Big Data Analytics

Lecture 3/B

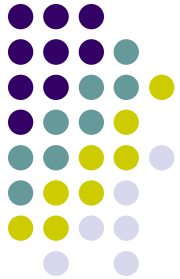
Supervised classification methods with human tagging (second part)

UNIVERSITÄT LUZERN

References

- ✓ Olivella, Santiago, and Shoub Kelsey (2020). Machine Learning in Political Science: Supervised Learning Models. In Luigi Curini and Robert Franzese (eds.), *SAGE Handbook of Research Methods is Political Science & International Relations*, London, Sage, chapter 56

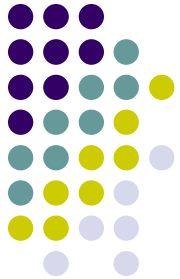
ML alghoritms



Let’s now discuss about a further and rather popular ML algorithm: the *Support Vector Machine* one

This is a non-probabilistic geometric/spatial model

Support Vector Machine (SVM)



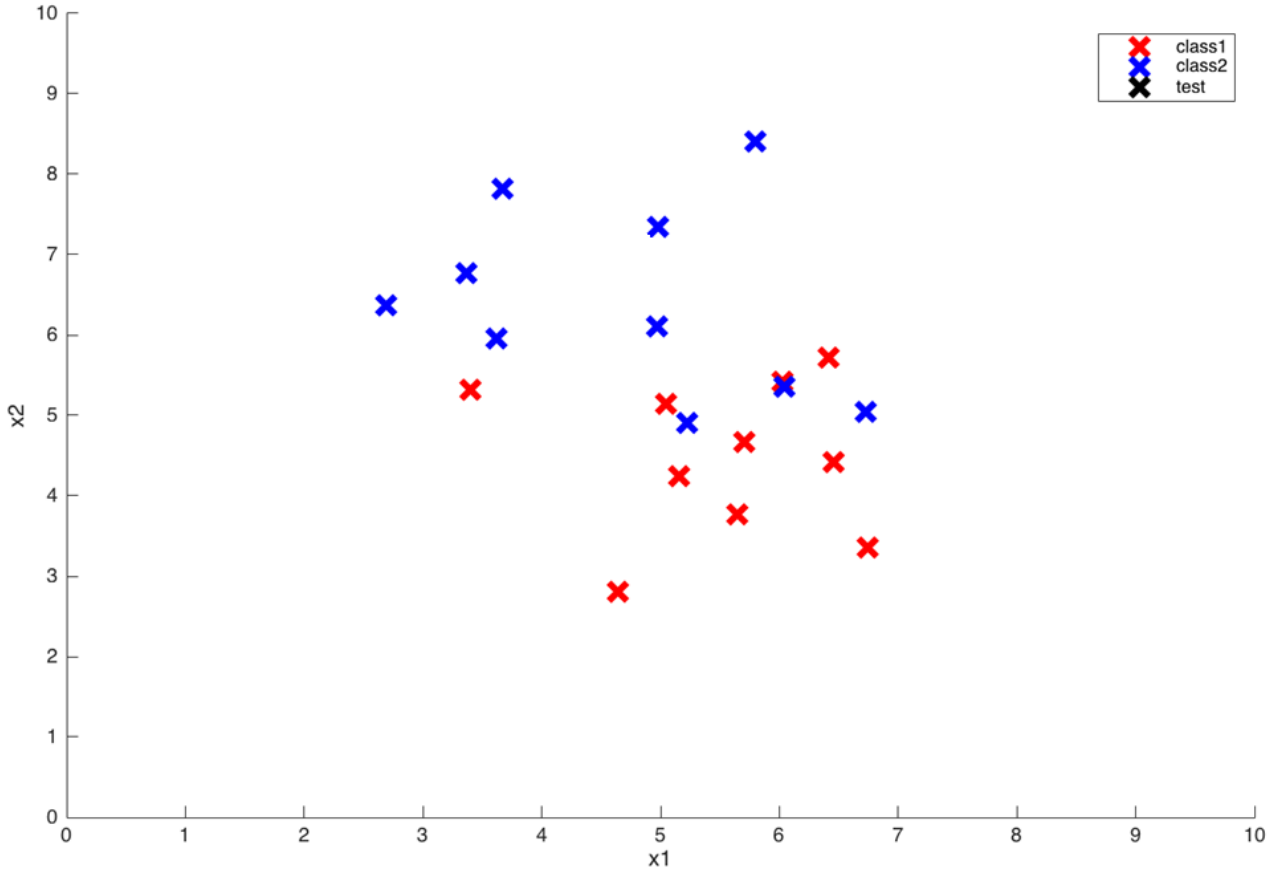
SVM is a generalization of **Nearest Neighbor** (NN) algorithm

NN is a rather simple algorithm. You are given a training data consisting of *m* training documents $\{(\boldsymbol{x}^1, y^1), (\boldsymbol{x}^2, y^2), \ldots (\boldsymbol{x}^m, y^m)\}$, where x is a vector of possible variables and y^i is the class label (the category) of i^{th}

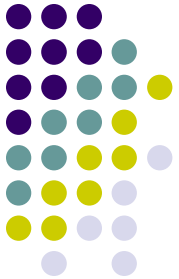
Support Vector Machine (SVM)

For example, in the figure below, we have two variables

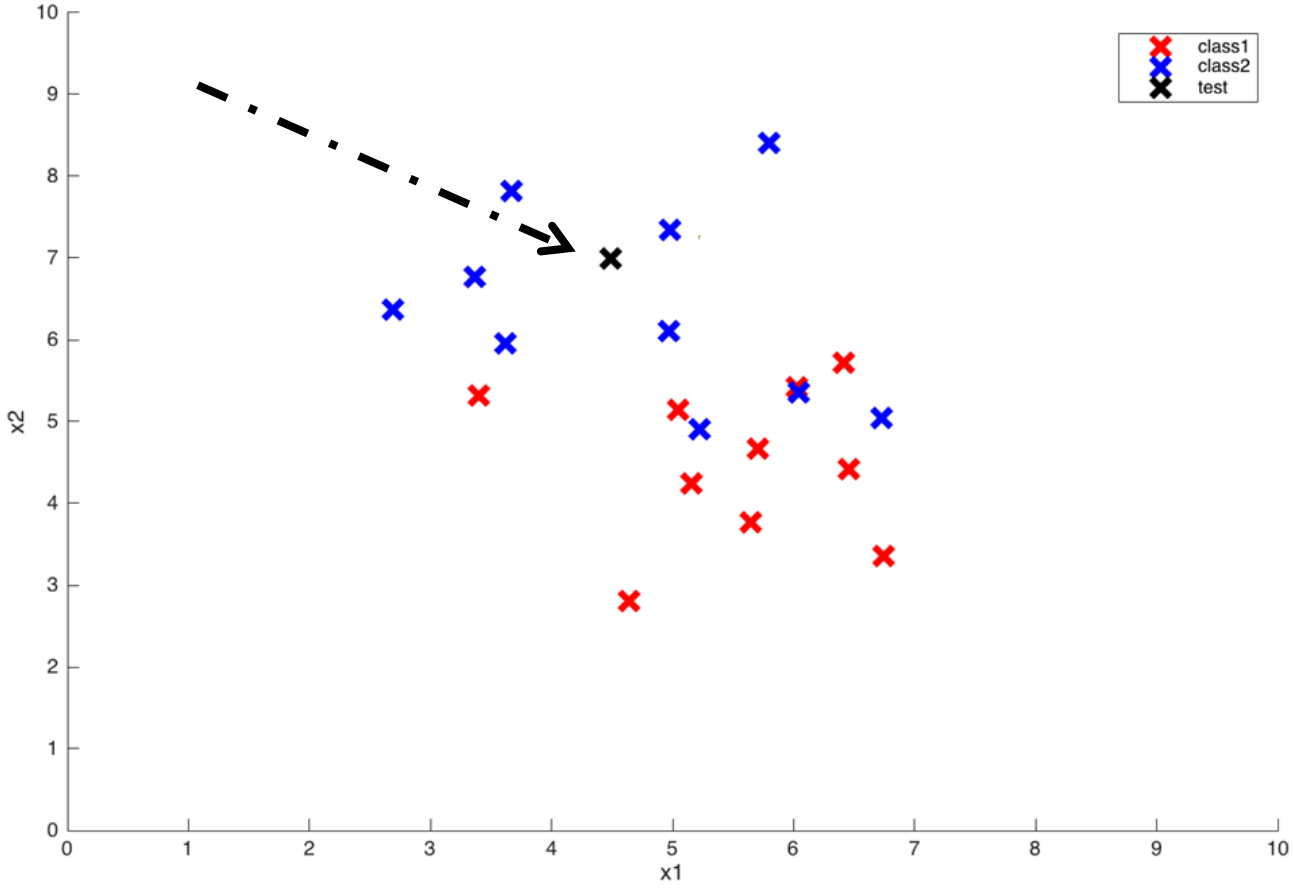
Xs: each document can be either label 1 (blue points), or label −1 (red points)



