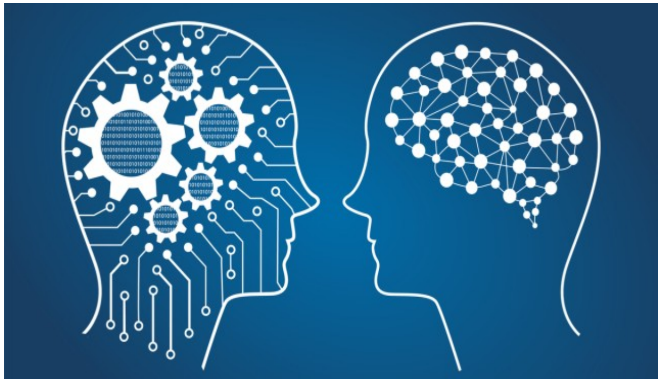


Support Vector Machines (SVMs)

Dr. Ammar Mohammed

Associate Professor of Computer Science

ISSR, Cairo University

PhD of CS (Uni. Koblenz-Landau, Germany)

Spring 2019

Contact:

mailto: Ammar@cu.edu.eg

Drammarcu@gmail.com

Classification Revisited

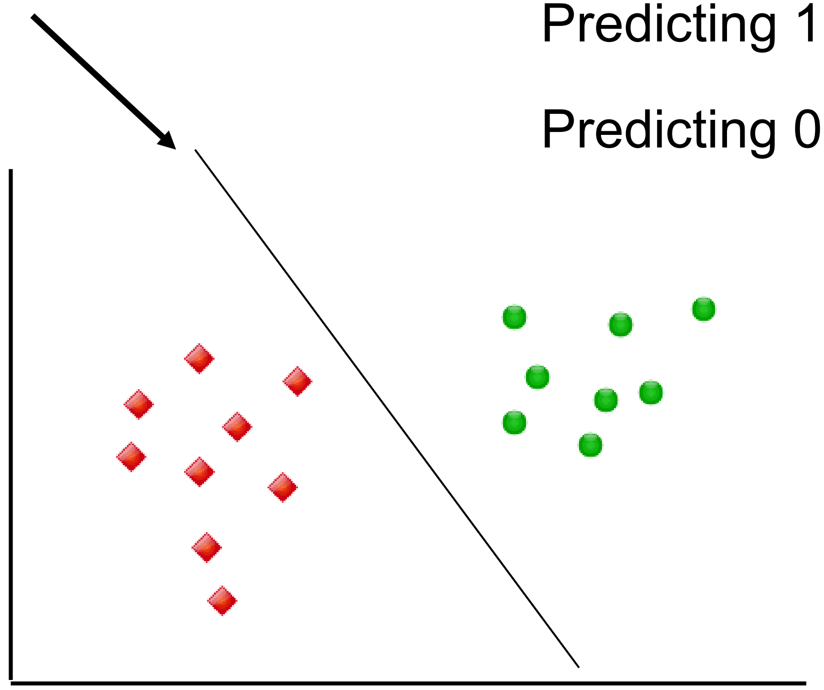
Binary Classification can be view as the task of Separating classes in feature spaces. For a hypothesis $h_\theta(x) = \theta^T x$

Decision Boundary

$\theta^T x = 0$

Predicting 1 (the green points) $\theta^T x > 0$

Predicting 0 (the red points) $\theta^T x < 0$



In other words, Classification for any unseen point x is :

$$f(x) = \text{sign}\,(\theta^T x)$$

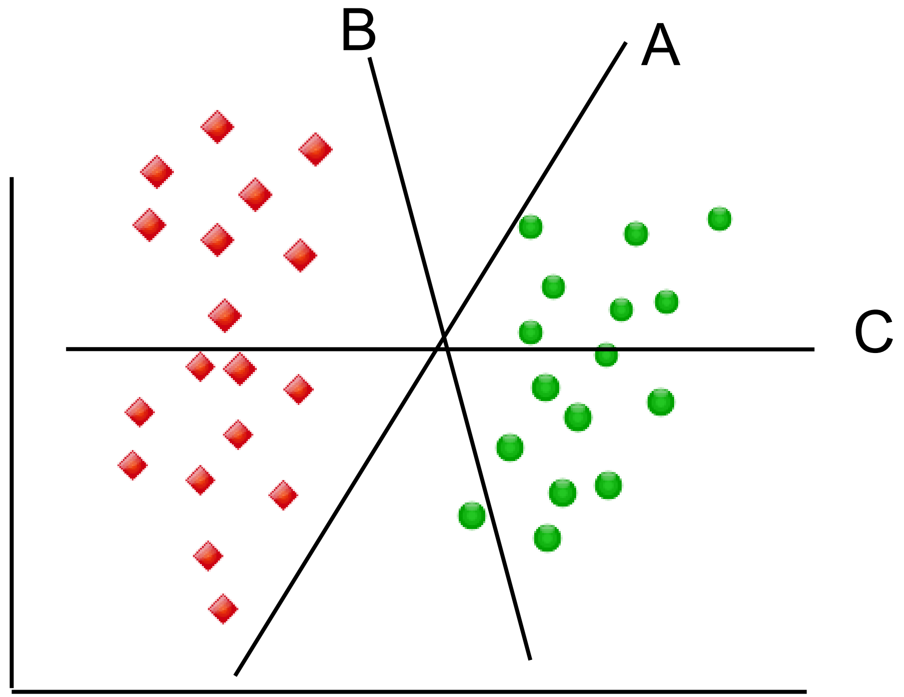
The goal of Classifier is to train a model that assigns a new unseen object into a specific category. It places the object above or below the separation hyper plane

Linear Separator

Classification is the process of categorizing the two classes with a hyper-plane. Now How can we Identify the right right (best) hyper-plane

Scenario 1

Identify the right hyper-plane



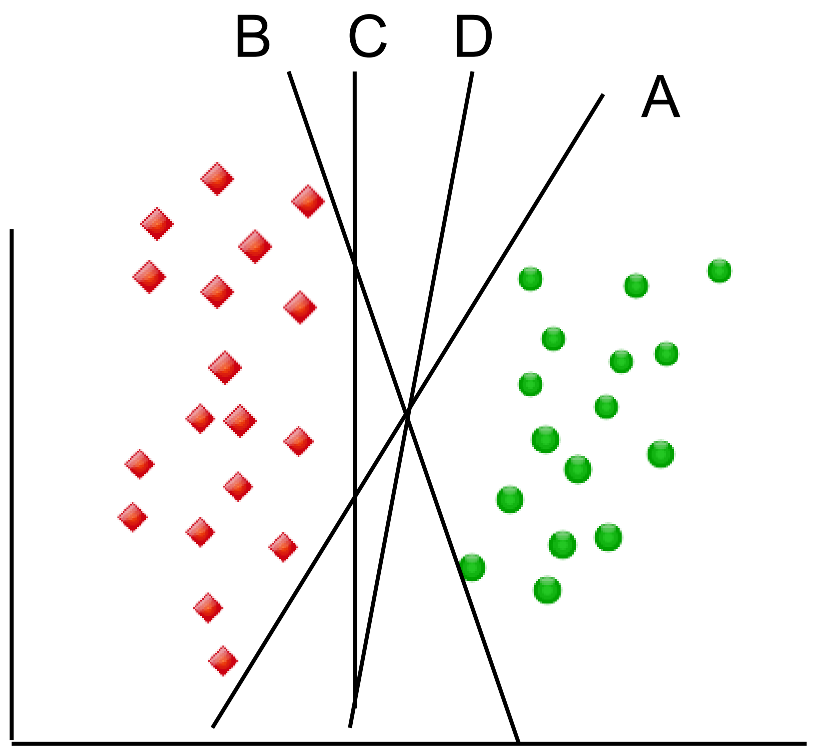
A hyper-plane A is correctly classifies the data points

Good vs Bad Classifiers

Classification is the process of categorizing the two classes with a hyper-plane. Now How can we Identify the right right (best) hyper-plane?

