Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland

MSE | MASTER OF SCIENCE IN ENGINEERING

Machine Learning

T-MachLe

7. Support Vector Machines - SVM

Christophe Gisler
Jean Hennebert
Andres Perez Uribe

Plan - Classification task with Support Vector Machines (SVM)

7.1 Recaps on classification task and logistic regression

7.2 Linear SVM

7.3 Nonlinear problems

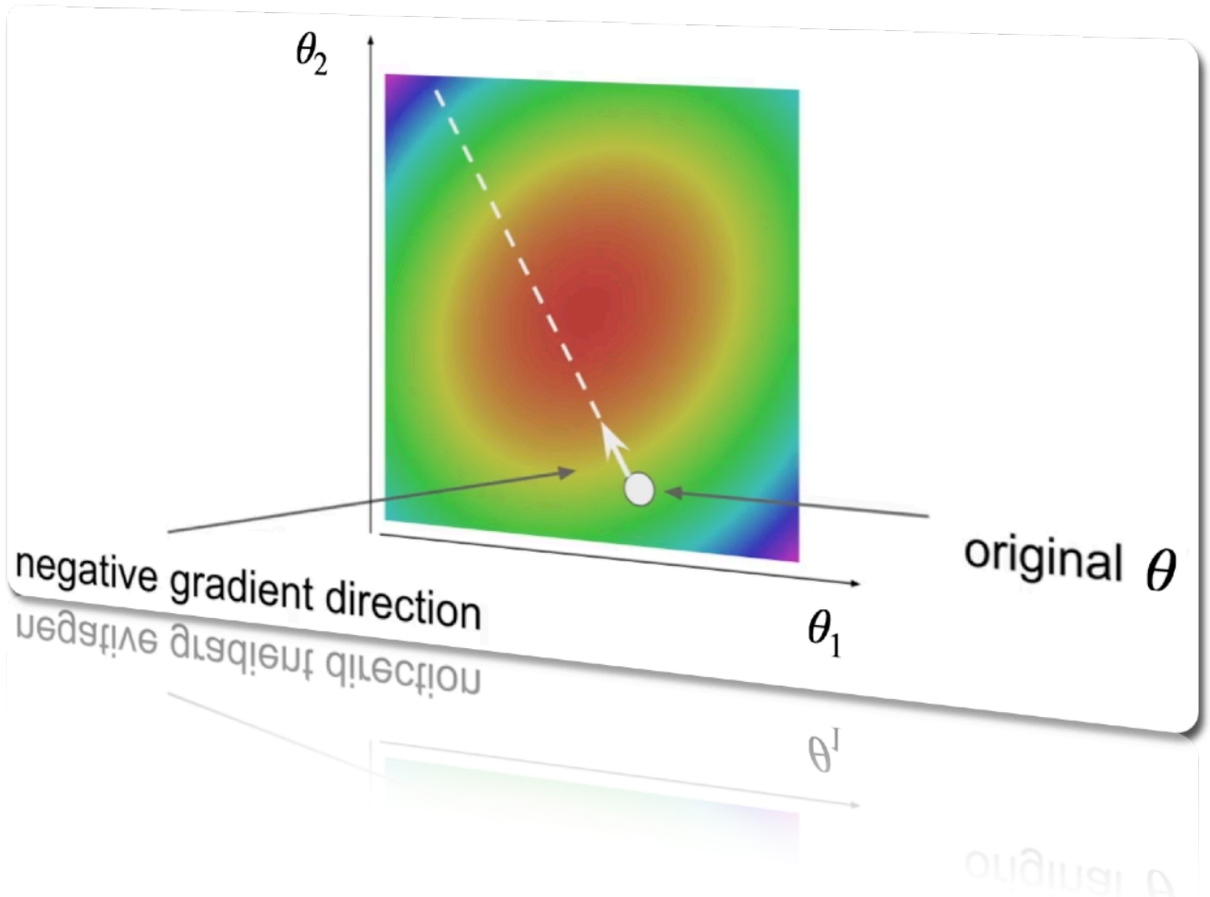
7.4 Multiclass SVM

7.5 History & References

Practical Work 7

7.1 Recaps and intro to SVM

Generative vs. Discriminative approaches



Classification task

- Example:
  - **2 classes**:

    Women (•)

    Men (•)
  - **2 features**:

    x_1 = Hair length (axe X)

    x_2 = Weight (axe Y)
  - **What about the new sample (•)?**
    Is it a woman or a man?

Bayesian model seen in Chap 3

Logistic regression seen in Chap 6, SVM seen today, ANNs in chapters >=9

Generative vs **discriminative models**

- **Generative models** are modelling the feature distributions in the respective classes, i.e. the likelihoods
  - training points $\mathbf{x}$ of class C_k is used to model $p(x \mid C_k)$
    - not very good usage of a given $\mathbf{x}$ as we "only" learn to represent the associated class, i.e. $\mathbf{x}$ of class C_k has no impact on classes other than k
- **Discriminative models** are modelling the border between classes
  - training points $\mathbf{x}$ of class C_k is used to model borders between the class C_k and the other classes C_i
    - good usage of a given $\mathbf{x}$ as it impacts the border between all classes

Bayesian model seen in Chap 3

Logistic regression seen in Chap 6, SVM seen today, ANNs in chapters >=9

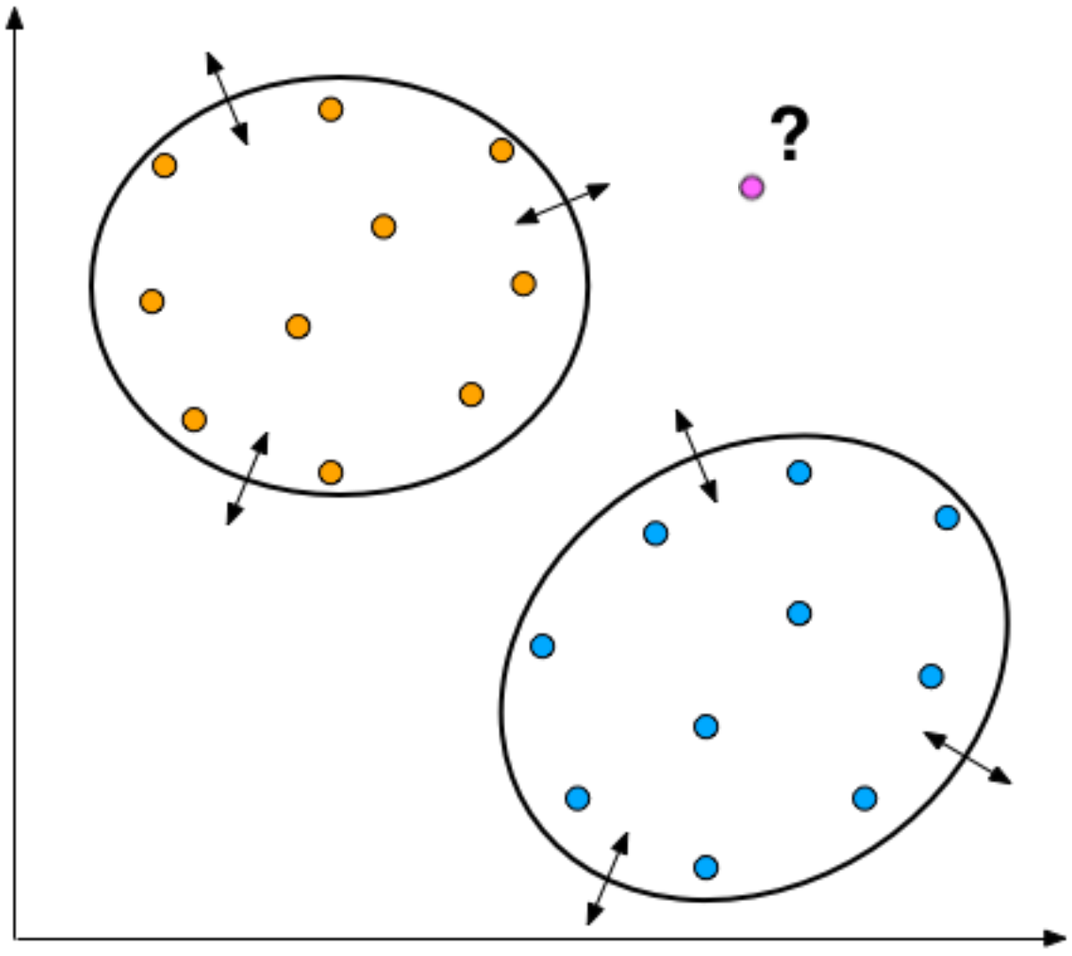
Generative vs discriminative models

K = number of classes, D = number of features

	Generative	Discriminative
Complexity inference	O(D.K) for inference, which is pretty slow if K and D are large	Generally better inference time as we model a “border”, close to O(D) for logistic regression, can tend to (K.D) for complex models (deep).
Complexity training	Depend to the form of likelihood estimator. Can be parallelized for each class. No need to retrain if a new class appears or if priors are changing.	Depend to the form of “border” estimator. Need to retrain if new class, or if priors are changing.
Performance	Usually decent performance are obtained. Facing difficulties if dim D of features is large: lots of training points is needed, simplifying hypotheses such as naive Bayes	Generally discriminative approaches perform better than generative. Cope well with larger D. Simplifying hypotheses are less frequently done.
Interpretability	Quite good. The mathematical background is strong. We can decompose the decision between priors and likelihoods.	Sometimes opaque. We fall on one side of the border but we often have difficulties to say why.
Sensitivity	Good behaviour with unbalanced classes. The unbalances are measured in the priors.	Difficulties with strong unbalanced classes. Risk of learning “just the priors”.

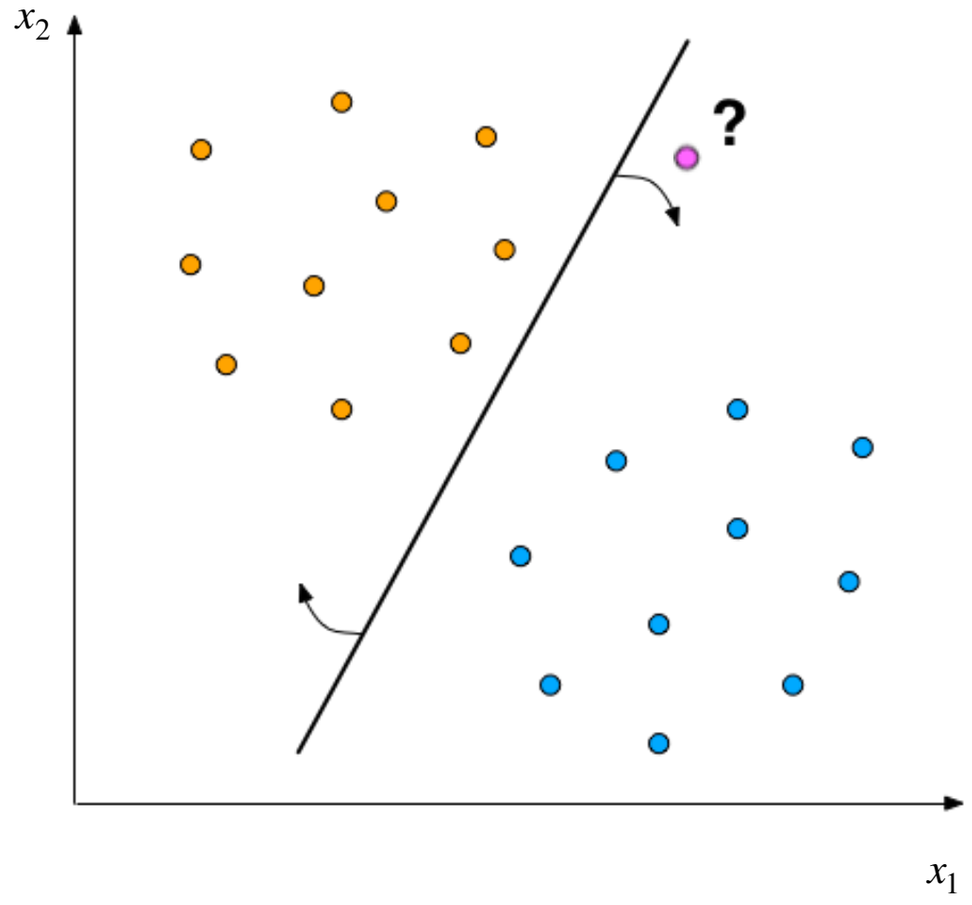
Classification task - generative models

- One **generative model per class**
- Typically the generative model estimates the distribution of x within the class
  - E.g. Bayesian approach with likelihood estimation using Gaussian Mixture Models (**GMM**)



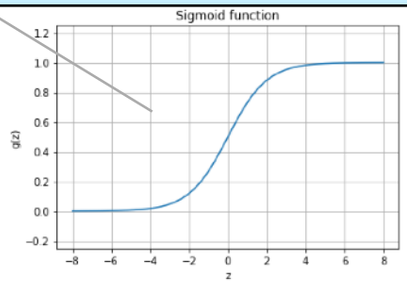
Classification task - discriminative models

- Some algorithms try to **discriminate classes**, i.e. to build a border defining the classes
- Last week we have seen **Logistic Regression**
  - Discover the equation of the border that maximise the correct classifications
  - The equation of the border is feeding a sigmoid so that on one side of the border the probability value is larger than 0.5 (class 1) or smaller (class 0)
  - Logistic regression may find many possibilities of hyperplanes
- Which is the **best separating hyperplane**?

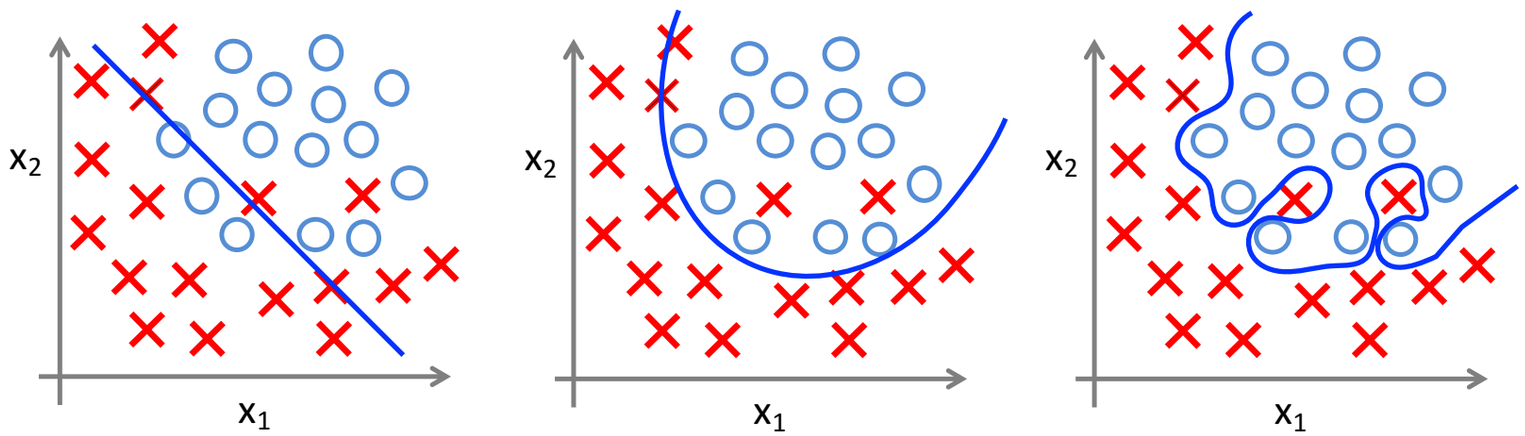


$$\hat{y} = P(C_1 \mid x) = 1 - P(C_0 \mid x)$$
$$\hat{y} = h_\theta(\mathbf{x}) = sigmoid(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

The sigmoid is a bit like a "sign" function



Logistic regression with non-linear decision boundaries



$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(g = sigmoid function)

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$

- To move to non-linear decision boundaries, just add features to the x array
  - e.g. representing dependencies to the squared

$$h_\theta(\mathbf{x}) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \dots)$$

$$h_\theta(\mathbf{x}) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \dots)$$

with $x_3 = x_1^2$

x1 = surf	x2 = rooms	x3 = surf^2	y = rented
26	1	26*26 =	0
37	1.5	1369	1
57	2	3249	0
48	2	2304	1
…	…	…	…

Training discriminative models

- A common strategy to find the best parameters is the **gradient descent**.

Or its negative version: an **objective** function $-J(\theta)$ to maximise.