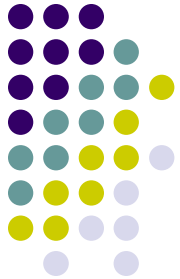
Support Vector Machine (SVM)



Now you are **given a test point** (the black x below), and you have to predict its class, whether it belongs to red class or blue class



Support Vector Machine (SVM)

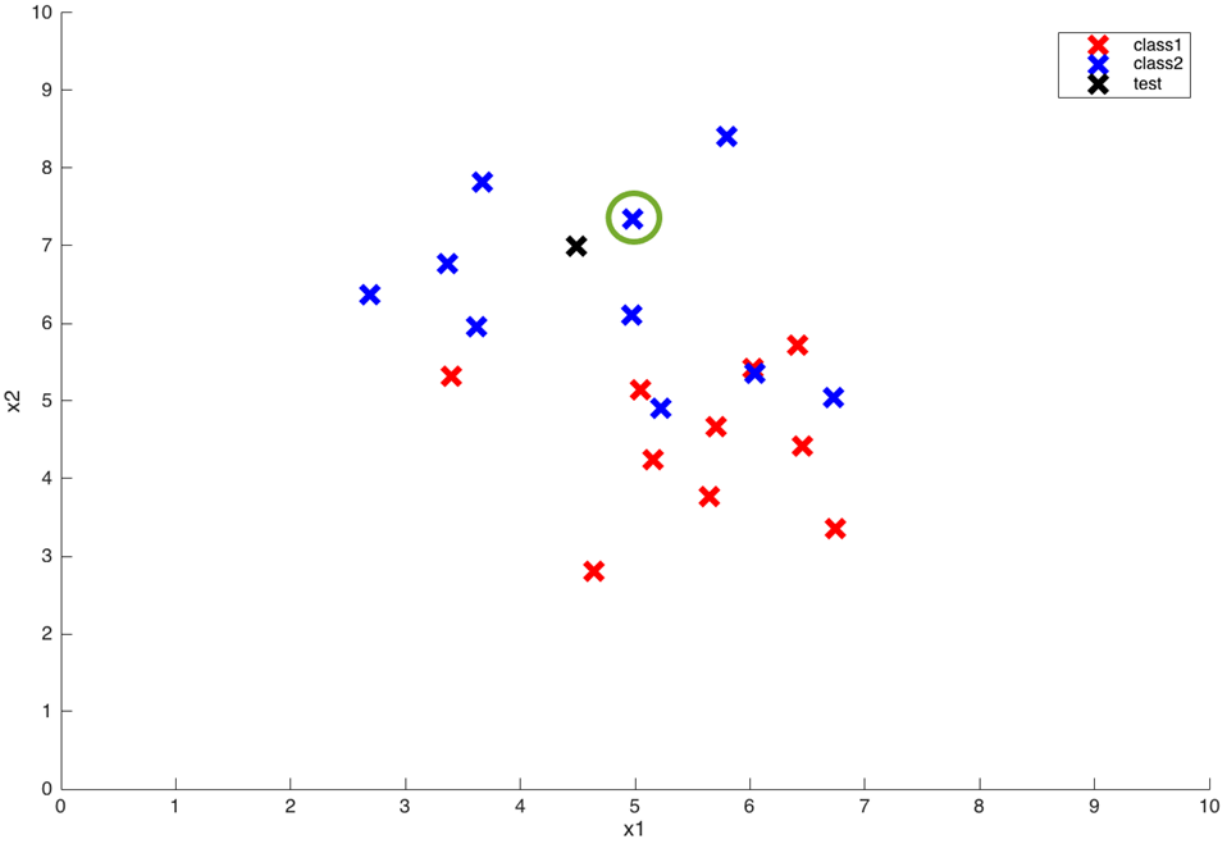


The NN algorithm finds the nearest training point to this test point (by measuring the distance of test point from **every** training point), and the class predicted of test point is the same as of nearest training point

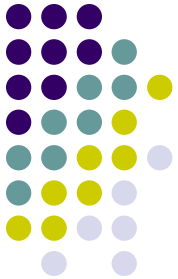
The lower the distance, the higher the **similarity** between two points

Support Vector Machine (SVM)

For example, the circled point is closest to test point, and hence class of test point is blue



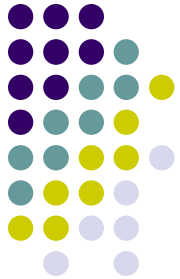
Support Vector Machine (SVM)



Two observations about NN algorithm:

- ✓ We don't do any computation alone with training points. Only when a test point comes, we compute similarity from **every** training point. This is a big disadvantage of NN algorithm
  - Consider having millions of training points, and for every test point, we have to calculate millions of similarities from test point. We calculate similarities from the training points which are very far from test points, which are not really required
- ✓ We don't give any importance to other training points **except** the nearest one

Support Vector Machine (SVM)



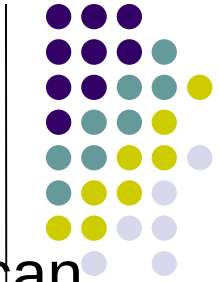
SVM remove each of these two problems

Instead of finding similarity from every training point of any test point, we calculate similarity from only a **subset** of training points (or documents, when dealing with text classification), which we compute in the **training phase**

These selected training points are called **support vectors,** since only these points will support our decision of selecting the class of a test point

Our hope is that our training phase finds as few as support vectors so that we have to compute fewer number of similarities