Scenario 2

Identify the right hyper-plane



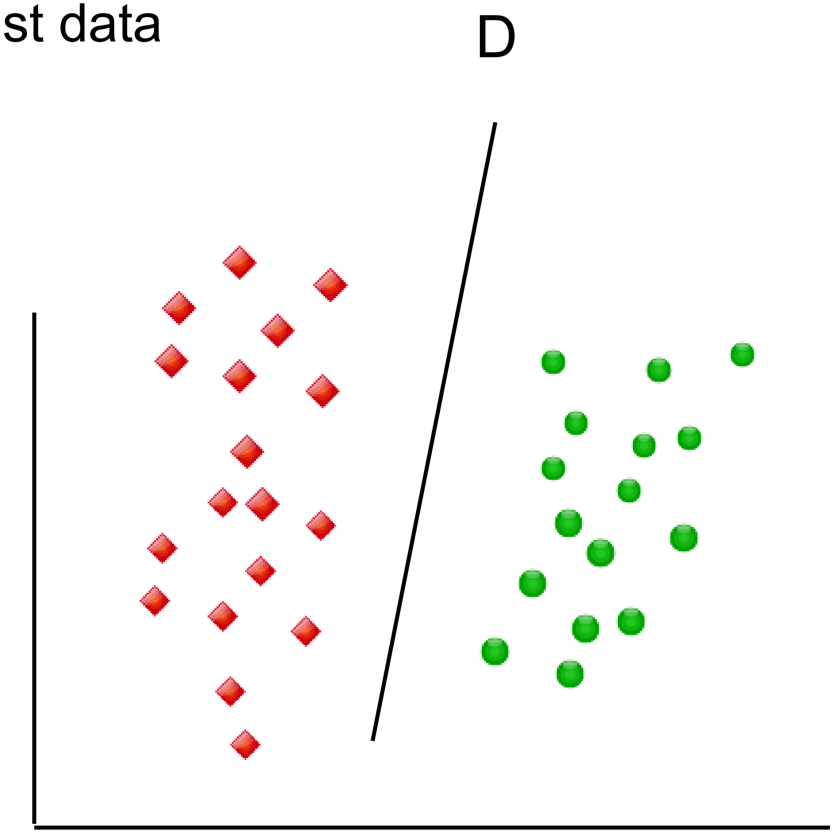
We have 4 hyper-planes A, B,C, and D. All are separating the classes well

Remember:The worth of a classifier is not in how well it separates the training data, but We want it eventually to classify unseen-data points. Given that, we want choose a hyper-plane that captures the general pattern in the training data, so there is a good chance it does well on the test data

Maximum Margin Classifier (MMC)

A,B, and C seem too close to the data points. Sure they appear in the training data perfectly, but when they see a test point, there is a good chance that it would get the wrong class (miss-classified).

D stays as far a way as possible from both classes. By being right in the middle of the two classes, it is less “risky” to miss-classify the unseen data. And thus generalizes well on test data

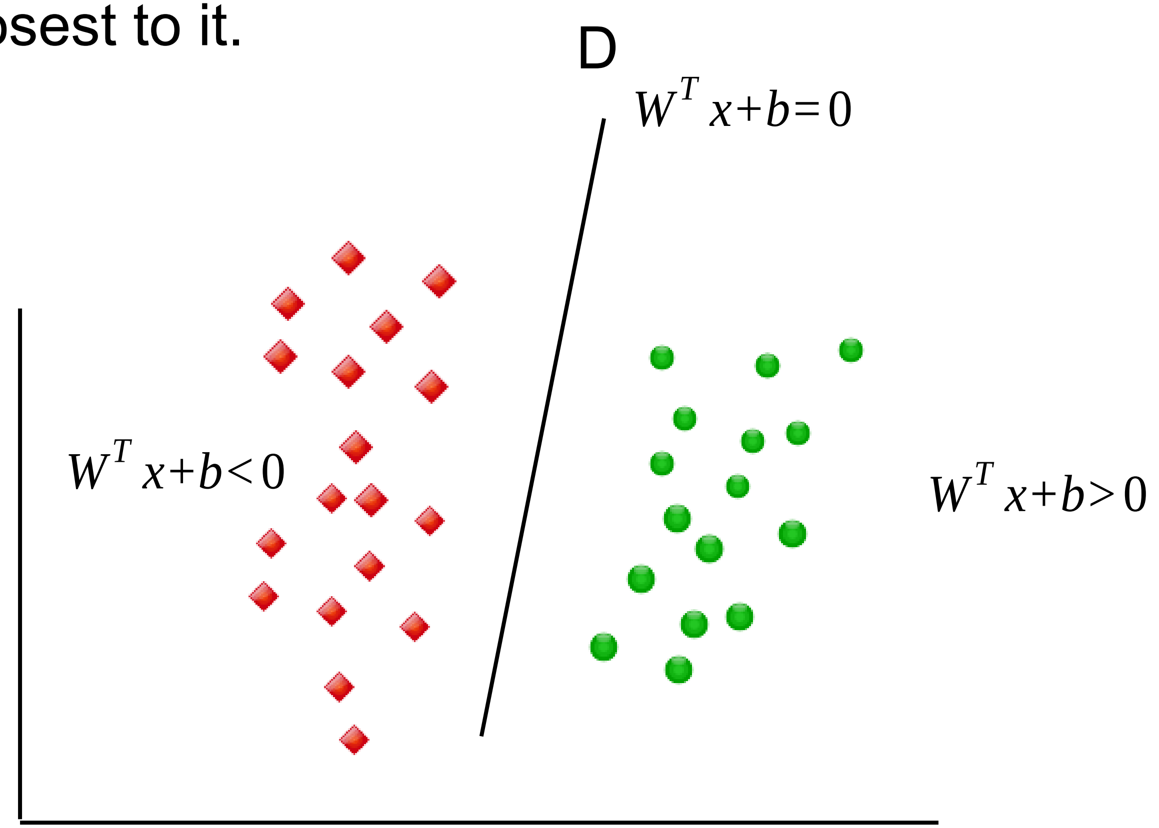


Maximum Margin Classifier (MMC)

MMC try to find the best separator hyperplane. Here is a simple version of What MMC (as a kind of SVMs) do:

1. Find hyper-planes that correctly classify the training data.

2. Among all such hyper-planes, pick the one that has the greatest distance to the points closest to it.

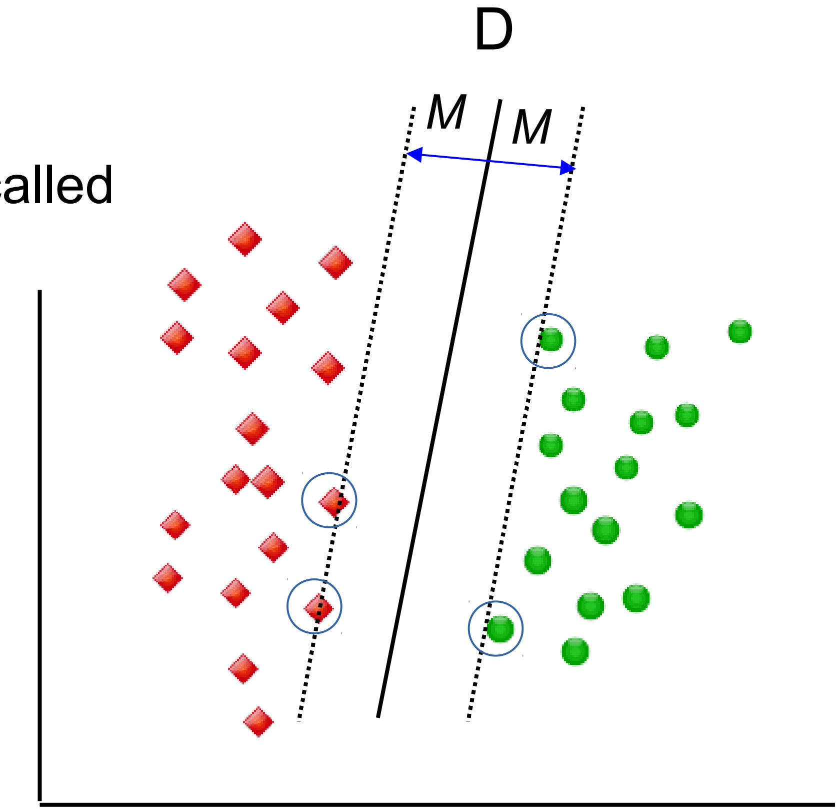


Essence of MMC

The closest points that identify this hyper-plane are knows as **support vectors**. The region they defined around the line is knows as the **Margin**.

- Only support vectors matter, other training examples are ignorable
- Data that can be separated by a hyper-plane is known as **Linearly separable data**
- The hyper-plane that classifies the linear separable data act as a linear classifier

- *2M is t*he margin
- Circled points are called support vectors



Optimization Problem

Maximize $M = \frac{1}{\|w\|}$

Subject to $y_i \left(w \cdot x_i + b \right) \geq M$

or

Minimize $\frac{1}{2} w \cdot w$

Subject to $y_i \left(w \cdot x_i + b \right) \geq M$

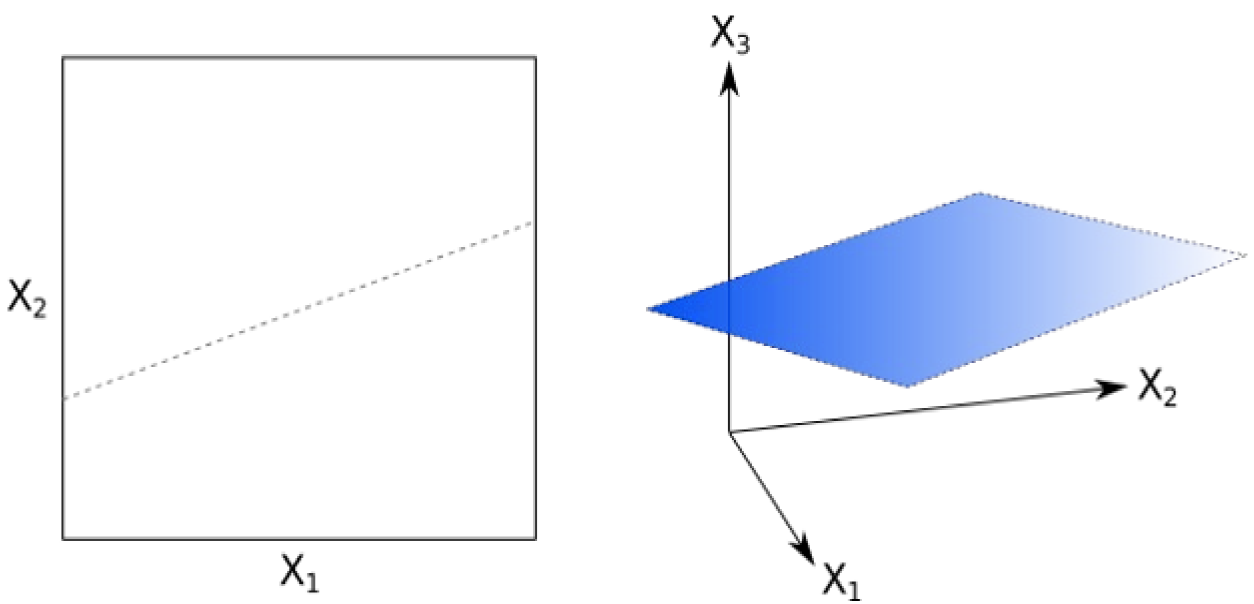
What IS Support vector Classifier (SVC)

SVC is a classifier formally defined by separating hyperplane $W^t x + b = 0$
A hyperplane is a subspace of ***one dimension*** less thank its ambient space. This means a hyperplane of two dimension space is one dimension separator (line). A hyperplane of three dimension space is two dimension separator (plane).

Elements above the hyperplane satisfy $w^t x + b > 0$

Elements below the hyperplane satisfy $w^t x + b < 0$

The weight vector ***W*** represents the orientation of the hyperplane and **b** represent the bias



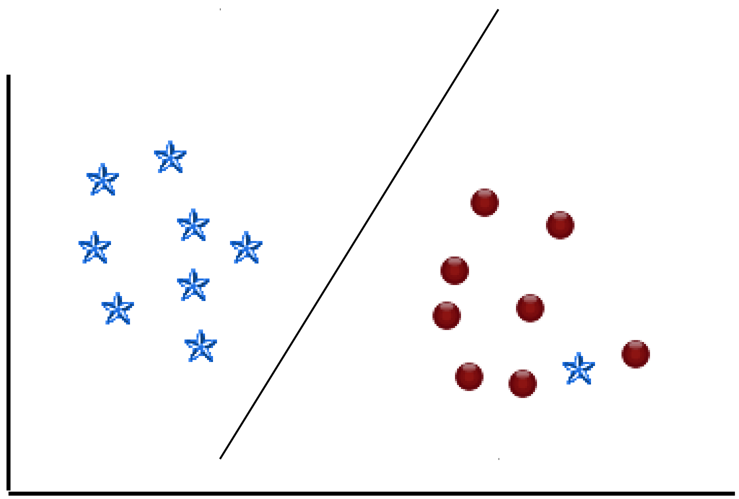
2 D space separated by line (left). 3 D space separated by plane (right)

Allowing for Errors: SVC

We look at the easy case of perfectly linear separable data. But real-world data is typically messy and almost few instances of data a linear classifier can't get right

Scenario 3

How can you find the right hyperplane ?



