

# SVM: Support Vector Machine

Diane Lingrand and many contributors



UNIVERSITÉ  
**CÔTE D'AZUR**

Master Data Science M1

2020 - 2021

1 SVM classification

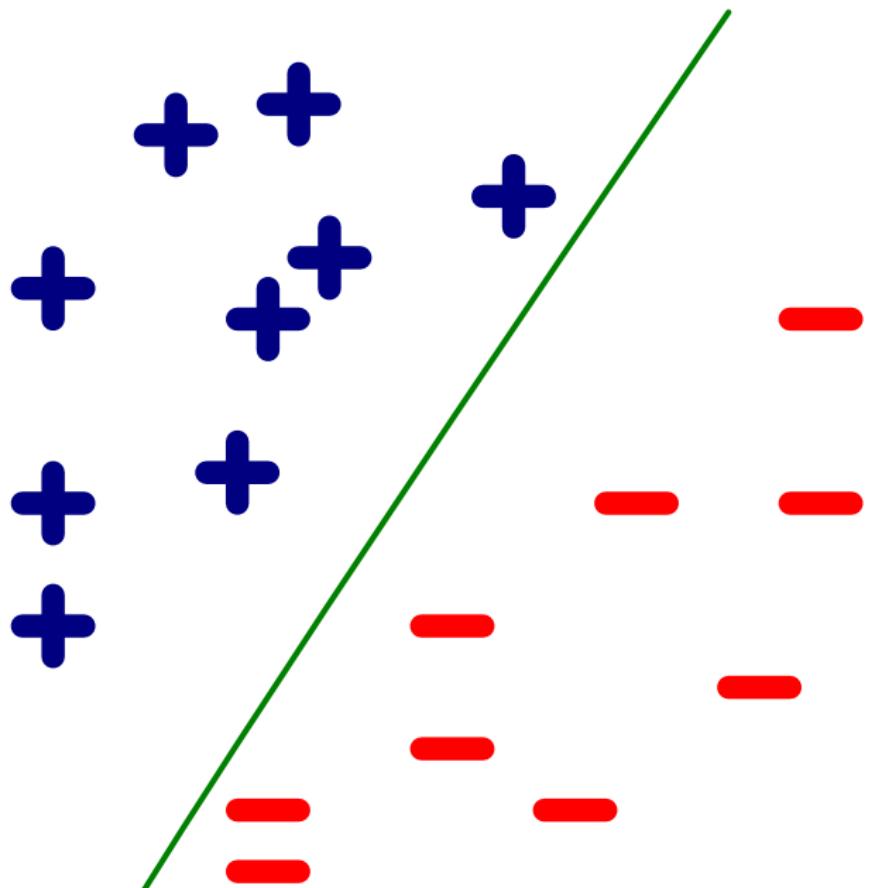
2 SVM regression

3 SVM one class

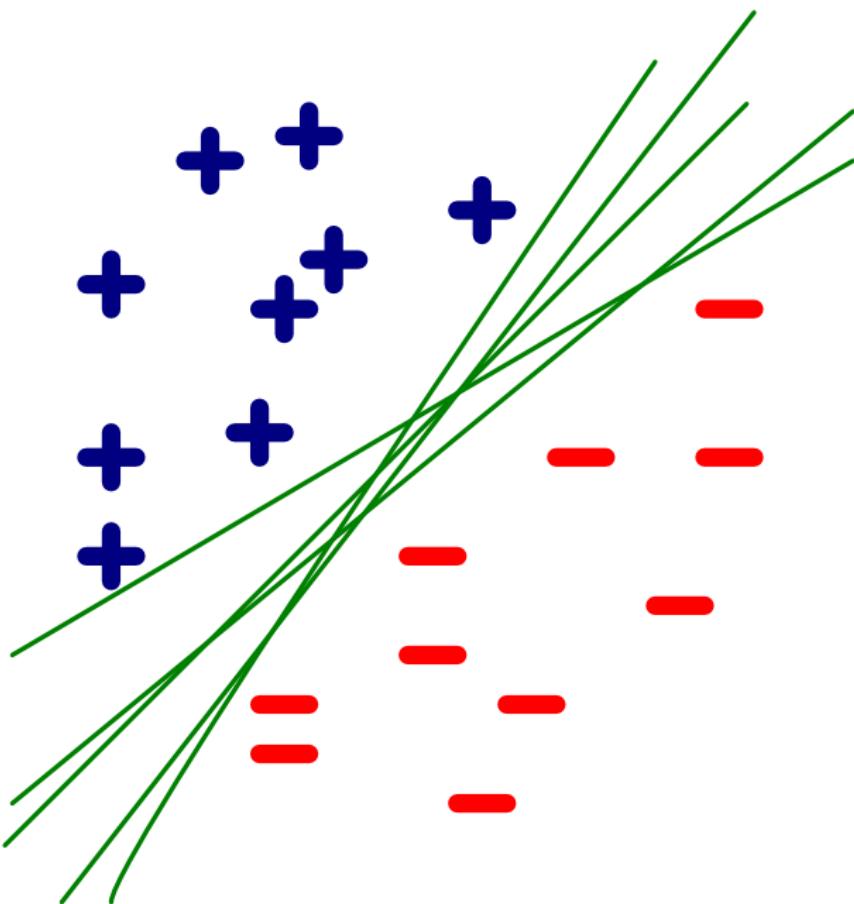
# Introduction

- Context
  - supervised learning
  - classification (or regression)
    - binary
    - extension : multiclass
- Why SVM ? Is that deep ?
- SVMs are important because of
  - theoretical reasons :
    - Robust to very large number of variables and small set of samples
    - Can learn both simple and highly complex classification models
    - Employ sophisticated mathematical principles to avoid overfitting
  - superior empirical results