Gradient descent involves the computation of two mathematical terms:

- A **loss function** $J(\theta)$
  - It expresses how bad we are with the current values of parameters θ on the train set
  - We want to minimise this function through the training procedure
- The **gradient of the loss** with respect to the parameters $\frac{\partial J}{\partial \theta}$

- The negative of the gradient will tell us in which direction to move the parameters to minimise the loss

Training with a gradient descent

- Update any parameters of your model in the opposite direction of the gradient of the loss w.r.t. weights

$$\text{param} \leftarrow \text{param} - \alpha \frac{\partial J}{\partial \text{param}}$$

In practice we have stopping criteria, e.g. epoch < max_epochs or loss_gain < ε

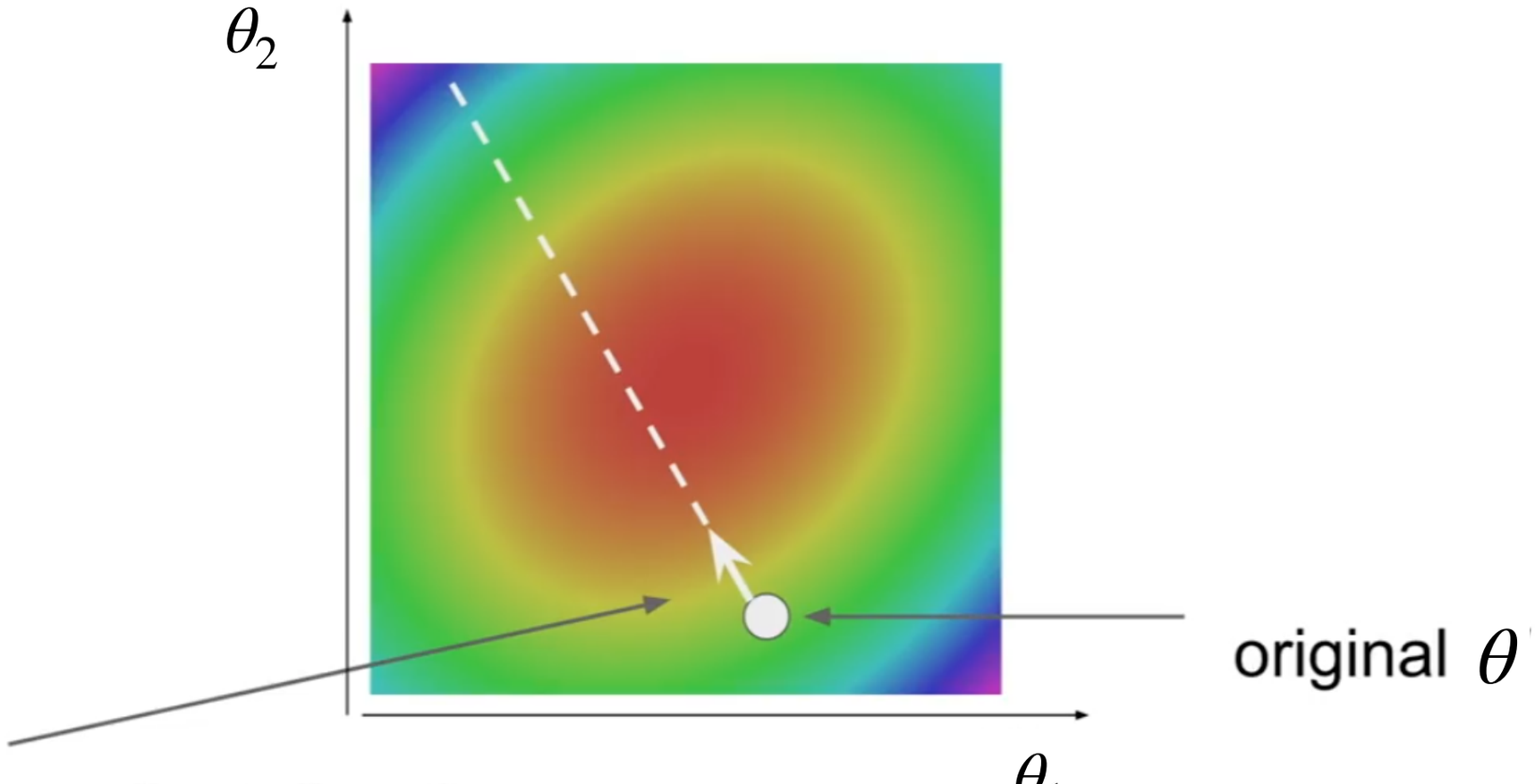
1 sample = stochastic gradient descent
B samples = mini-batch gradient descent
N samples = (full) batch gradient descent