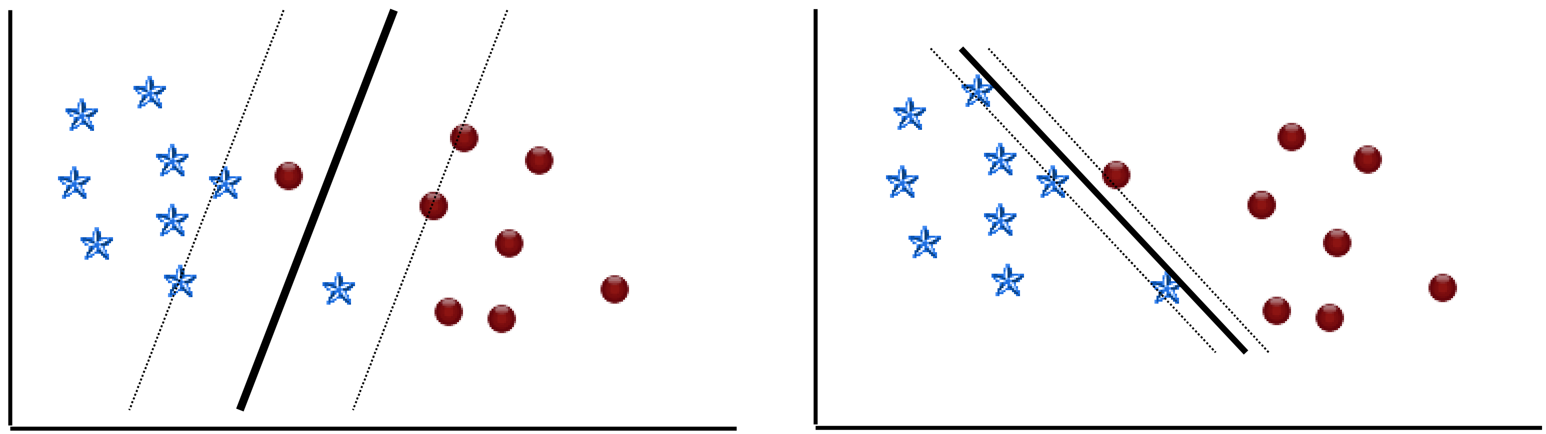
SVC has feature to ignore outliers and find the hyperplane that has maximum margin. i.e SVC (hence SVM)is robust to outliers

Allowing for Errors: SVC

We look at the easy case of perfectly linear separable data. But real-world data is typically messy and almost few instances of data a linear classifier can't get right

Scenario 4

Which one is the best ?



Soft margin Classifier

Will you maximize the margin and allow misclassified instance. Or will you choose to correctly classify with less margin ?

This is a trade-off

Allowing for Errors: SVC

- How SVC let you handle this situation ? It allows you to specify how many errors you are willing to accept.

- Providing a hyper parameter called ‘ ***C***’ to your SVM. This allows to control the trade-off between:
  - A wide Margin
  - Correctly classify the training data

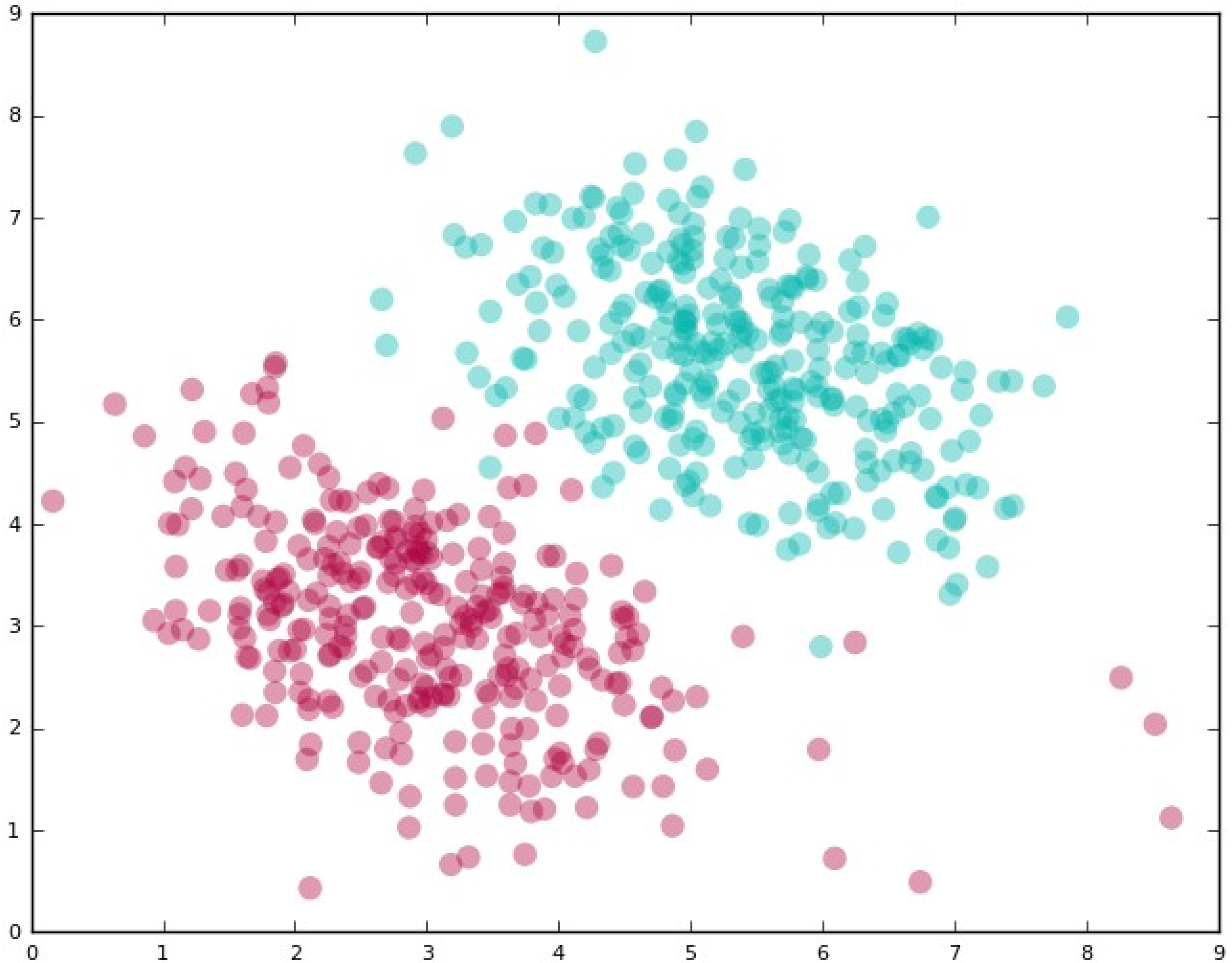
- C is a non-negative “Tuning” parameter. If C=0, implies that no violation of the margin is possible (in this case, we have MMC situation)

Usually in addition to C, The SVC introduces a parameter ϵ_i called slack variable to each data point x_i. It allows the data points to be on the wrong side of the margin or hyperplane. $\sum_{i=1}^{n} \epsilon_i \leq C$