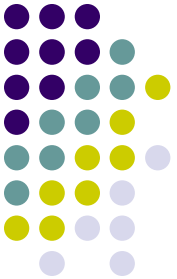
Support Vector Machine (SVM)



Moreover, once we have selected support vectors, we can assign a **weight** to each support vector, which basically tells how much importance we want to give to that support vector while making our decision

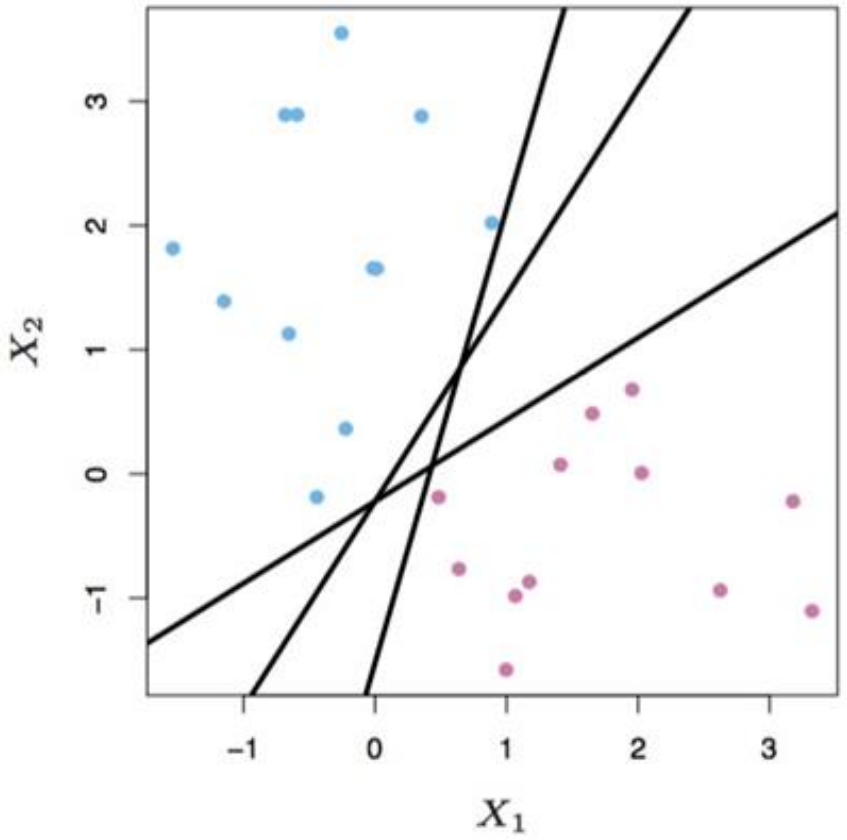
So unlike NN, we don't give importance to only a single training point (i.e., *document*, given that we are dealing with text classification), instead we give each support vector a separate importance

Support Vector Machine (SVM)

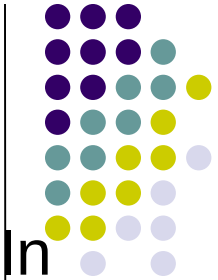


But how to find a **support vector**?

Intuition: first, we need to find the best line that separates observations belonging to different classes!



Support Vector Machine (SVM)

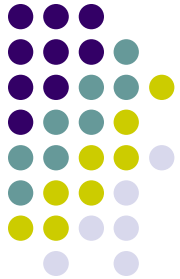


This is harder to visualize in more than two dimensions. In this case you need an hyperplane…a what?!?

An hyperplane is n-1 dimensional subspace of an n-dimensional space (a line in 2D, a plane in 3D and an hyperplane in higher dimensions)

But not only that…

Support Vector Machine (SVM)



We want to find an hyperplane that **best separates two classes of points with the maximum margin** (i.e., we try to find that separating hyperplane from which distance of closest training points is maximum) thus producing the “cleanest" possible sorting of observations

That is, our goal is to identify the hyperplane that *maximizes* the total distance between the line and the closest point in each class

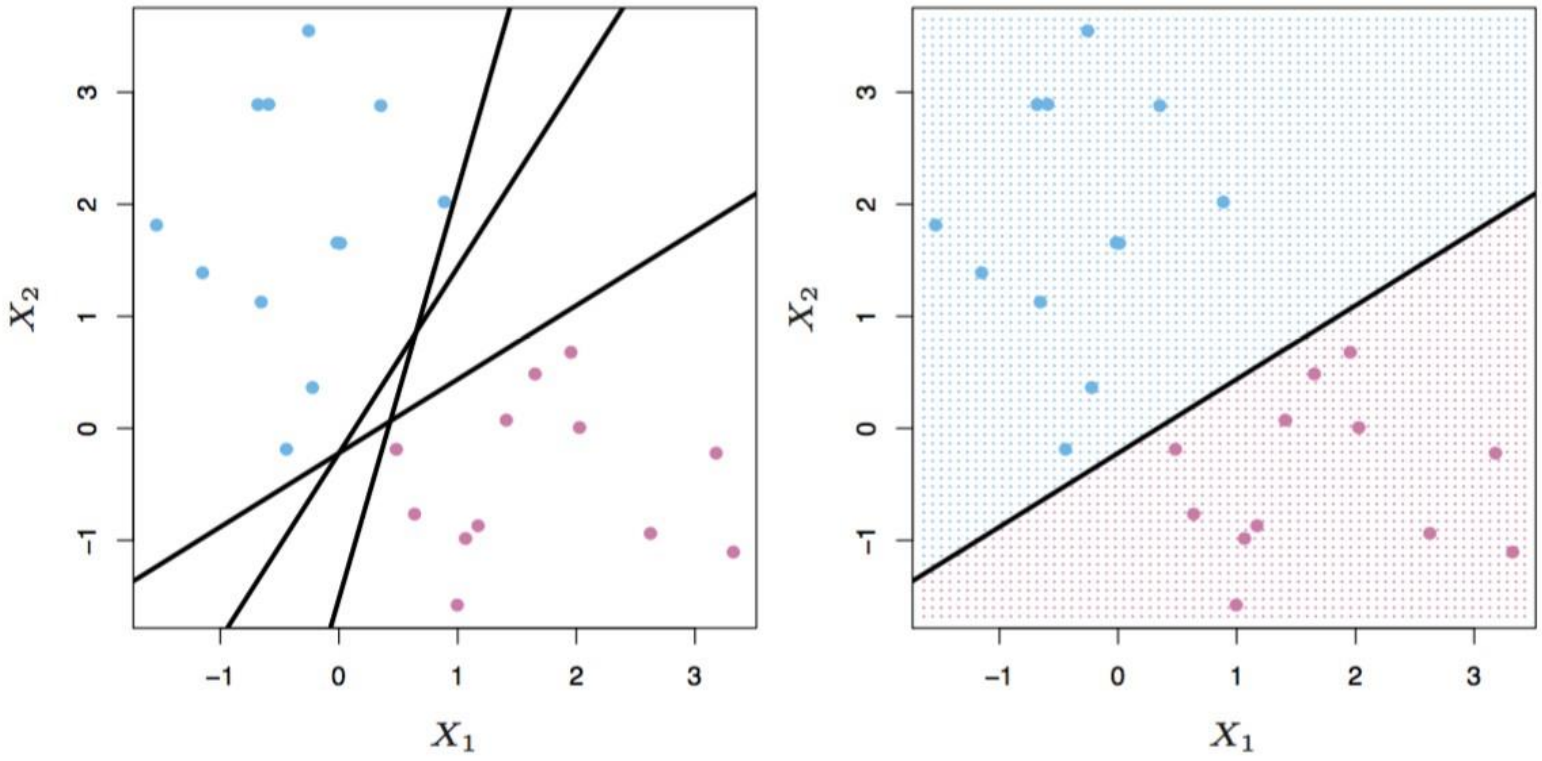
Support Vector Machine (SVM)

Essentially, this is a **constrained optimization problem** where the **margin is maximized** subject to the constraint that it **perfectly classifies the data**

Intuitively, the "maximum-margin" line allows for noise and is most tolerant to mistakes on either side. So that a good thing to look for!

