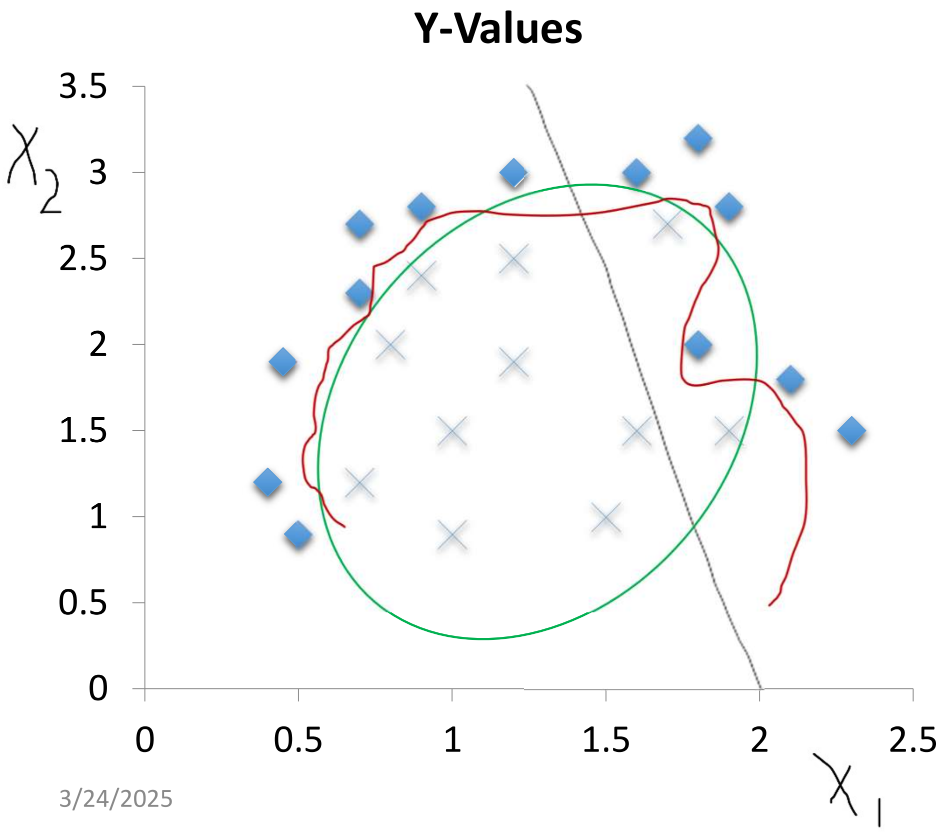
Classification-5

Support Vector Machine

Support Vector Machine (SVM)-Introduction



$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

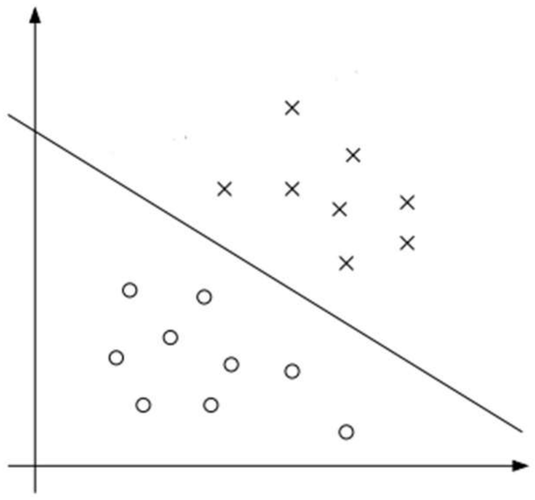
$$\emptyset(x) = \begin{bmatrix} X_1 \\ X_2 \\ X_1^2 \\ X_2^2 \\ X_1X_2 \end{bmatrix}$$

Support Vector Machine (SVM)-Introduction

- SVMs have a clever way to prevent overfitting
- They can use many features without requiring too much computation.
- Turn-key algorithm

Optimal margin classifier (Separable case)

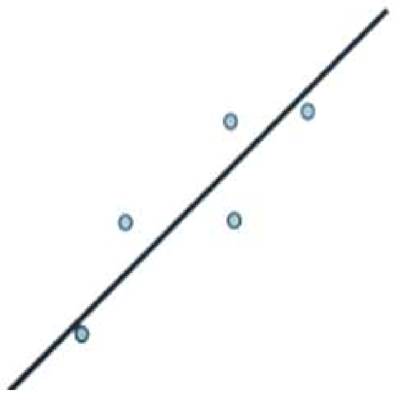
- Basic building block for support vector machine



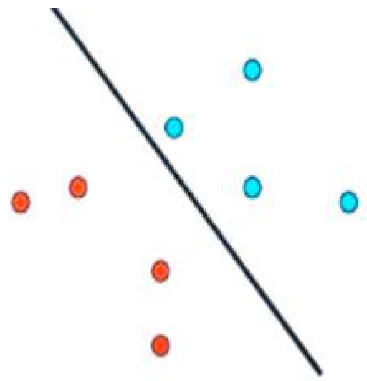
Linearly Separable case

Finding lines/decision boundaries

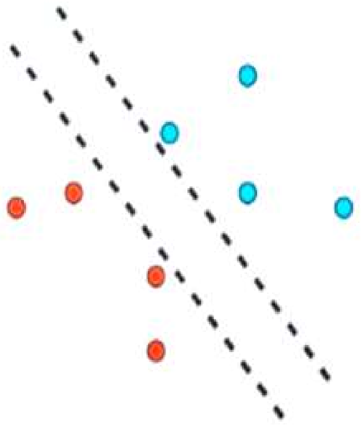
Linear Regression



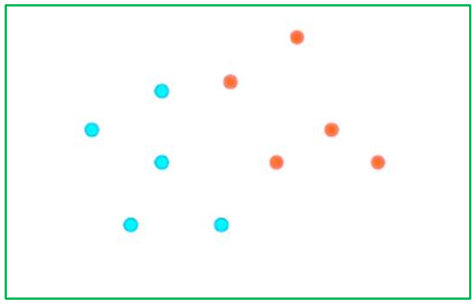
Logistic Regression



Support Vector Machine (SVM)

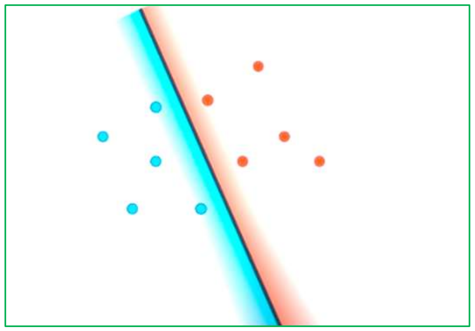


Classification goal: to split the data



- Objective in perceptron algorithm or logistic regression:

to find the line which splits the dataset in best possible way, a line also with positive and negative sides.

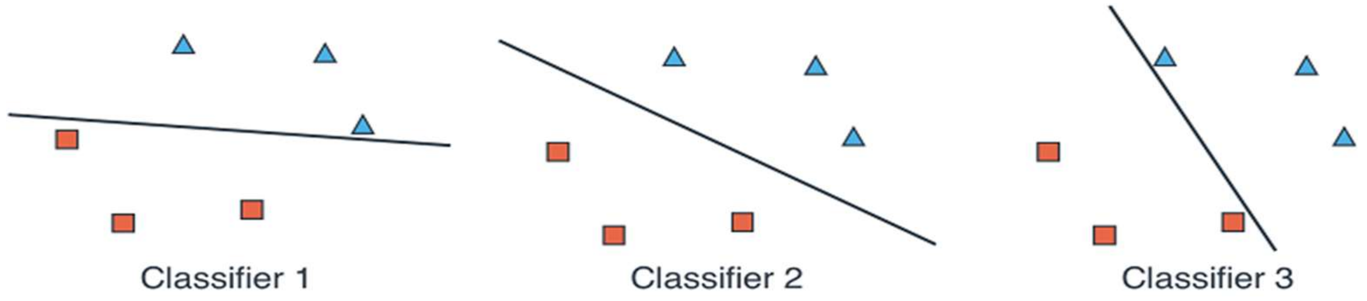


SVMs - Introduction

- A very powerful classification algorithm.
- *A support vector machine not only aims to find a separation boundary, it aims to find the one that is the farthest away from the points.*
- SVMs have a clever way to prevent overfitting
- They can use many features without requiring too much computation.
- In linear classifiers in two dimensions are simply defined by a line that best separates a dataset of points with two labels (binary classification).
- However, you may have noticed that many different lines can separate a dataset, and this raises the question: How to find which one is the best line in two dimensions?

SVMs – Introduction (cont.)

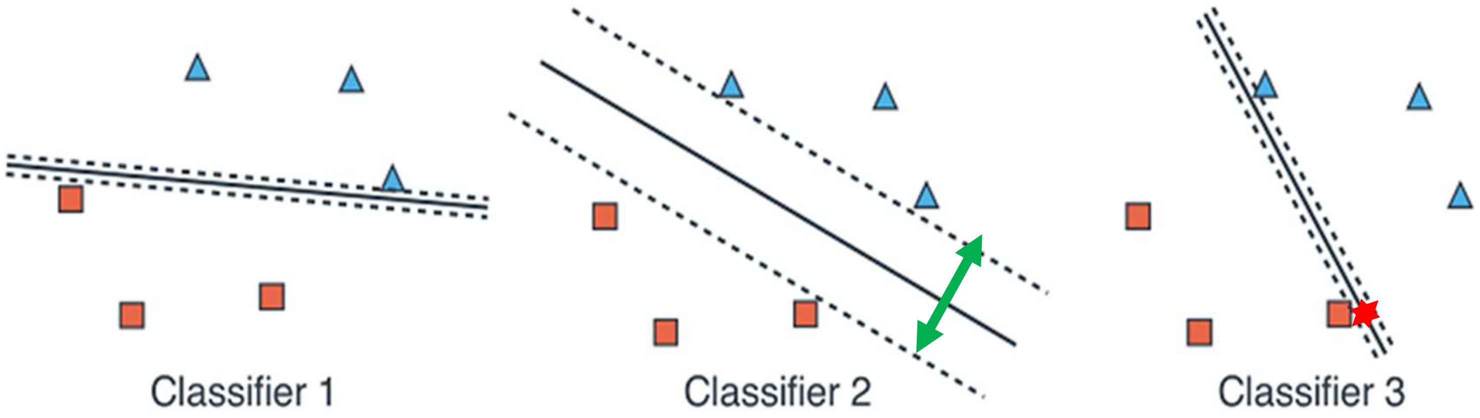
- Example: Three classifiers that classify dataset correctly. Which one is best Classifier 1, 2, or 3?



- All three lines separate the dataset well, but the second line is better placed,
  - the first and the third line are very close to some of the points, while the second line is not close to any points.
  - If we were to wiggle the three lines around a little bit, the first and the third may go over one of the points, misclassifying them in the process, while the second one will still classify them all correctly.
- *This is where SVM is different. The SVMs employ two parallel lines, and try to space them up apart as much as possible while still classifying the points correctly.*

SVMs – Introduction (cont.)

- Now, observe classifier as two parallel lines, as far apart from each other as possible.
- Note that Classifier 2 is the one where the parallel lines are farther from each other. This means that the main line in Classifier 2 is the one best located between the points.



- How to build such a classifier? In a very similar way as before, only with a slightly different error function (or loss function), and a slightly different iterative step (gradient descent).