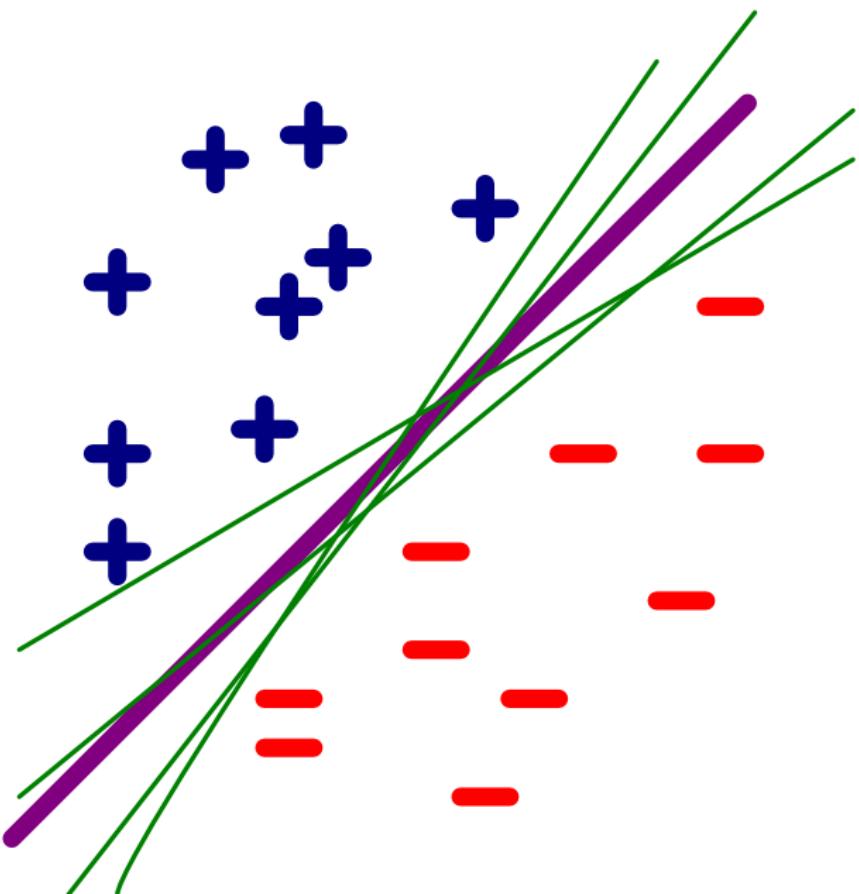
## Principle : linear separation



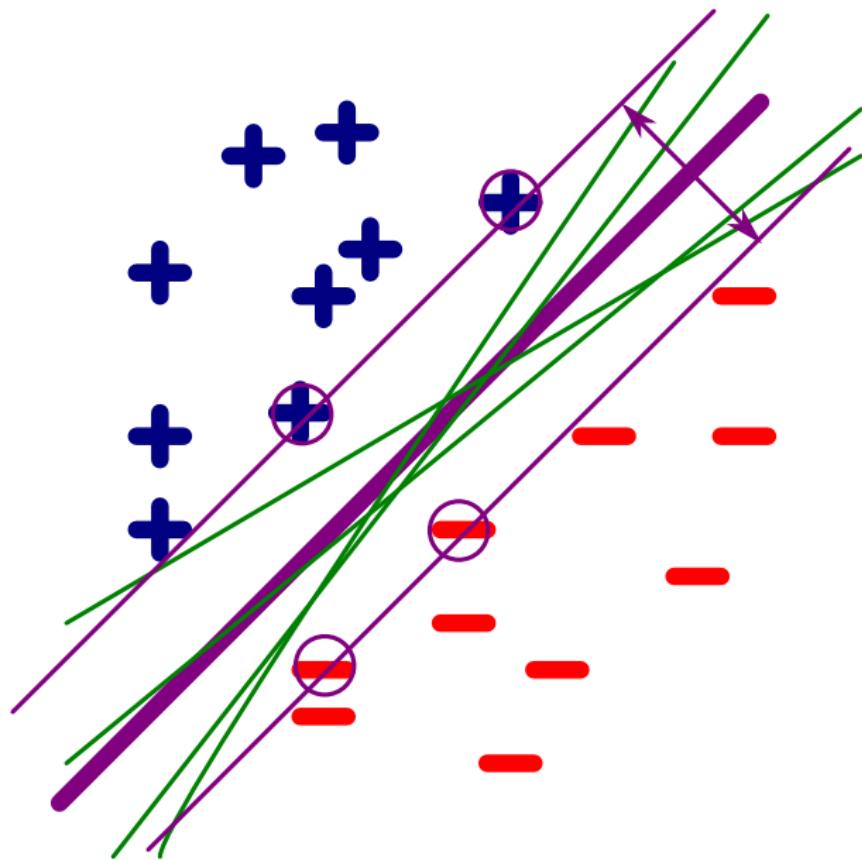
# Principle : many solution



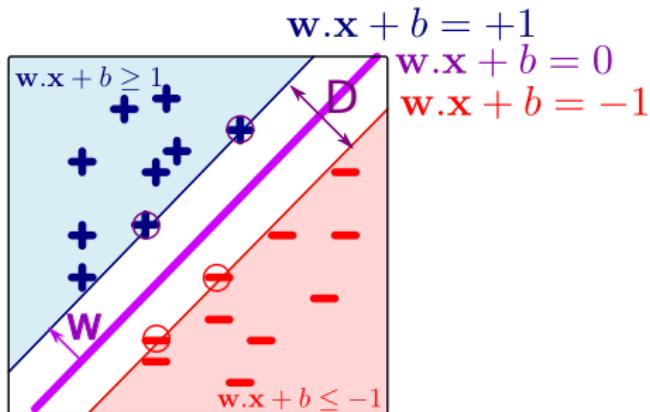
## Principle : find the best one



# Principle : maximising the margin



# Problem formalisation



- Margin maximisation :  $D = \frac{2}{\|w\|}$ 
  - minimization of  $\|w\|$  or  $\frac{1}{2}\|w\|^2$
- Labeled data :
  - positive samples :  $y = +1$ 
    - $w.x + b \geq 1$
  - negative samples :  $y = -1$ 
    - $w.x + b \leq -1$
  - thus :  $y w.x + b \geq 1$

SVM problem :

minimisation of  $\frac{1}{2}\|w\|^2$  under the constraint  $\forall i, y_i (w.x_i + b) \geq 1$

Prediction :  $sign(w.x + b)$

## dual problem

- Lagrangian : maximisation of  $\frac{1}{2} \|\mathbf{w}\|^2 - \sum_i \alpha_i (y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1)$  with  $\forall i \alpha_i \geq 0$
- annulation of derivatives with respect to  $\mathbf{w}$  and  $b$  :
  - $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$
  - $\sum_i \alpha_i y_i = 0$
- the dual problem is :

### dual problem

maximisation of  $\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_k \alpha_i \alpha_k y_i y_k \mathbf{x}_i \cdot \mathbf{x}_k$  under the constraints  
 $\forall i \alpha_i \geq 0$  and  $\sum_i \alpha_i y_i = 0$