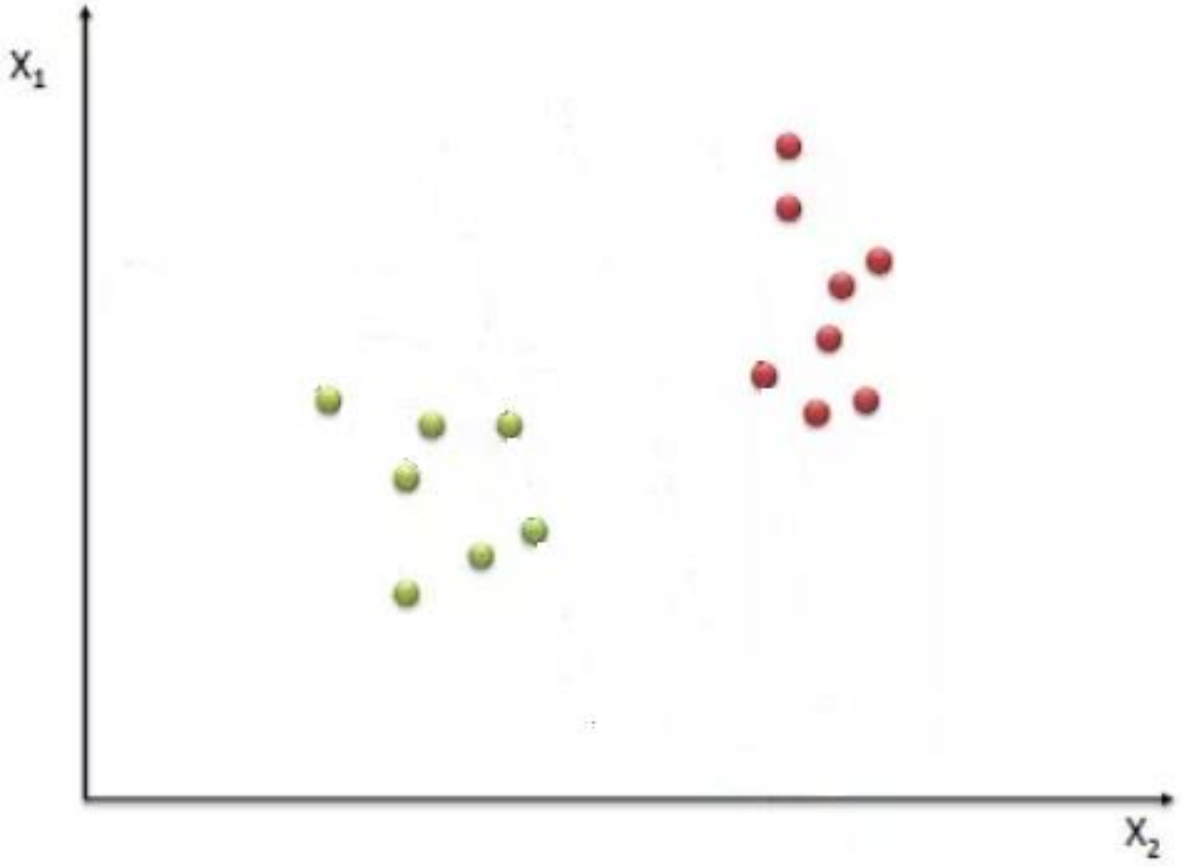
Support Vector Machine (SVM)

In the previous example, there are an infinite number of lines that will accomplish this task, but only **one** "maximum-margin" line

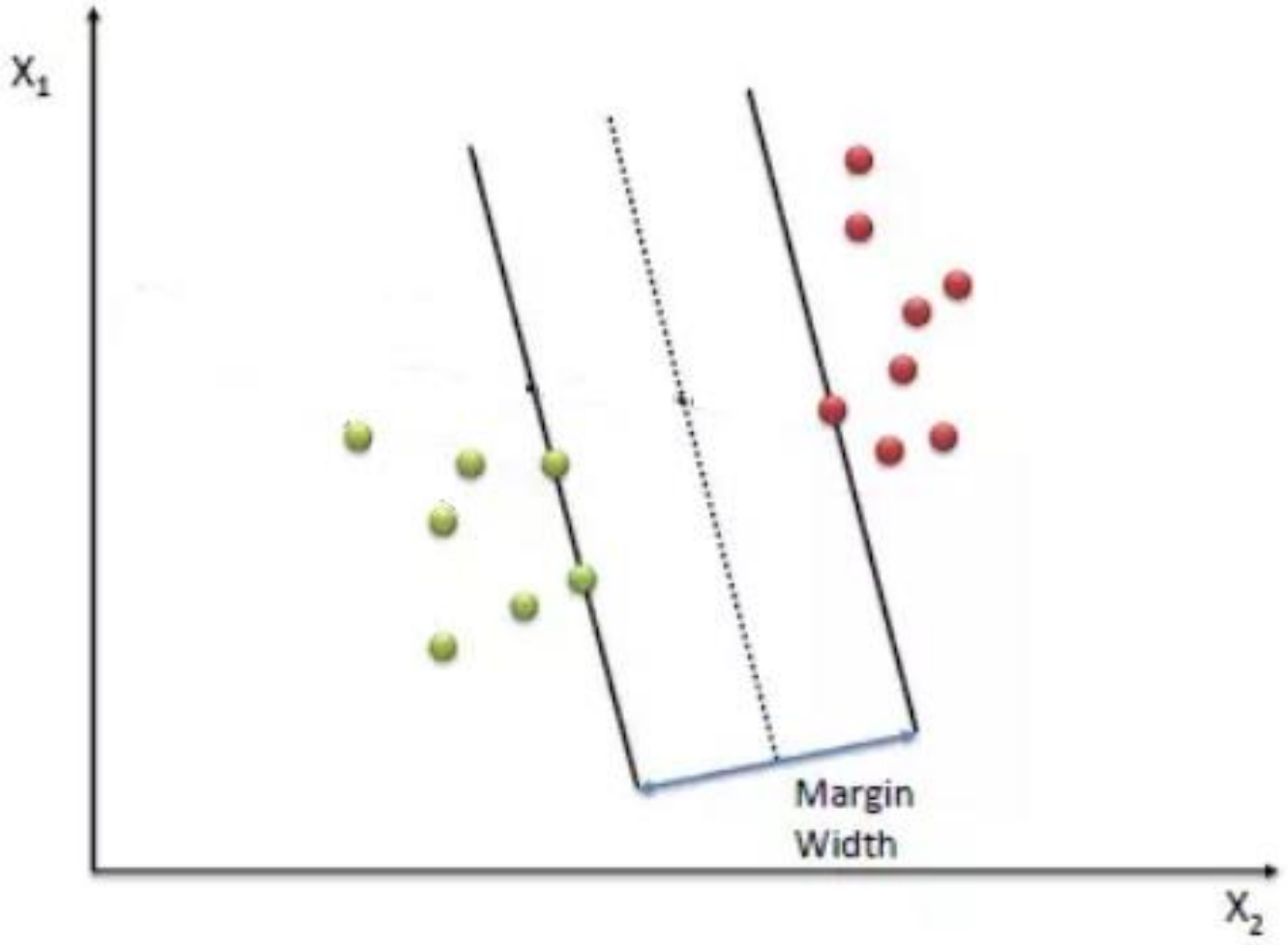


Support Vector Machine (SVM)