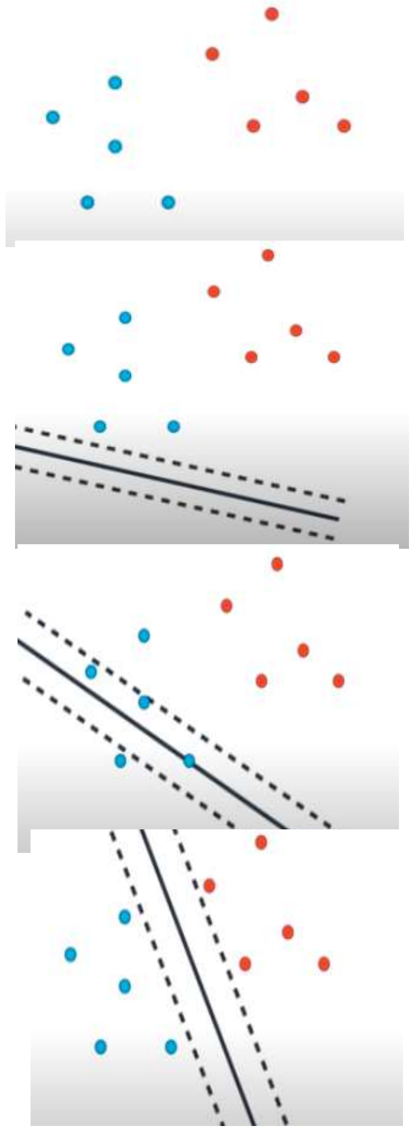
How to build SVM classifier

- Split data – separate lines

Step 1. Draw a random line and then draw two parallel lines surround it with some random distance.

Step2: Keep separating the lines till best classification..

What to learn – How to separate lines?



Developing error function to build SVM classifiers

- As it is common in machine learning models, SVMs are also defined using an error function.
- The error function of SVMs, tries to maximize two things at the same time: correct classification and distance between the points.
- Build an error function that will create a classifier consisting of two lines, as spaced apart as possible.
- For building an error function, : “What to achieve?”- The error function should penalize any model that doesn’t achieve those things.
- The following are the two objectives for error function:
  - the two lines to classify the points as best as possible.
  - the two lines to be as far away from each other as possible.

Developing error function to build SVM classifiers

- Since two objectives to be met, SVM error function should be the sum of two error functions:
  - The first error function penalizes points that are misclassified, and
  - the second penalizes lines that are too close to each other.

- Therefore, our error function should look like this:
  - Error = Classification Error + Distance (or margin) Error.

- Let's develop each one of these two error functions separately:
  - Classification error function - trying to classify the points correctly.
  - Distance (or margin) error function - trying to space our two lines as far apart as possible.

Classification error function
- trying to classify the points correctly

- This is the part of the error function that tries to make sure our points are classified correctly.
- One difference between SVMs and other linear classifiers is that SVMs use two parallel lines instead of one.
- The two parallel lines have very similar equations, they have the *same weights, but a different bias*.
- Thus, in SVM, use the central line as a frame of reference L with equation: $w_1\ x_1 + w_2\ x_2 + b = 0$

and construct two lines, one above it and one below it, with the respective equations

L+: $w_1\ x_1 + w_2\ x_2 + b = 1$

L-: $w_1\ x_1 + w_2\ x_2 + b = -1$

Classification error function
- trying to classify the points correctly

- Example:

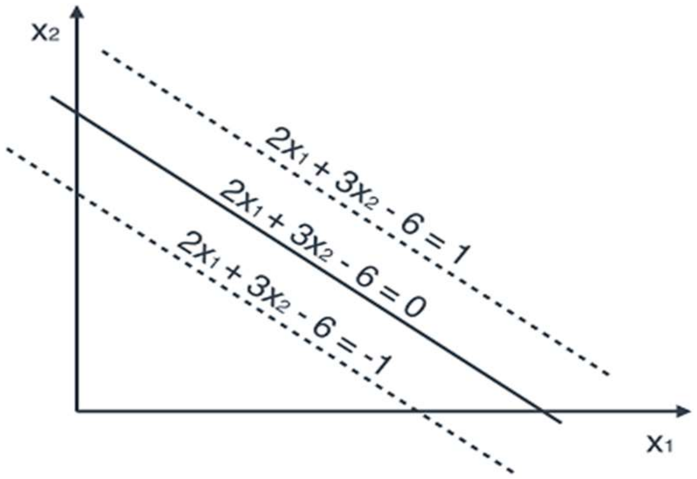
L: $2x_1 + 3x_2$ - 6 = 0

L+: $2x_1 + 3x_2$ - 6 = 1

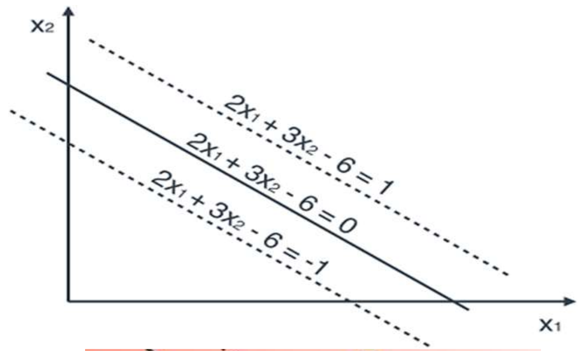
L-: $2x_1 + 3x_2$- 6 = -1

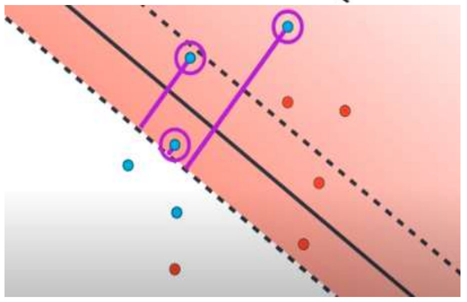
Reference hyperplane L is the one in the middle as shown below:

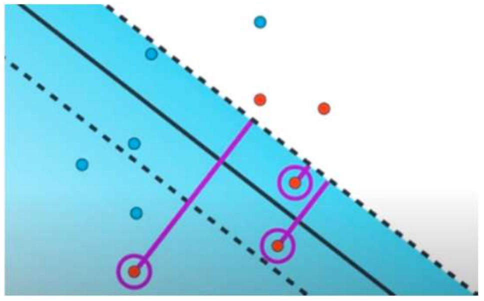
Building the two parallel boundary lines L+ and L- by slightly changing the equation of L.



Classification error function
- trying to classify the points correctly



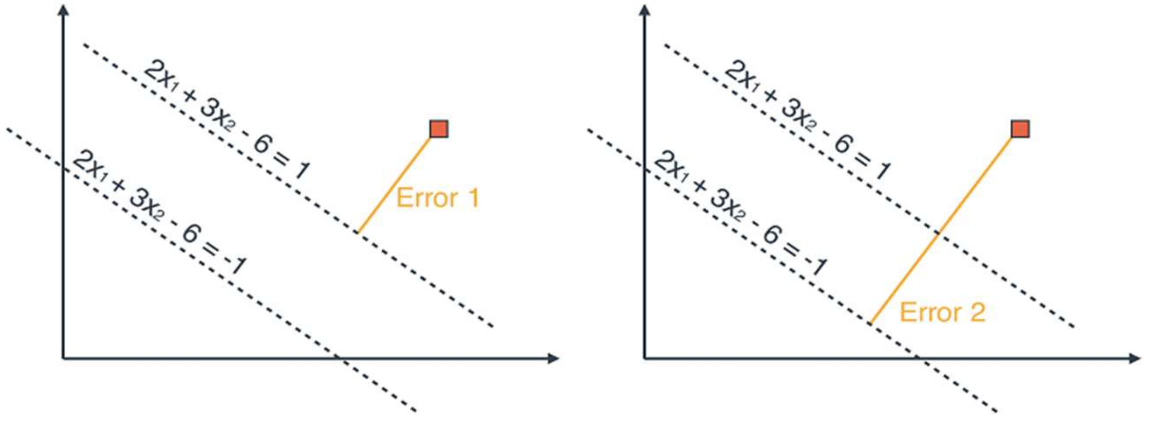




- Now, classifier will consist of the lines L+ and L-.
- The goal of this classifier is to have as few points as possible in between the two lines, and to have the two lines separate the points as best as possible.
- In order to measure this, the classifier act as two different classifiers, one for L+ and one for L-.
- *The classification function is defined as the **sum** of the error functions for both classifiers.*

Classification error function
- trying to classify the points correctly

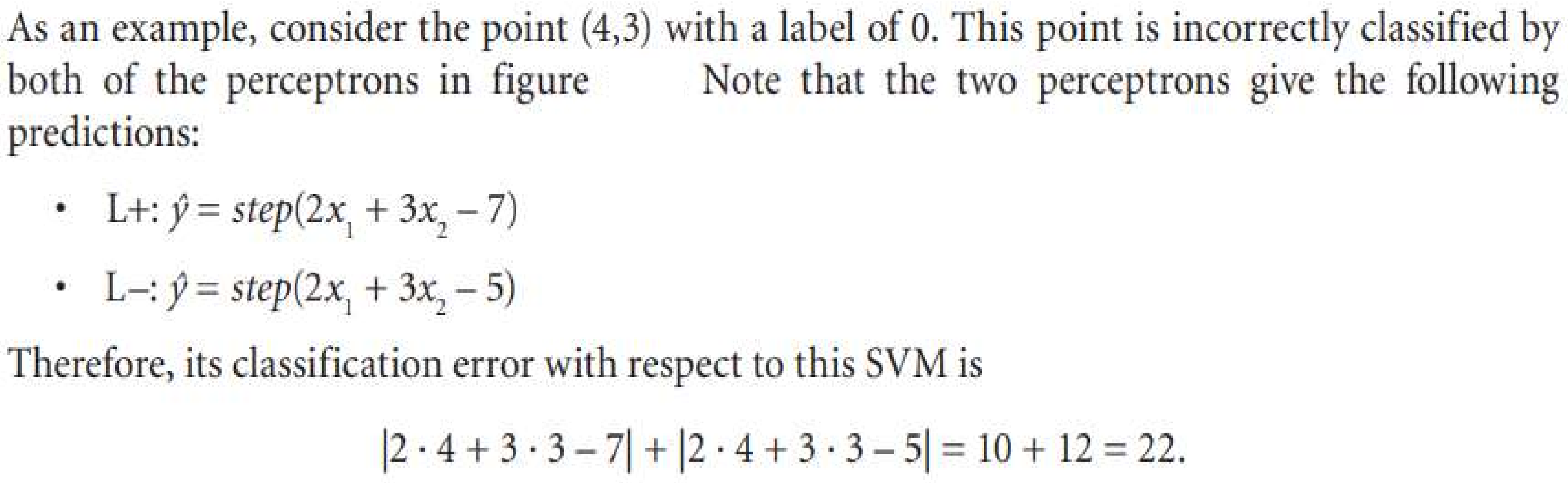
- Now that classifier consists of two lines, the error of a misclassified point is measured with respect to both lines.
- Then **add** the two errors to obtain the classification error.



Error = Error 1 + Error 2

- Note that in an SVM, both lines have to classify the points well. Therefore, *a point that is between the two lines is always misclassified by one of the lines*, so the classifier does not classify it correctly.

Classification error function
- trying to classify the points correctly



As an example, consider the point (4,3) with a label of 0. This point is incorrectly classified by both of the perceptrons in figure   Note that the two perceptrons give the following predictions:

- L+: $\hat{y} = step(2x_1 + 3x_2 - 7)$
- L–: $\hat{y} = step(2x_1 + 3x_2 - 5)$

Therefore, its classification error with respect to this SVM is

$$|2 \cdot 4 + 3 \cdot 3 - 7| + |2 \cdot 4 + 3 \cdot 3 - 5| = 10 + 12 = 22.$$

Distance (or margin) error function

- trying to space our two lines as far apart as possible

- Now that we have developed an error function that takes care of classification errors, next to build one that takes care of the distance between the two lines.
- Here, developing simple error function which is large when the two lines are close, and small when the lines are far.

- This error function is so simple that you have already seen it before; it is the regularization term. More specifically, if our lines have equations

$$w_1\ x_1 + w_2\ x_2 + b = 1$$
$$w_1\ x_1 + w_2\ x_2 + b = -1$$

- then the error function (or margin error) is $w_1^2 + w_2^2$.
- Why so?