# SVM : a unique solution

If, to the dual problem, we add the Karush–Kuhn–Tucker condition :

$$\forall i \ \alpha_i(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1) = 0$$

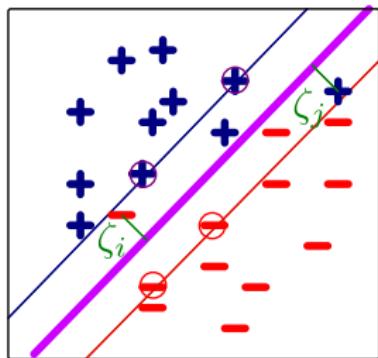
there exists an optimal solution

- if  $\alpha_i > 0$  :  $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 = 0$  : on the margin
- if  $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) > 1$  :  $\alpha_i = 0$
- thus  $\mathbf{w} = \sum_{i, \alpha_i > 0} \alpha_i y_i \mathbf{x}_i$
- for a new data  $\mathbf{x}$  :  $sign(\mathbf{w} \cdot \mathbf{x} + b) = sign\left(\sum_{i, \alpha_i > 0} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b\right)$

## First step in SVM using python : using scikitlearn

```
# pip3 install scikit-learn
from sklearn import svm
X = [[0, 0], [1, 1]]
y = [0,1]
classif = svm.SVC(kernel='linear')
classif.fit(X, y)
print('prediction class for [2,2]', classif.predict([[2., 2.]]))
print('support vectors: ', classif.support_vectors_)
```

# Accepting errors : soft margin



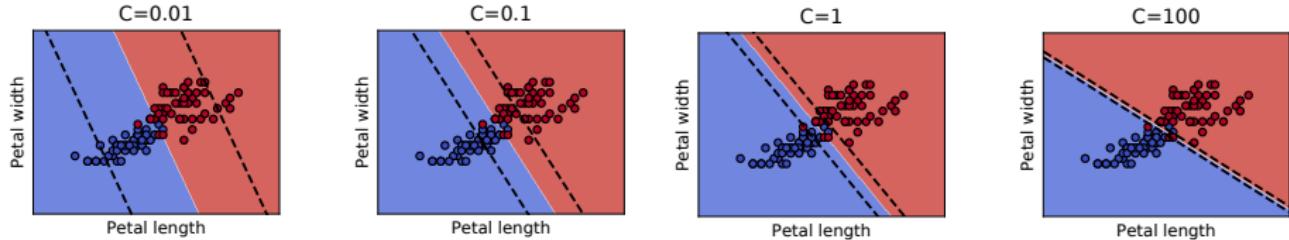
## Soft margin

minimisation of  $\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_i \zeta_i$  under the constraint  
 $\forall i y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \zeta_i$  and  $\forall i \zeta_i \geq 0$

## Dual formulation

maximisation of  $\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_k \alpha_i \alpha_k y_i y_k \mathbf{x}_i \cdot \mathbf{x}_k$  under the constraints  
 $\forall i C \geq \alpha_i \geq 0$  and  $\sum_i \alpha_i y_i = 0$

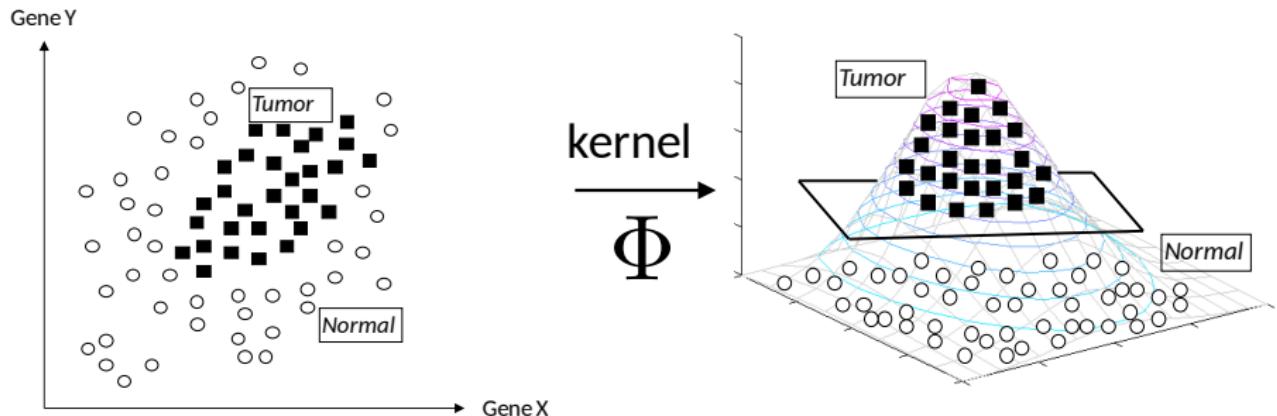
# Soft margin : parameter C



- A small C value will give a wider margin, at the cost of some misclassifications.
- A huge C value will give the hard margin classifier and tolerates zero constraint violation.
- Find the C value such that noisy data does not impact the solution too much.

- the idea is to increase the cost of bad classification for the smallest class
- $C$  is replaced by  $C^+$  for positive data and  $C^-$  for negative data.
- implementation in `sklearn` (from the documentation)
  - SVC implements a keyword `class_weight` in the `fit` method. It's a dictionary of the form `class_label : value`, where `value` is a floating point number strictly positive that sets the parameter  $C$  of class `class_label` to  $C * value$ .
  - SVC implements also weights for individual samples in method `fit` through keyword `sample_weight`. Similar to `class_weight`, these set the parameter  $C$  for the  $i^{th}$  example to  $C * sample_weight[i]$ .