Another example: suppose that we want to split the below red circles from the green ones by drawing a line



Support Vector Machine (SVM)

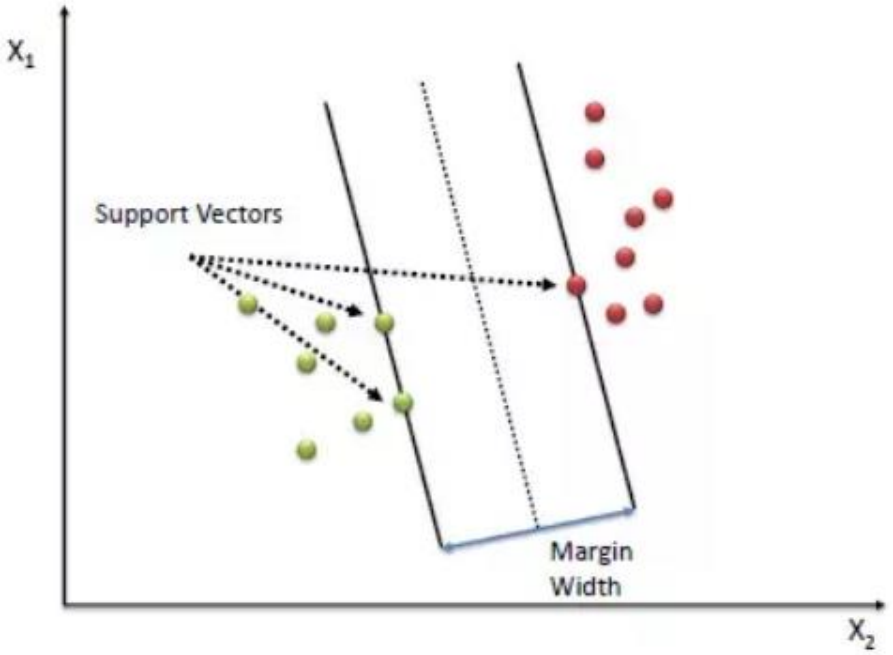


Support Vector Machine (SVM)

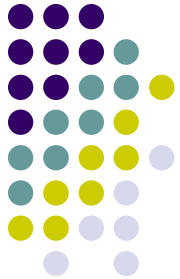
The data points that kind of "support" this hyperplane on either sides (i.e., the closest training points to the line) are called the **support vectors**

Support Vectors are simply the co-ordinates of individual observations (i.e., in text analysis: documents)

In the figure, there are 3 support vectors, so at the test time, we will compute similarity test point **on only these 3 support vectors**



Support Vector Machine (SVM)



So far, we have assumed that a hyperplane can **perfectly separate instances across classes**

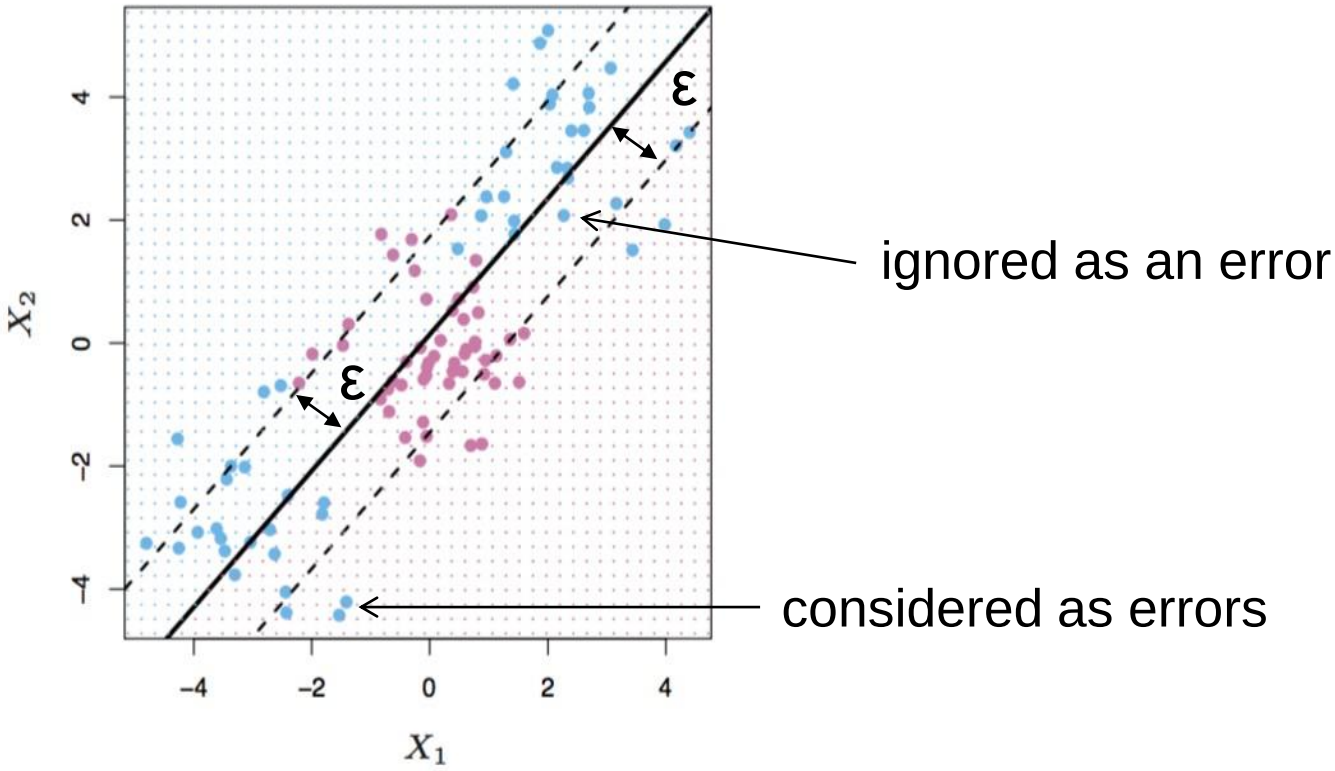
When this is not the case, we must relax the constraint imposed on the distances between points and the hyperplane, and allow for a **certain amount of slack**

This slack will allow for instances to be within the margin, or even to cross the (quasi)separating hyperplane

Support Vector Machine (SVM)

