$$

Here, $g_i(w^*)$ represents a set of constraints.

8 Applying Lagrange's method to SVM

From the quadratic convex optimization problem obtained for SVM earlier, using the set of constraints, let:

$$g_i(w) = 1 - d_i\left(w^T x_i + b\right) \leq 0$$

According to the 3rd KKT condition listed above, if $\alpha_i^* > 0$, then $g_i(w^*) = 0$. Using our definition of $g_i(w)$, $g_i(w)$ will be 0 for only those training examples, whose functional margin is exactly equal to 1. These points are support vectors, as explained in definition 4.1. This implies that $\alpha_i > 0$ only for support vectors.

We construct the Lagrangian for the optimization problem as:

$$J(w, \alpha, b) = \frac{1}{2}\|w\|^2 + \sum_{i=1}^{N} \alpha_i \left(1 - d_i\left(w^T x_i + b\right)\right)$$
$$\text{subject to } \alpha_i \geq 0 \quad \forall i$$

$$\therefore J(w, \alpha, b) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \alpha_i d_i\left(w^T x_i + b\right) + \sum_{i=1}^{N} \alpha_i \tag{1}$$

$$\text{subject to } \alpha_i \geq 0 \quad \forall i$$

To find the dual form of the problem (which is easier to solve), we need to minimize $J(w, \alpha, b)$ w.r.t. $w and \beta$, while keeping α fixed. So now we use the step of Lagrange's method that involves finding partial derivatives and equating them to 0.

$$\frac{\partial J}{\partial w} = 0$$
$$\therefore \frac{2}{2} w - \sum_{i=1}^{N} \alpha_i d_i x_i + 0 = 0$$

$$\therefore w = \sum_{i=1}^{N} \alpha_i d_i x_i \tag{2}$$

$$\frac{\partial J}{\partial b} = 0$$
$$\therefore 0 - 0 + \sum_{i=1}^{N} \alpha_i d_i + 0 = 0$$

$$\therefore \sum_{i=1}^{N} \alpha_i d_i = 0 \tag{3}$$

We now substitute equation (3) in equation (1),

$$J(w,\alpha,b) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N}\alpha_i d_i w^T x_i - \sum_{i=1}^{N}\alpha_i d_i b + \sum_{i=1}^{N}\alpha_i$$
$$\therefore J(w,\alpha,b) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N}\alpha_i d_i w^T x_i + \sum_{i=1}^{N}\alpha_i$$

Substituting equation (2) in the above equation,

$$J(w,\alpha,b) = \frac{1}{2}w^T w - w^T w + \sum_{i=1}^{N}\alpha_i$$
$$\therefore J(w,\alpha,b) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}w^T w$$
$$\therefore J(w,\alpha,b) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j d_i d_j x_i^T x_j$$

This equation is the dual form of the quadratic optimization problem. We pose the problem properly, along with constraints:

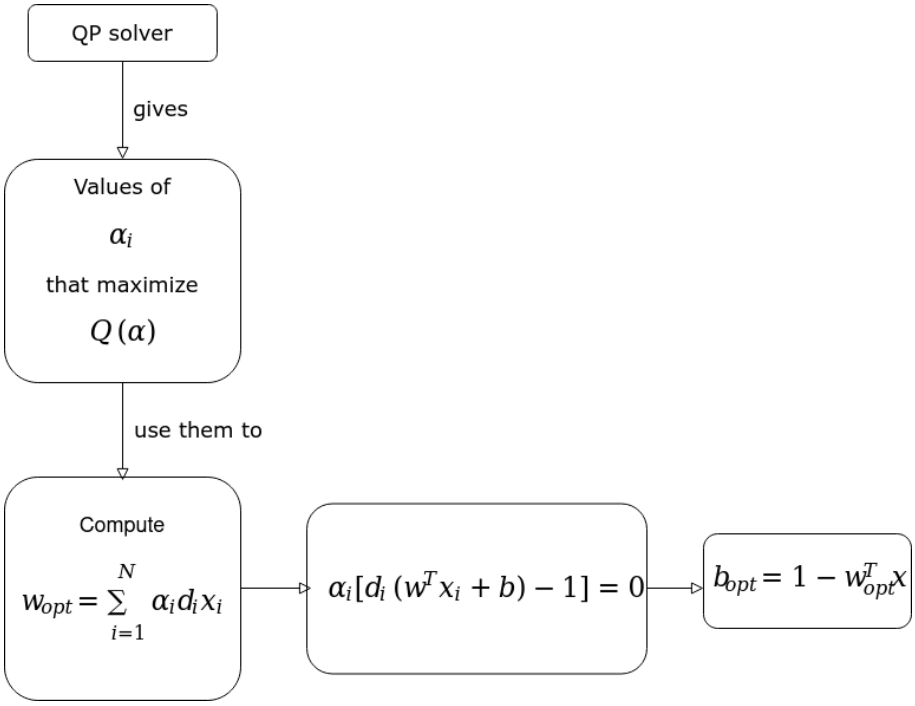
$$\max_{\alpha} \quad Q(\alpha) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j d_i d_j x_i^T x_j$$
$$\text{subject to } \alpha_i \geq 0, \quad i = 1,\ldots,N$$
$$\sum_{i=1}^{N}\alpha_i d_i = 0$$

Now, once we find the values of α_is that maximize $W(\alpha)$, then we can use equation (2) to find the optimal value of w.
The values of α_i can be found using a QP (Quadratic Programming) solver — a software meant for solving optimization problems (such software usually requires the problem to be in convex form and have linear constraints), and the software might employ some algorithm such as SMO (sequential minimal optimization).