Current weighs

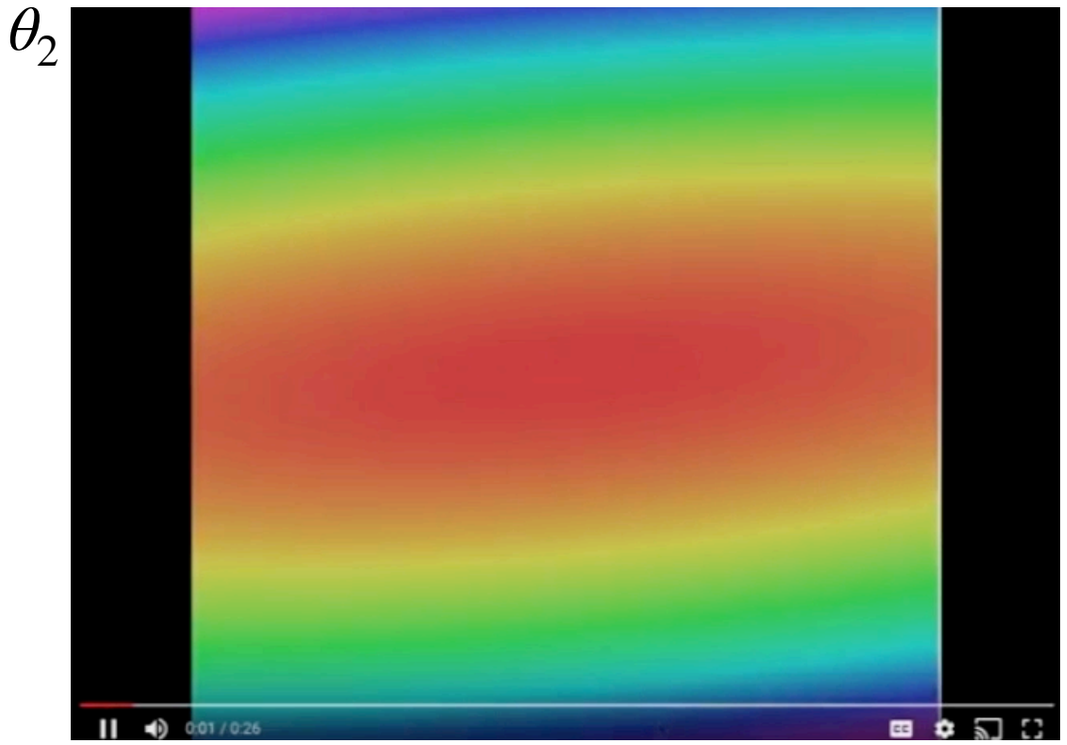
```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

The loss function



From Fei-Fei Li & Justin Johnson & Serena Yeung, April 2018: CS231n Stanford



Loss function and gradient for logistic regression

- We want to maximise the number of correct classifications, i.e. the product over the training set of the a posteriori probabilities.
  - As the derivation of the product is not easy, we took the log of the product which transformed as a sum of the logs.

"Cross Entropy Loss"

$$J(\theta) = -\frac{1}{N}\sum_{n=1}^{N} J(\theta, x_n) = -\frac{1}{N}\sum_{n=1}^{N} y_n \log h_\theta(\mathbf{x_n}) + (1 - y_n)\log(1 - h_\theta(\mathbf{x_n}))$$

Note : in the last class, the negative term was not there and we had to maximise the function J

When $y_n = 1$ $J(\theta, x_n) = -\log(h_\theta(\mathbf{x_n})) = -log(\hat{y}_n)$

When $y_n = 0$ $J(\theta, x_n) = -\log(1 - h_\theta(\mathbf{x_n})) = -\log(1 - \hat{y}_n)$

- After some mathematical developments, the gradient of the loss w.r.t. the parameter is

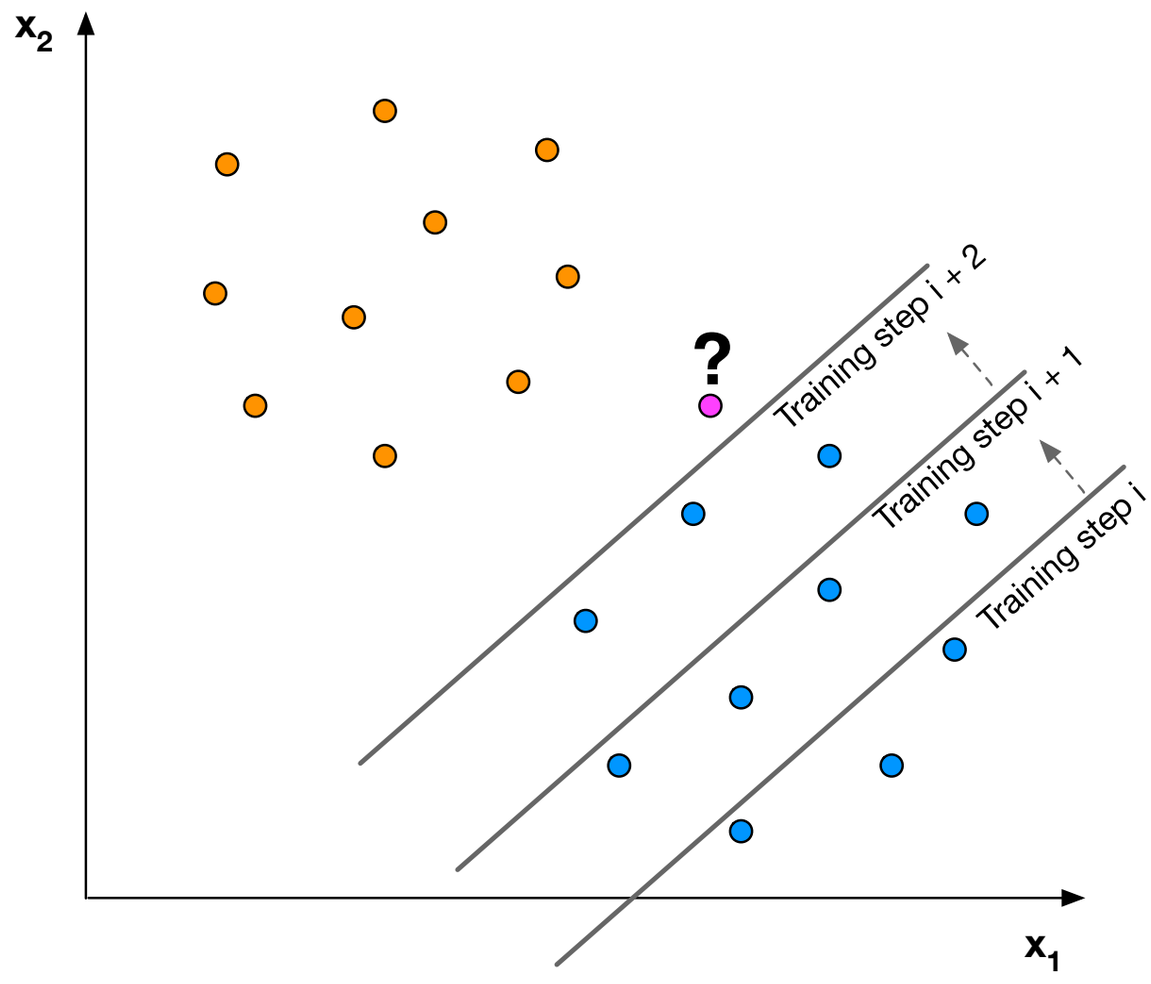
$$\frac{\partial J(\theta)}{\partial \theta_i} = \frac{1}{N}\sum_{n=1}^{N}(y_n - h_\theta(\mathbf{x}_n))x_{n,i}$$

Target value

Gotten output

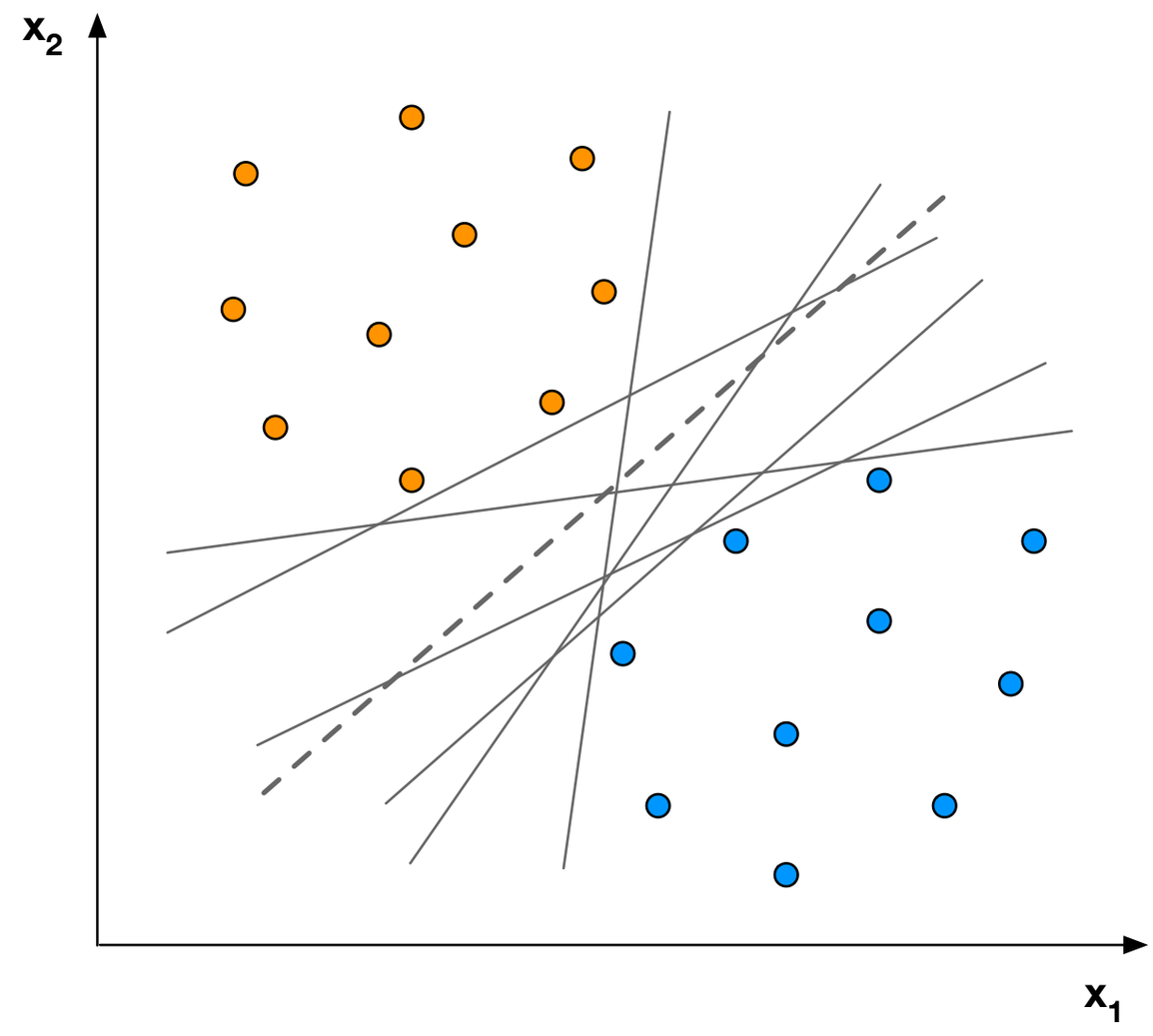
Problem 1 with logistic regression

- If the problem is linearly separable, the border will move until all training samples will be correctly classified, then it will quickly stop.
  - In this example, $J(\theta) \approx 0$ in training step i + 2 and there is no more update of the parameters.
- The new (●) input is classified to the yellow class, which is probably not correct in this case.



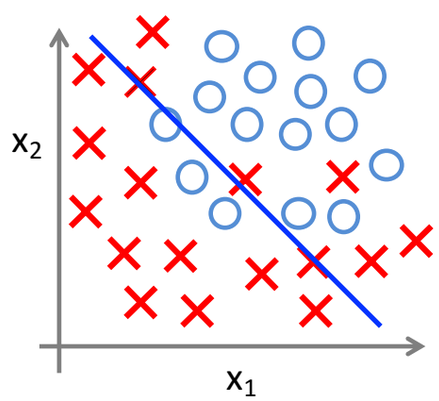
Problem 1 with logistic regression

- Actually all the borders here give $J(\theta) \approx 0$ and are potential solutions to the problem
- Is the dashed solution a better solution?
  - Probably yes as it "*maximises the margin*" between the border and the samples of the classes



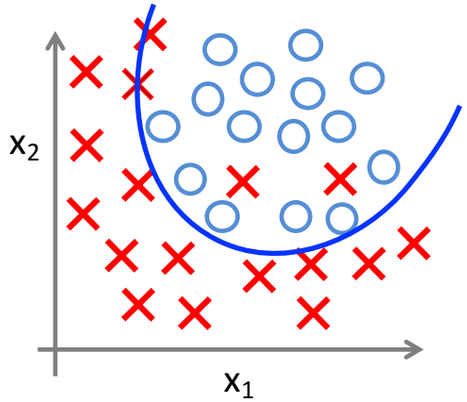
Idea 1 for SVM: let's include a performance term in the loss that will "*maximise the margin*".

Problem 2 with logistic regression

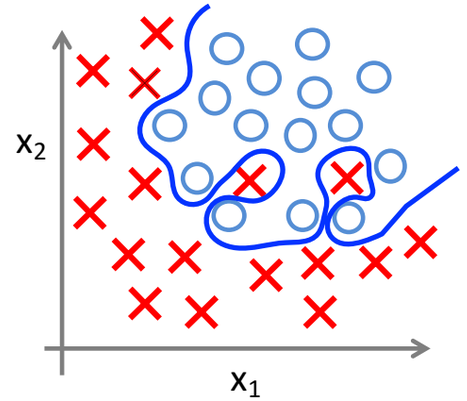


$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(g = sigmoid function)



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$



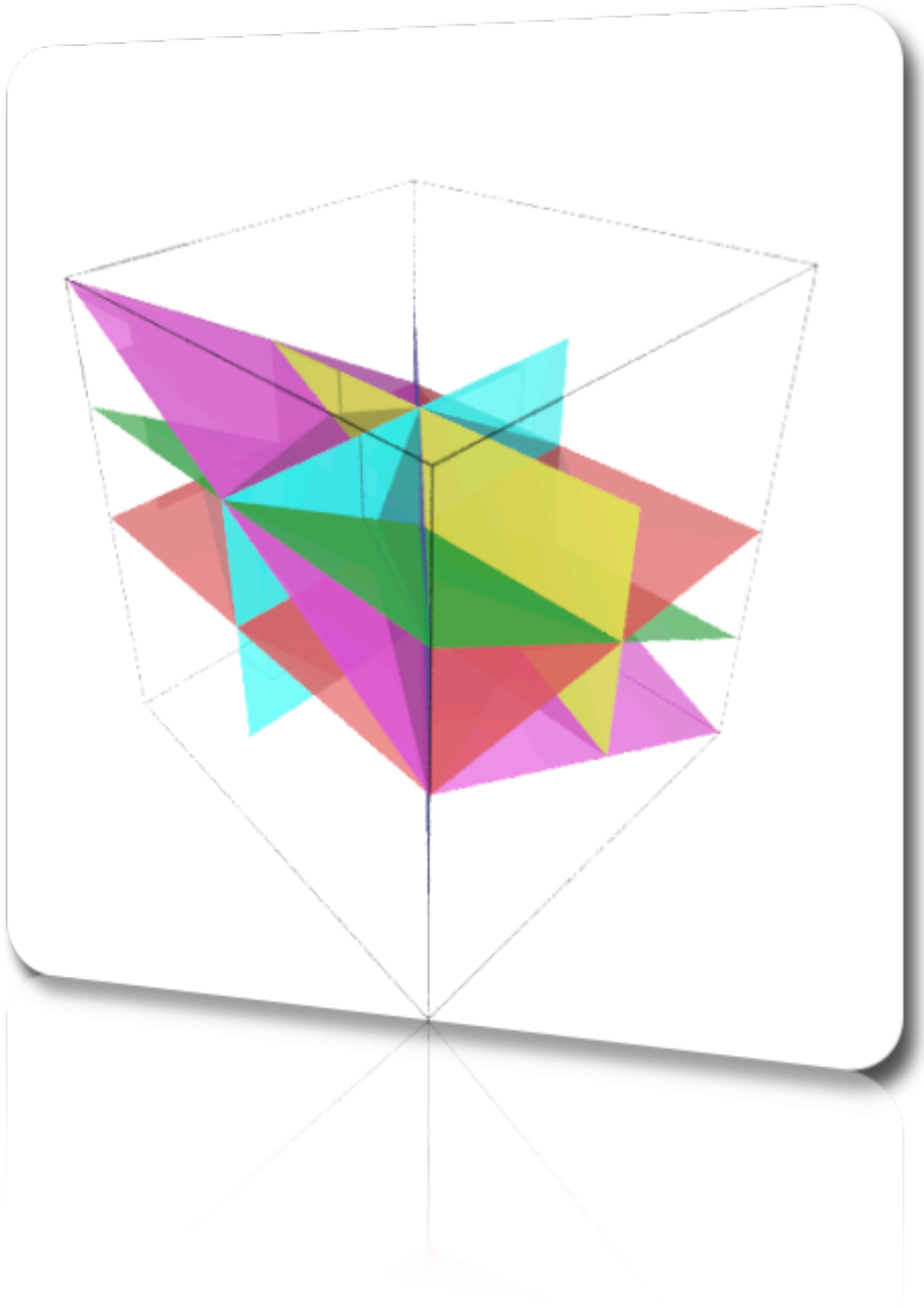
$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$

- We need to *manually* find the best feature "tricks" that will allow to separate non-linearly separable problems:
  - Try with x_1, x_2 : poor performances (left Fig.)
  - Try with x_1, x_2, x_1^2 : medium performances
  - Try with x_1, x_2, x_1^2, x_2^2 : better performances
  - Try with $x_1, x_2, x_1^2, x_2^2, x_1 x_2$: even better performances (middle Fig.)
  - …
  - Try with $x_1, x_2, x_1^2, x_1^2 x_2, x_1^2 x_2^2, x_1^2 x_2^3, x_1^3 x_2, \dots$: overfit (right Fig.)