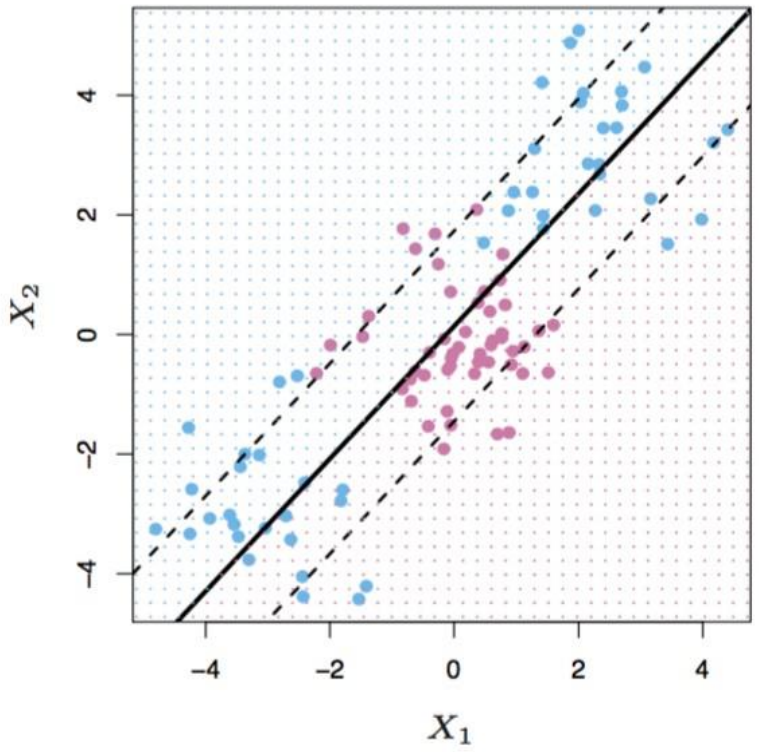
In this respect we can define a *loss function* that *ignores* those errors which are situated within the certain distance of the true value

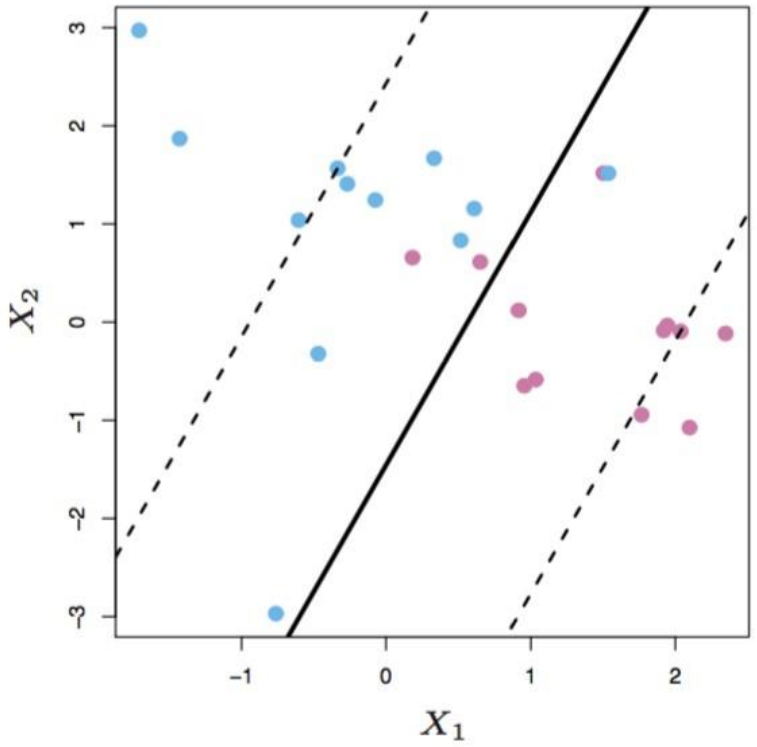
This type of function is often called “epsilon (ε) intensive-loss function”



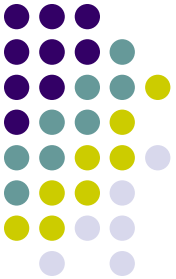
Support Vector Machine (SVM)

The width of the intensive-loss zone can of course be (very) different!





Support Vector Machine (SVM)

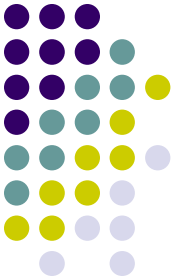


This function allows us to identify the **cost** of the errors on the training points

These are zero for all points that are inside the band (i.e., that are within ε distance of the observed value), and larger than 0 for all points outside of it

This penalty for the errors is known as *C* (i.e., cost) and it quantifies the *penalty* associated with having an observation on the wrong side

Support Vector Machine (SVM)



With no perfect separation, the goal is therefore to minimize our sum of classification errors, conditioning on the **tuning parameter** *C* (i.e., cost) that indicates tolerance to errors

In particular, parameter *C* determines the trade-off between the *model complexity* and the degree to which deviations larger than epsilon are *tolerated* in optimization formulation

Support Vector Machine (SVM)

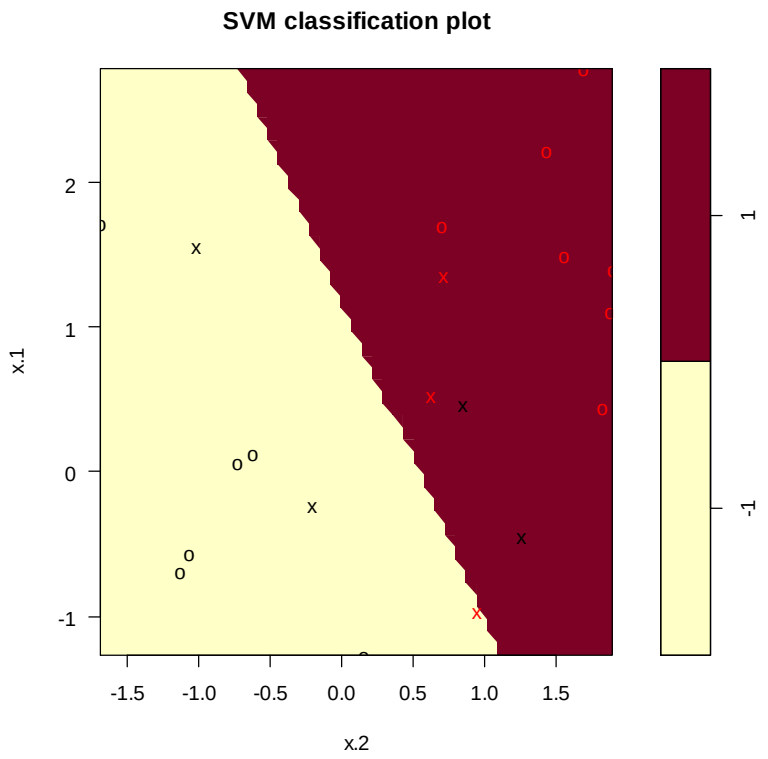
Larger values of *C* (ex. *C* = 1000 , a value which penalizes a lot the model for misclassified observations) thus result in greater focus of attention on the points located **very close** to the decision boundary (for a given ε), i.e., only those **instances near the class boundary play a big role its definition**, while those that remain far away from the boundary have **little effect** on its location and direction

…while smaller values of *C* (ex. *C* = 0.01, a value which doesn't penalize the model much for misclassified observations) involve an attention also on data points **farther away** (for a given ε). It is these points that now can also become support vectors