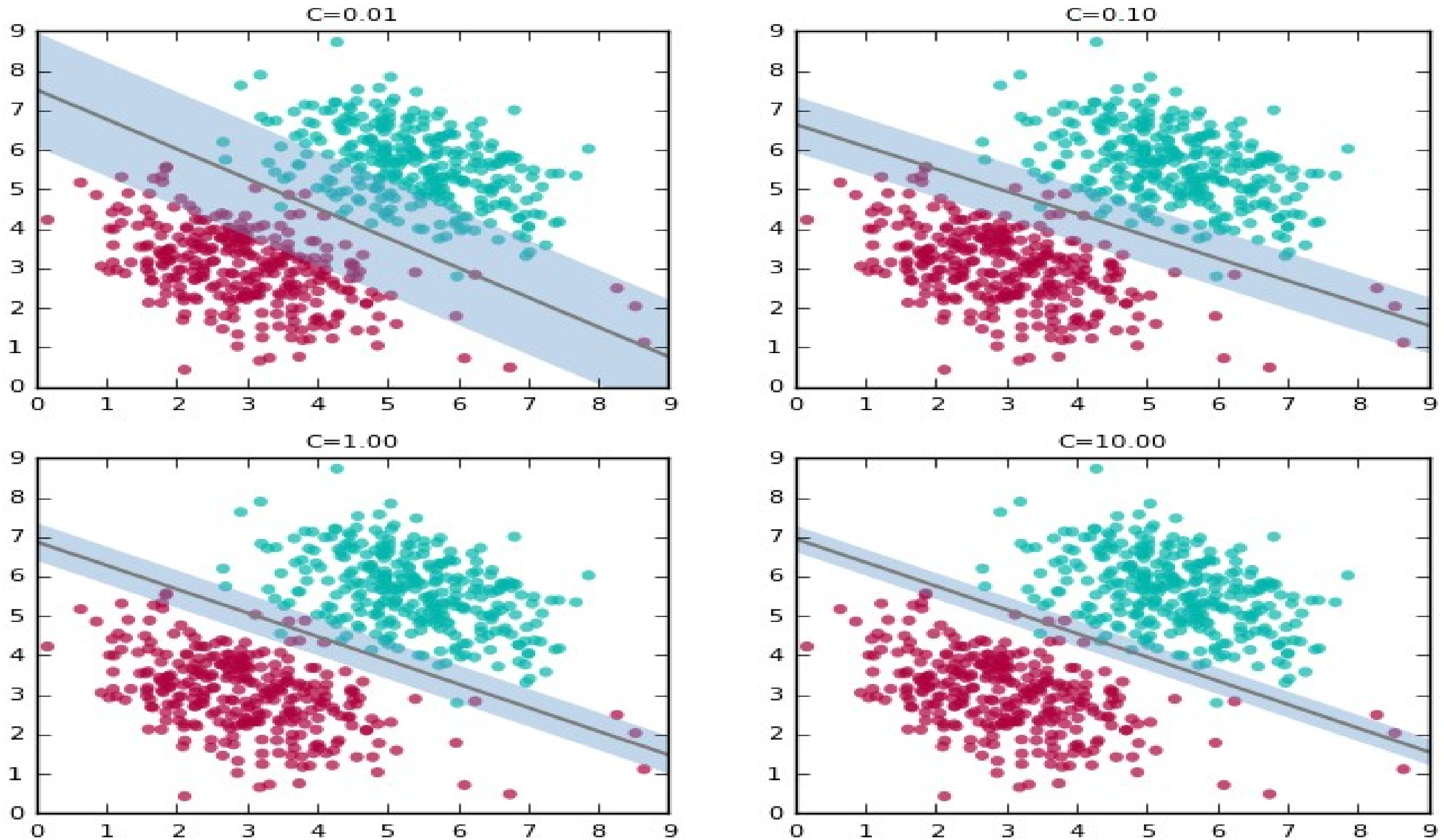
If $\epsilon_i = 0$, it states that the training point x_i is on the correct side of the margin, for $\epsilon_i > 0$ means x_i on the wrong side of the **margin**. $\epsilon_i > 1$ Means x_i on the wrong side of the hyperplane

Allowing for Errors: SVC

Example to separate the Data



Allowing for Errors: SVC



The first plot c=0.01 capture general trend better, although it suffers from low accuracy on the training data compared to higher value for C

Bias-variance trade off again

- **Large C:**
  - Small Margin
  - Allow for more violation of Margin
  - More Support Vectors
  - Less variance, more stable
  - High bias
- **Small C:**
  - Large Margin
  - Less violation on training data
  - Low training error, less bias
  - Fewer support vectors
  - Higher variance

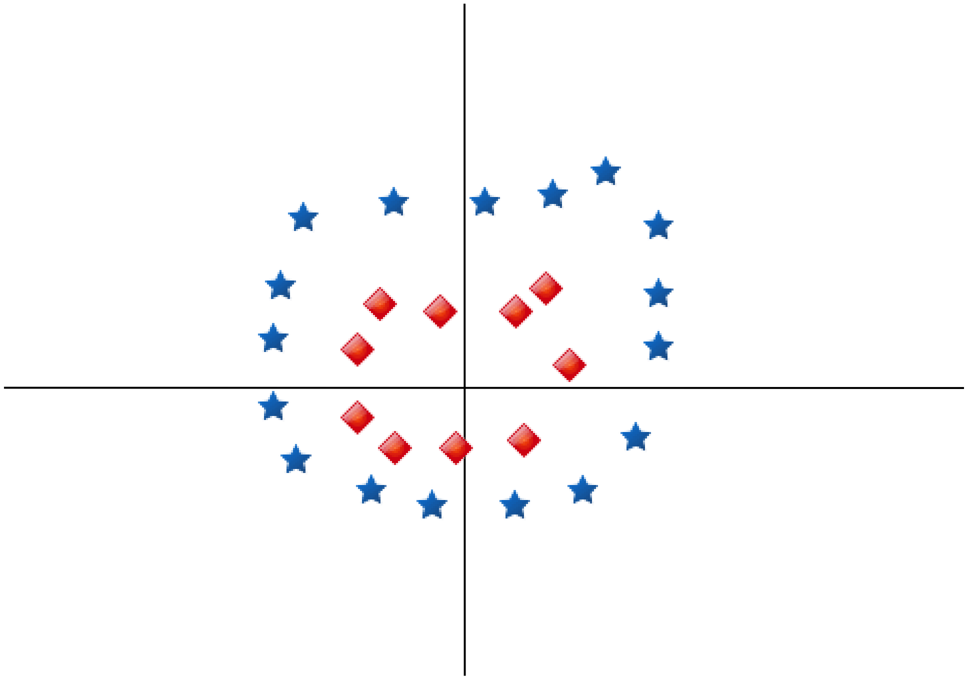
How do we Choose C in practice ?