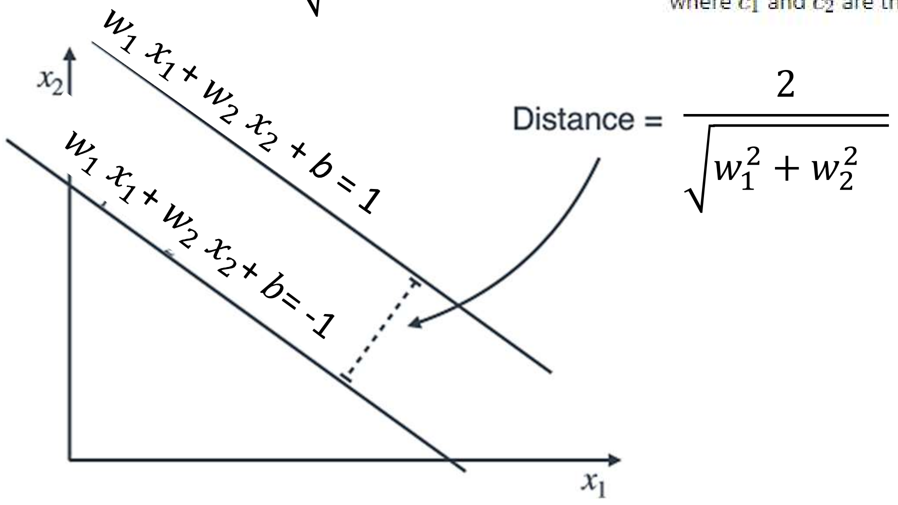
Distance (or margin) error function
- trying to space our two lines as far apart as possible

- The perpendicular distance or margin between the two lines is precisely

$$\frac{2}{\sqrt{w_1^2 + w_2^2}}$$

$$\text{Distance} = \frac{|c_1 - c_2|}{\sqrt{w_1^2 + w_2^2}}$$

where c_1 and c_2 are the constant terms in the line equations.

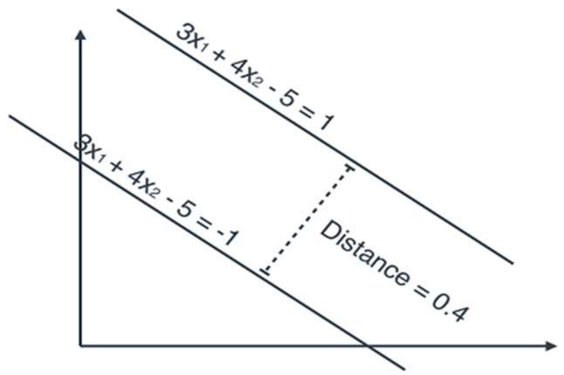

$w_1 x_1 + w_2 x_2 + b = 1$
$w_1 x_1 + w_2 x_2 + b = -1$
x_2
x_1
Distance = $\frac{2}{\sqrt{w_1^2 + w_2^2}}$

- Knowing this
- When $w_1^2 + w_2^2$ is large, error is large $\leftrightarrow$distance is small.
- When $w_1^2 + w_2^2$ is small, error is small $\leftrightarrow$ distance is large.

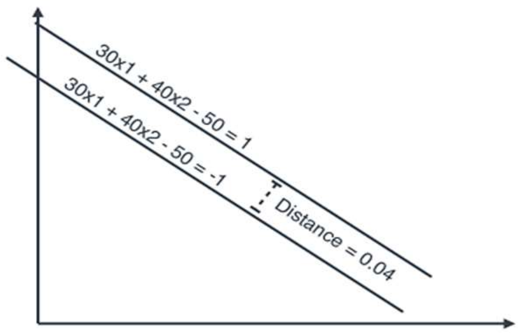
Distance (or margin) error function

- trying to space our two lines as far apart as possible

- Since lines required to be as far apart as possible, this term $w_1^2 + w_2^2$ is a good error function, as it provides large values for the bad classifiers (those where the lines are close), and small values for the good classifiers (those where the lines are far).
- Consider two examples of classifiers, one with a large error function and one with a small error function. Recall that a classifier's job is to make a prediction of a label based on the features.



Error function = 25
Good classifier

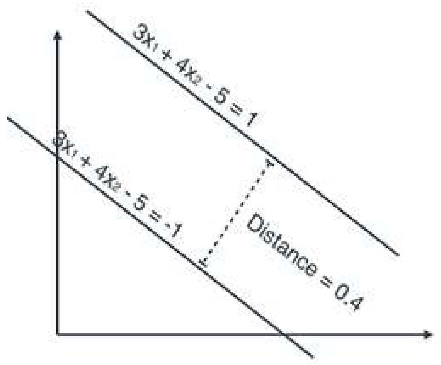


Error function = 2500
Bad classifier

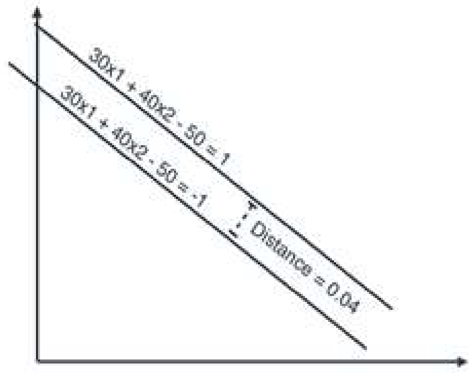
Distance (or margin) error function
- trying to space our two lines as far apart as possible

- In this case, the line is a classifier because it predicts the points on one side of the line as having a positive label, and the points on the other side as having a negative label. These two classifiers have the following equations:

- **Classifier 1:**
- Line 1: $3x_1$+ $4x_2$ - 5 = 1
- Line 2: $3x_1$+ $4x_2$ - 5 = -1



- **Classifier 2:**
- Line 1: $30x_1$+ $40x_2$ - 50 = 1
- Line 2: $30x_1$+ $40x_2$ - 50 = -1



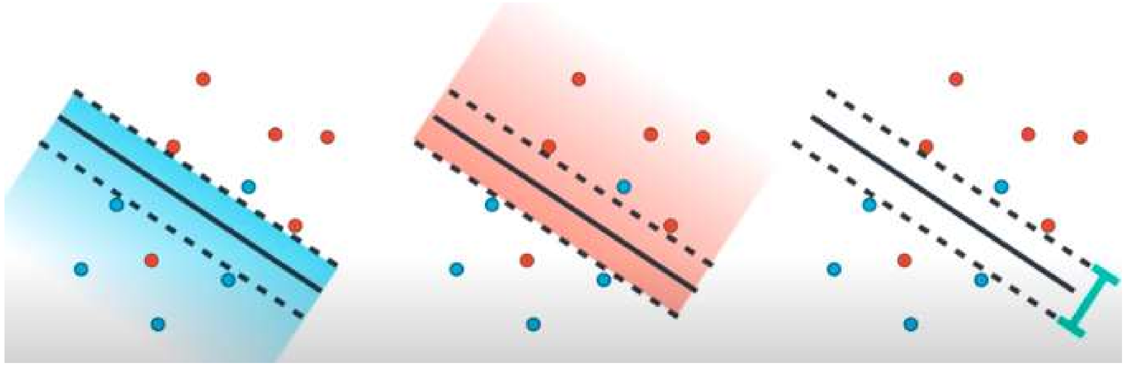
Distance (or margin) error function
- trying to space our two lines as far apart as possible

- Calculate the distance error function of each one.
- Classifier 1:
- Distance error function = 25
- Classifier 2:
- Distance error function = 2500

- Note that the lines are much closer in classifier 2 than in classifier 1, which makes classifier 1 a much better classifier (from the distance perspective).
  - The distance between the lines in classifier 1 is 0.4, whereas in classifier 2 it is 0.04.

Adding the two error functions - SVM error function

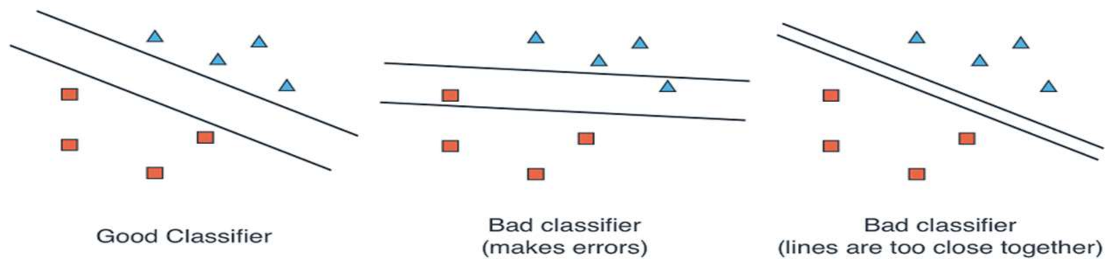
- Combine classification error function and a distance error function to build an error function that make sure that both goals achieved: classifying points in best possible way, and with boundaries that are far apart from each other.
- In order to obtain this error function, add the classification error function and the distance (or margin) error function. Then
- Error = Classification Error + Distance (or margin) Error



- *A **good classifier** must then try to make very few classification errors, as well as try to keep the lines as far apart as possible.*

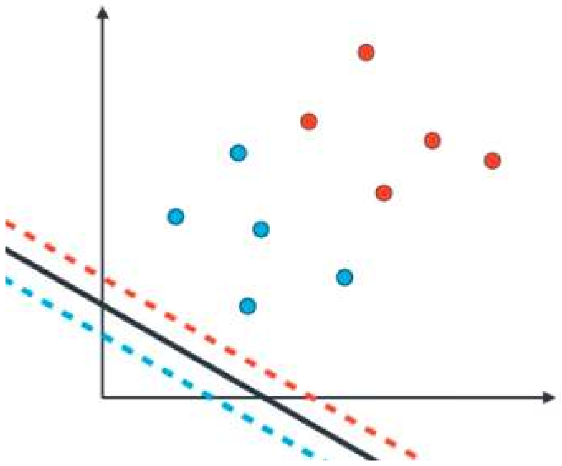
Example:

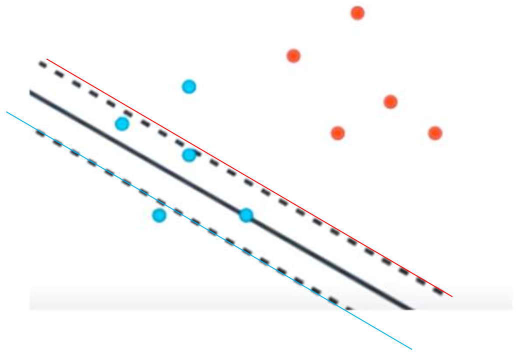
- Left: a good classifier, which consists of two well spaced lines and classifies all points correctly. Middle: A bad classifier which misclassifies two points. Right: A bad classifier which consists of two lines that are too close together.



- In Figure, three binary classifiers for the same 2D dataset.
- The one on the left is a good classifier, since it classifies the data well, and the lines are far apart, reducing the likelihood of errors.
- The classifier in the middle makes some errors (since there is a triangle underneath the top line, and a square over the bottom line), so it is not a good classifier.
- The classifier in the right classifies the points correctly, but the lines are too close together, so it is also not a good classifier.

SVM Algorithm





Step 1: Start with a random line of equation:

$$w_1\ x_1 + w_2\ x_2 + b = 0$$

Draw parallel lines with equations:

$$w_1\ x_1 + w_2\ x_2 + b = +1, \text{ and}$$

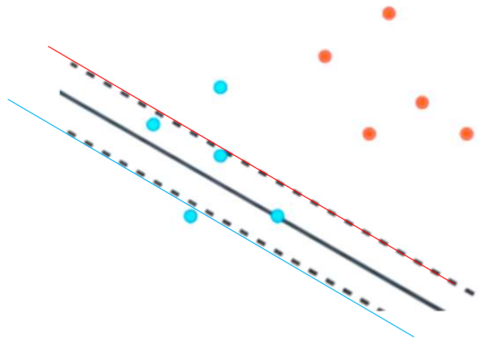
$$w_1\ x_1 + w_2\ x_2 + b = -1$$

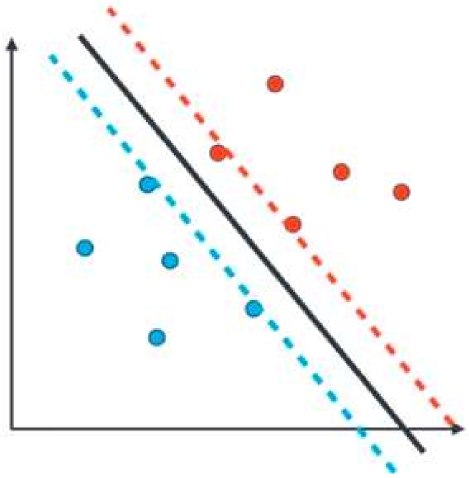
Step 2: Pick a large number. 1000 (number of repetitions, or epochs)

Step 3: Pick a learning rate 0.01

)

SVM Algorithm (cont.)