# Non linearly separable data : the kernel trick



Data is not linearly separable  
in the input space

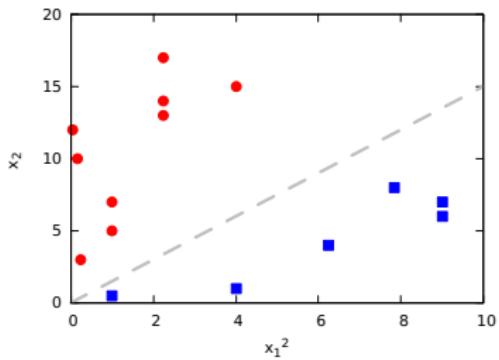
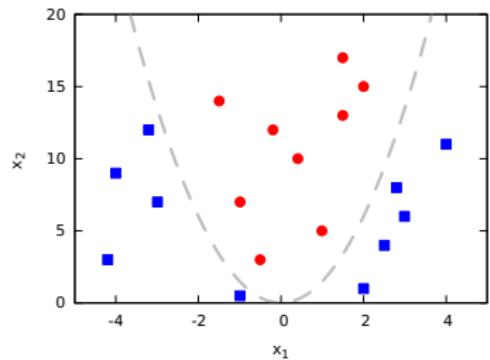
Data is linearly separable in the  
feature space obtained by a kernel

$$\Phi : \mathbf{R}^N \rightarrow \mathbf{H}$$

# kernel trick

- Here, we define  $\Phi$  explicitly

- input space  $x = [x_1, x_2]$  (2 dimensions)
- feature space  $\Phi(x) = [x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, 1]$  (6 dimensions)



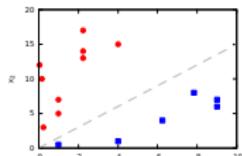
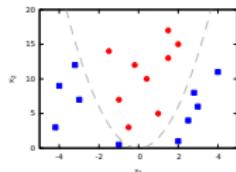
- SVM solution in the induced space (feature space) :

$$f(x) = \sum_{i \in \text{supp. vect}} \alpha_i (y_i \Phi(x_i) \cdot \Phi(x) + b)$$

- Instead of explicitly defining  $\Phi$ , we rather define  $K$  :

$$K(x, x') = \Phi(x) \cdot \Phi(x')$$

- Allow avoiding computation in the input space
  - useful, mainly if  $\dim(\Phi) = \infty$
- Back to our example :



$$\begin{aligned}\Phi(x) &= [x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, 1] \\ \Phi(x') &= [x'_1^2, x'_2^2, \sqrt{2}x'_1, \sqrt{2}x'_2, \sqrt{2}x'_1x'_2, 1] \\ K(x, x') &= (x \cdot x' + 1)^2\end{aligned}$$

## Dual formulation

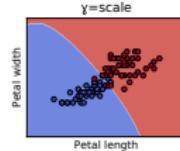
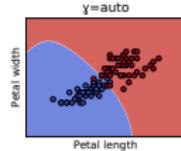
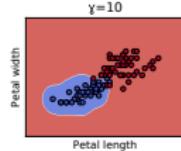
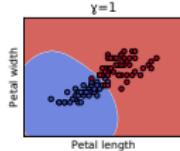
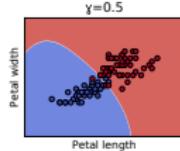
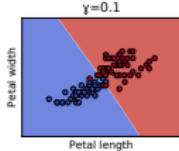
maximisation of  $\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_k \alpha_i \alpha_k y_i y_k K(x_i, x_k)$  under the constraints  
 $\forall i \ C \geq \alpha_i \geq 0$  and  $\sum_i \alpha_i y_i = 0$

## Prediction

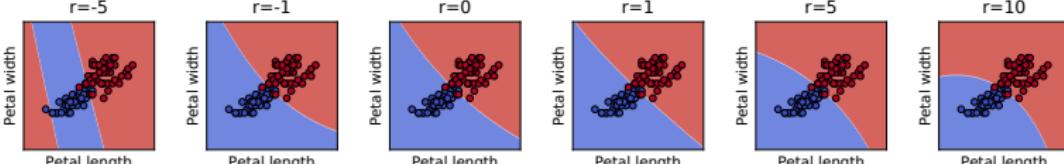
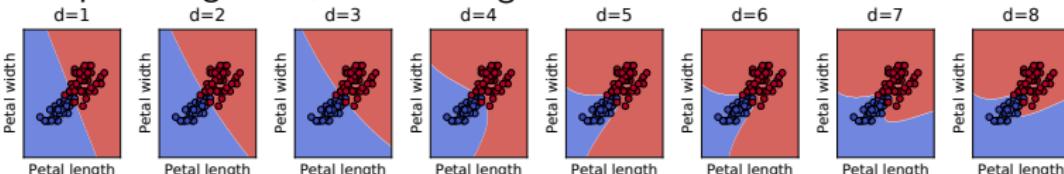
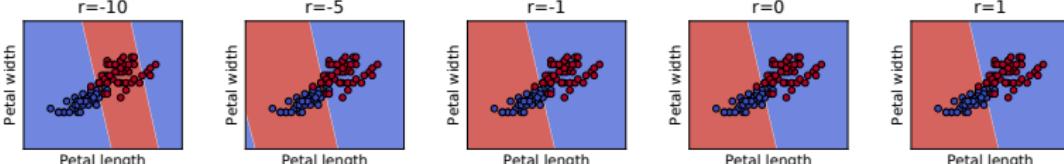
$$\text{sign} \left( \sum_{j \text{ supp. vect}} \alpha_j y_j K(x_j, x) + b \right)$$

# Predefined kernels (1)

- 'linear'
- 'rbf' (radial basis function) :
  - $\exp(-\gamma \|x - x'\|^2)$  (gamma for  $\gamma$ )
  - in scikit-learn :
    - gamma = 'auto' :  $\frac{1}{n}$  where  $n$  is the dimension of samples
    - gamma = 'scale' :  $\frac{1}{\sigma n}$  where  $\sigma$  is the standard variation of  $x$
- Examples using a RBF kernel with  $C = 1$  :



# Predefined kernels (2)

- 'polynomial':
    - $(\gamma \langle x, x' \rangle + r)^d$  (degree for  $d$ ; coef0 for  $r$ )
    - examples using degree 3,  $C = 1$  and gamma = 'scale'
  - examples using  $r = 0$ ,  $C = 1$  and gamma = 'scale'
  - 'sigmoid'
    - $\tanh(\gamma \langle x, x' \rangle + r)$
    - examples using  $C = 1$  and gamma = 'scale'
- 
- 
- 

- need to find the best kernel for your problem
- compare kernels with best parameters
  - $C, \gamma, \dots$
- methods of parameters estimation
  - exhaustive search (GridSearchCV)

```
param_grid = [  
    {'C': [0.1, 0.2, 0.5, 1, 2, 5, 10], 'kernel': ['linear']},  
    {'C': [0.5, 1, 5, 10], 'degree': [2, 3], 'coef0': [-1, 0, 1], 'kernel': ['poly']}  
]
```