Cross validation

Non-Linearly Separable Data

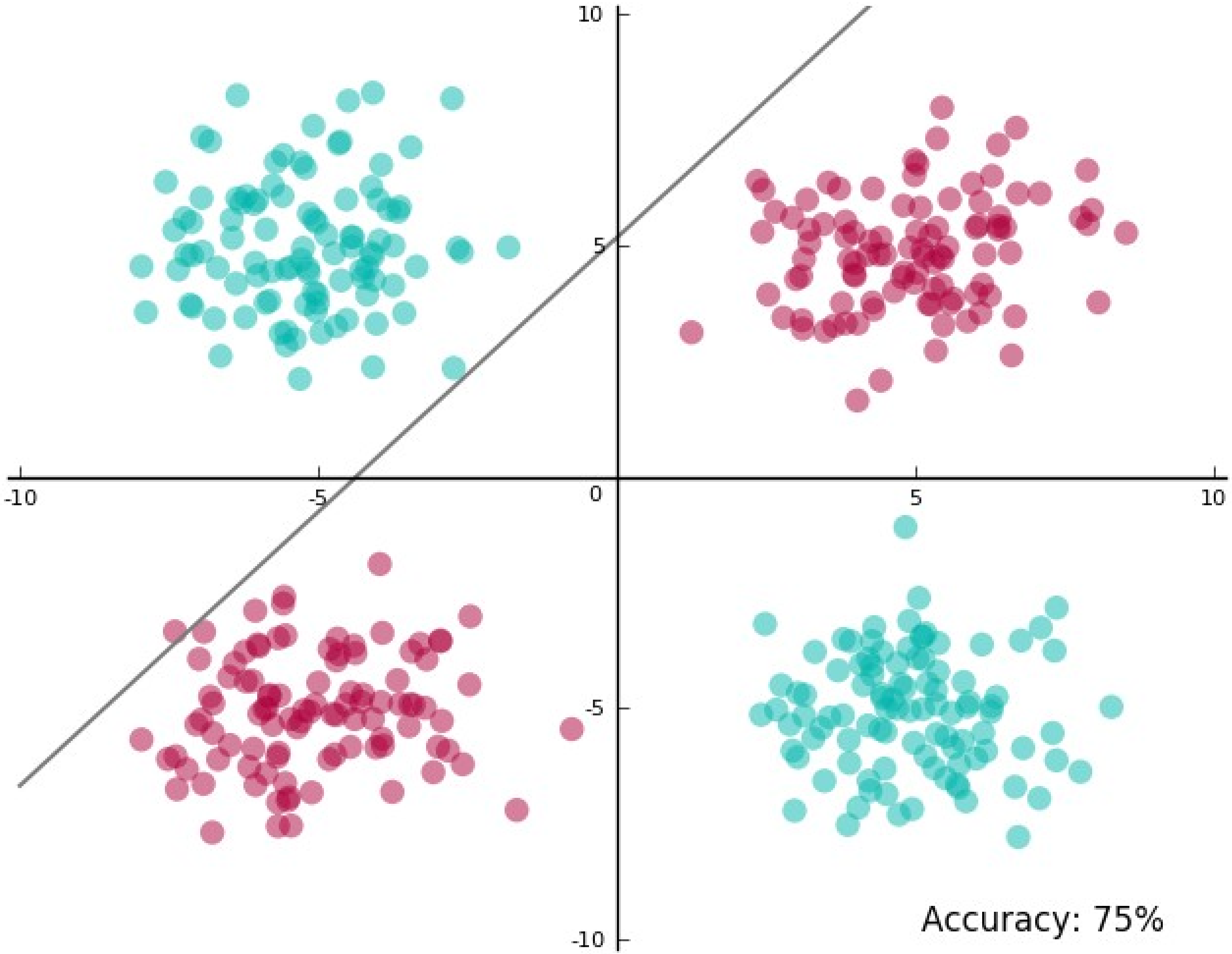
Scenario 5

We can't have a linear Hyperplane between the two classes. How does SVM classify theses two Classes ?



Non-Linearly Separable Data

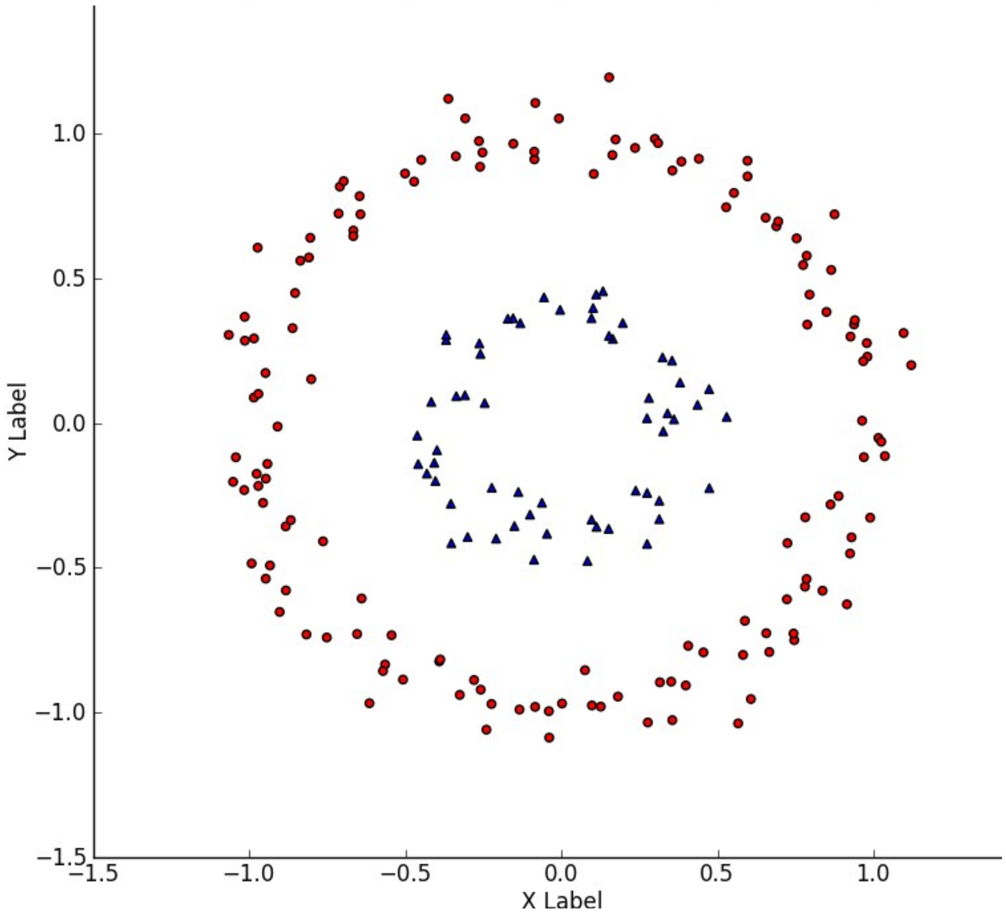
A lot if real-world data are non-linearly separable. Here an example XOR Data set.



If we use the SVC, it would give extremely poor performance. In the example, the accuracy almost 75 % on the training data

Support Vector Machines

Although the data is non-linearly separable, We have a good technique at finding hyperplane using **SVM** by Extending a SVC is to allow non-linear decision boundary