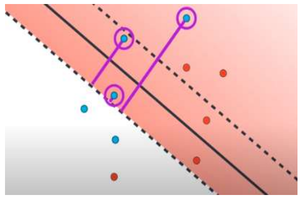




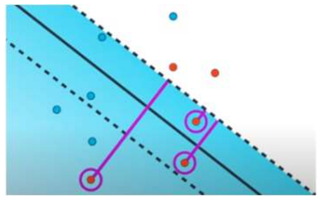
Step 5: (repeat 1000 times)

- Pick random point (p,q)
- If point is correctly classified
  - Do nothing

- If point is blue, and $w_1\ x_1 + w_2\ x_2 + b > 0$
  - Subtract 0.01p to w_1
  - Subtract 0.01q to w_2
  - Subtract 0.01 to b



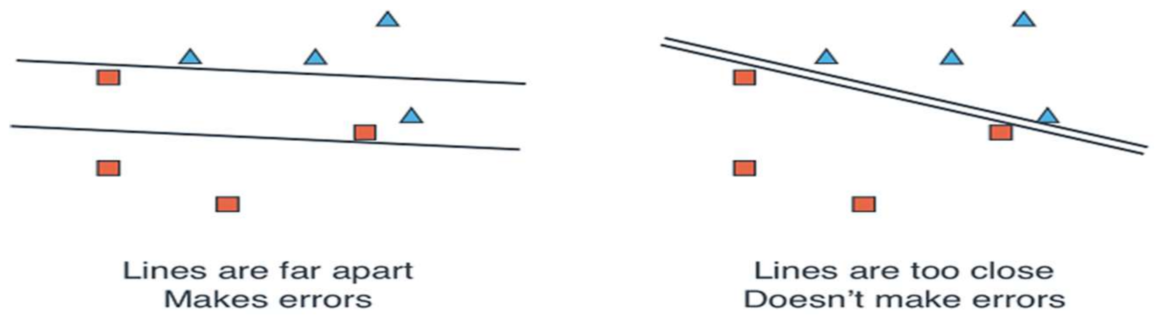
- If point is red, and $w_1\ x_1 + w_2\ x_2 + b < 0$
  - Add 0.01p to w_1
  - Add 0.01q to w_2
  - Add 0.01 to b



Step 6: Use the lines that separate the data.

Using the C parameter to tune and improve model

- The C parameter is used in cases where required to train an SVM which pays more weight to classification than to distance
- To build a good SVM classifier - the classifier with few classification errors as possible while keeping the lines as far apart as possible.
- But what if we have to sacrifice one for the benefit of the other?
- Example: Both of these classifiers have one strength and one weakness. The one in the left has the lines well spaced (strength), but it misclassifies some points (weakness). The one in the right has the lines too close together (weakness), but it classifies all the points correctly (strength).



Using the C parameter to tune and improve model

- Sometimes we want

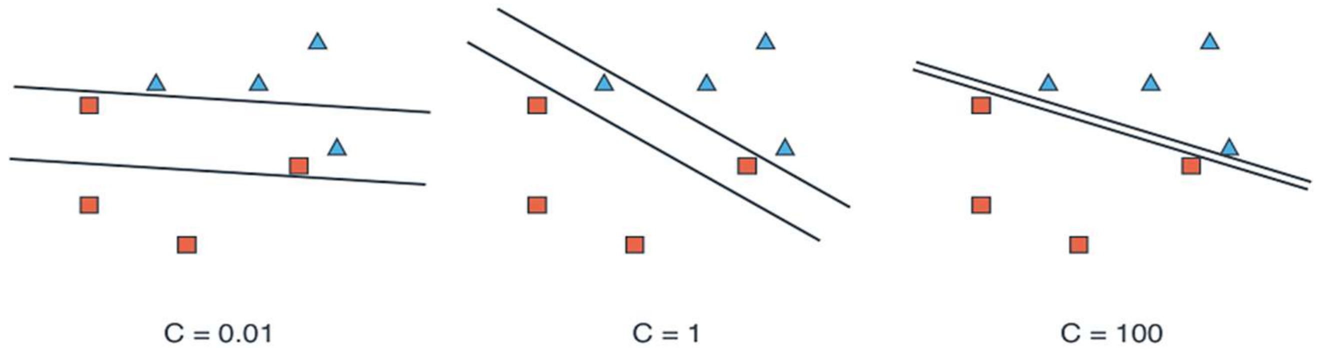
  - a classifier that makes as few errors as possible, even if the lines are too close, OR

  - a classifier that keeps the line apart, even if it makes a few errors.

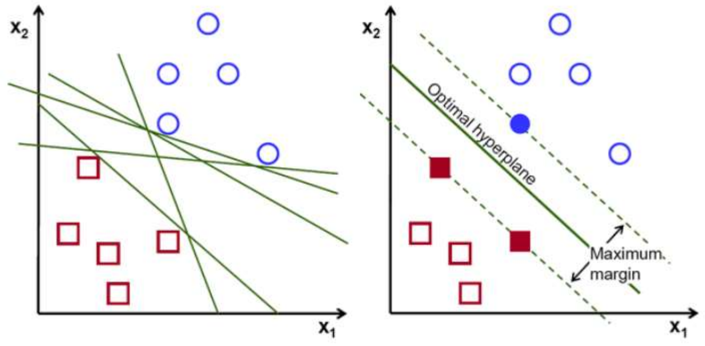
- How to control this? - Use a parameter C and update the error formula by multiplying the classification error by C:
  - Error formula = C * (Classification Error) + (Distance Error).
- If C is a large value, then the error formula is dominated by the classification error, so our classifier will focus more on classifying the points correctly.
- If C is a small value, then the formula is dominated by the distance (or margin) error, so our classifier will focus more on keeping the lines far apart.

Using the C parameter to tune and improve model

- Example: Different values of C toggle between a classifier with well spaced lines, and one that classifies points correctly.
- The classifier on the left has a small value of C = 0.01, and the lines are well spaced, but it makes mistakes.
- The classifier on the right has a large value of C = 100, and it classifies points correctly, but the lines are too close together.
- The classifier in the middle has a medium value of C = 1, is a good middle ground between the other two.
- In real life, C is a hyperparameter that train using grid search.



Support Vectors (points closest to decision boundary)



- Minimum Distance of the training instance to the decision surface, is known as margin
- The line that maximizes the minimum margin- Optimal hyperplane
- This maximum-margin separator is determined by a subset of the datapoints or vectors.
  - called "support vectors"
  - Support vectors actually determine the equation of the hyperplane(since these vectors support the hyperplane)
  - we use the support vectors to decide which side of the separator a test case is on.

SVM-Mathematical Properties

- Functional Margin
- Geometric Margin

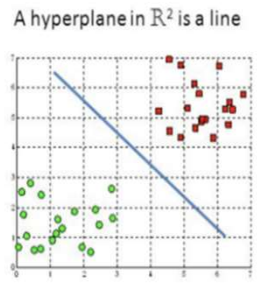
Functional Margin

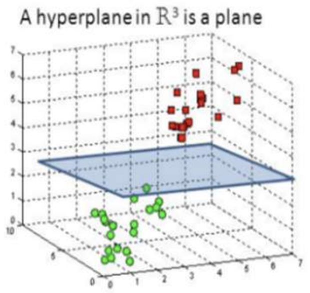
- Given a **single** training example ($x^{(i)}$, $y^{(i)}$), we define the **functional margin of Hyperplane defined by (w, b)** with respect to the training example as

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b)$$

if $y^{(i)} = 1$, want $w^T x^{(i)} + b >> 0$

if $y^{(i)} = -1$, want $w^T x^{(i)} + b << 0$





- If $\hat{\gamma}^{(i)} >> 0$ we classified ($x^{(i)}$, $y^{(i)}$) Correctly.

  $h(x^{(i)}) = y^{(i)}$ $\hat{\gamma}^{(i)} >> 0$

- Hence, a large functional margin represents a confident and a correct prediction.

Functional Margin (Formal Definition)

- Given a **training set** S = $\{(x^{(i)}, y^{(i)}) ; i = 1, . . . , m\}$, **the function margin** of (w, b) with respect to S can therefore be written:
- This represents the lowest relative "confidence" of all classified points,

Problem:

$$w \longrightarrow 2w$$
$$b \longrightarrow 2b$$

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b)$$

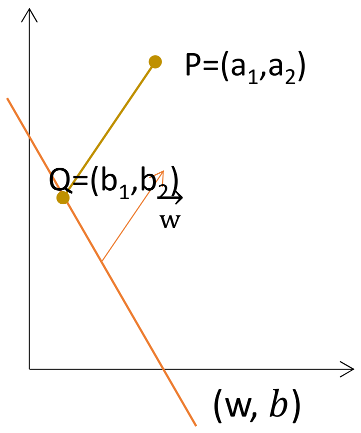
Solution:

$||w||_2 = 1$; where $||w||_2$ is the euclidean length

$$(w, b) = \left(\frac{w}{||w||_2}, \frac{b}{||w||_2}\right)$$

Geometric Margin

- But we do not just rely on if we are correct or not in this classification. We need to know how correct we are, or what is the degree of confidence that we have in this classification.
- Geometric margin gives a magnitude to our confidence.
- Geometric margin represents the smallest distance from to the hyperplane.
    - For a decision surface defined by (w, b)
    - the vector orthogonal to it is given by w.
    - The unit length orthogonal vector is $\frac{\mathrm{w}}{\|w\|}$
    - $P = Q + \gamma \frac{\mathrm{w}}{\|w\|}$



Geometric Margin

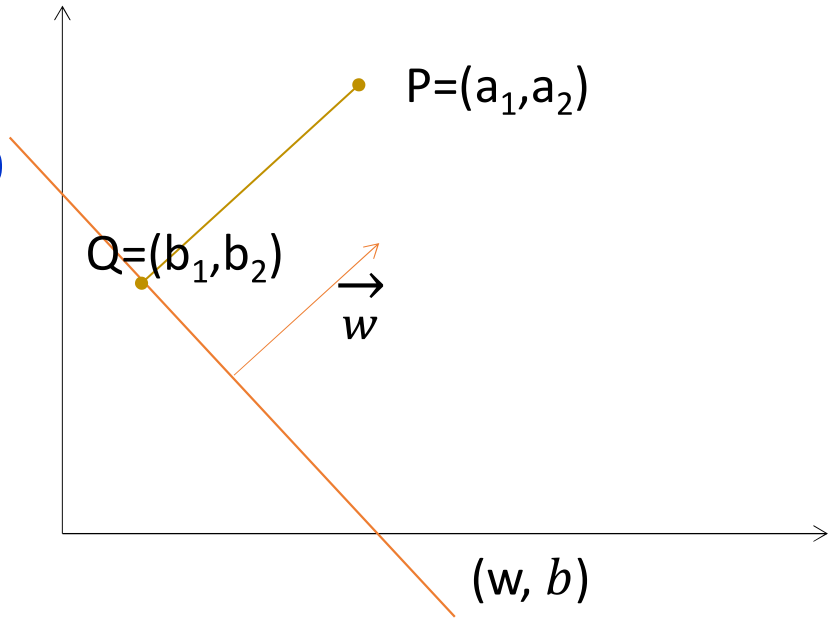
$$P = Q + \gamma \frac{\mathrm{w}}{\|w\|}$$

$$\left(b_1^{(i)}, b_2^{(i)}\right) = \left(a_1^{(i)}, a_2^{(i)}\right) - \gamma^{(i)} \frac{\mathrm{w}}{\|w\|}$$

$$\rightarrow \mathrm{w}^{\mathrm{T}}\left(\left(a_1^{(i)}, a_2^{(i)}\right) - \gamma^{(i)} \frac{\mathrm{w}}{\|w\|}\right) + b = 0$$

$$\rightarrow \gamma^{(i)} = \frac{\mathrm{w}^{\mathrm{T}}\left(a_1^{(i)}, a_2^{(i)}\right) + b}{\|w\|}$$

$$= \frac{w}{\|w\|}^{T}\left(a_1^{(i)}, a_2^{(i)}\right) + \frac{b}{\|w\|}$$



$$\gamma^{(i)} = t.\left(\frac{w}{\|w\|}^{T}\left(a_1^{(i)}, a_2^{(i)}\right) + \frac{b}{\|w\|}\right)$$

Geometric margin of (w, b) wrt training set $\{(x_1, t_1), (x_2, t_2), \dots, (x_N, t_N)\}$

- smallest of the geometric margins of individual points.