- randomized search (RandomizedSearchCV)
  - distributions for parameters

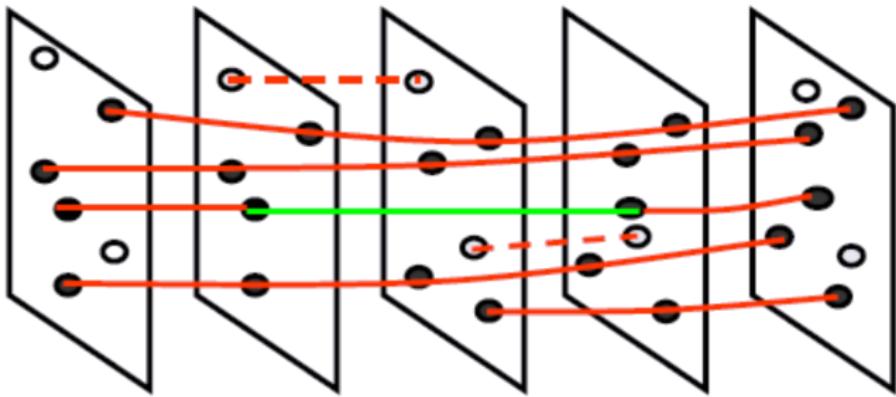
## hand make kernel

```
def my_kernel(X, Y):
    return np.dot(X, Y)
classif = svm.SVC(kernel=my_kernel)
```

- for text or genes
- example for image : [https://www.tensorflow.org/tutorials/representation/kernel\\_methods](https://www.tensorflow.org/tutorials/representation/kernel_methods) (Fourier kernel)
- for video

## a video kernel (1)

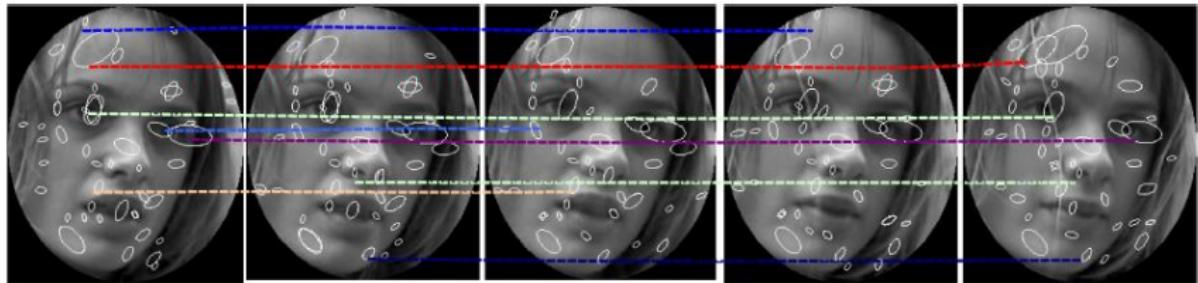
- intra-tube chain tracking
- consistent chain extraction



Solid lines: consistent chains, dash lines: noise, green lines: linking two short chains

- Tube  $T_i$  : a set of chains  $C_{ri}$  of SIFT descriptors
- $T_i = \{C_{1i}, \dots, K_{ki}\}$  and  $C_{ri} = \{SIFT_{1r_i}, \dots, SIFT_{pr_i}\}$

## a video kernel (2)



The major kernel on tubes is then defined as:

$$K'_{pow}(T_i, T_j) = \left( \sum_r \sum_s \frac{|C_{ri}|}{\sqrt{|T_i|}} \frac{|C_{sj}|}{\sqrt{|T_j|}} k'(C_{ri}, C_{sj})^q \right)^{\frac{1}{q}} \quad (1)$$

with the following minor kernel on chains:

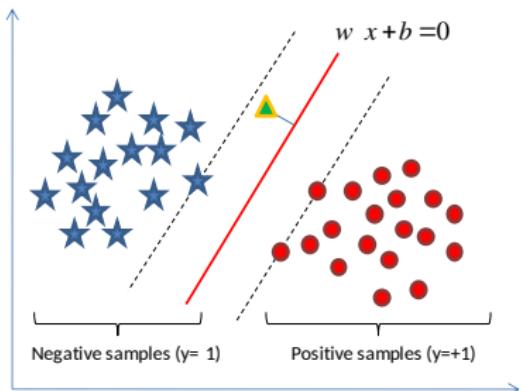
$$k'(C_{ri}, C_{sj}) = \exp \left( -\frac{1}{2\sigma^2} \chi^2 \left( \overline{C}_{ri}, \overline{C}_{sj} \right) \right) e^{-\frac{(\bar{x}_{ri}-\bar{x}_{sj})^2 + (\bar{y}_{ri}-\bar{y}_{sj})^2}{2\sigma_2^2}}$$

# Output of SVM

- not a probability
- need to be transformed (Platt)
- sklearn constructor option probability set to True

# Output of SVM classifier

- ① SVMs output a class label (positive or negative) for each sample sign( $w \cdot x + b$ )
  - ② One can also compute distance from the hyperplane that separates classes, e.g.  $w \cdot x + b$ . These distances can be used to compute performance metrics.
- 
- Question : How can one use SVMs to estimate posterior class probabilities, i.e.  $P(\text{positive class} \mid \text{sample } x)$  ?

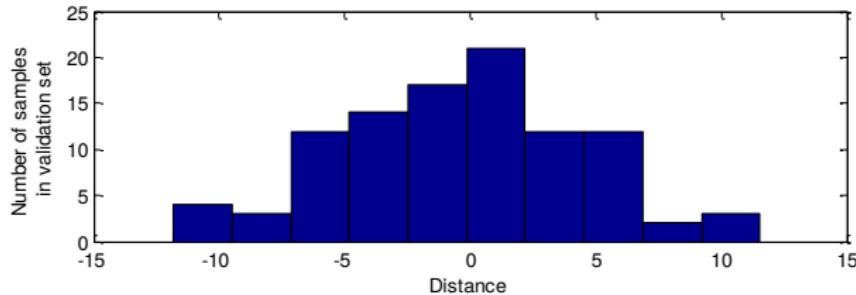


# Simple binning method

1. Train SVM classifier in the *Training set*.
2. Apply it to the *Validation set* and compute distances from the hyperplane to each sample.