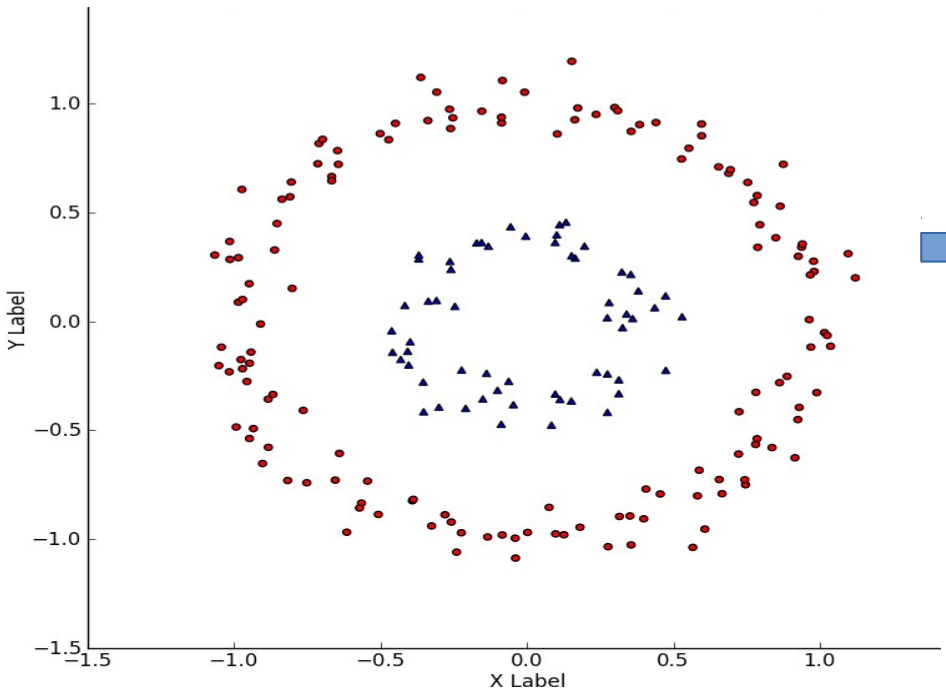
How

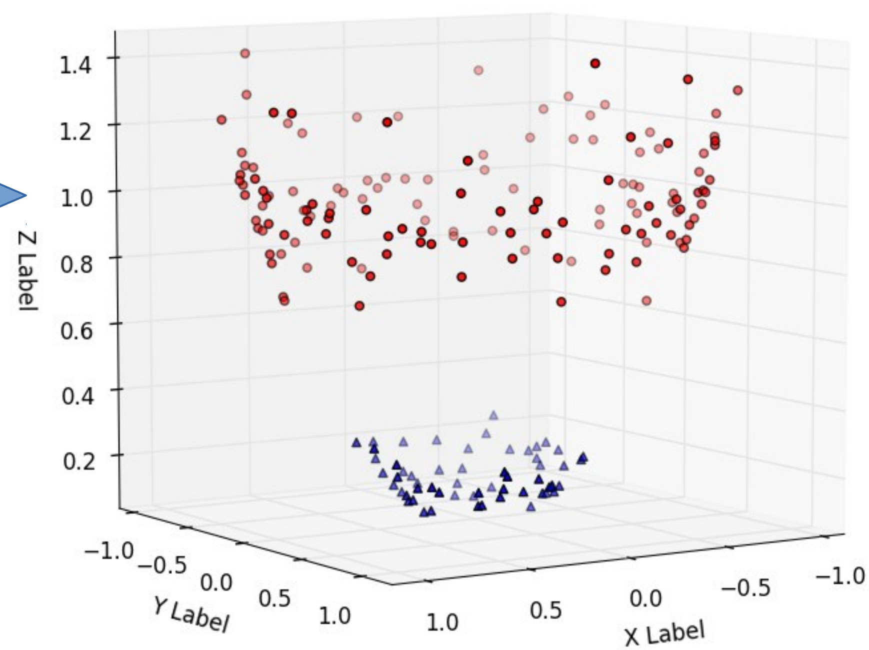
Idea: Project the data into another dimensional space where it is linearly separable and then find the hyperplane in this new space

Support Vector Machines

Upgrade to a higher dimension

Transform data from 2D space to 3D Space

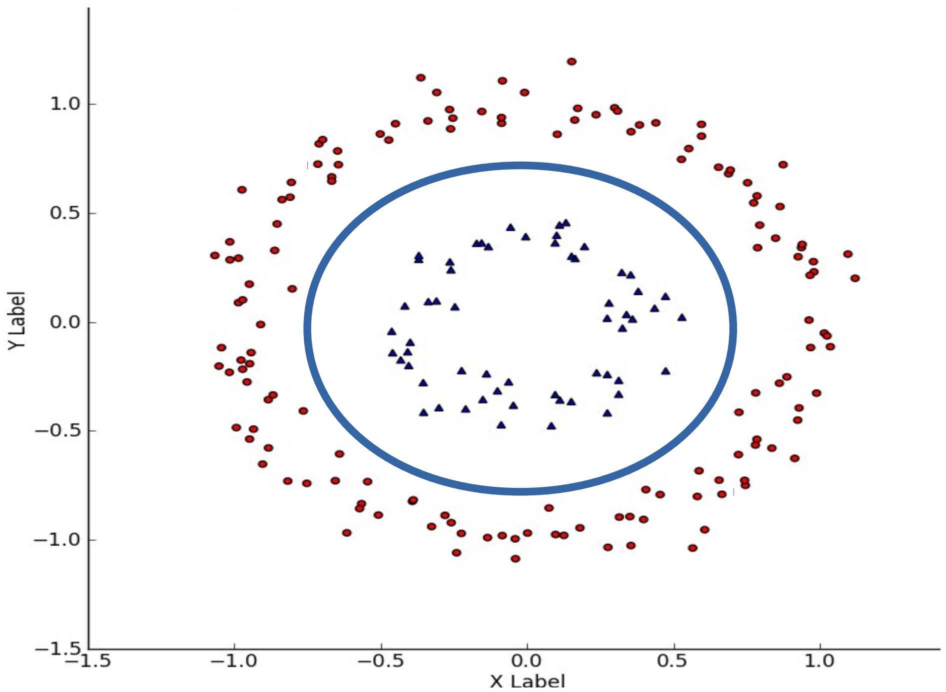


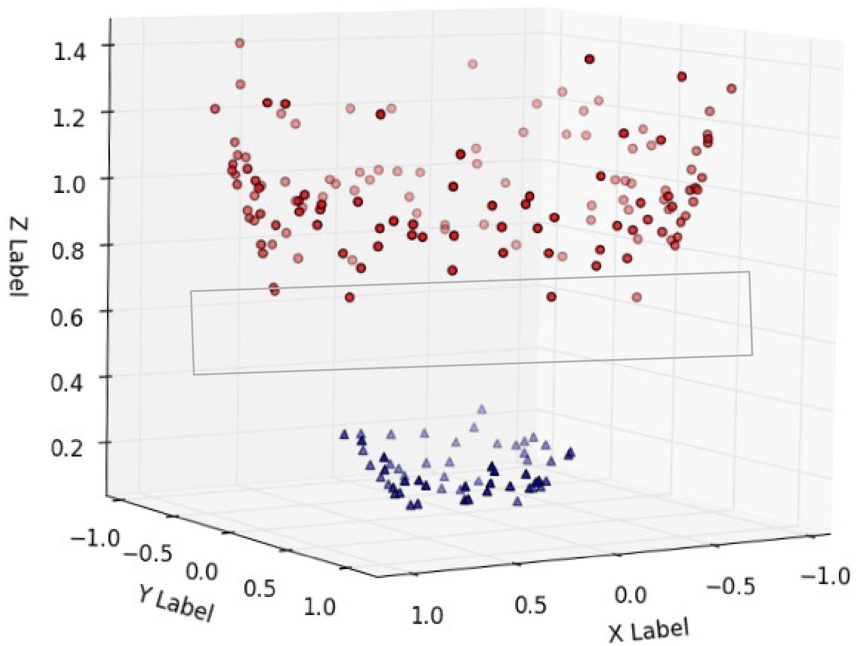


Data now is linearly separable by a hyperplane. The plane in 3D space can separate the data