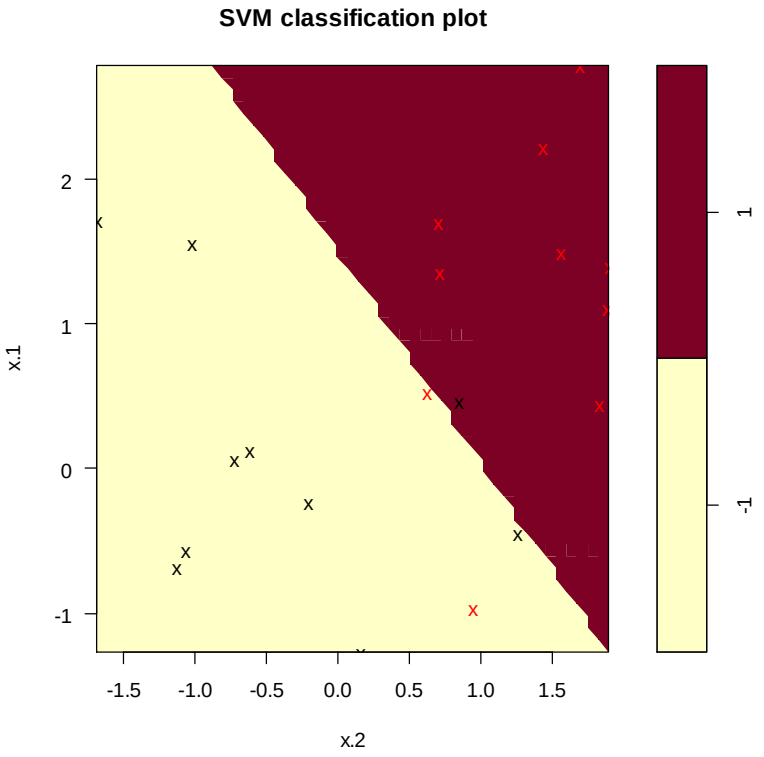
Support Vector Machine (SVM)

An example with two labels (red and black points) and two different values for *C* (with epsilon fixed to 0.1)

C=10; SV points (i.e., the x)=7



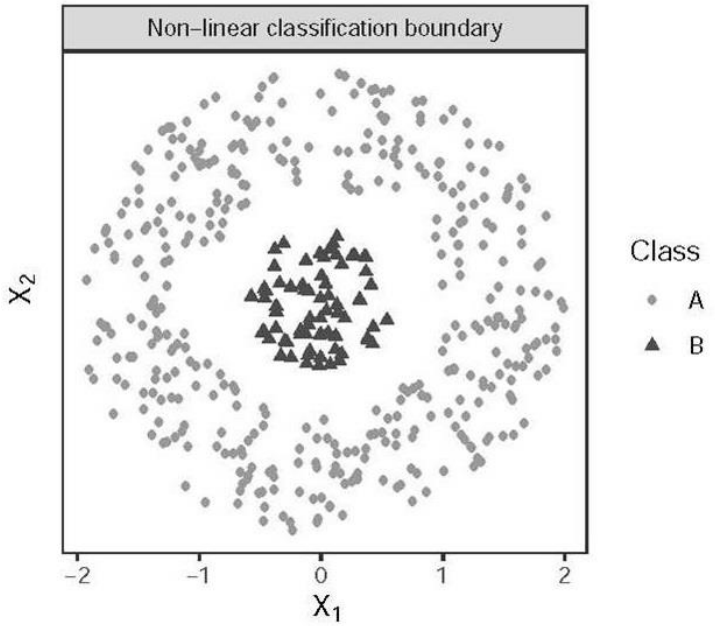
C=0.01; SV points=ALL! (although of course with a different weight!)



Support Vector Machine (SVM)

But what if we don't want (or **cannot**) fit a straight line to find support vectors (for example in 2 dimensions)?

For instance, in the figure below, although the two classes are easily recognized as occupying different regions of feature space, no hyperplane across it would result in a good separation, The optimal decision boundary, which in this case corresponds to a circle, is not linear



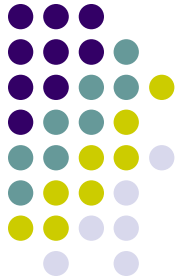
Support Vector Machine (SVM)

So what to do? **Move to non-linearity**!

We can achieve this however *not* by drawing curves, but by "lifting" the features we observe into higher dimensions…

….i.e., instead of operating on the space defined by the *original set of predictors* (where no linear boundary can correctly separate classes of the target outcome), we can operate on a tra*nsformed space of higher dimensions* in which linear separability becomes (again) possible

Support Vector Machine (SVM)



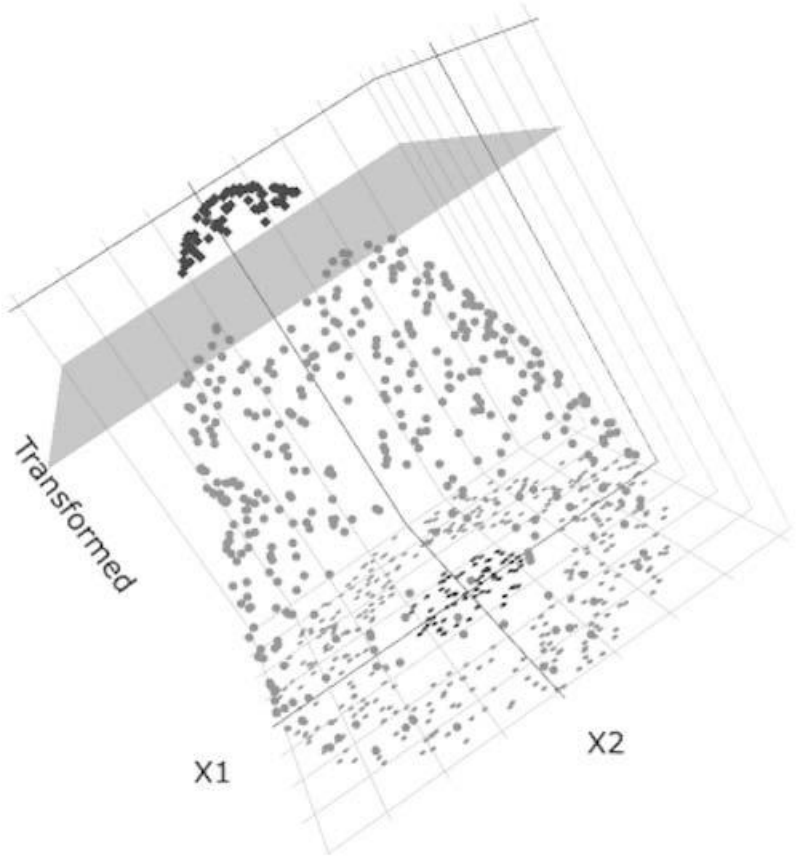
For example, if we can't draw a line in the space (x1, x2) then we may try adding a third dimension, (x1, x2, x1 * x2)

This is known as a **kernel trick**, that can be linear, radial, polynomial, etc.