Sample #	1	2	3	4	5	...	98	99	100
Distance	2	1	8	3	4	...	2	0.3	0.8

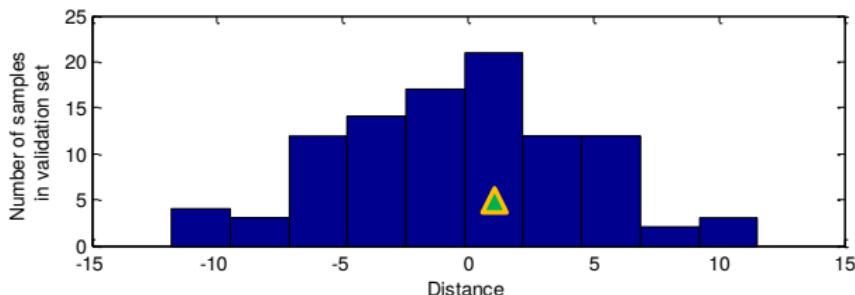
3. Create a histogram with Q (e.g., say 10) bins using the above distances. Each bin has an upper and lower value in terms of distance.



## Simple binning method

- Given a new sample from the *Testing set*, place it in the corresponding bin.

E.g., sample #382 has distance to hyperplane = 1, so it is placed in the bin [0, 2.5]



5. Compute probability  $P(\text{positive class} \mid \text{sample \#382})$  as a fraction of true positives in this bin.

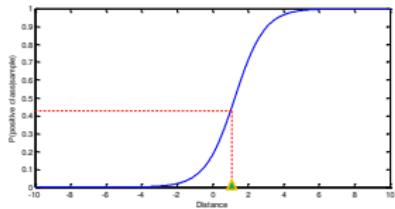
E.g., this bin has 22 samples (from the *Validation set*), out of which 17 are positive ones , so we compute  $P(\text{positive class} \mid \text{sample \#382}) = 17/22 = 0.77$

# Platts's method

- convert distances output by SVM to probabilities :
  - sigmoid filter :

$$P(\text{positive class}|\text{sample}) = \frac{1}{1 + \exp(Ad + B)}$$

where  $d$  is the distance from hyperplane,  $A$  and  $B$  parameters



- target probabilities :  $t_i = \frac{y_i+1}{2}$
- minimization :

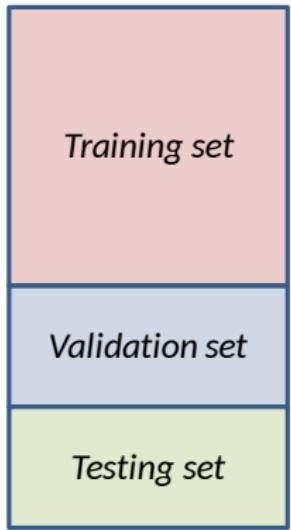
$$\min - \sum_i t_i \log(p_i) + (1 - p_i) \log(1 - t_i) \text{ where } p_i = \frac{1}{1 + \exp(Ad_i + B)}$$

# Platt's method

1. Train SVM classifier in the *Training set*.
2. Apply it to the *Validation set* and compute distances from the hyperplane to each sample.

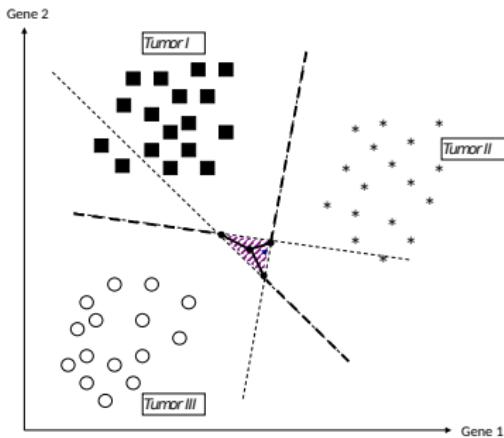
Sample #	1	2	3	4	5	...	98	99	100
Distance	2	1	8	3	4	...	2	0.3	0.8

3. Determine parameters  $A$  and  $B$  of the sigmoid function by minimizing the negative log likelihood of the data from the *Validation set*.
4. Given a new sample from the *Testing set*, compute its posterior probability using sigmoid function.

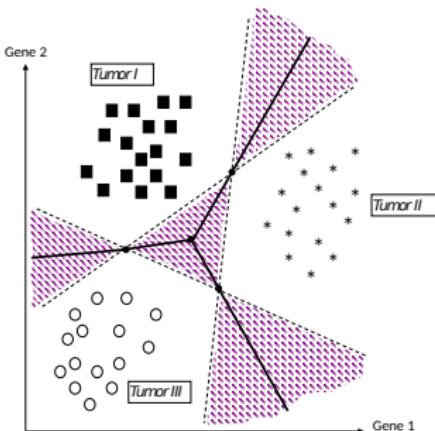


# Multiclass SVM

- $n$  classes
- two methods (`decision_function_shape(...)`)

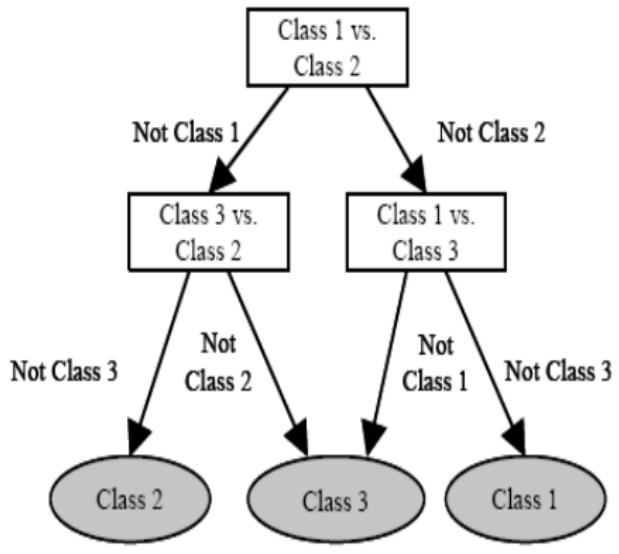


- one versus one ('ovo')
  - build  $n(n - 1)/2$  SVM classifiers
  - maximum vote from classifiers