Support Vector Machines

Project the hyperplane back to the original dimension

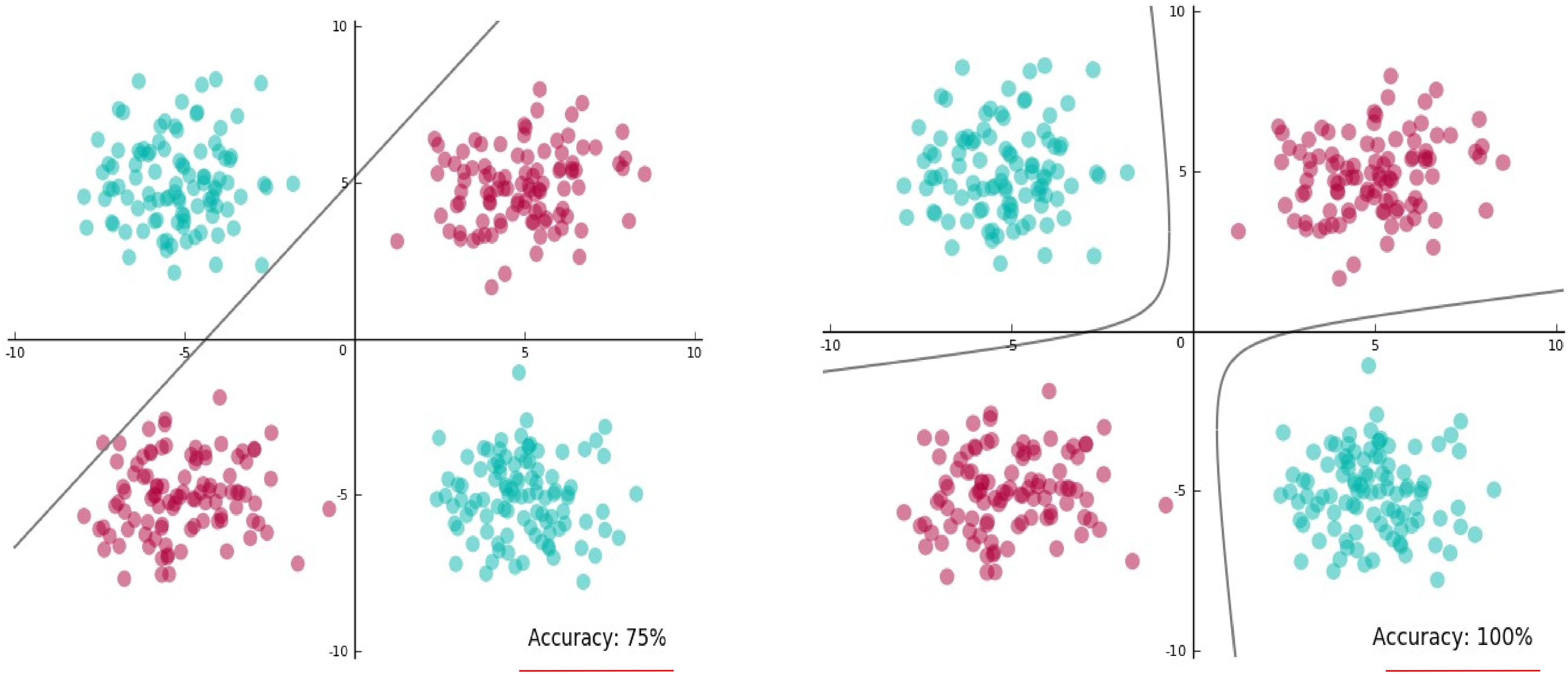




Non-linear separation

When mapping the decision boundary back to the original space, the separating boundary is not a line anymore

SVMs



The shape of the separating boundary in the original space depends on the **projection**(the mapping function) in the projection space.

SVMs: Mapping to higher Dimension

Try to find a mapping function **f** that takes the input spaces to a higher dimension feature space

$$f : R^n \Rightarrow R^m$$

- **f** is a map from ***n***-dimension to ***m***-dimension. Usually m > n

- instead of finding the non-linear separability hyperplane in **n**-dimension, we try to find a hyperplane linearly in **m**- dimension

Minimize $\frac{1}{2} w \cdot w$

Subject to $y_i \left(w \cdot x_i \right) \geq M$

map →

Minimize $\frac{1}{2} f\left(w\right) \cdot f\left(w\right)$

Subject to $y_i \left(f\left(w\right) \cdot f\left(x_i\right) \right) \geq M$

That is cool 

But costly and need extra effort

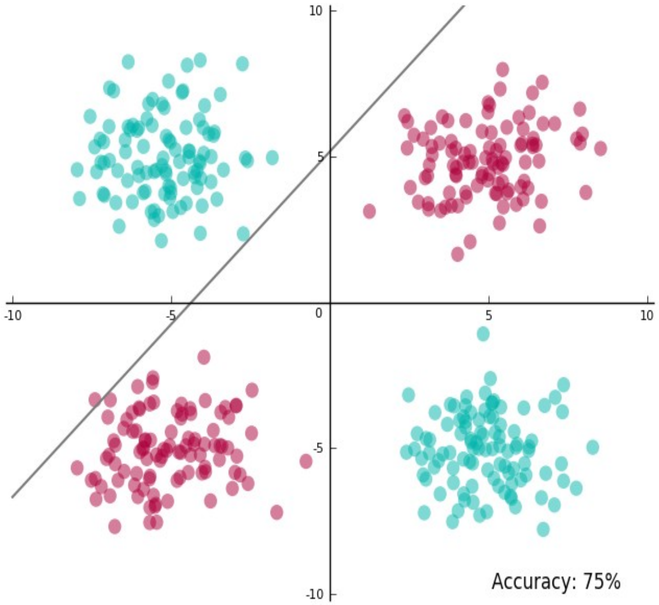
SVMs: Kernel

In essence, SVMs are an extension of SVC that results from enlarging the features space through the use of functions known as **Kernels**

Let x, y are n dimensional inputs. The kernel function **K** on x,y is the dot product on the mapping function **f** on x and y
- define: K(x,y)= **f**(x) . **f**(y)

Normally, we need to map each data point form the dimension n to dimensional m using f and apply SVC (dot product optimization) on the new dimension

We give an example to illustrate the power of kernel



$X=(x_1,x_2)$ 2D feature space. Let f be the mapping on 3D space

$$f(x)=\left(x_{1,}^2 x_2^2, \sqrt{2}\, x_1 x_2\right)$$

$$f: R^2 \Rightarrow R^3$$

The cost of mapping

$$X_i = (x_{i1}, x_{i2}) \longrightarrow \hat{X}_i = (x_{i1}^2, x_{i2}^2, \sqrt{2}\, x_{i1} x_{i2})$$

To compute the projection (mapping) we need to perform the following operation:
- to Get the new first dimension: 1 Multiplication
- second dimension: 1 Multiplication
- third dimension: 2 Multiplications

In all, 1+1+2= **4 Multiplications**

Since the most important operation of SVC is the dot product between any two data points, let see the cost of the dot product in the new dimension

$$\hat{X}_i \cdot \hat{X}_j = X_{i1} X_{j1} + X_{i2} X_{j2} + X_{i3} X_{j3}$$

To compute this dot product for point I and j, we need to compute their projection first,so that is 4+4 =8 Multiplications. The dot product needs 3 multiplications and 2 additions. In all
8 (for projections) + 3(multiplications)+ 2 additions = 13 operations

Now, what if we use the kernel $K(x_i, x_j) = (x_i \cdot x_j)^2$

Kernel Trick

$$K(x_i, x_j) = (x_i \cdot x_j)^2$$

Let us expand the kernel function

$$K(x_i, x_j) = (x_i \cdot x_j)^2 \longrightarrow \textbf{(1)}$$

$$= (x_{i1} x_{j1} + x_{i2} x_{j2})^2 \longrightarrow \textbf{(2)}$$

$$= x_{i1}^2 x_{j1}^2 + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{i2} x_{j1} x_{j2} \longrightarrow \textbf{(3)}$$

$$= (x_{i1}^2, x_{i2}^2, \sqrt{2}\, x_{i1} x_{i2}) \cdot (x_{j1}^2, x_{j2}^2, \sqrt{2}\, x_{j1} x_{j2}) \longrightarrow \textbf{(4)}$$

$$= f(x_i) \cdot f(x_j) \longleftarrow$$

This is magic. The Kernel Trick

How many operation to compute equation (2) ?

2 multiplications+ 1 addition+ 1 for squaring the result = 4 operations

If we use the kernel function of the mapping, it would make 31% reduction of of the operations that we calculated before. It look faster to use a kernel function to compute the dot products. We do not need even to map the data into the other dimension **magic ;-)**

SVMs with Kernel

A kernel function computes what the dot product would be if you had actually projected the data

A kernel Trick means a kernel function transforms the data into a higher dimensional feature space to make it possible to perform linear separation on the data.

Some Popular Kernels

- **Polynomial Kernel** $K(x,y) = (x^T y + 1)^d$
- **Radial Base Function Kernel** $K(x,y) = \exp(-\gamma \|x - y\|^2)$
- **Linear Kernel** $K(x,y) = x^T y$