Idea 2 for SVM: let's find a way to reduce this manual feature engineering trial and error by including a set of functions applied to the features that "project" the features in higher dimensions where it is simpler to separate the classes, i.e. the *kernel trick*.

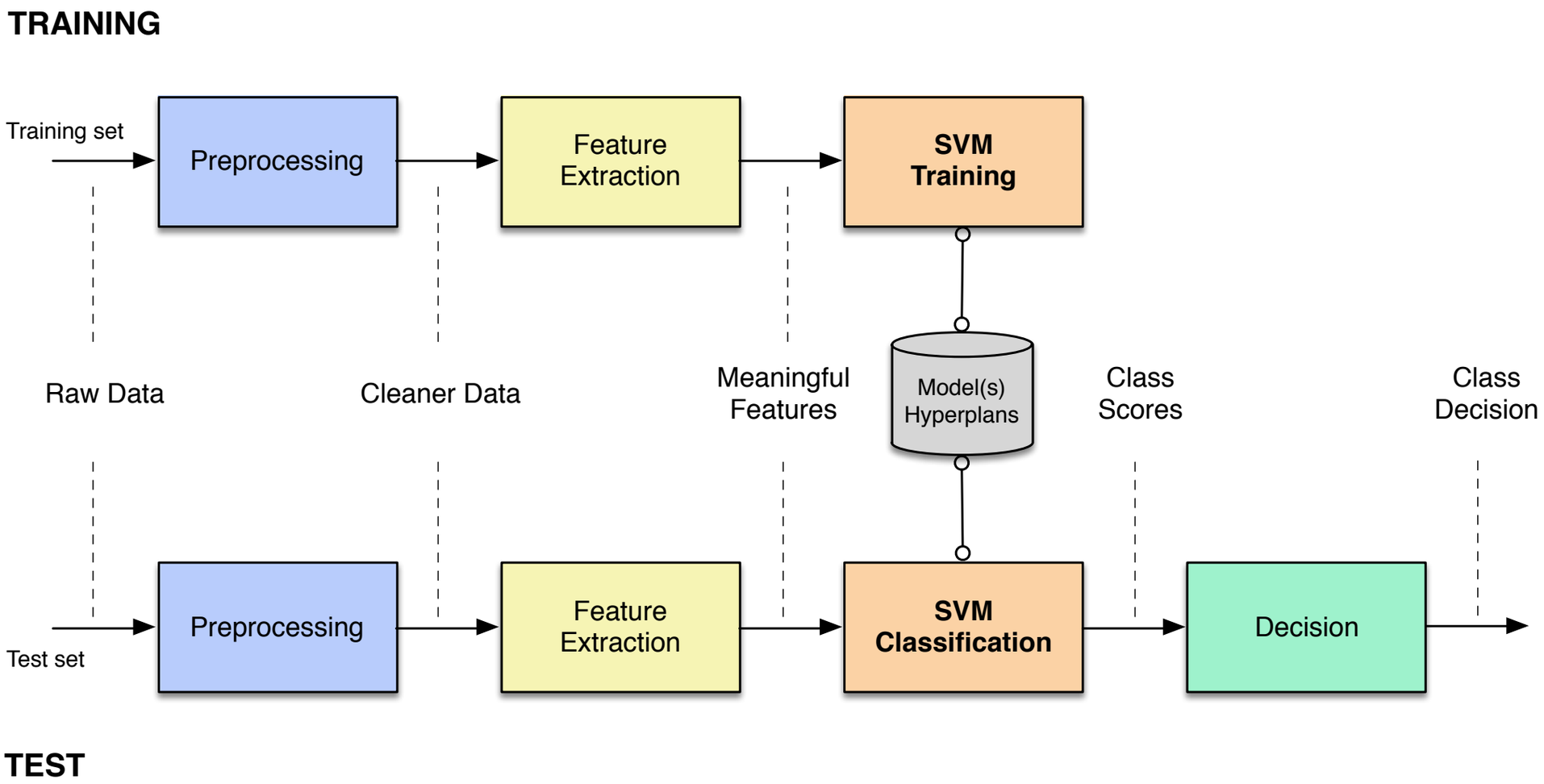
7.2 Linear SVM

- Hyperplane that maximises the margin
- Hinge loss
- SVM loss function
- Minimising loss with SciKit Learn



By Kilom691 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=37508909

Classification task



Linear SVM

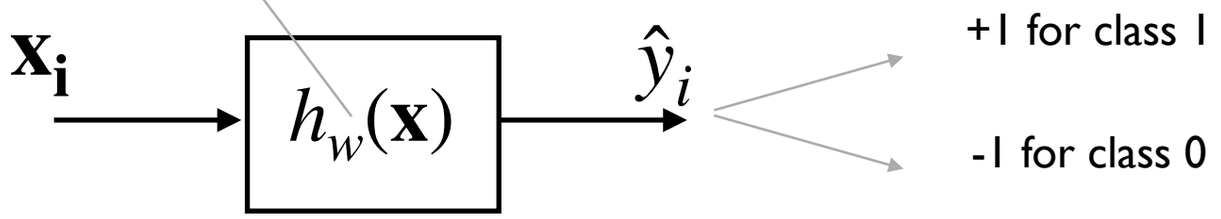
- Given:
  - Training samples $\mathbf{x_i} \in \mathbb{R}^D$
  - A two class problems with targets $y_i \in \{-1, +1\}$ associated to classes C_0 , C_1
  - Number of samples : N with i = 1,…,N
  - The hypothesis function:

$$h_w(\mathbf{x}) = sign(b + w_1 x_1 + \ldots + w_D x_D)$$
$$h_w(\mathbf{x}) = sign(b + \mathbf{wx})$$

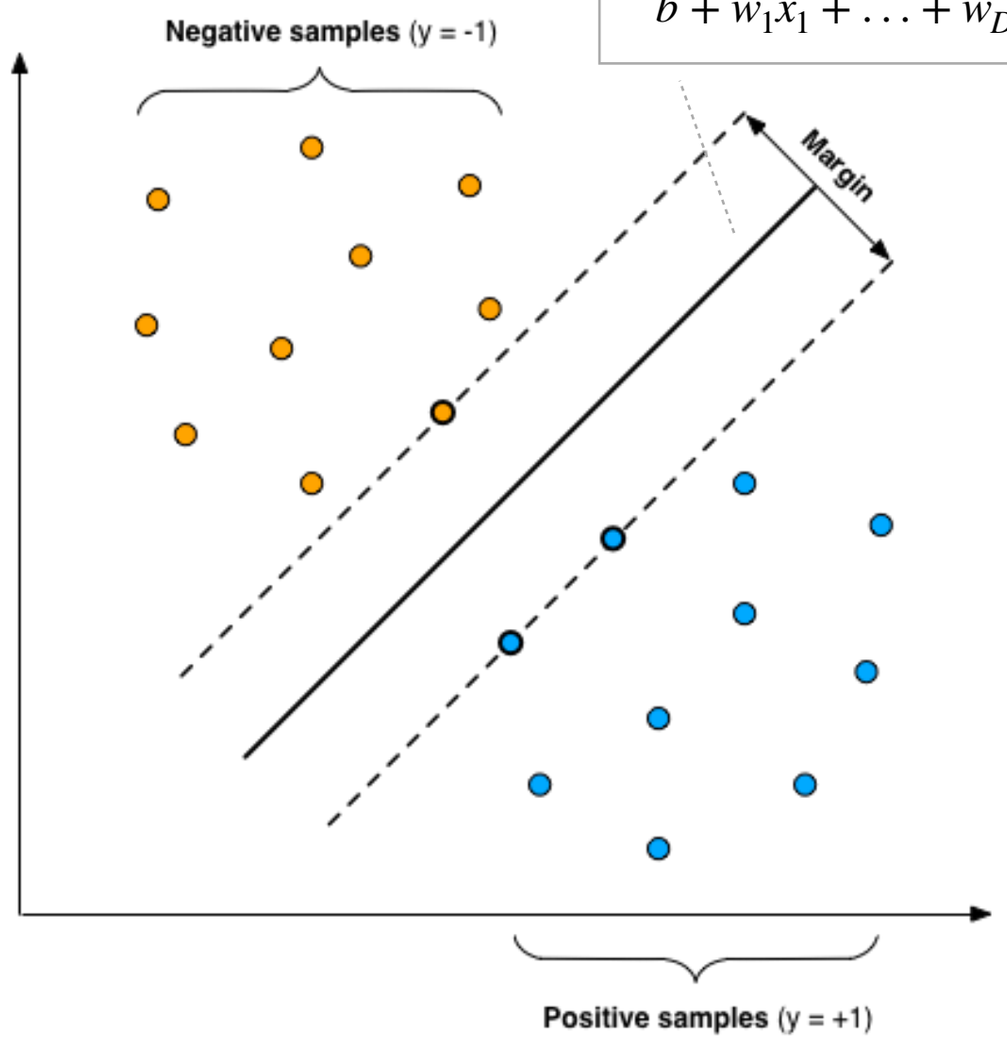
A linear SVM tries to find the **hyperplane** that **separates** the 2 classes and that **maximizes** the **margin** between the 2 classes

In previous classes, we used symbol θ instead of **w**

In logistic regression, we had +1, 0



Hyperplane equation:
$b + w_1 x_1 + \ldots + w_D x_D = 0$



Linear SVM

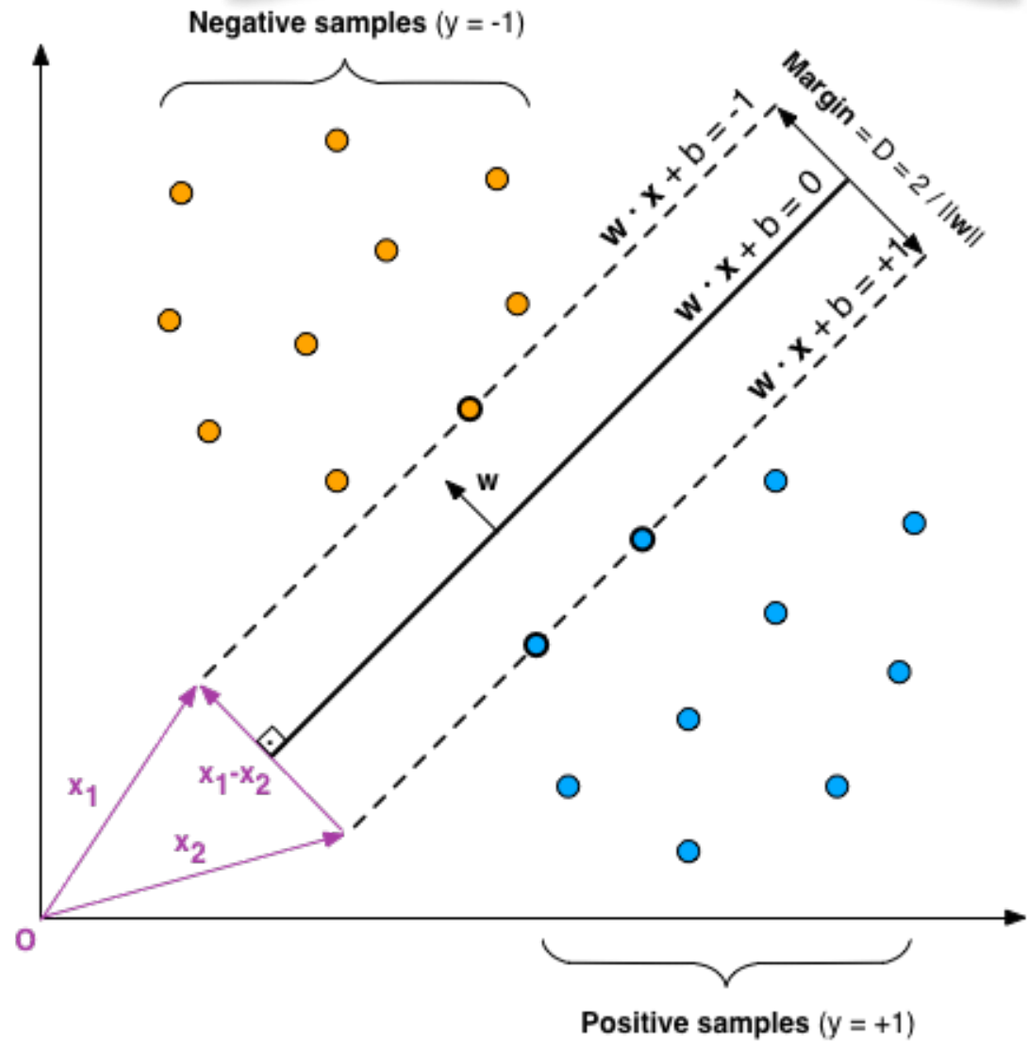
What does it mean to maximise the margin?

- Margin = distance M between the 2 parallel hyperplanes (on boundaries):
  $\mathbf{w} \cdot \mathbf{x} + b = -1$ (a is a constant >0)
  $\mathbf{w} \cdot \mathbf{x} + b = +1$
- Let's define $\mathbf{x_1}$, $\mathbf{x_2}$ as points on these 2 hyperplanes defining a perpendicular vector to the hyperplanes with : $\mathbf{x_1} - \mathbf{x_2} = t \cdot \mathbf{w}$ (t is a scalar)
- So we have: $\mathbf{x_1} = \mathbf{x_2} + t \cdot \mathbf{w}$
  $M = ||\mathbf{x_1} - \mathbf{x_2}|| = ||t \cdot \mathbf{w}|| = t \cdot ||\mathbf{w}||$ (1)
- We also have:
  $\mathbf{w} \cdot \mathbf{x_1} + b = -1$ (2)
  $\mathbf{w} \cdot \mathbf{x_2} + b = +1$ (3)
- (3) - (2) gives: $\mathbf{w} \cdot (\mathbf{x_2} - \mathbf{x_1}) = 2$
  $\iff \mathbf{w} \cdot (\mathbf{x_1} + t \cdot \mathbf{w} - \mathbf{x_1}) = 2$
  $\iff \mathbf{w} \cdot t \cdot \mathbf{w} = 2 \iff t\,(\mathbf{w} \cdot \mathbf{w}) = 2$
  $\iff t\,||\mathbf{w}||^2 = 2 \iff t = 2 / ||\mathbf{w}||^2$ (4)
- (4) in (1) gives: $M = 2 / ||\mathbf{w}||$
- Hence to maximize M, we have to minimize $||\mathbf{w}||$ while respecting the correct classification constraints (see next slide)

By geometric definition **w** is the normal vector of the hyperplane, i.e. perpendicular to the hyperplane

$||\mathbf{x}||$ is the Euclidian norm of vector **x**

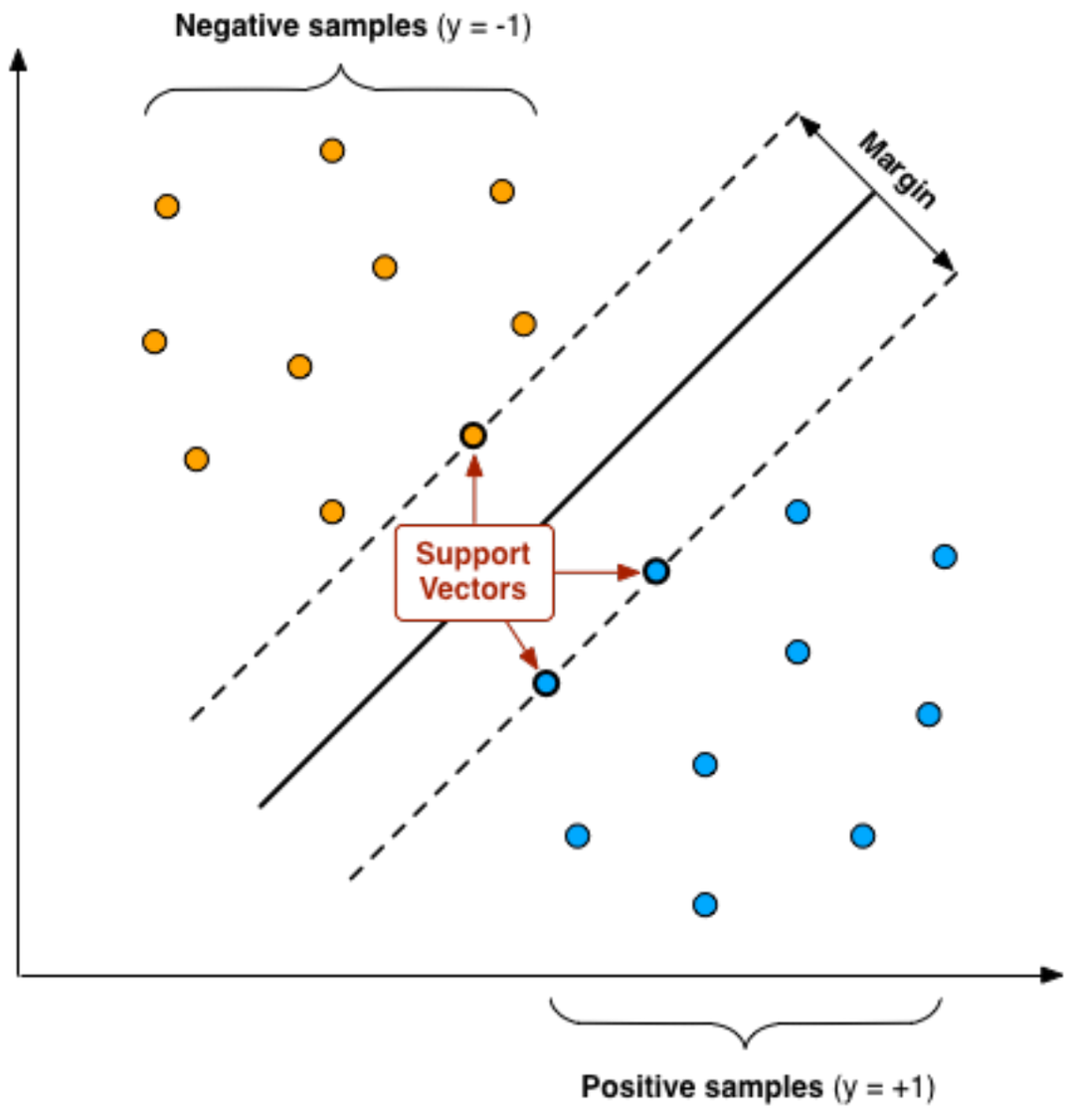
$$\|\boldsymbol{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}.$$



In other words, a term on the loss function will include $||\mathbf{w}||$

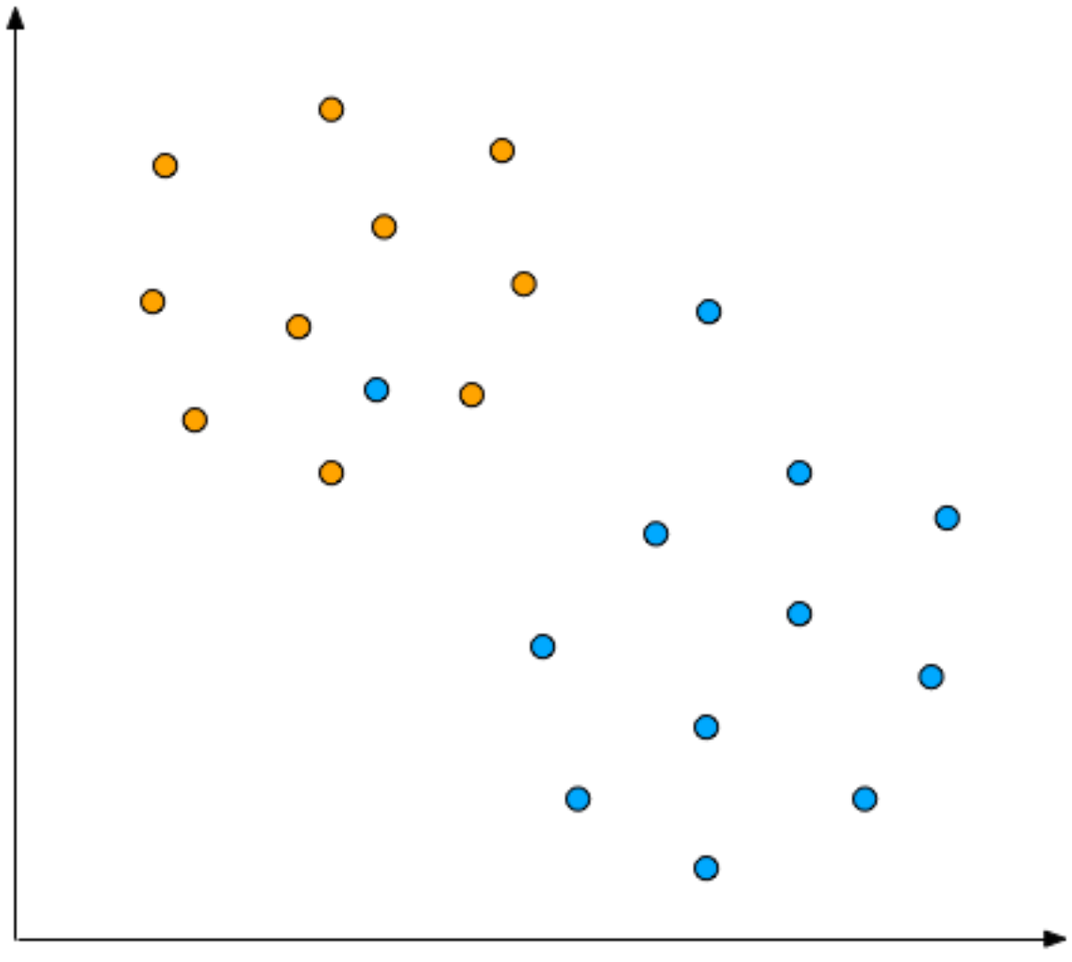
Linear SVM

- Training samples on the margin boundaries are called the **support vectors**



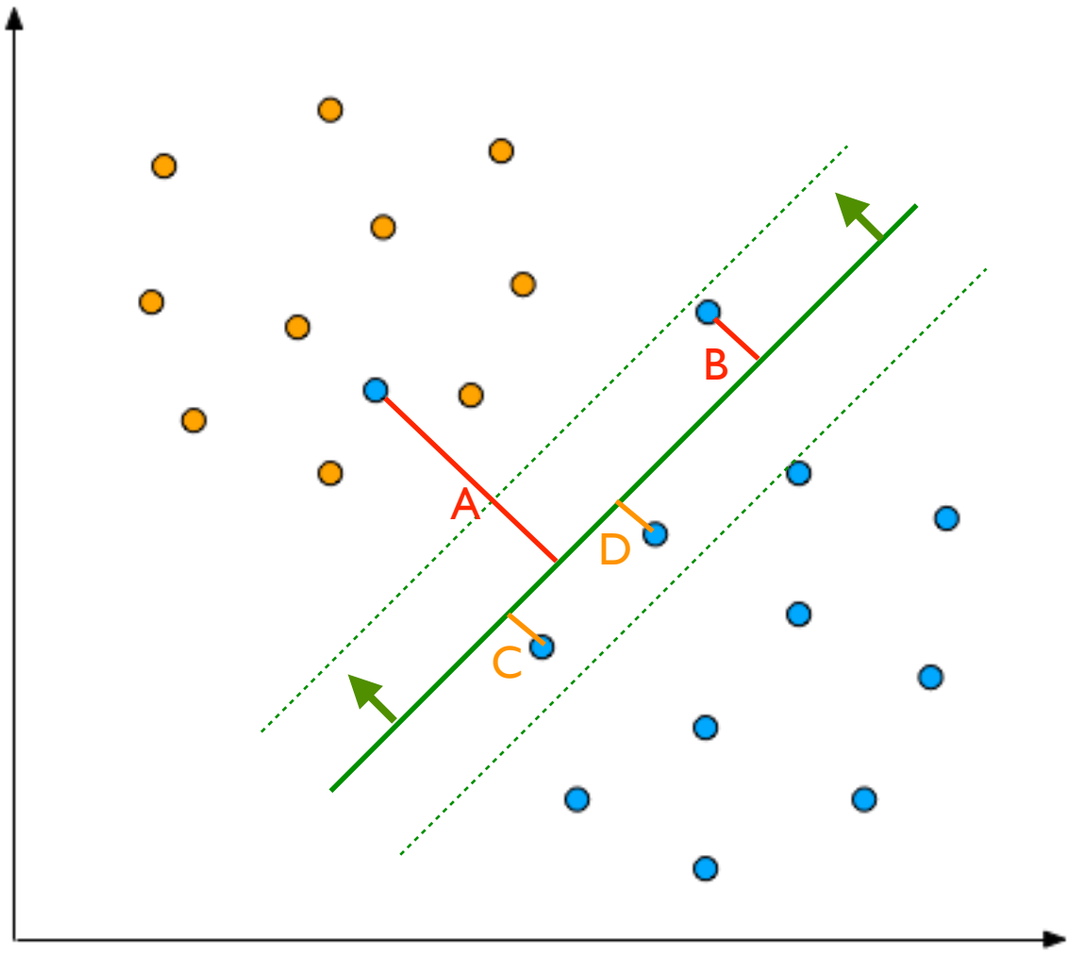
Linear SVM for not linearly separable data

- **Problem:** how to linearly separate these 2 classes?
- We need to inject another term in the cost function that will minimise the number of incorrectly classified samples.