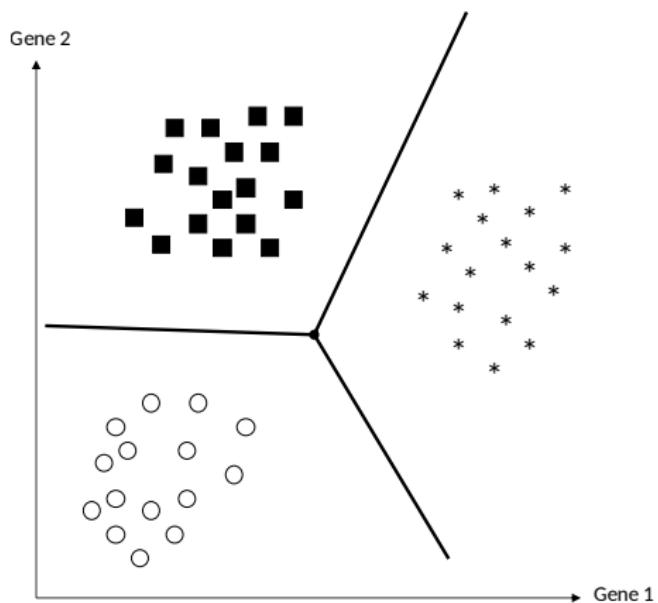
- one versus the rest ('ovr')
  - build  $n$  SVM classifiers
  - unbalanced classes

# DAGSVM multiclass



c

- Constructs a decision tree
- Each node is a binary SVM for a pair of classes
- $k$  leaves :  $k$  classification decisions
- non-leaf  $(p, q)$  : two edges
  - left edge : not  $p$  decision
  - right edge : not  $q$  decision

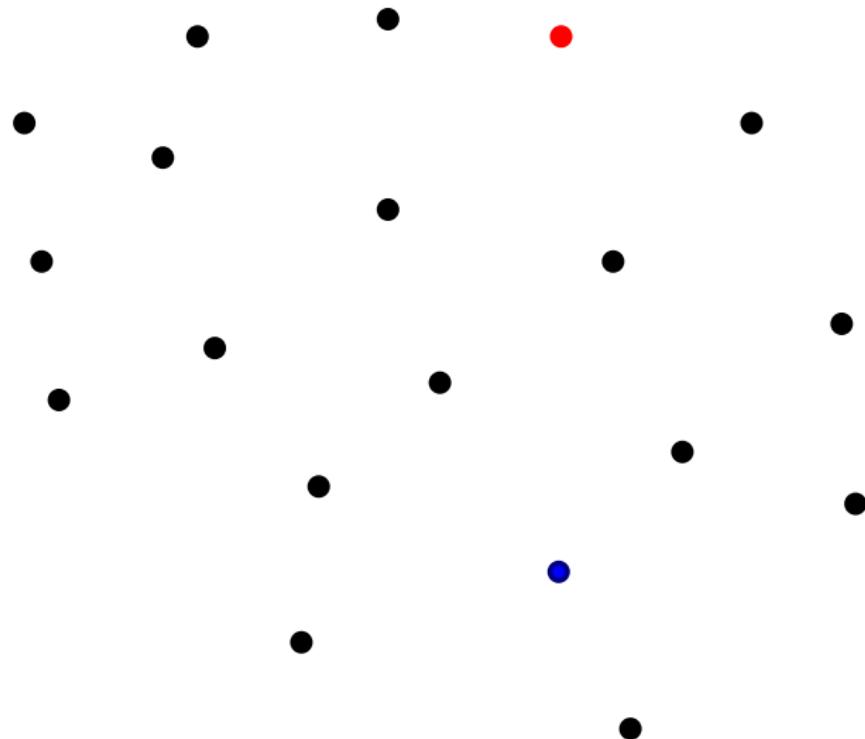


- Constructs a single classifier by maximizing the margin between all the classes simultaneously
- Both require the solution of a single QP problem of size  $(k-1)n$ , but the CS MC-SVM uses less slack variables in the constraints of the optimization problem, thereby making it computationally less expensive

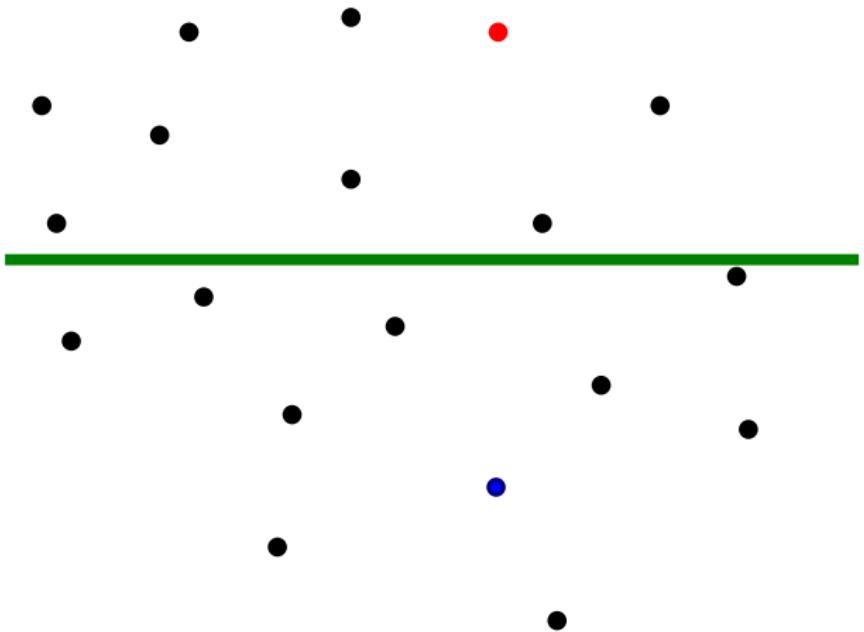
# Active learning with SVM

- hyperplane estimation using small set of labelled data
- estimation of distance for non-labelled data
- selection of data close to the hyperplane (could use a constraint on diversity)
  - ask the user for label (or to correct labels)
- refine the estimation