For $\|w\|$=1, Functional Margin = Geometric Margin

Optimal margin classifier

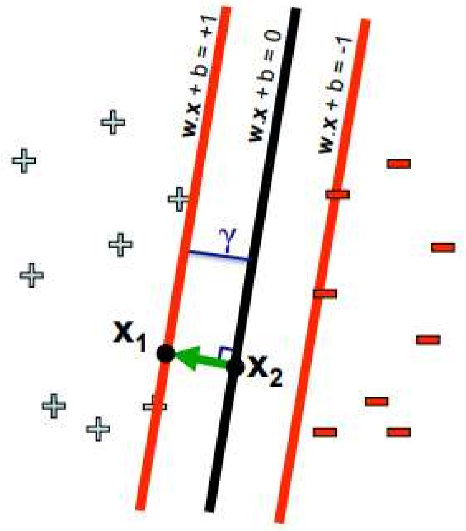
- Objective: a classifier (linear separator) with as big a margin as possible.
- We are only interested in solutions for which all data points are correctly classified, so that

$$\max_{\gamma, w, b} \gamma$$

$$\text{s.t}\; y^{(i)}(w^T x^{(i)} + b) \geq \widehat{\gamma}^{(i)} \quad i = 1, .., m$$

$$\|w\| = 1$$

ϒ(geometric margin) as a function of w



$$\begin{array}{r} w \cdot x_1 + b = 1 \\ - \quad w \cdot x_2 + b = 0 \\ \hline w \cdot (x_1 - x_2) = 1 \end{array}$$

Plug in

We also know that:

$$x_1 - x_2 = \gamma \frac{w}{||w||}$$

$$1 = w \cdot \left(\gamma \frac{w}{||w||} \right) = \frac{\gamma}{||w||} w \cdot w = \gamma ||w||$$

$$\boxed{\text{So, } \gamma = \frac{1}{||w||}}$$

(assuming there is a data point on the **w.x** + b = +1 or -1 line)

So maximum margin is $\frac{2}{||w||}$

Optimal margin classifier

$$\max_{\gamma, w, b} \gamma$$

$$\text{s.t}\; y^{(i)}(w^T x^{(i)} + b) \geq \hat{\gamma}^{(i)} \quad i = 1,..,m$$

$$\text{s.t}\; y^{(i)}(w^T x^{(i)} + b) \geq 1 \quad \text{As } \gamma = \frac{1}{\|w\|}$$

Scale such that γ = 1

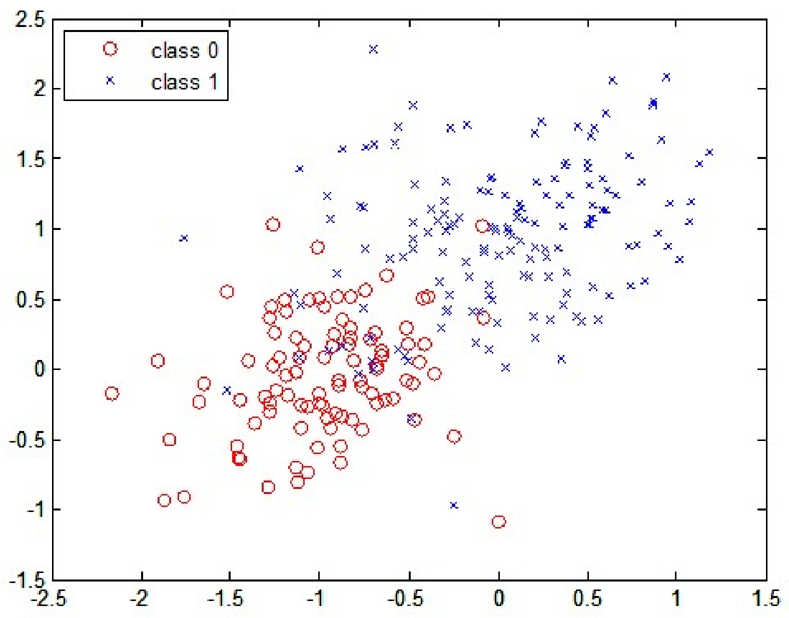
Maximize $\frac{1}{\|w\|}$ which is equivalent to minimize $\|w\|$

since||w|| is not differentiable at 0, we'll minimize (1/2)*||w||² instead, whose derivative is just w. Optimization algorithms work much better on differentiable functions

SVM Primal Optimization problem:

$$\boxed{\min \frac{\|w\|^2}{2} \quad \text{such that } y^{(i)}(w^T x^{(i)} + b) \geq 1 \quad i = 1,..,m}$$

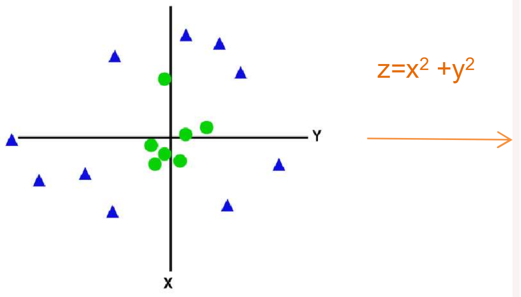
Limitations of previous SVM formulation

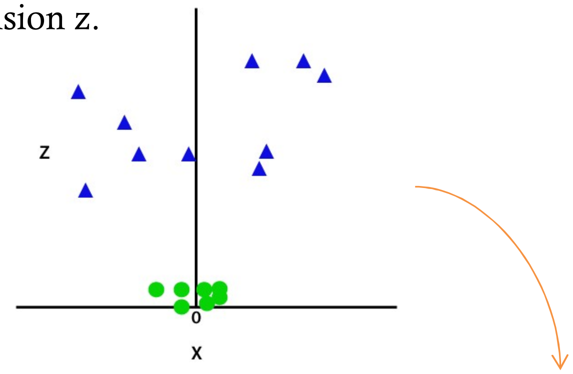


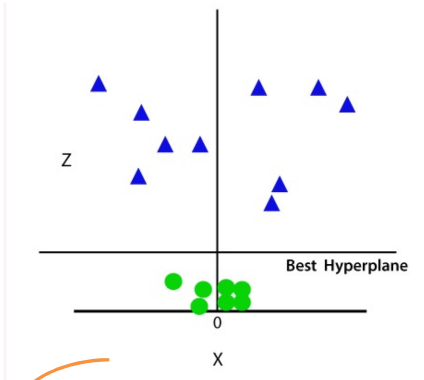
- What if the data is not linearly separable?
- Or noisy data points?

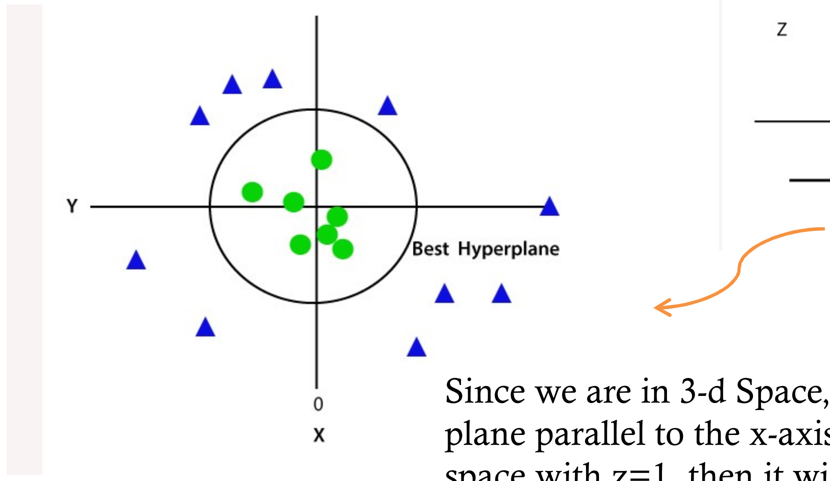
Idea: Map data to a high dimensional space where it is linearly separable.

Add a third dimension z.



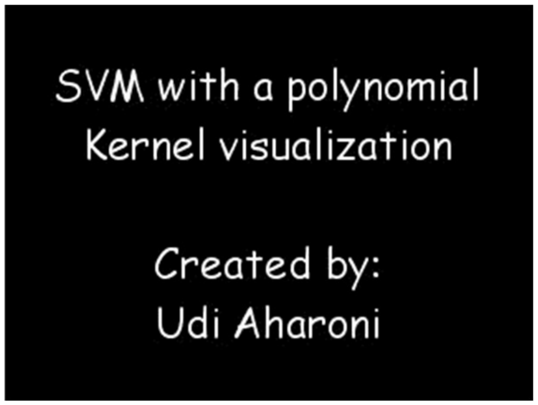






Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:

Visualization



SVM with a polynomial
Kernel visualization

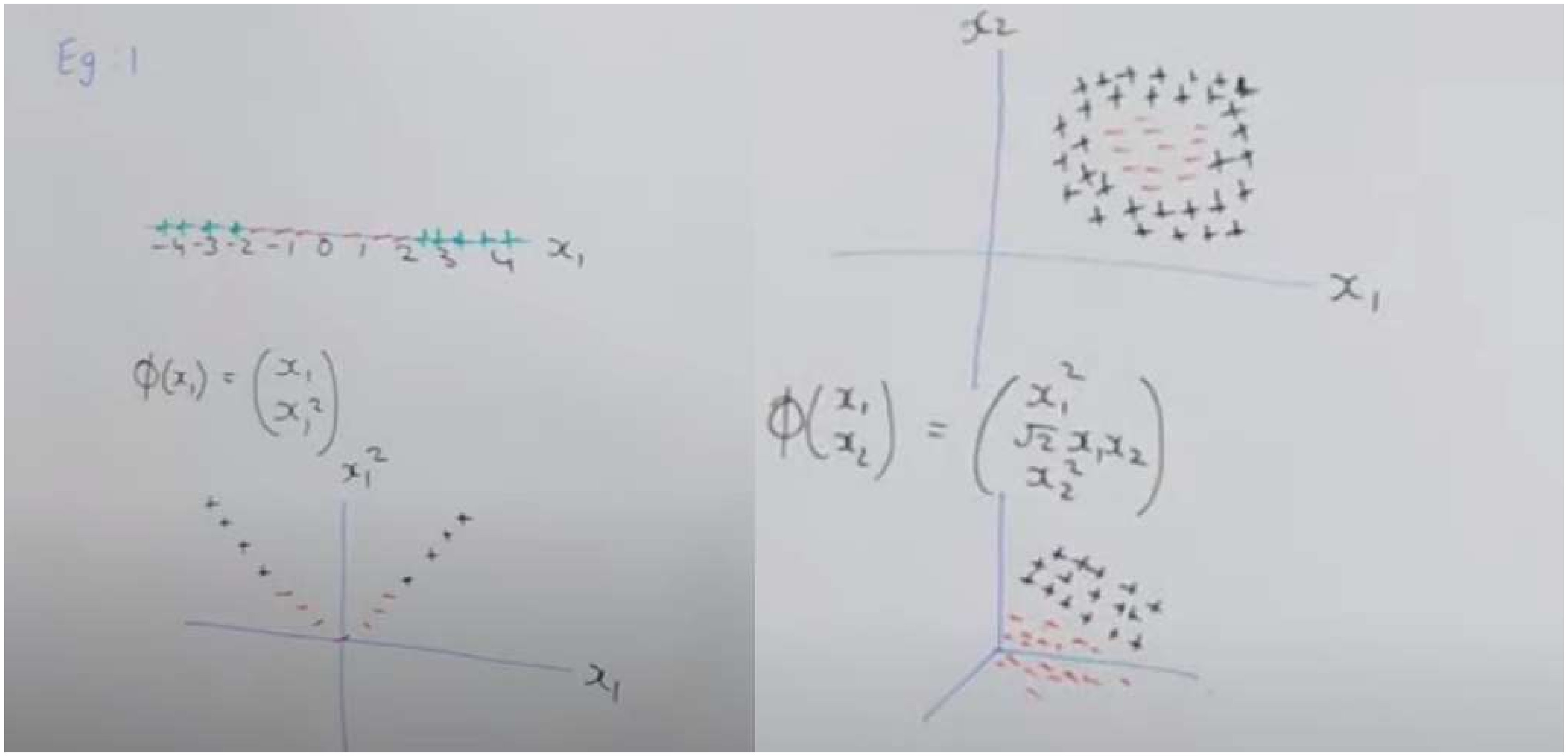
Created by:
Udi Aharoni

What if the decision boundary is truly non-linear?