Support Vector Machine (SVM)

Going back to the previous figure, suppose we add a third feature equal to the negative sum of squares of the original predictors (i.e., $x_1 + x_2 – (x_1^2 + x_2^2)$)

This results in the 3D scatterplot depicted on the figure below



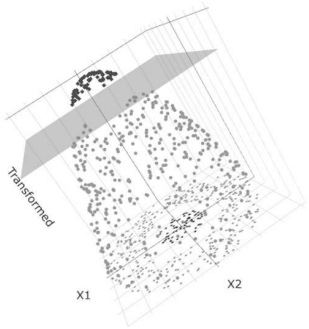
Support Vector Machine (SVM)

In this new, three-dimensional feature space, observations are now arrayed on a conical surface, with instances of class B (i.e., the triangles) rising to its apex

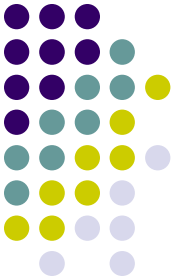
It is now easy to define a *plane*, depicted in gray, that cuts the top of this cone and separates instances of the two classes

The projection of this separating plane back onto the original two-dimensional space generates the circular decision boundary we needed

Once again, the SVM's goal is to learn this separating hyperplane



Support Vector Machine (SVM)



You can employ different Kernel transformations:

Popular SVM Kernels

Kernel	Form
Linear	$K(x, x') = \langle x, x' \rangle$
dth Degree Polynomial	$K(x, x') = (1 + \langle x, x' \rangle)^d$
Radial Basis Function	$K(x, x') = \exp(-\|\|x - x'\|\|^2/c)$

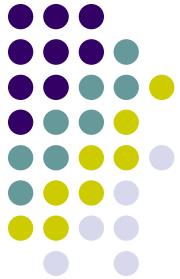
The *Linear* kernel is the simplest kernel function. It is given by the inner product <x,y> plus an optional constant

Linear SVMs are usually preferable to other kernels when the number of features in your problem is larger than the number of instances (“large p, small n”)

This is the typical situation you have with text-analytics!

So, a linear Kernel can be considered as the (possible) first-best when dealing with texts

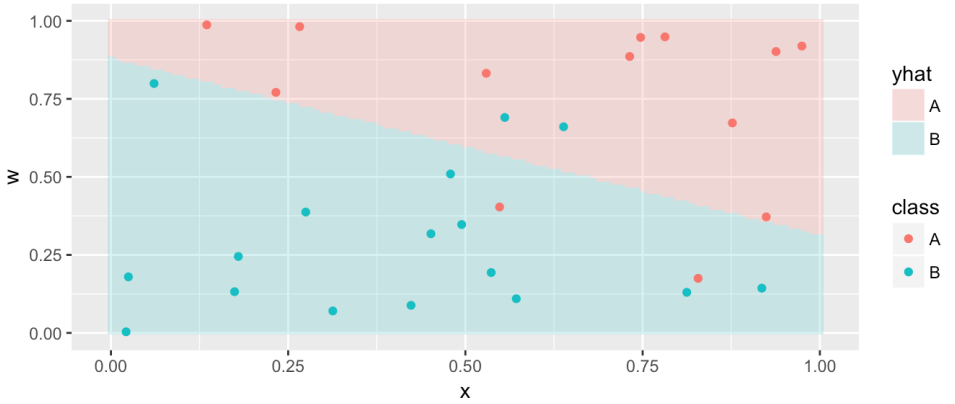
Support Vector Machine (SVM)



With other type of analysis, the Radial Kernel is a reasonable (and popular) first choice

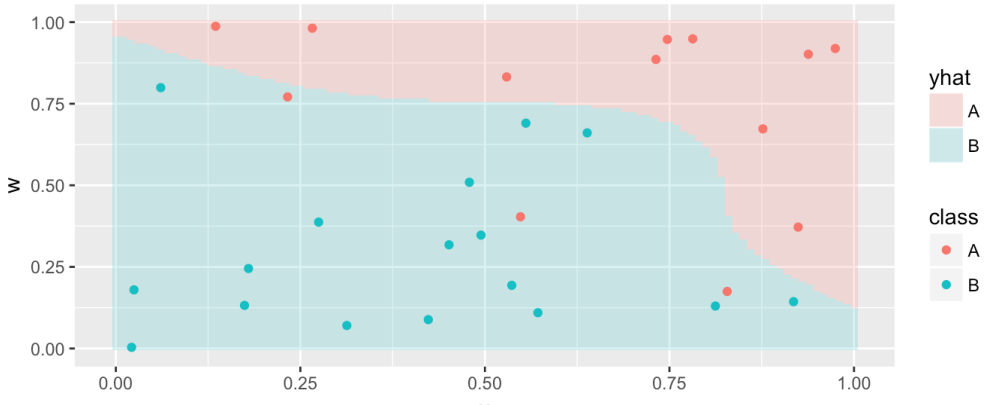
In particular, in those cases where groups are not nicely separated by lines or (hyper)planes, the Radial Kernel is able to carry out non-linear partitioning

Support Vector Machine (SVM)

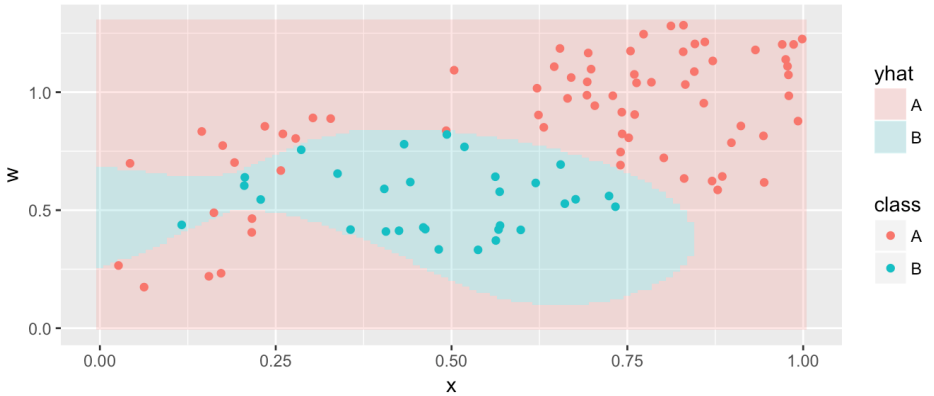


Linear Kernel

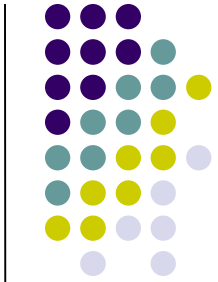
Polynomial Kernel



Radial Kernel



Support Vector Machine (SVM)



How does SVM work with a multi-category classes? SVM implements a one-vs-one approach, that is a SVM classifier is trained to distinguish one class from another

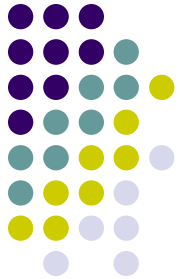
For M classes (larger than 2), however, you have M*(M-1)/2 combinations, which is also the number of resulting classifiers

For example, if we had 3 classes we get 3 classifiers, i.e., class1-vs-class2, class1-vs-class3, class2-vs-class3

Practically, in each case you suppose that only 2 classes exist, and you train a classifier to distinguish among them

During the prediction phase, you then choose the class with a majority vote among all the classifiers

Support Vector Machine (SVM)



SVMs have been most successfully used to solve classification problems, particularly when the number of predictive features is much larger than the number of observations n (as it happens with texts!!!)

For medium-sized datasets, Support Vector Machines (SVMs) provides, quite often, an excellent choice