- For SVM, the “Hinge” loss is used
  - It takes into account the notion of margin



Intuition behind the “hinge” loss

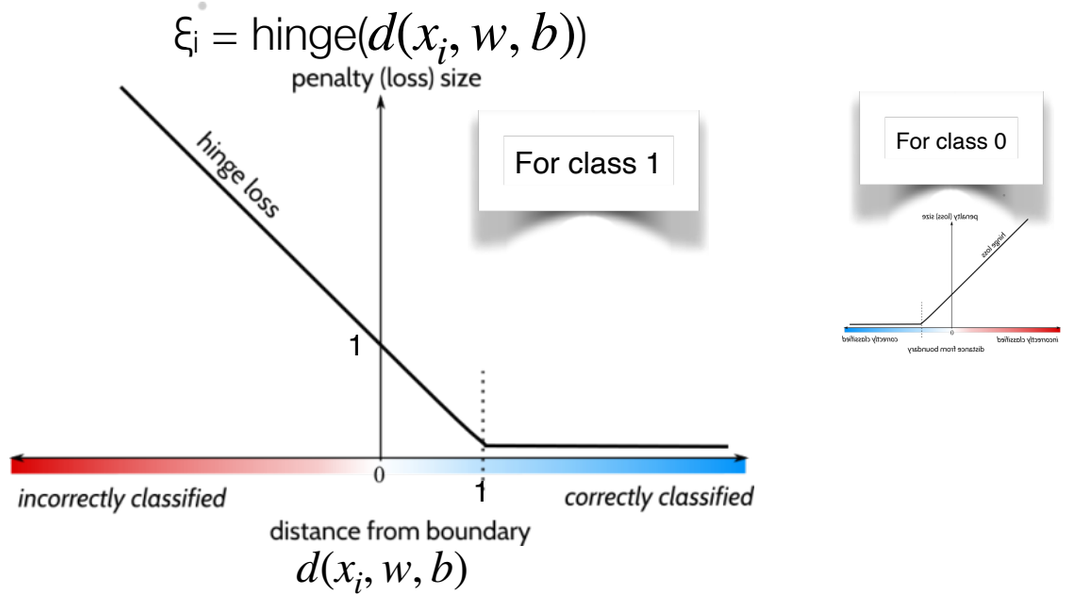
- At a given training epoch, we get a decision border (in green) with the margin (dashed green)
- Each training point contribute to the loss
  - Points falling on the good side and out of the margin gives no penalty, loss=0.0
  - Points falling on the wrong side gives a penalty increasing with the distance to the border and the minimum is 1.0, loss = A + B = 4.5 + 1.5
  - Points falling on the right side but in the margin gives a small penalty between 0.0 (=on the margin) and 1.0 (=on the border), loss = C + D = 0.5 + 0.5

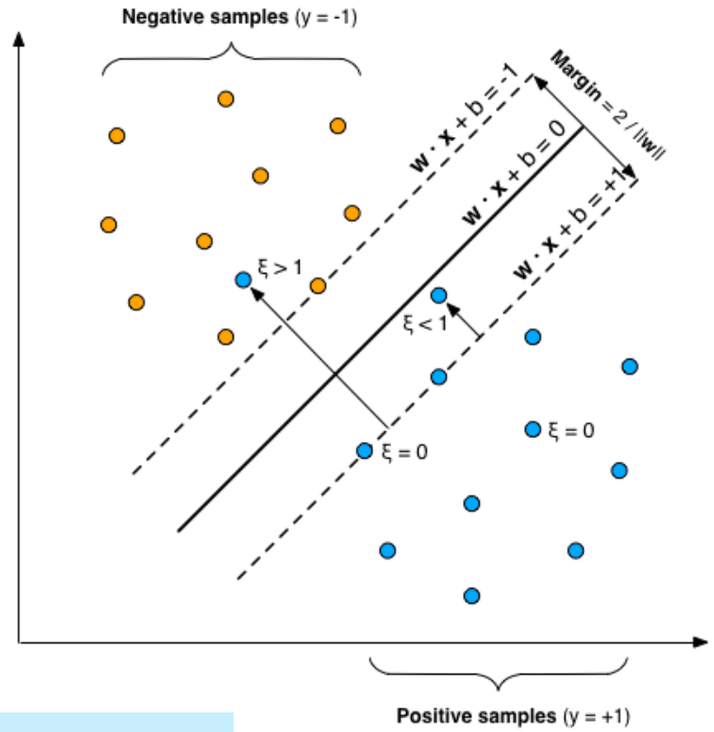


- In this case, the hinge loss J will be = 7.0 and at the next iteration, the update of the border will **move** it to reduce the loss

Linear SVM for not linearly separable data

- For each sample $\mathbf{x}_i$, compute the Hinge loss ξ_i
  - = 0 if the point falls above the margin
  - = 1 if the point falls on the hyperplane
  - > 1 if the point is on the wrong side of the hyperplane
- The $\xi \geq 0$ measure the degree of misclassification of samples in terms of distance to the plane
- Overall loss is quantified by: $\sum \xi_i$ for i=1,…,N
- In the SVM terminology, the ξ_i are also called the **slack** variables.

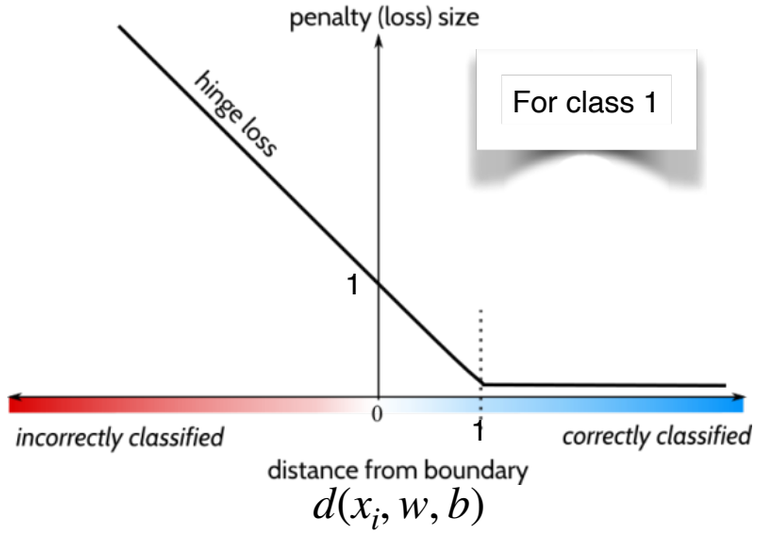


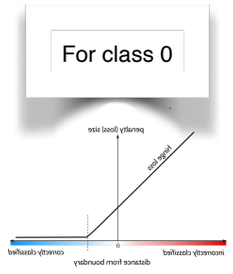


In other words, a term on the loss function will include $\sum \xi_i$

SVM loss function

- We can then define the SVM loss function taking into account of two terms to minimise:
  - One based on the Hinge loss for both classes C_1 and C_0
  - One based on ||**w**||
    - For mathematical reasons, we instead minimise ||**w**||^2/2





Depending to the sample class label, one of the terms disappears, i.e. when $y_i = 1$ (class 1) or when $y_i = 0$ (class 0)

$$J(\theta) = C\left[\frac{1}{N}\sum_{i=1}^{N} y_i \,\text{hinge}_{C1}(d(x_i, w, b)) + (1 - y_i)\,\text{hinge}_{C0}(d(x_i, w, b))\right] + \left[\frac{1}{2}\|w\|^2\right]$$

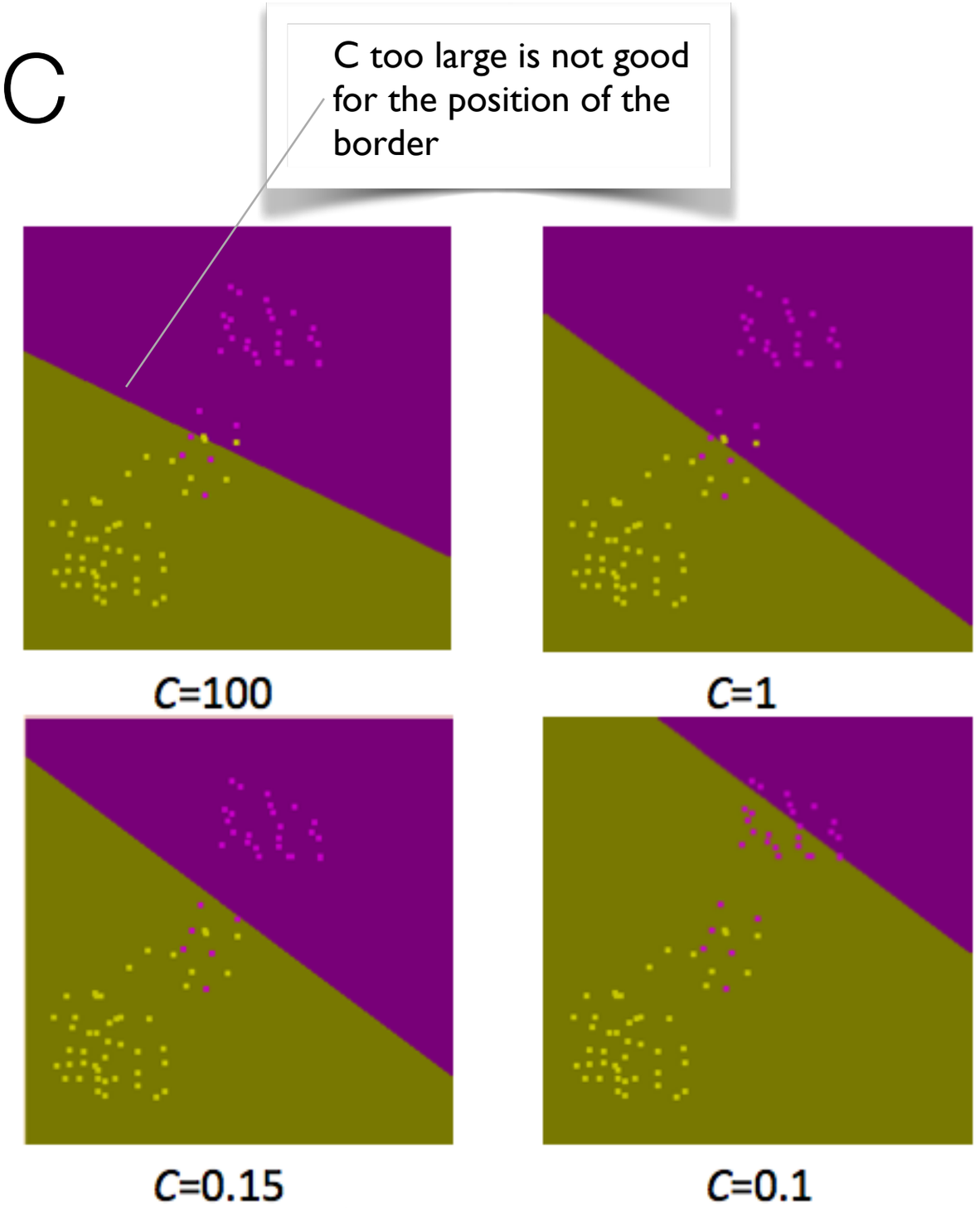
Trade-off coef : giving more or less importance to perf term

Performance term: on the data, how far are we predicting from the ground truth?

Regularisation term:
(1) impeach too large values of **w**,
(2) minimising **w** will maximize margin

Impact of parameter C

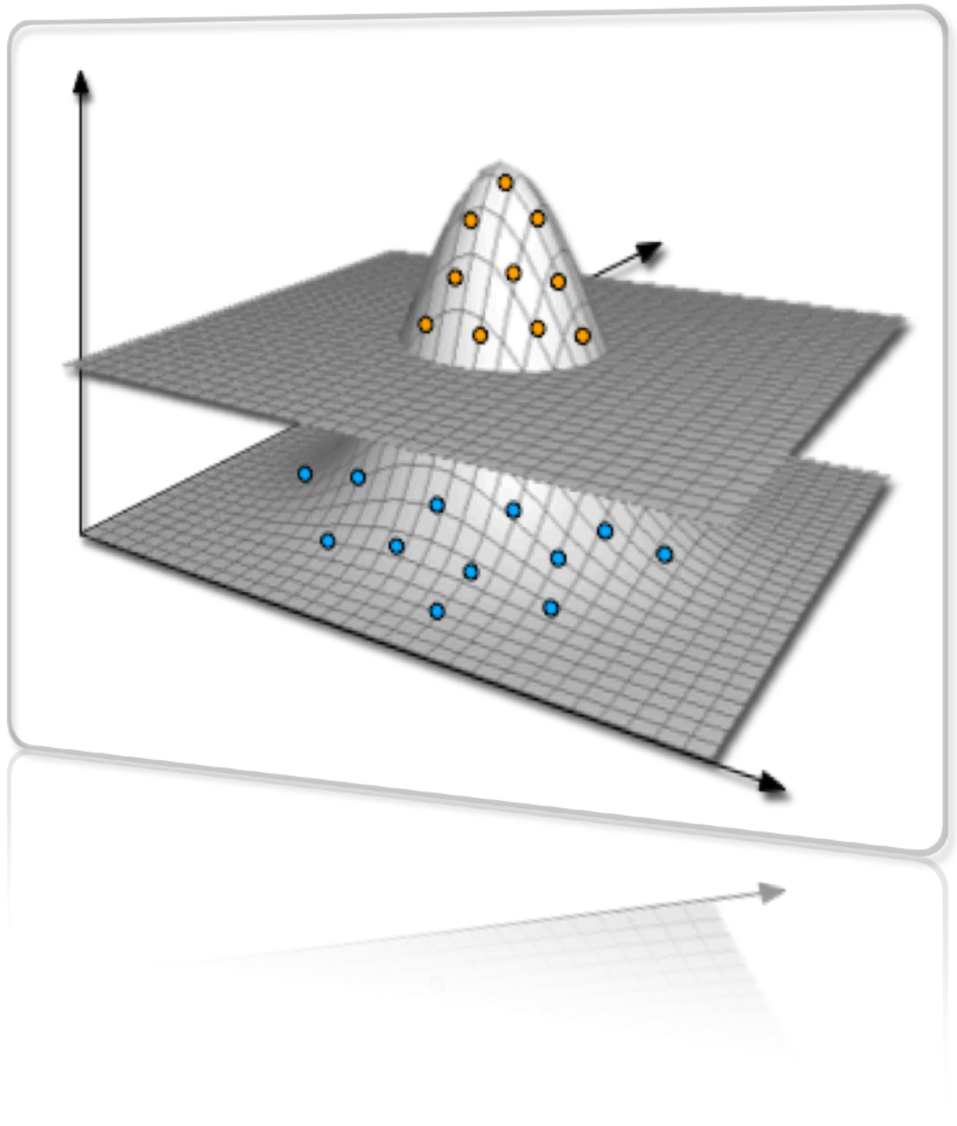
- The factor C is a regularization para-meter which trades off the margin size and the training error
- The smaller C, the greater the number of admitted misclassified train samples



Minimizing the loss function

- 2 possibilities
- Use math toolboxes for minimisation problems under constraints
  - For example: SciKit Learn **`svm.SVC()`** based on the popular library libsvm
    - http://scikit-learn.org/stable/modules/svm.html
    - http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
    - https://www.csie.ntu.edu.tw/~cjlin/libsvm/
  - Practical considerations: usually less tuning to perform, libsvm is a well optimised and stabilised library