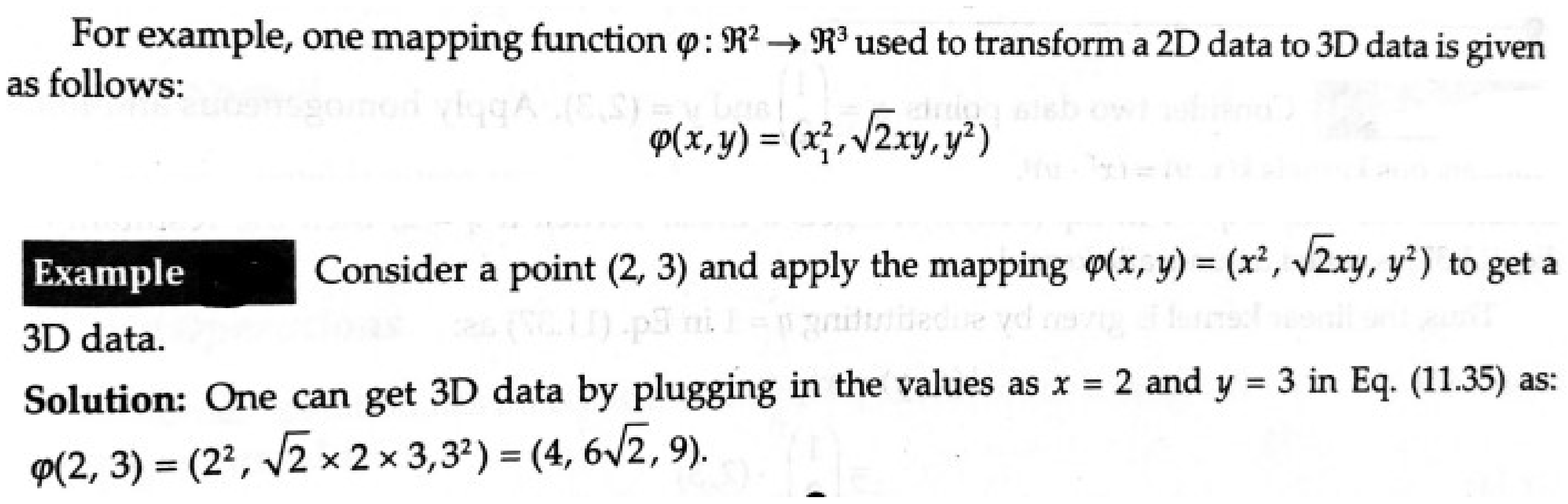
Idea: Map data to a high dimensional space where it is linearly separable.

- Using a bigger set of features will make the computation slow?
- The “kernel” trick to make the computation fast.

Dimension Mapping


Eg : 1
-4 -3 -2 -1 0 1 2 3 4
x_1
$\phi(x_1) = \begin{pmatrix} x_1 \\ x_1^2 \end{pmatrix}$
x_1^2
x_1
x_2
x_1
$\phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \end{pmatrix}$

Dimension Mapping



For example, one mapping function $\varphi : \Re^2 \rightarrow \Re^3$ used to transform a 2D data to 3D data is given as follows:

$$\varphi(x, y) = (x_1^2, \sqrt{2}xy, y^2)$$

Example Consider a point (2, 3) and apply the mapping $\varphi(x, y) = (x^2, \sqrt{2}xy, y^2)$ to get a 3D data.

Solution: One can get 3D data by plugging in the values as $x = 2$ and $y = 3$ in Eq. (11.35) as:
$\varphi(2, 3) = (2^2, \sqrt{2} \times 2 \times 3, 3^2) = (4, 6\sqrt{2}, 9)$.

Limitation of Dimension Mapping

- Mapping involves more computation and learning costs
- There is no general rule , that what transformations should be applied
- If the data is large, mapping process takes huge amount of time.

Kernel

- Kernels are used to compute the value without transforming the data.
- The SVM kernel is a function that takes low dimensional input space and transforms it into higher-dimensional space, i.e it converts non separable problem to separable problem. It is mostly useful in non-linear separation problems.
- Kernels are used to manipulate data using dot product at higher dimensions.
- Different kernel functions can be specified for the decision functions and its possible to specify custom kernels.

Commonly-used kernel functions

- Linear kernel: $K\left(x_i, x_j\right) = x_i . x_j$
- Polynomial of power *p*:

$$K\left(x_i, x_j\right) = (1 + x_i . x_j)^p$$

- Gaussian (radial-basis function):

$$K\left(x_i, x_j\right) = e^{-\frac{\left\|x_i - x_j\right\|^2}{2\sigma^2}}$$

- Sigmoid

$$K\left(x_i, x_j\right) = \tanh(\beta_0 x_i . x_j + \beta_1)$$

In general, functions that satisfy *Mercer's condition* can be kernel functions.

Linear Kernel

Linear kernels are of the type $k(x, y) = x^T y$ where x and y are two vectors.

Polynomial Kernel

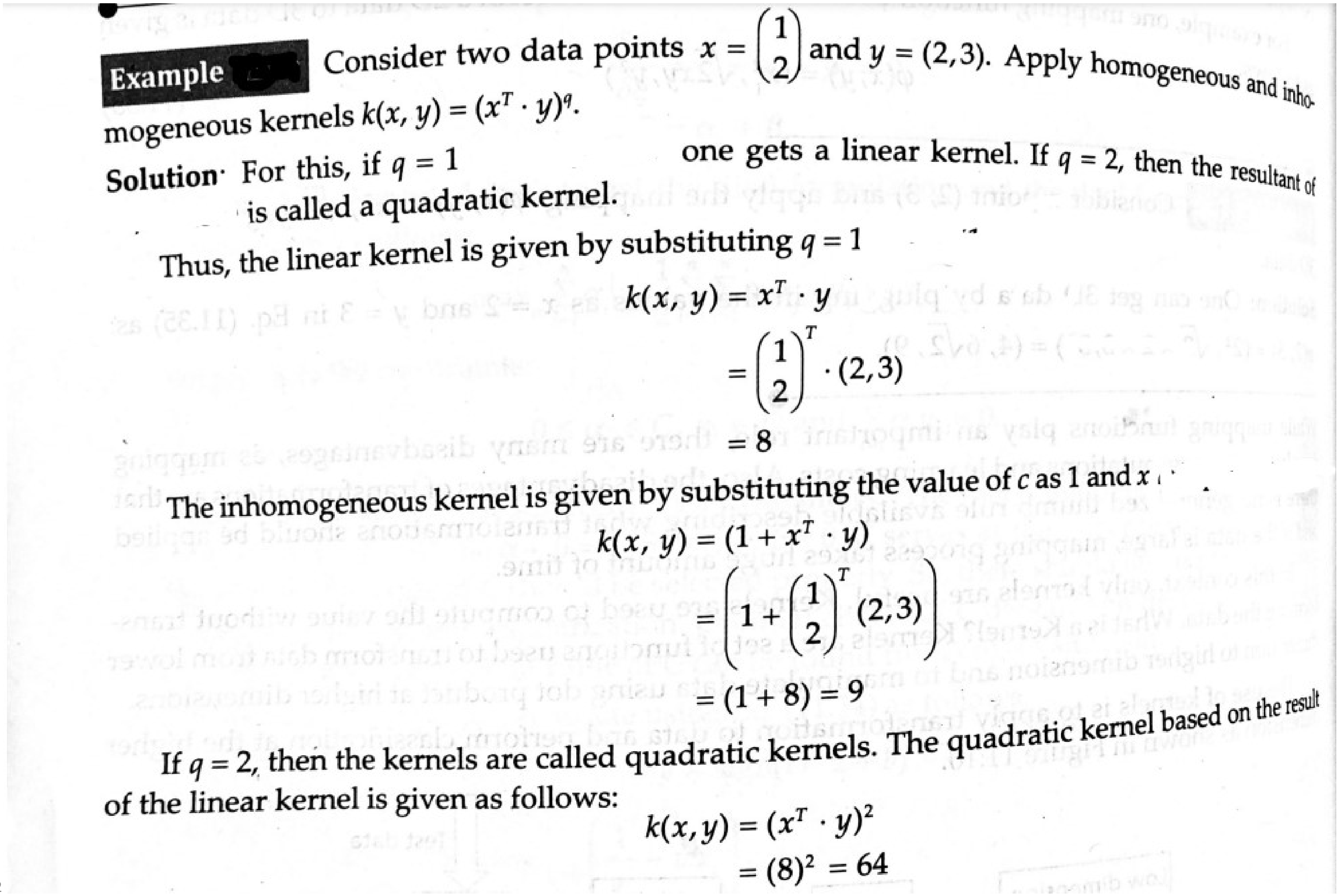
Polynomial kernels are of the type:

$$k(x, y) = (x^T \cdot y)^q.$$

This is called homogeneous kernel.

For inhomogeneous kernels, this is given as:

$$k(x, y) = (c + x^T \cdot y)^q.$$



Example [redacted] Consider two data points $x = \begin{pmatrix}1\\2\end{pmatrix}$ and $y = (2,3)$. Apply homogeneous and inhomogeneous kernels $k(x, y) = (x^T \cdot y)^q$.

Solution: For this, if $q = 1$ one gets a linear kernel. If $q = 2$, then the resultant of is called a quadratic kernel.

Thus, the linear kernel is given by substituting $q = 1$

$$k(x,y) = x^T \cdot y$$
$$= \begin{pmatrix}1\\2\end{pmatrix}^T \cdot (2,3)$$
$$= 8$$

The inhomogeneous kernel is given by substituting the value of c as 1 and x

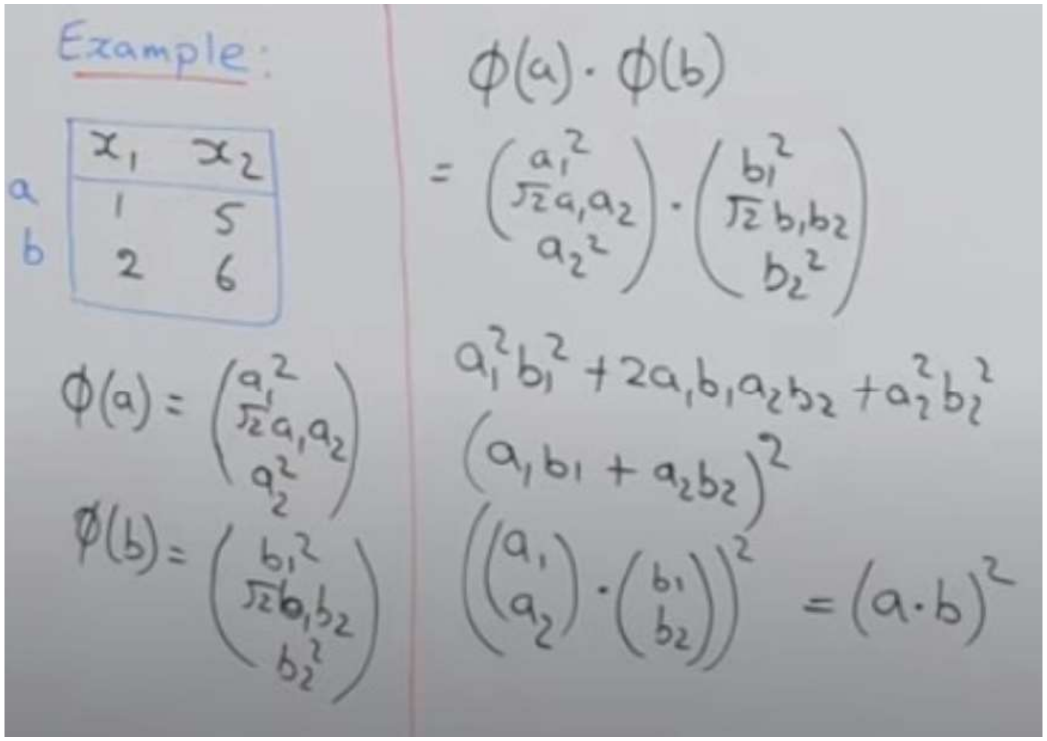
$$k(x, y) = (1 + x^T \cdot y)$$
$$= \left(1 + \begin{pmatrix}1\\2\end{pmatrix}^T (2,3)\right)$$
$$= (1 + 8) = 9$$

If $q = 2$, then the kernels are called quadratic kernels. The quadratic kernel based on the result of the linear kernel is given as follows:

$$k(x,y) = (x^T \cdot y)^2$$
$$= (8)^2 = 64$$

Kernel trick

- Replace the dot product in mapping function by kernel function.