Once we have the optimal value of the weight vector w, we can get the optimal value of bias b in this way:

$$b_{opt} = 1 - w_{opt}^T x$$

, since $w_{opt}^T x + b_{opt} = 1$. Here, for x, we can take any training example which is a support vector, so that the value of $\alpha > 0$. That is also the reason why we have equated $w_{opt}^T x + b_{opt}$ to 1, because we take x as a support vector, for which the functional margin is exactly 1.



The process of solving the dual form of the problem

9 Using SVM for prediction on test data

Once we have found the optimal values of w and b for a training dataset, we can use them to predict whether a new point x belongs to the positive class or to the negative class. First, we need to calculate $g(x) = \left(w^T x + b\right)$, and then if $g(x) > 0$, then we predict that x belongs to the positive class, else it belongs to the negative class.

However, using equation (2), we can write the equation for the classification decision as:

$$w^T x + b = \left(\sum_{i=1}^{N} \alpha_i d_i x_i\right)^T x + b$$
$$\therefore w^T x + b = \sum_{i=1}^{N} \alpha_i d_i \left(x_i x\right) + b$$

This shows that, since we have the values of α_i, so to make a prediction, we don't even need to use the value of w, we simply need to take a dot product between the new training example x, and the points x_i in the training set.

Importantly, α_i will be 0 for all x_i other than the support vectors, since $\alpha_i > 0$ only for support vectors, as explained earlier. This implies that **we only need to take the dot product between the new training example x and each of the support vectors, in order to make a prediction**.

10 Example of SVM

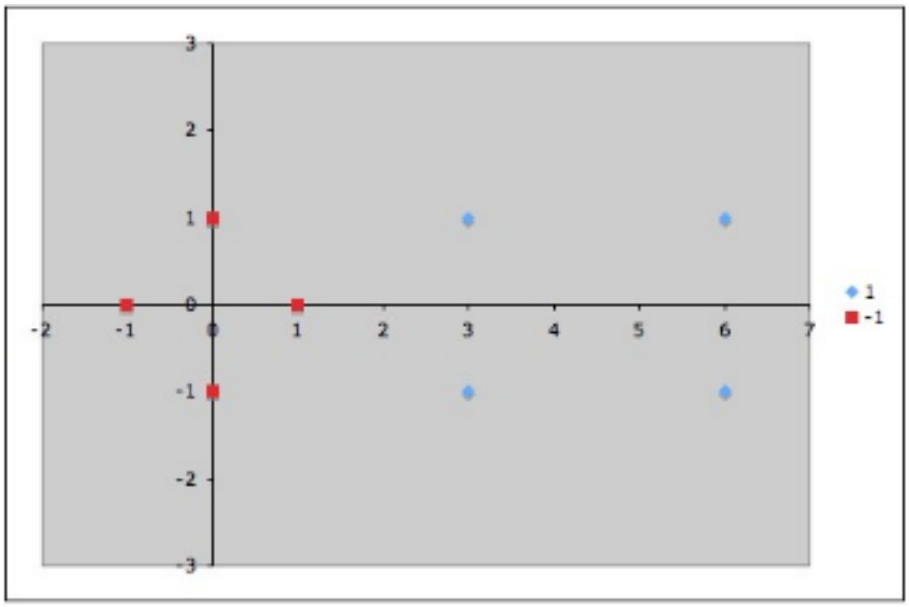
Suppose we are given the following positively labeled data points in $\Re^2$:

$$\left\{\begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ -1 \end{pmatrix}, \begin{pmatrix} 6 \\ 1 \end{pmatrix}, \begin{pmatrix} 6 \\ -1 \end{pmatrix}\right\}$$

and the following negatively labeled data points in $\Re^2$:

$$\left\{\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix}\right\}$$

We can plot them like this:



Plotting the given positive and negative training examples

Here, the blue points represent positive training examples, and the red points represent negative training examples.

We want to find a simple SVM that accurately discriminates between the two classes. Since the data is linearly separable, we can use a linear SVM (using the kernel trick is not required here).