- Use gradient descent approaches: compute the gradient of the loss w.r.t. the parameters **w** and apply gradient descent as usual
  - For example: SciKit Learn **`linear_model.SGDClassifier()`** with parameters: `loss='hinge'` and `penalty='l2'`. Note: instead of parameter C giving weight to the classification performance, they use parameter alpha giving weight to the regularisation term.
    - http://scikit-learn.org/stable/modules/sgd.html#sgd
    - http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
  - Practical considerations - pros: could be advantageous for very large training sets and cases where incremental learning is needed, cons: we need to tune the learning rate

7.3 SVM for non-linear problems

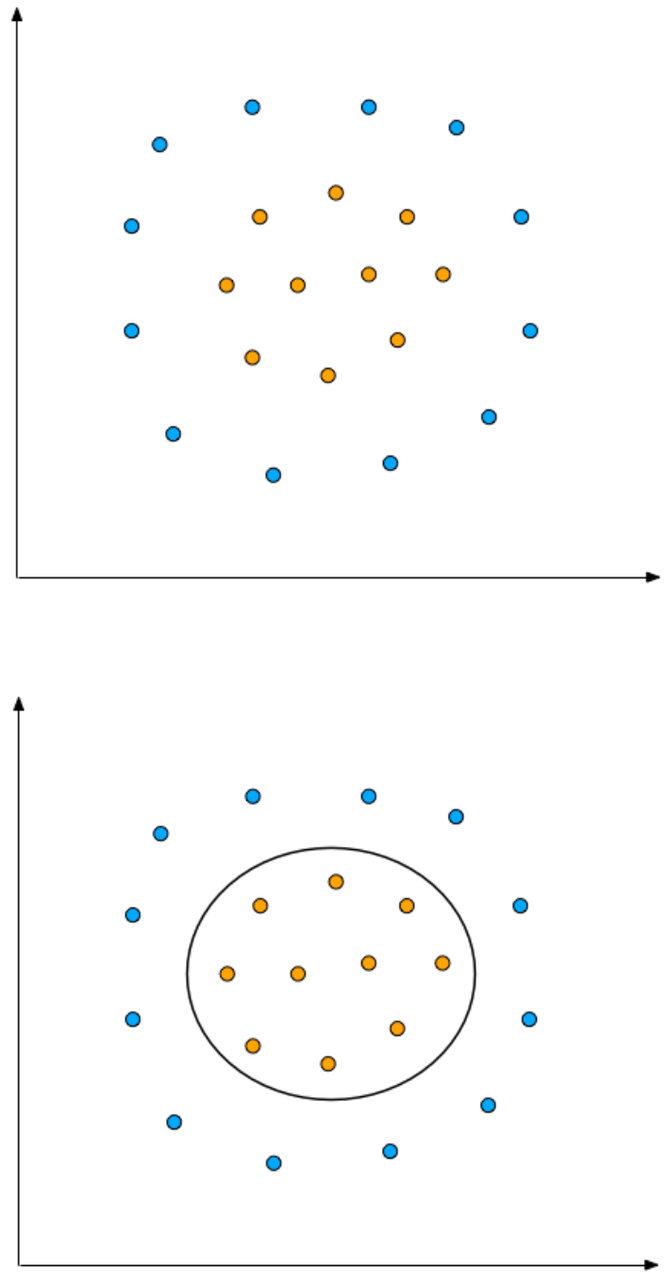


Kernel trick

Common kernel

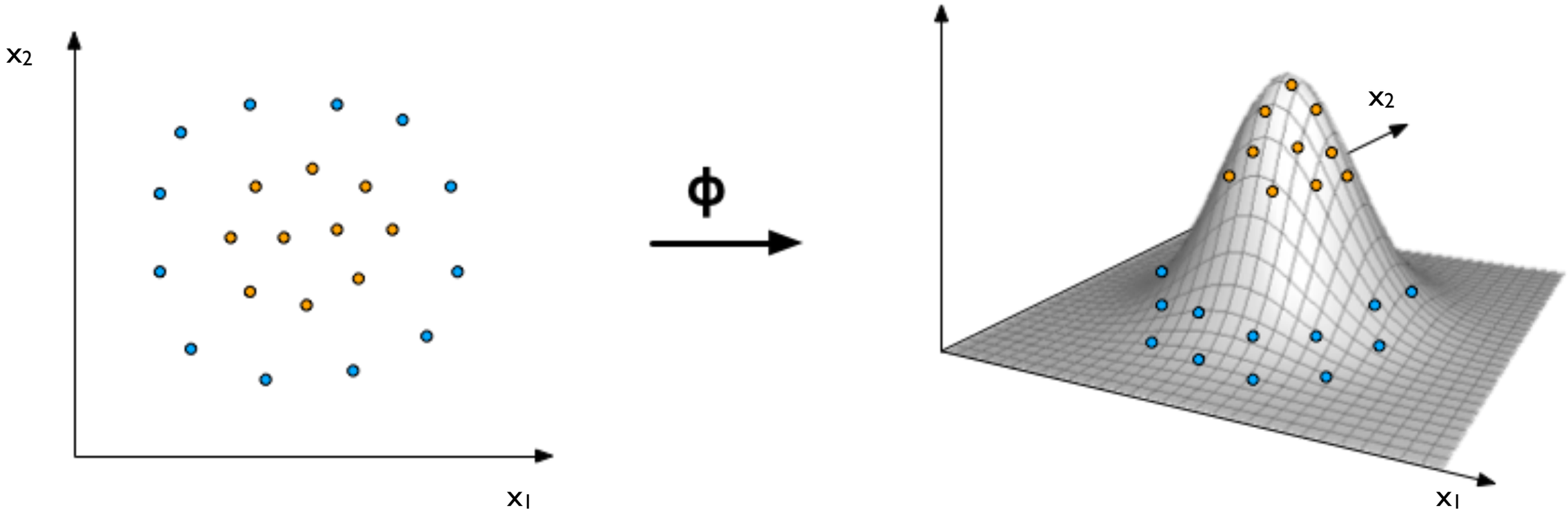
Non-linear problems

- **Problem**: how to linearly separate these 2 classes of samples?
- 2 solutions
  - move to non-linear decision boundaries by adding non-linear features to the $\mathbf{x}$ array and then use a linear SVM as usual - in a similar way as what we did with logistic regression (see slide 9)
  - use non-linear SVMs with **kernels**
    - **see next slides**



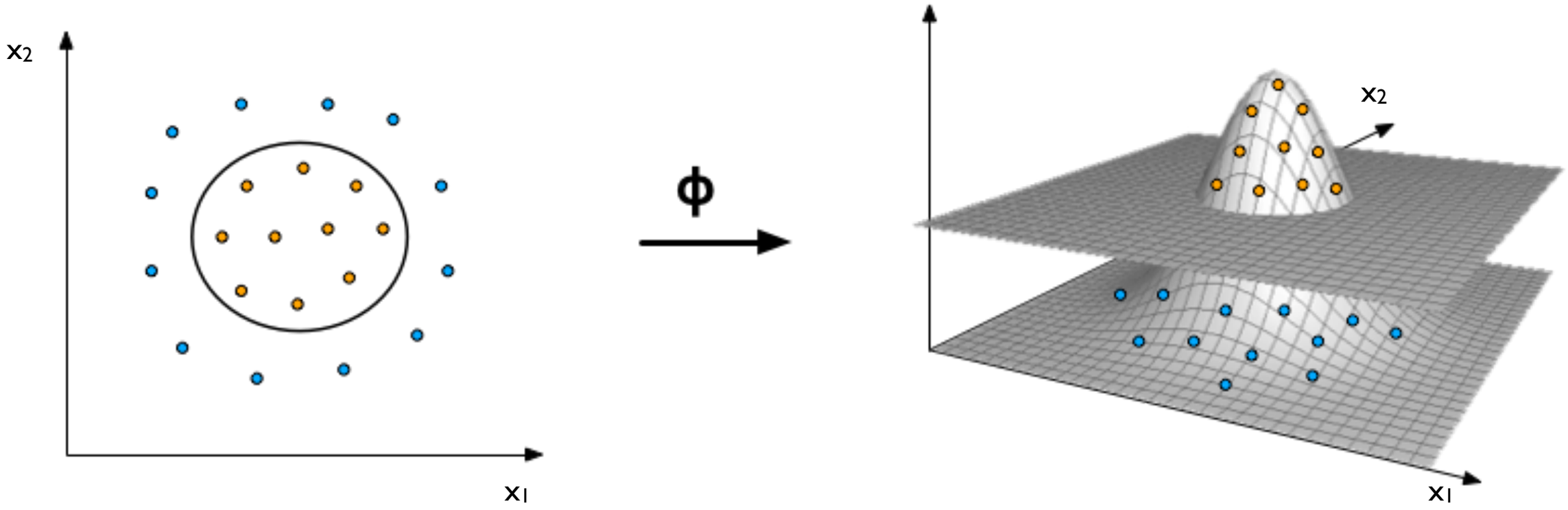
Nonlinear SVM for not linearly separable data

- **Map** the sample **input space to** a higher dimensional space (called **feature space**) with a function **ϕ**…



Nonlinear SVM for not linearly separable data

- …where samples are linearly separable by SVM !



Nonlinear SVM for not linearly separable data

- In summary, the SVM problem remains the same as before except that **x** is replaced by $\phi(\mathbf{x})$ in all equations

- The vector **w** defining the **optimal hyperplane** is found through the learning process using
$\mathbf{w} \cdot \phi(\mathbf{x}) + b = 0$

- Then a new test sample $\mathbf{x_t}$ is classified by computing
sign($\mathbf{w} \cdot \phi(\mathbf{x_t}) + b$)

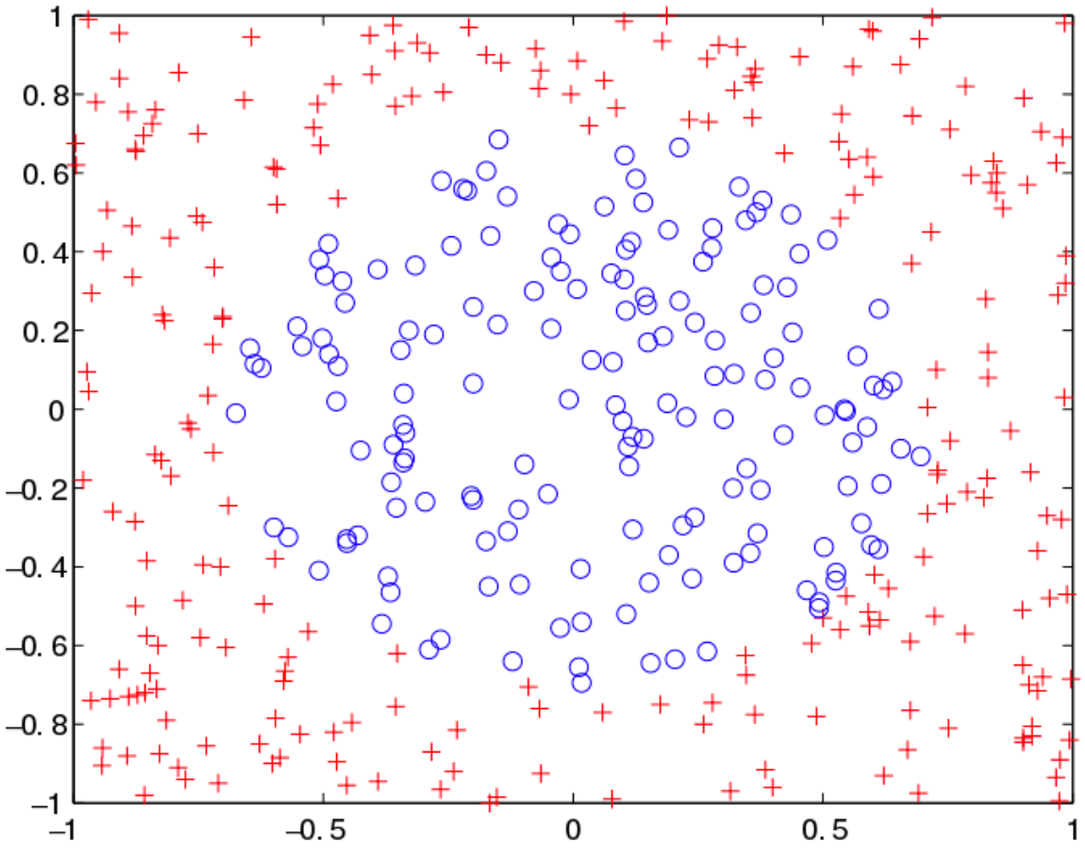
In red what changes from previous methods

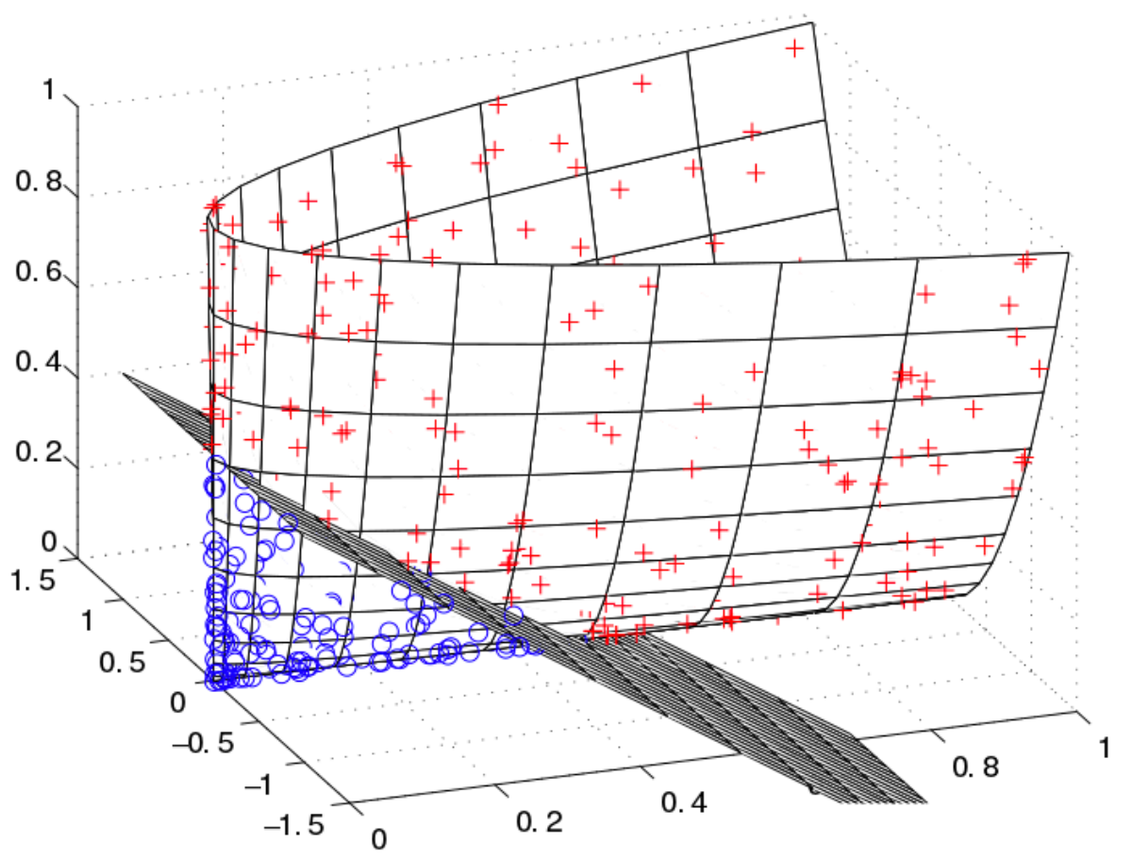
Nonlinear SVM for not linearly separable data

- The functions ϕ are computed through *kernel functions* located at each training points $\mathbf{x_i}$
  - A new test sample $\mathbf{x_t}$ is classified by computing the sign of $\mathbf{w} \cdot \phi(\mathbf{x_t}) + b = \sum \alpha_i y_i K(\mathbf{x_i}, \mathbf{x_t}) + b$

- **Common kernels** $K(\mathbf{x_i}, \mathbf{x_j})$: $(i, j = 1,\ldots,N)$
  - Linear: $K(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i} \cdot \mathbf{x_j}$
  - Polynomial: $K(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$ (where d is the polynom degree)
  - Gaussian or radial basis function (RBF):
    $K(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$ (where $\gamma = 1/2\sigma^2 > 0$)
  - Hyperbolic tangent:
    $K(\mathbf{x_i}, \mathbf{x_j}) = \tanh(k\, \mathbf{x_i} \cdot \mathbf{x_j} + c)$ (for some $k > 0$ and $c < 0$)

Example





From https://commons.wikimedia.org/wiki/File:Nonlinear_SVM_example_illustration.svg