Example:

	x_1	x_2
a	1	5
b	2	6

$$\phi(a) = \begin{pmatrix} a_1^2 \\ \sqrt{2}a_1a_2 \\ a_2^2 \end{pmatrix}$$

$$\phi(b) = \begin{pmatrix} b_1^2 \\ \sqrt{2}b_1b_2 \\ b_2^2 \end{pmatrix}$$

$$\phi(a) \cdot \phi(b)$$

$$= \begin{pmatrix} a_1^2 \\ \sqrt{2}a_1a_2 \\ a_2^2 \end{pmatrix} \cdot \begin{pmatrix} b_1^2 \\ \sqrt{2}b_1b_2 \\ b_2^2 \end{pmatrix}$$

$$a_1^2b_1^2 + 2a_1b_1a_2b_2 + a_2^2b_2^2$$

$$(a_1b_1 + a_2b_2)^2$$

$$\left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 = (a \cdot b)^2$$

Kernel

- Original input attributes is mapped to a new set of input features via feature mapping Φ.
- Since the algorithm can be written in terms of the scalar product, we replace $x_a . x_b$ with $\phi(x_a) . \phi(x_b)$
- For certain Φ's there is a simple operation on two vectors in the low-dim space that can be used to compute the scalar product of their two images in the high-dim space

$$K(x_a, x_b) = \phi(x_a) . \phi(x_b)$$

Let the kernel do the work rather than do the scalar product in the high dimensional space.

Kernel Trick

This simple trick allowed the linear SVM to capture non-linear relationship in the data.

- Write the whole algorithm in terms of $<x^{(i)},x^{(j)}>$
- Let there be some mapping from your original input features x to some high dimensional set of features $\phi(x)$.
- Find a way to compute $K(x,z)= \phi(x)^T\phi(z)$, this is called the kernel function.
- Replace $<x,z>$ in algorithm with $K(x,z)$

Optimal margin classifier

- Hard margin (hard condition, no relaxation) SVM optimization objective

$$\min \frac{\|w\|^2}{2} \quad \text{such that } y^{(i)}(w^T x^{(i)} + b) \geq 1 \quad i = 1,..,m$$

- $x^{(i)} \in R^{100}$
- Assumption: $w = \sum_{i=1}^{m} \alpha_i \; x^{(i)}$ W can be expressed as linear combination of training examples

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

$y^{(i)}$=+/- 1

Primal optimization problem

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \qquad \text{.....................(1)}$$

Optimization Problem:

$$\min \frac{\|w\|^2}{2} \quad \text{such that } y^{(i)}(w^T x^{(i)} + b) \geq 1 \quad i = 1,..,m \qquad \text{.....................(2)}$$

As we know that,

$$\|w\|^2 = w^T w \qquad \text{.....................(3)}$$

Substituting eqn (1) in eqn (2) & eqn (3) :

- Substituting eqn (1) in eqn (2) & eqn (3) :

$$\min_{w,b} \frac{1}{2}\left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}\right)^T \left(\sum_{j=1}^{m} \alpha_j y^{(j)} x^{(j)}\right)$$

$$\min_{w,b} \frac{1}{2}\left(\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)T} x^{(j)}\right)$$

$=<x^{(i)}, x^{(j)}>$

Inner product notation

such that $y^{(i)}(w^T x^{(i)} + b) \geq 1$

Expressing algorithm in the terms of inner product between features x
Is the key step need to derive kernels

$$y^{(i)}\left(\left(\sum_{j} \alpha_j y^{(j)} x^{(j)}\right)^T x^{(i)} + b\right) \geq 1$$

$$y^{(i)}\left(\sum_{j} \alpha_j y^{(j)} < x^{(j)}, x^{(i)} > + b\right) \geq 1$$

Dual optimization problem

Using Convex optimization theory, we get dual optimization function as:

$$\text{Max}_\alpha \ W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} y^{(i)} y^{(j)} \alpha_i \alpha_j \left\langle x^{(i)}, x^{(j)} \right\rangle$$

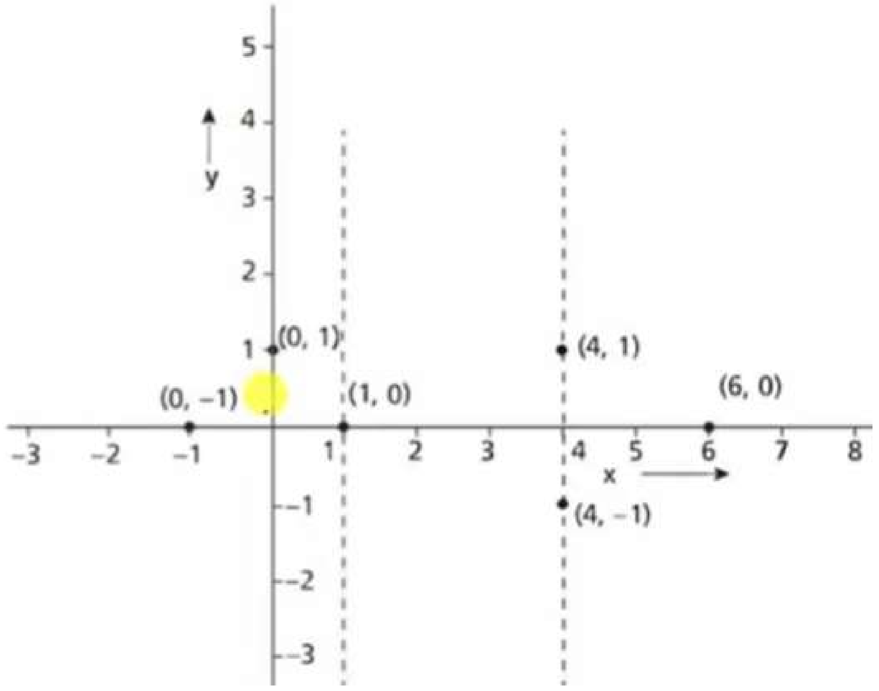
$$\text{s.t } \alpha_i \geq 0 \quad \sum \alpha_i y^{(i)} = 0$$

Instead of directly solving for **w and b**, it introduces **Lagrange multipliers αi**.

The dual problem **only depends on dot products** between feature vectors, i.e., ⟨x(i),x(j)⟩.

Allows the use of **Kernels**, enabling SVM to work in high-dimensional and non-linearly separable spaces.

Numerical: Points are given on 2-d plane, Draw an optimal hyperplane to classify the points



Sol: step 1: Observe support vectors

$$s_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, s_2 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}, s_3 = \begin{pmatrix} 4 \\ -1 \end{pmatrix}$$

- The augmented vector can be obtained by adding bias

$$\tilde{s}_1 = \begin{pmatrix}1\\0\\1\end{pmatrix},\ \tilde{s}_2 = \begin{pmatrix}4\\1\\1\end{pmatrix},\ \tilde{s}_3 = \begin{pmatrix}4\\-1\\1\end{pmatrix}$$

From these, a set of three equations can be obtained based on these three support vectors as follows:

$$\alpha_1\tilde{s}_1\tilde{s}_1 + \alpha_2\tilde{s}_2\tilde{s}_1 + \alpha_3\tilde{s}_3\tilde{s}_1 = -1$$
$$\alpha_1\tilde{s}_1\tilde{s}_2 + \alpha_2\tilde{s}_2\tilde{s}_2 + \alpha_3\tilde{s}_3\tilde{s}_2 = +1$$
$$\alpha_1\tilde{s}_1\tilde{s}_3 + \alpha_2\tilde{s}_2\tilde{s}_3 + \alpha_3\tilde{s}_3\tilde{s}_3 = +1$$

Substituting we get

$$\alpha_1\begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} + \alpha_2\begin{pmatrix}4\\1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} + \alpha_3\begin{pmatrix}4\\-1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix}$$
$$= 2\alpha_1 + 5\alpha_2 + 5\alpha_3 = -1$$

$$\alpha_1\begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}4\\1\\1\end{pmatrix} + \alpha_2\begin{pmatrix}4\\1\\1\end{pmatrix}\begin{pmatrix}4\\1\\1\end{pmatrix} + \alpha_3\begin{pmatrix}4\\-1\\1\end{pmatrix}\begin{pmatrix}4\\1\\1\end{pmatrix}$$
$$= 5\alpha_1 + 18\alpha_2 + 16\alpha_3 = +1$$

$$\alpha_1\begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}4\\-1\\1\end{pmatrix} + \alpha_2\begin{pmatrix}4\\1\\1\end{pmatrix}\begin{pmatrix}4\\-1\\1\end{pmatrix} + \alpha_3\begin{pmatrix}4\\-1\\1\end{pmatrix}\begin{pmatrix}4\\-1\\1\end{pmatrix}$$
$$= 5\alpha_1 + 16\alpha_2 + 18\alpha_3 = +1$$

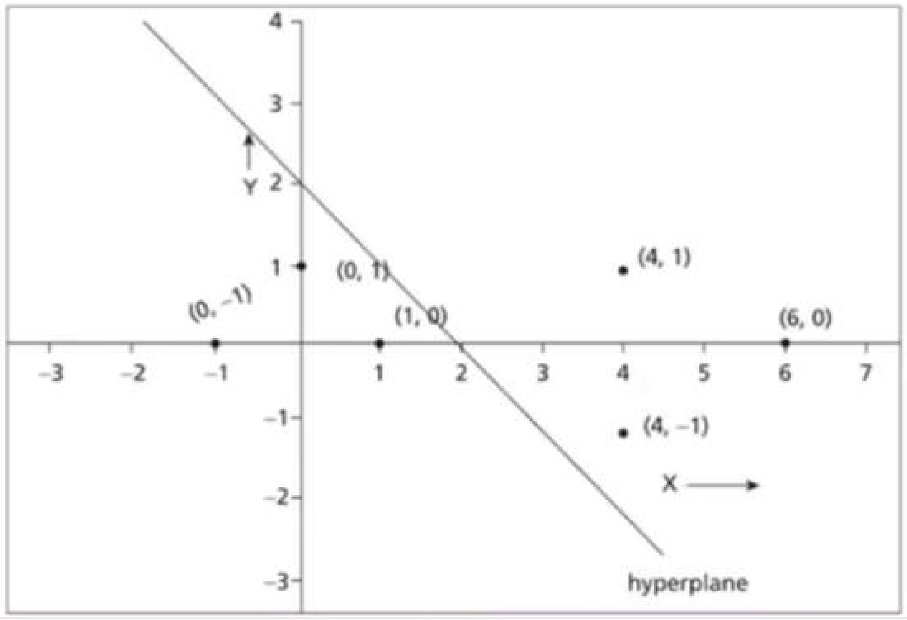
Solving these three simultaneous equations with three unknowns yields the values:

$$\alpha_1 = -3$$
$$\alpha_2 = +1$$
$$\alpha_3 = 0$$

The optimal Hyperplane is given as:

$$w = \sum_{1}^{3} \alpha_i \times \tilde{s}_i$$
$$= -3 \times \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + 1 \times \begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix} + 0 \times \begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ -2 \end{pmatrix}$$

The hyperplane is (1, 1) with an offset -2.