Since the blue points and red points are neatly separated in this simple example, we just need to find a line(hyperplane) that lies between these 2 groups. As can be seen in the graph, the support vectors (points which would be the closest to the hyperplane) are the points (1, 0), (3,1) and (3, -1).
Let:

$$\left\{s_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, s_2 = \begin{pmatrix} 3 \\ 1 \end{pmatrix}, s_3 = \begin{pmatrix} 3 \\ -1 \end{pmatrix}\right\}$$

We will now use vectors augmented with a 1 as a bias input, and for clarity we will differentiate these with an over-tilde. So, if $s_1 = (10)$, then $\tilde{s_1} = (101)$. Our task is to find values for the α_i such that

$$\begin{aligned}
\alpha_1 \tilde{s_1} \cdot \tilde{s_1} + \alpha_2 \tilde{s_2} \cdot \tilde{s_1} + \alpha_3 \tilde{s_3} \cdot \tilde{s_1} &= -1 \\
\alpha_1 \tilde{s_1} \cdot \tilde{s_2} + \alpha_2 \tilde{s_2} \cdot \tilde{s_2} + \alpha_3 \tilde{s_3} \cdot \tilde{s_2} &= +1 \\
\alpha_1 \tilde{s_1} \cdot \tilde{s_3} + \alpha_2 \tilde{s_2} \cdot \tilde{s_3} + \alpha_3 \tilde{s_3} \cdot \tilde{s_3} &= +1
\end{aligned}$$

How did we get these equations? By using $\sum_{i=1}^{N} \alpha_i d_i (x_i x) + b$. Note that in this example we have included the bias values in the feature vectors. Also, in the first equation, we have factored out $d_i = -1$ from each of the 3 terms, and then multiplied both sides of the equation by -1.

In the 2nd and 3rd equations, $d_i = 1$ anyway, so even on factoring out, the RHS remains 1.

Solving these equations:

$$\alpha_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = -1$$
$$\alpha_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} = +1$$
$$\alpha_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} = +1$$

Computing the dot product of each pair of vectors, we get:

$$\begin{aligned} 2\alpha_1 + 4\alpha_2 + 4\alpha_3 &= -1 \\ 4\alpha_1 + 11\alpha_2 + 9\alpha_3 &= +1 \\ 4\alpha_1 + 9\alpha_2 + 11\alpha_3 &= +1 \end{aligned}$$

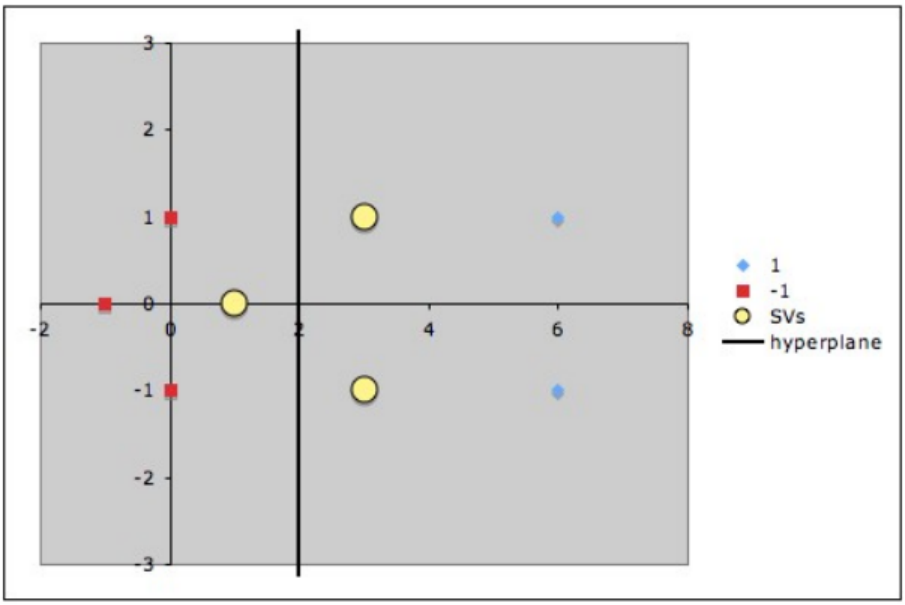
Solving this system of equations gives $\alpha_1 = -3.5, \alpha_2 = 0.75$ and $\alpha_3 = 0.75$.

Now that we have the α_i, how do we find the hyperplane that discriminates the positive from the negative examples? It turns out that:

$$\begin{aligned} \tilde{w} &= \sum_i \alpha_i \tilde{s}_i \\ &= -3.5 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + 0.75 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + 0.75 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix} \end{aligned}$$

Finally, since our vectors are augmented with a bias, the last entry in $\tilde{w}$ is the hyperplane offset b and so we write the separating hyperplane equation $y = wx + b$ as $w = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $b = -2$.

We can plot the hyperplane this way:



Plotting the equation of the hyperplane (SVs = Support Vectors)

References

[1] Lecture notes on SVM, made by the authors of this document.

[2] Lecture notes on SVM by Andrew Ng : http://cs229.stanford.edu/notes/cs229-notes3.pdf

[3] An example of SVM: http://axon.cs.byu.edu/Dan/678/miscellaneous/SVM.example.pdf

[4] https://medium.com/data-science-group-iitr/support-vector-machines-svm-unraveled-e0e7e3ccd49b

[5] https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989

[6] http://fuzihao.org/blog/2017/11/18/SVM-Revisit/