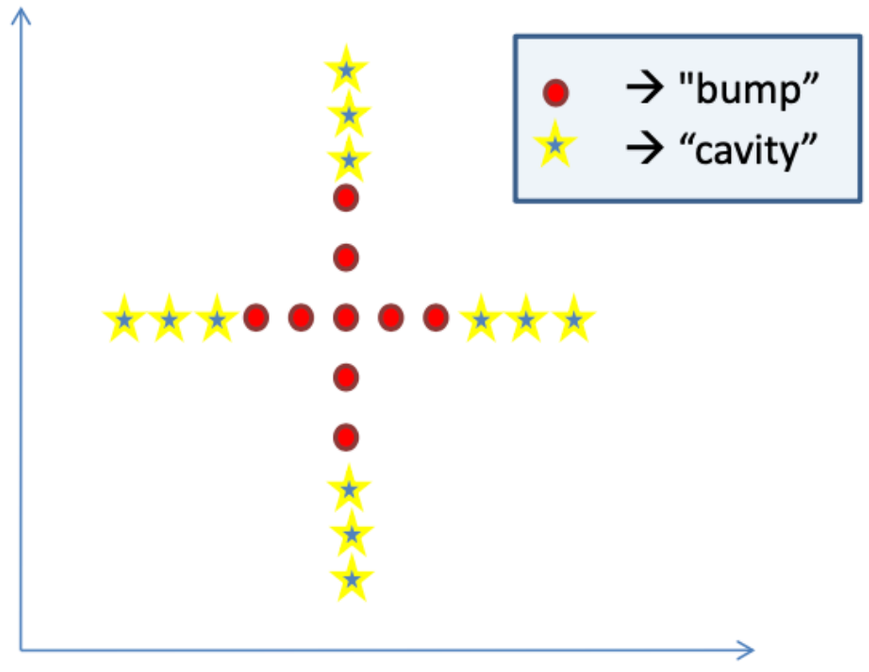
Description	**English:** An illustration of kernel transformation for nonlinear SVM. Original 2D data (left) are linearly nonseparable. Transformed 3D data are separable. Used kernel function: $K(x_i, x_j) = (x_i \cdot x_j)^2$, mapping $\Phi(\vec{x}) = (x_1^2, \sqrt{2}x_1 x_2, x_2^2)$

Intuition behind Gaussian kernel

Understanding the Gaussian kernel

Consider Gaussian kernel: $K(\vec{x}, \vec{x}_j) = \exp(-\gamma \left\| \vec{x} - \vec{x}_j \right\|^2)$

Geometrically, this is a "bump" or "cavity" centered at the training data point $\vec{x}_j$:



The resulting mapping function is a **combination** of bumps and cavities.

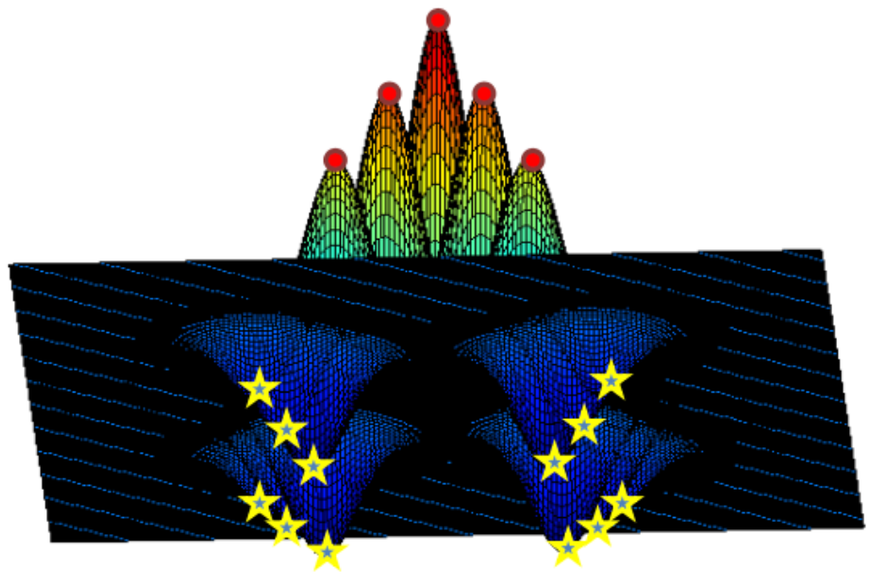
61

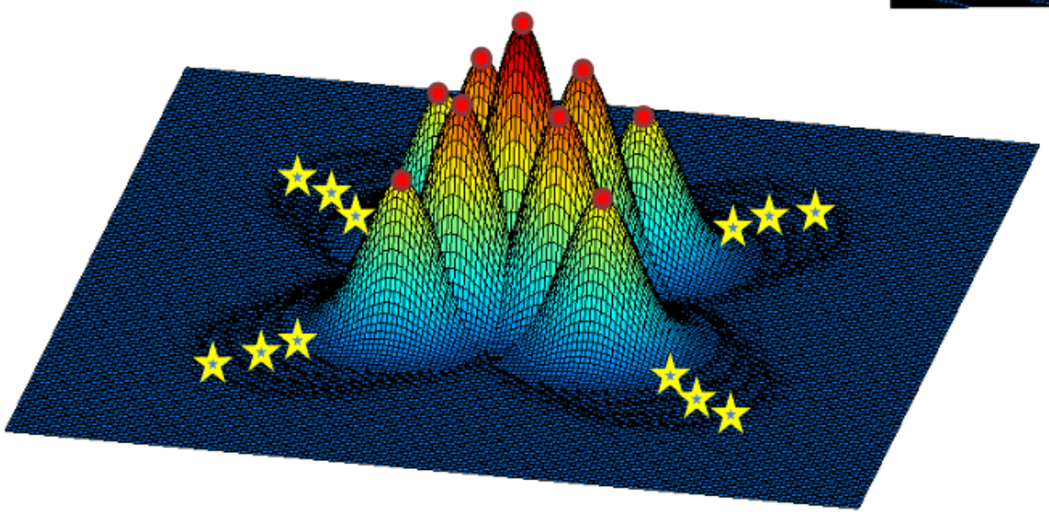
From *A Gentle Introduction to Support Vector Machines in Biomedicine*, https://www.eecis.udel.edu/~shatkay/Course/papers/UOSVMAlliferisWithoutTears.pdf

Intuition behind Gaussian kernel

Understanding the Gaussian kernel

Several more views of the data is mapped to the feature space by Gaussian kernel

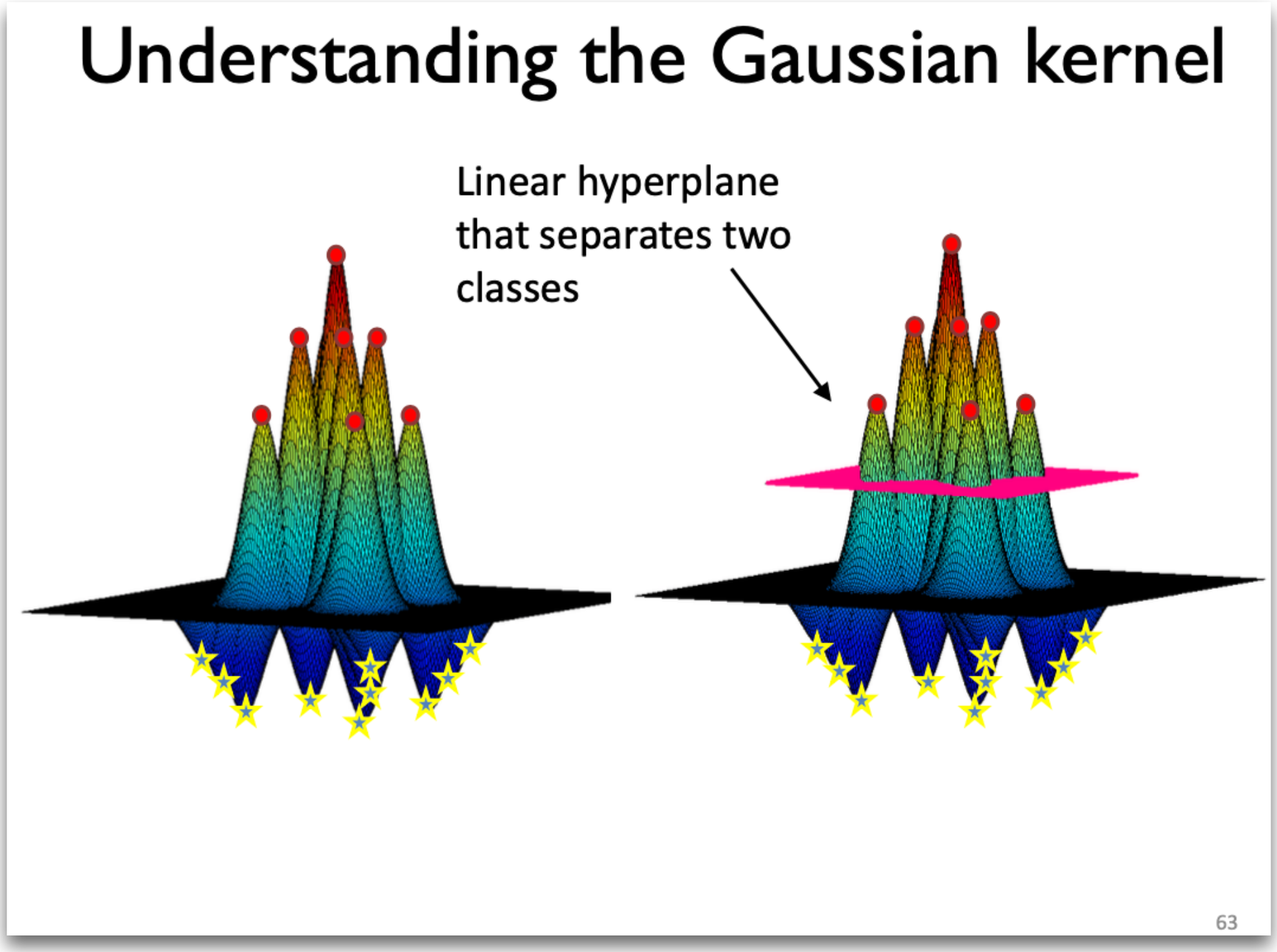




62

From *A Gentle Introduction to Support Vector Machines in Biomedicine*, https://www.eecis.udel.edu/~shatkay/Course/papers/UOSVMAlliferisWithoutTears.pdf

Intuition behind Gaussian kernel

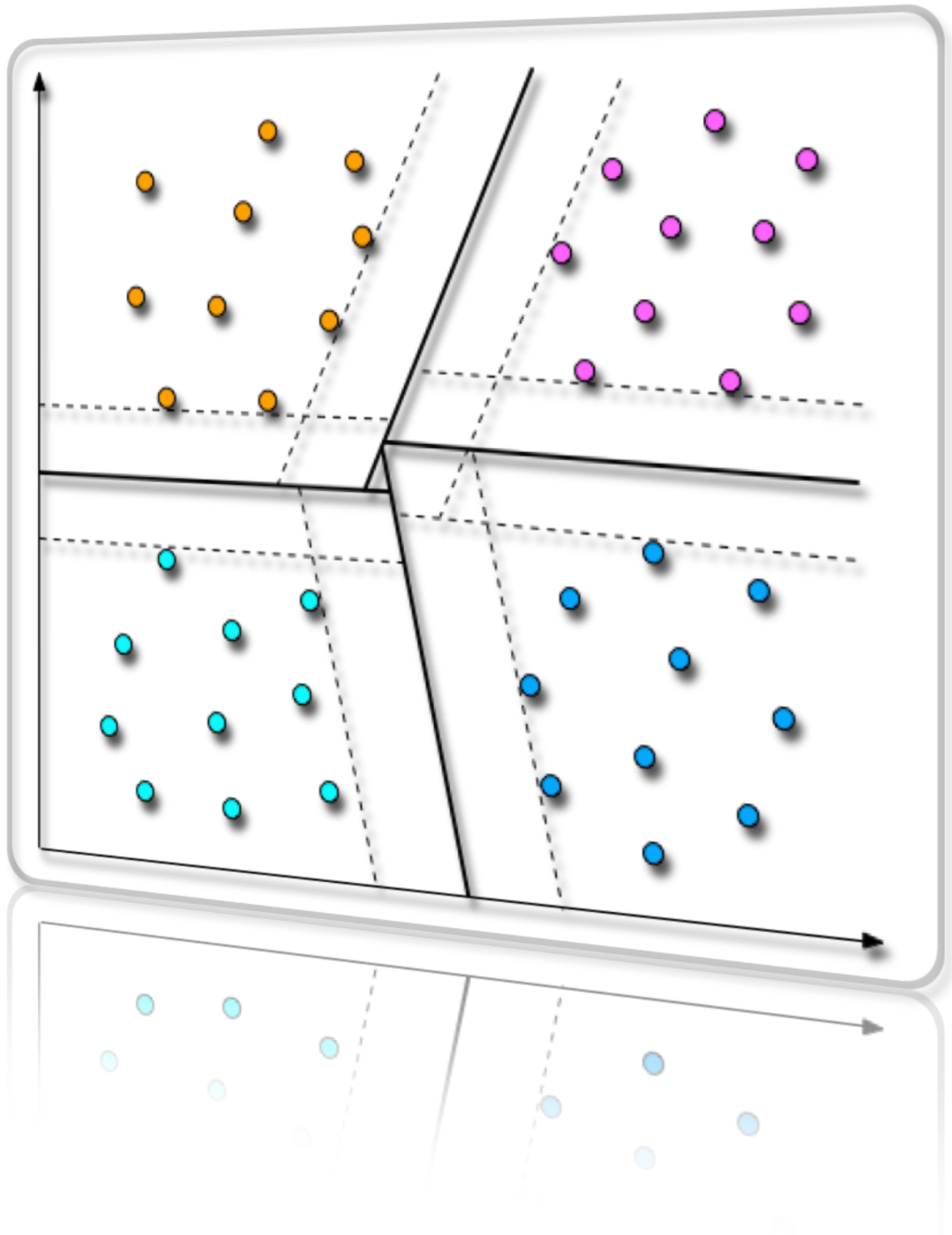


From *A Gentle Introduction to Support Vector Machines in Biomedicine*,
https://www.eecis.udel.edu/~shatkay/Course/papers/UOSVMAlliferisWithoutTears.pdf

7.4 Multiclass SVM

One vs all

One vs one

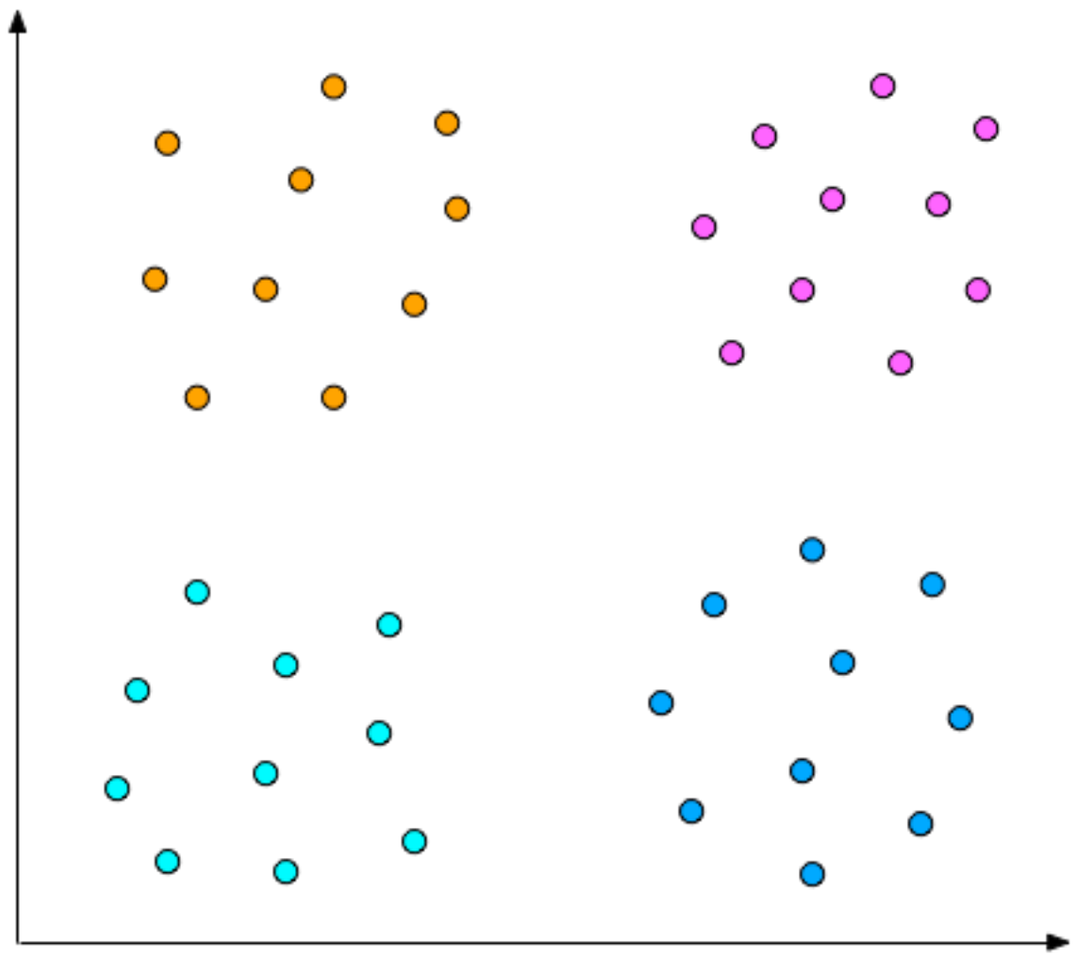


By Balu Ertl https://upload.wikimedia.org/wikipedia/commons/5/54/Euclidean_Voronoi_diagram.svg

Multiclass SVM

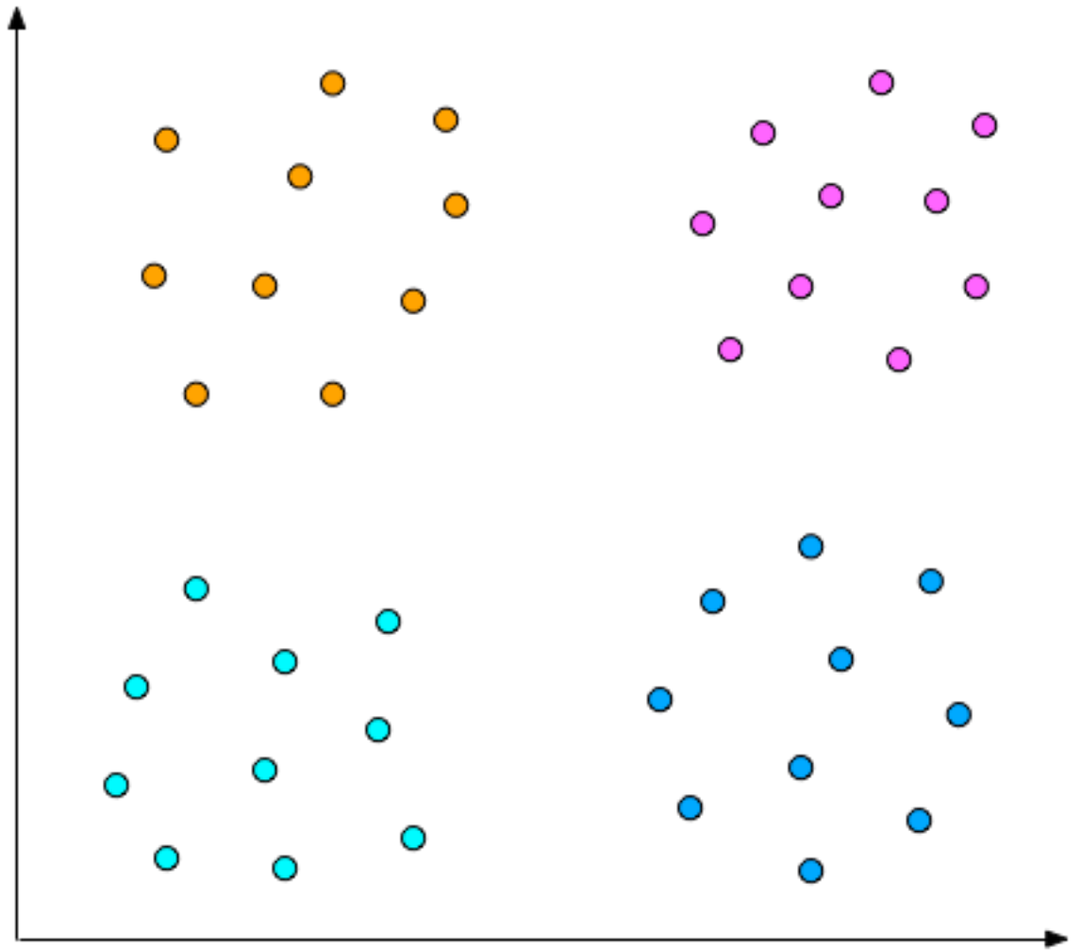
- Initially, **SVM** are a **binary classifier**, i.e. can separate 2 classes of samples

- **Problem:** how to separate multiple classes?



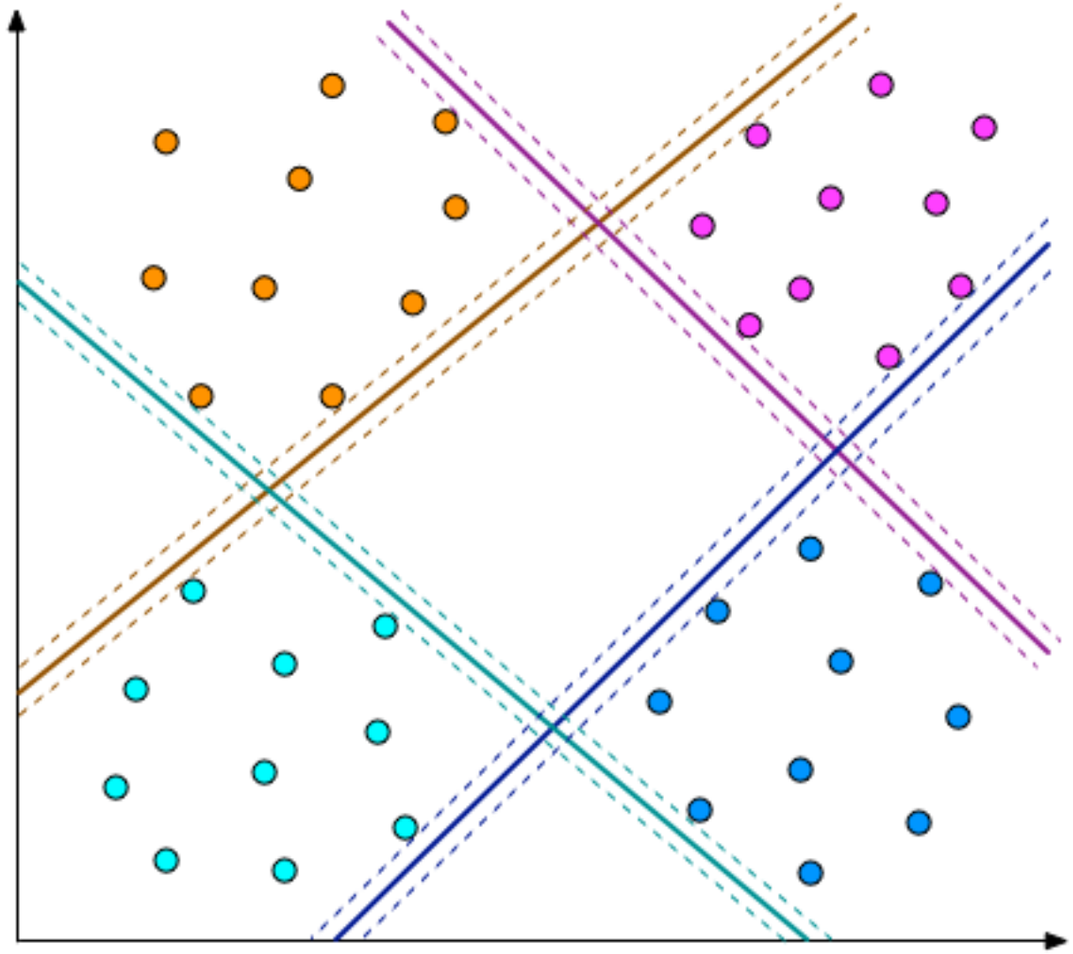
Multiclass SVM

- **Solution:** reduce the single multiclass problem into **multiple binary classification problems**



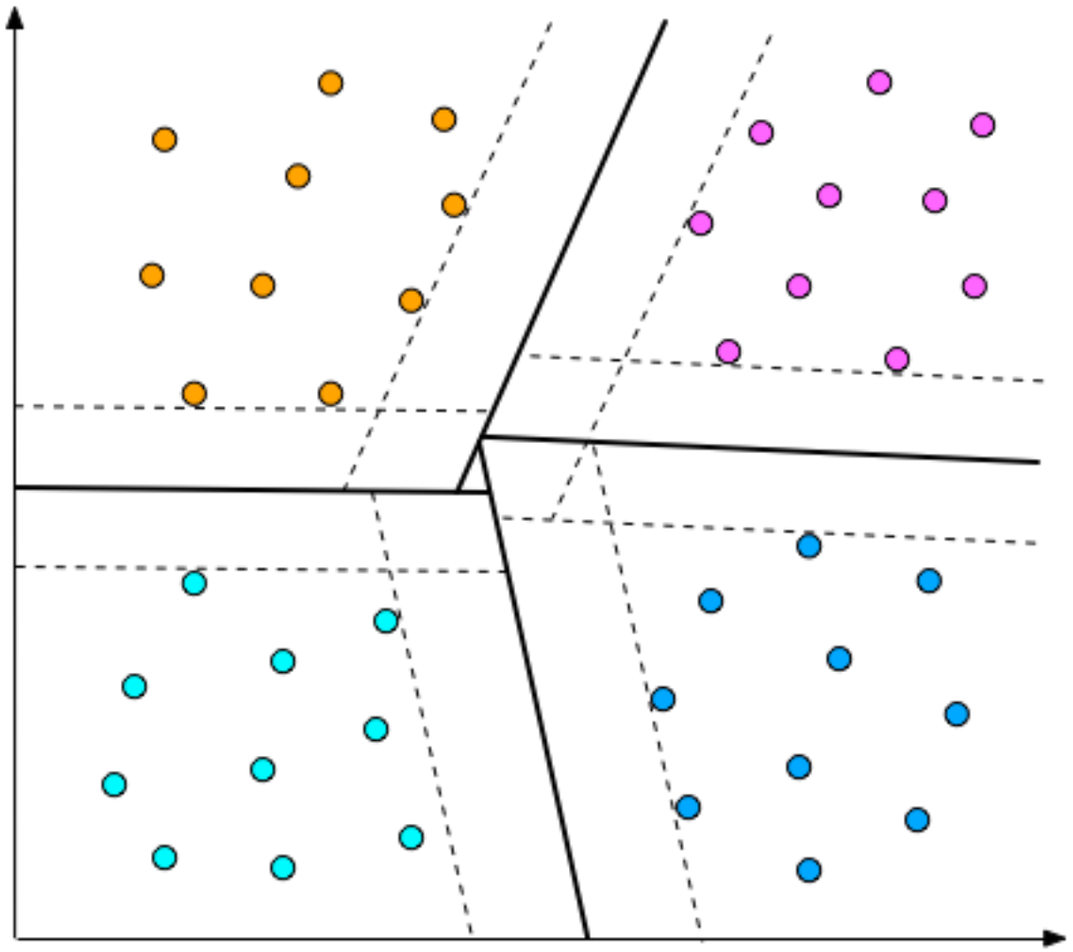
Multiclass SVM

- **One-vs-all method:** the classification of new samples is done by a winner-takes-all strategy, in which **the SVM with the highest output value assigns the class to a given sample**

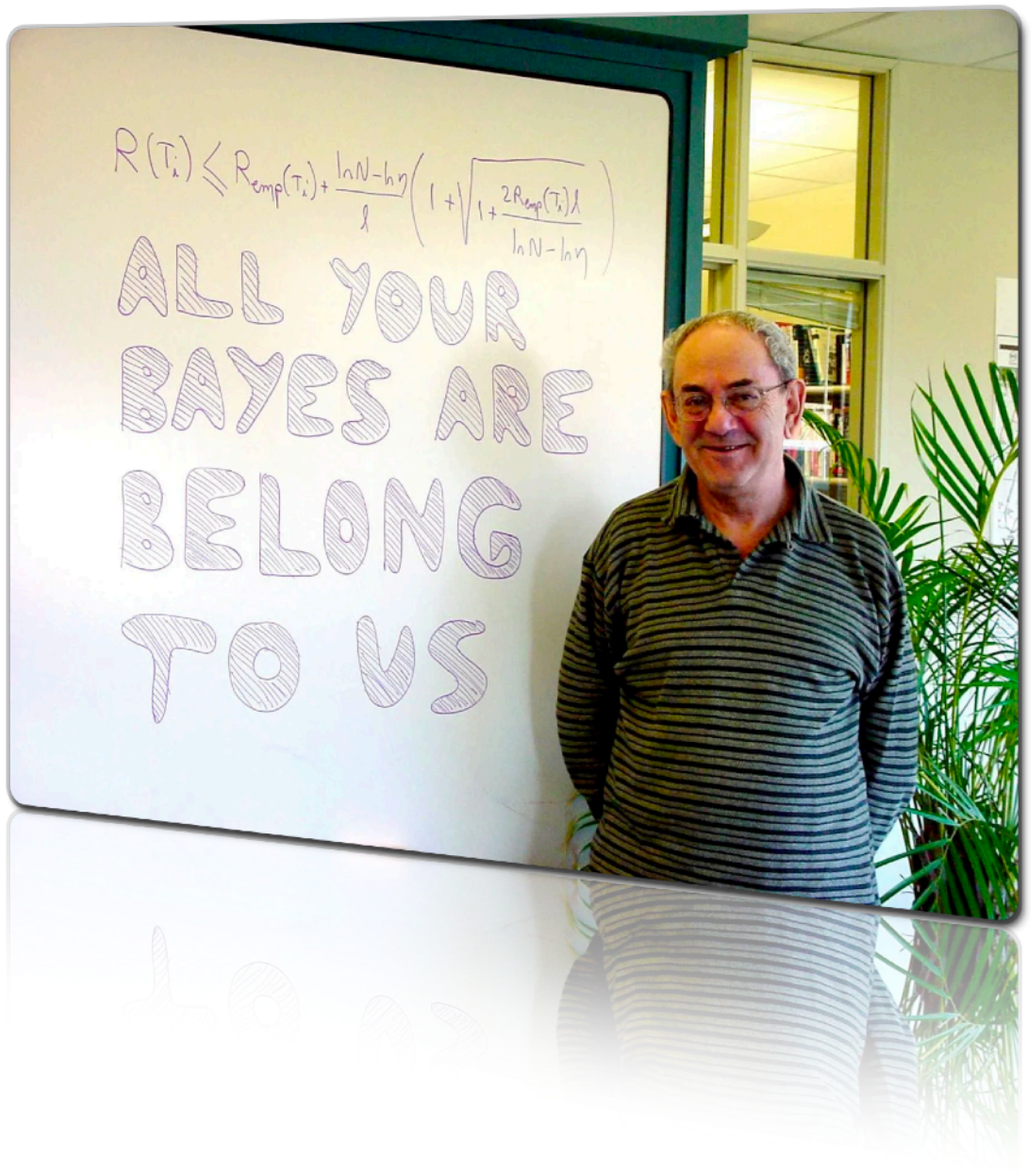


Multiclass SVM

- **One-vs-one method:** the classification of new samples is done by a max-wins voting strategy, in which **every SVM classifier assigns a given sample to one of the two classes**, then the number of votes for the assigned class is increased by one, and finally **the class with the highest number of votes is assigned to the sample**



7.5 History & References



From https://www.reddit.com/r/math/comments/7up5o/vladimir_vapnik_the_svm_guy_best_pic/

History of SVM

- **1960**: Beginning of SVM development
- **1963**: Original **linear SVM** algorithm proposed by Vladimir N. Vapnik
- **1964**: **Kernel trick** first published by M. Aizerman, E. Braverman, and L. Rozonoer