Kernels as similarity metrics

- If x,z are "similar", then K(x,z)=$\phi(x)^T\phi(z)$ is large.
- If x,z are "dissimilar", then K(x,z)=$\phi(x)^T\phi(z)$ is small.

$$K(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$$

Gaussian kernel
or squared exp kernel

- When x and z are close, then K(x,z) is close to 1.
- When x and z are far apart, then K(x,z) is close to 0.

Necessary conditions for K(x,z) to be a KERNEL Function

1. K should be symmetric. $K(x,z)=K(z,x)$

2. $K(x,z)=\phi(x)^T\phi(z)$

   $\Rightarrow K(x,x)= \phi(x)^T\phi(x) \geq 0$

Let's consider $\{x^{(1)},....x^{(d)}\}$

Let's define $K \in R^{dxd}$, "kernel matrix"

Such that $K_{ij} = K(x^{(i)}, x^{(j)})$

Necessary conditions for K(x,z) to be a KERNEL Function

1. K should be symmetric. $K(x,z)=K(z,x)$

2. $K(x,z)=\phi(x)^T\phi(z)$

   $\Rightarrow K(x,x)= \phi(x)^T\phi(x) \geq 0$

Let's consider $\{x^{(1)},\dots.x^{(d)}\}$

Let's define $K \in R^{dxd}$, "kernel matrix"

Such that $K_{ij} = K(x^{(i)}, x^{(j)})$

- For any vector z

$$
\begin{aligned}
z^T K z &= \sum_i \sum_j z_i K_{ij} z_j \\
&= \sum_i \sum_j z_i \phi(x^{(i)})^T \phi(x^{(j)}) z_j \\
&= \sum_i \sum_j z_i \sum_k \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\
&= \sum_k \sum_i \sum_j z_i \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\
&= \sum_k \left(\sum_i z_i \phi_k(x^{(i)}) \right)^2 \\
&\geq 0
\end{aligned}
$$

This proves that $K \geq 0$

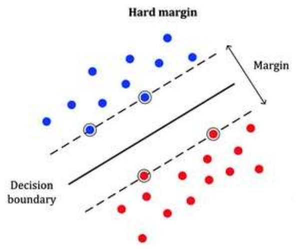
The kernel matrix K is positive semi-definite : **Sufficient Conditiion** for any function k to be a **valid kernel function**

Sufficient Condition

- Theorem (Mercer). Let $K : R^d \times R^d \rightarrow R$ be given. Then for K to be a valid (Mercer) kernel, it is necessary and sufficient that for any d points $\{x^{(1)}, \ldots, x^{(d)}\}$, $(d < \infty)$, the corresponding kernel matrix is symmetric positive semi-definite $(K \geq 0)$.

Hard Margin SVM

- Linear separable problem can be easily classified by hard margin SVM



- In the case of separable classes, we implicitly used an error function that gave infinite error if a data point was misclassified and zero error if it was classified correctly, and then optimized the model parameters to maximize the margin.
- Minimize $\frac{1}{2}\|\mathrm{w}\|^2 = \frac{1}{2}\mathrm{w.w}$ and *number of misclassifications*, i.e., minimize

$$\frac{1}{2}\ \mathrm{w.w} + \#(\text{training errors})$$

The hard margin SVM has two very important limitations:

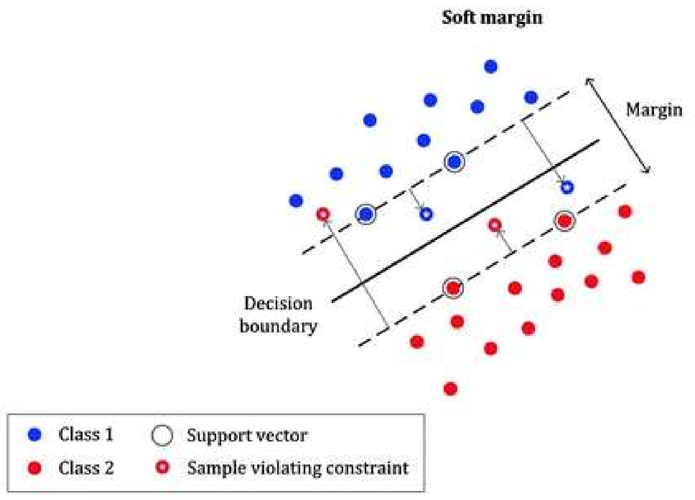
- it only works on linearly separable data;

- it is very sensible to outliers.

- But, In real-life applications we don't find any dataset which is linearly separable.

SVM – Maximum Margin with Noise
How to formulate?

- A way to deal with noise, we need to modify the SVM so as to allow some of the training points to be misclassified.
- We now modify this approach so that *data points are allowed to be on the 'wrong side' of the margin boundary, but with a penalty that increases with the distance from that boundary*.



Objective: To be minimized

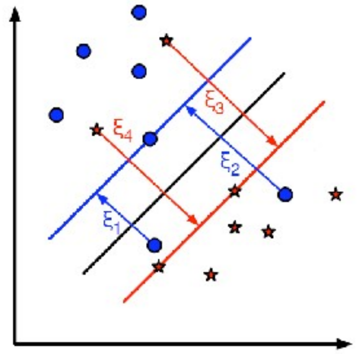
To make it flexible towards outliers or misclassification, Soft margin SVM is introduced.

- using the concept of slack variables $\epsilon^{(i)}$(which represents how much that particular data point can violate the margin)

we still want the margins to be as wide as possible, and we want the slack variables to be as small as possible to prevent margins' violations.

- Minimize

$$\frac{1}{2}\mathrm{w.w} + C(\text{distance of error points to their correct zones})$$



Soft margin SVM classifier objective:

~~Basic~~ optimization Problem:

$$\min_{\xi, w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{m} \xi_i$$

$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i$$

$$i = 1, \dots, m$$

$$\xi_i \geq 0, \quad i = 1, \dots, m$$

A new hyperparameter C is introduced that lets us control the tradeoff between the conflicting objectives. As C grows, it forces the optimization algorithm to find smaller values for ϵ.

Slack variables

- We introduce slack variables, $\xi_n \geq 0$ where $n = 1, \ldots, N$, with one slack variable for each training data point .
- These are defined by $\xi_n = 0$ for data points that are on margin boundary or correct side of the margin boundary and
- $\xi_n = | t_n - y(\mathrm{x}_n) |$ for other points.
- Thus a data point that is on the decision boundary $y(\mathrm{x}_n) = 0$ will have $\xi_n = 1$, and points with $\xi_n > 1$ will be misclassified.
- The exact classification constraints are then replaced with

$$t_n y(\mathrm{x}_n) \geq 1 - \xi_n, \; n = 1, \ldots, N$$

  in which the slack variables are constrained to satisfy $\xi_n \geq 0$.
- Data points for which $\xi_n = 0$ are correctly classified and are either on the margin or on the correct side of the margin.

Slack variables (cont.)

- Points for which $0 < \xi_n \leq 1$ lie inside the margin, but on the correct side of the decision boundary, and
- those data points for which $\xi_n > 1$ lie on the wrong side of the decision boundary and are misclassified.
- This is sometimes described as relaxing the hard margin constraint to give a soft margin and allows some of the training set data points to be misclassified.
- Note that while slack variables allow for overlapping class distributions (not linearly separable data), this framework is still sensitive to outliers because the penalty for misclassification increases linearly with ξ.
- **Objective:** To maximize the margin while softly penalizing points that lie on the wrong side of the margin boundary.

Dual optimization problem

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} y^{(i)} y^{(j)} \alpha_i \alpha_j \left\langle x^{(i)}, x^{(j)} \right\rangle$$

$$\text{s.t } \alpha_i \geq 0 \quad \sum \alpha_i y^{(i)} = 0$$

$$\max_{\alpha} \; w(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} y^{(i)} y^{(j)} \alpha_i \alpha_j \left\langle x^{(i)}, x^{(j)} \right\rangle$$

$$\text{s.t } \; 0 \leq \alpha_i \leq C, i = 1, \ldots m$$

$$\boxed{\sum_{i=1}^{m} \alpha_i y^{(i)} = 0}$$

- The solution of the discriminant function is

$$g(x) = \sum_{i \epsilon SV} \alpha_i K(x_i, x_j) + b$$

The SMO algorithm

The SMO algorithm can efficiently solve the dual problem.
First we discuss Coordinate Ascent.

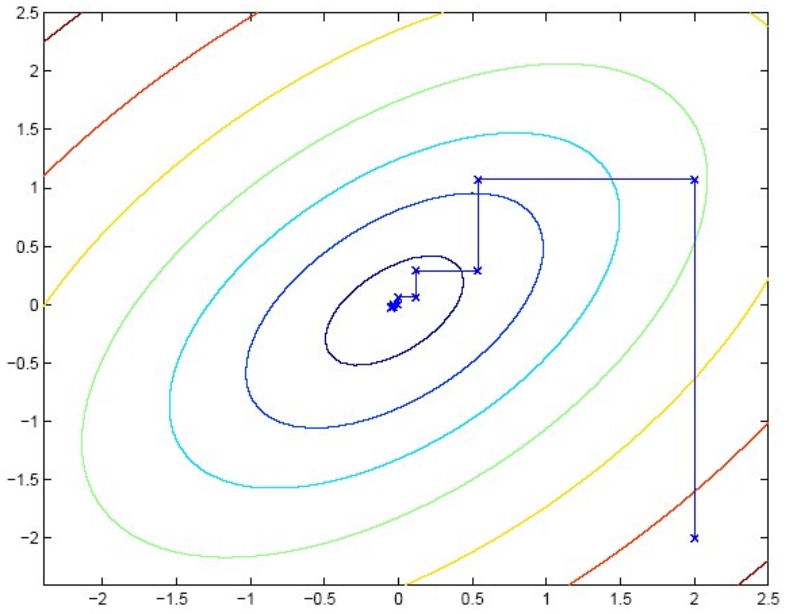
Coordinate Ascent

- Consider solving the unconstrained optimization problem:

$$\max_{\alpha} W(\alpha_1, \alpha_2, \dots, \alpha_n)$$

```
Loop until convergence: {
        for $i = 1\ to\ n$ {
            $\alpha_i = arg\,\max_{\widehat{\alpha_i}} W(\alpha_1, \dots, \widehat{\alpha}_i, \dots, \alpha_n)$ ;
        }
}
```

Coordinate ascent



- Ellipses are the contours of the function.
- At each step, the path is parallel to one of the axes.

Sequential minimal optimization

- Constrained optimization:

$$\max_{\alpha}\ \mathrm{J}(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i . \mathbf{x}_j)$$

$$\text{s.t.} \quad 0 \le \alpha_i \le C, \quad i = 1, \ldots, m$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0.$$

- Question: can we do coordinate along one direction at a time (i.e., hold all $\alpha_{[-i]}$ fixed, and update α_i?)

The SMO algorithm

$$\max_{\alpha} \mathrm{W}(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i . \mathbf{x}_j)$$

$$\text{s.t.} \quad 0 \le \alpha_i \le C, \quad i = 1, \ldots, m$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0.$$

- Choose a set of α_1's satisfying the constraints.
- α_1 is exactly determined by the other α's.
- We have to update at least two of them simultaneously to keep satisfying the constraints.

The SMO algorithm

Repeat till convergence {

1. Select some pair α_i and α_j to update next (using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum).
2. Re-optimize W(α) with respect to α_i and α_j, while holding all the other α_k 's ($k \neq i; j$) fixed.

}

- The update to α_i and α_j can be computed very efficiently.