- **1992**: **Nonlinear SVM** (using **kernel trick**) proposed by Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik
- **1995**: Current **standard SVM** (using **soft margin**) proposed by Vladimir N. Vapnik and Corinna Cortes
- **1996**: SVM for regression (**SVR**) proposed by Vladimir N. Vapnik
- **2008**: Vladimir N. Vapnik and Corinna Cortes received the ACM Paris Kanellakis Award for their scientifical contribution
- **Today**: SVM are widely used because of their efficiency

References

- **SVM on Wikipedia:**
  http://en.wikipedia.org/wiki/Support_vector_machine
- **SVM tutorials:**
  http://www.svms.org/tutorials
  http://www.kernel-machines.org/tutorials
- **Nice SVM presentation (with biomedical application):**
  A. Statnikov, D. Hardin, I. Guyon, C. F. Aliferis,
  A Gentle Introduction to Support Vector Machines in Biomedicine,
  https://www.eecis.udel.edu/~shatkay/Course/papers/UOSVMAlliferisWithoutTears.pdf (also on moodle)
- **Books:**
  C. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006
  R. O. Duda, P. E. Hart, D. G. Stork, *Pattern Classification (2nd Ed.)*, Wiley-Interscience, 2001

Conclusions - what you should know

- SVM are discriminative models defining a border between 2 classes - similar to logistic regression
- SVM solves two problems of logistic regression
  - Find a more optimal border by maximising the margin between classes
  - Ease the search for feature combinations by proposing “swiss-army knife” projection kernels
- SVM functioning principles:
  - **Linear SVM** tries to deal with **linearly separable data**, finding a hyperplan with maximal margin between classes.
  - **Linear SVM** tries to deal with **not linearly separable data**, finding a hyperplan minimising the **hinge loss**.
  - **Non-linear SVM** first maps the samples to higher dimensional space with kernel functions **ϕ**
    - A set of common kernel types is used and work well in many situations
    - The type of kernel is a hyper parameter. The best type is usually found by testing all types.
- SVM models can be **trained** with
  - A gradient descent on a loss function including 2 terms: hinge loss and regularisation term
  - Numerical procedures (minimisation problems under constraints - not seen in the context of this class)
- **Multiclass** problems can be dealt with one-vs-all and one-vs-one methods

In practice

- SVM are often better performing than logistic regression
- SVM work well on datasets with D numerical features, D from small to medium sizes
- SVM tuning:
  - Actually not so intensive - we find quickly a viable model
  - Hyperparameters:
    - C in the loss function, defining the weight on the performance vs regularisation term
    - Type of kernels to use
- Popular toolkits:
  - Based on numerical procedures: **libsvm** - encapsulated in C/C++, Java, Python
    - For example in SciKit Learn **`svm.SVC()`**
      - http://scikit-learn.org/stable/modules/svm.html
  - Based on gradient descent
    - For example in SciKit Learn **`linear_model.SGDClassifier()`** with parameters: `loss='hinge'` and `penalty='l2'`.