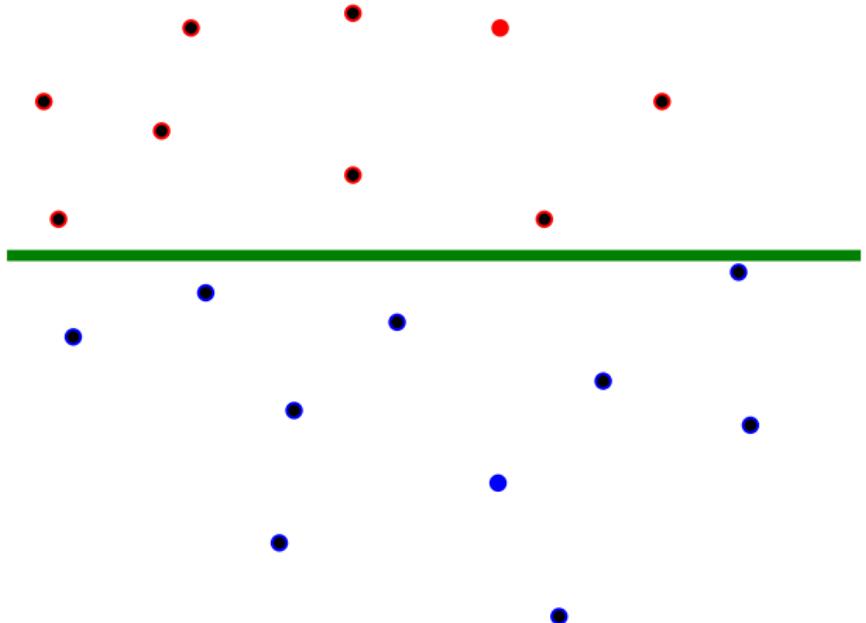
## Active learning example : start with 2 annotated samples



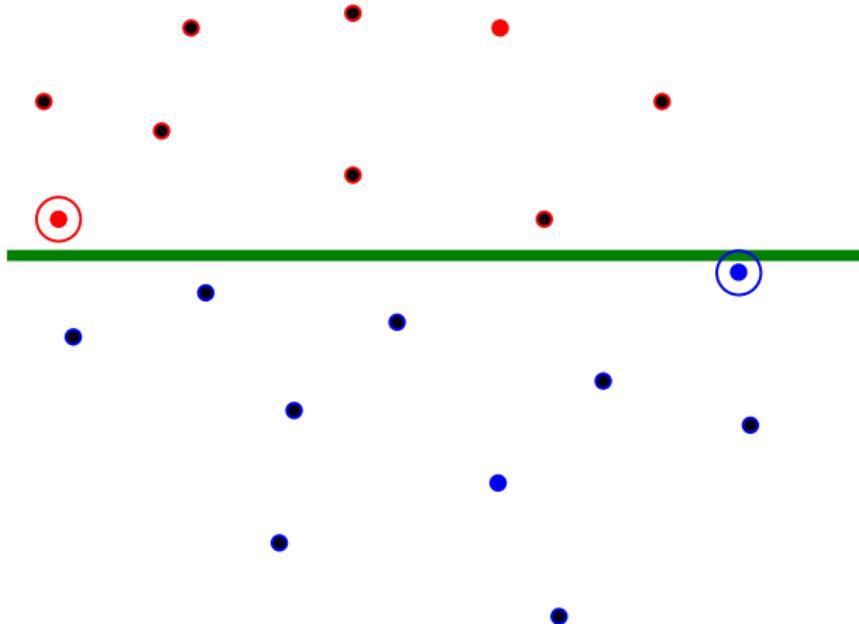
## Active learning example : linear SVM classification



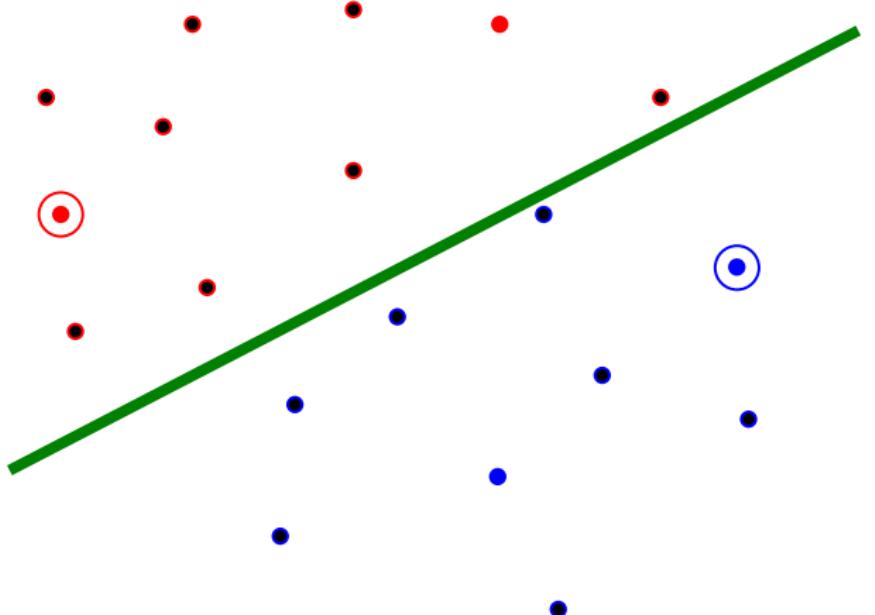
# Active learning example : label all data



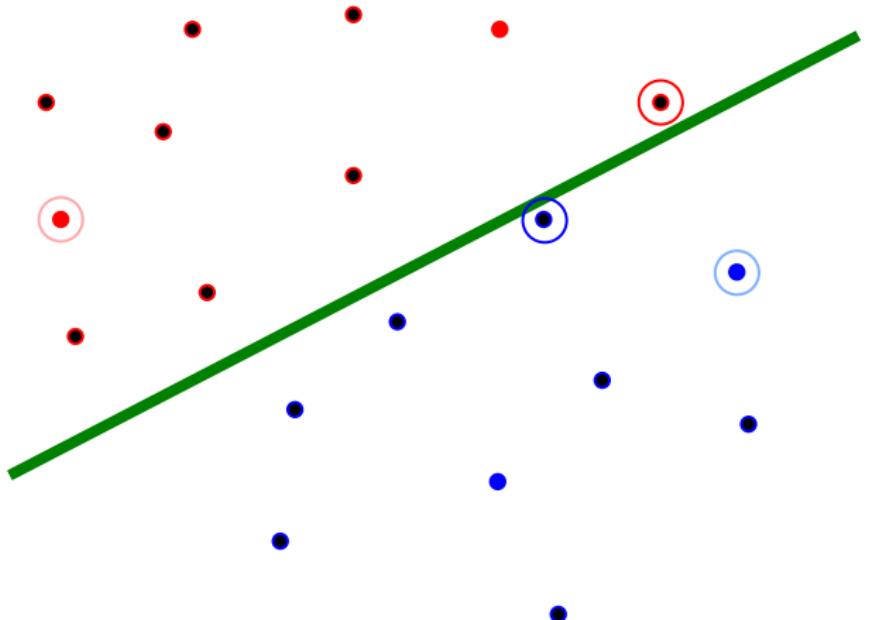
# Active learning example : ask for labels of the 2 closest



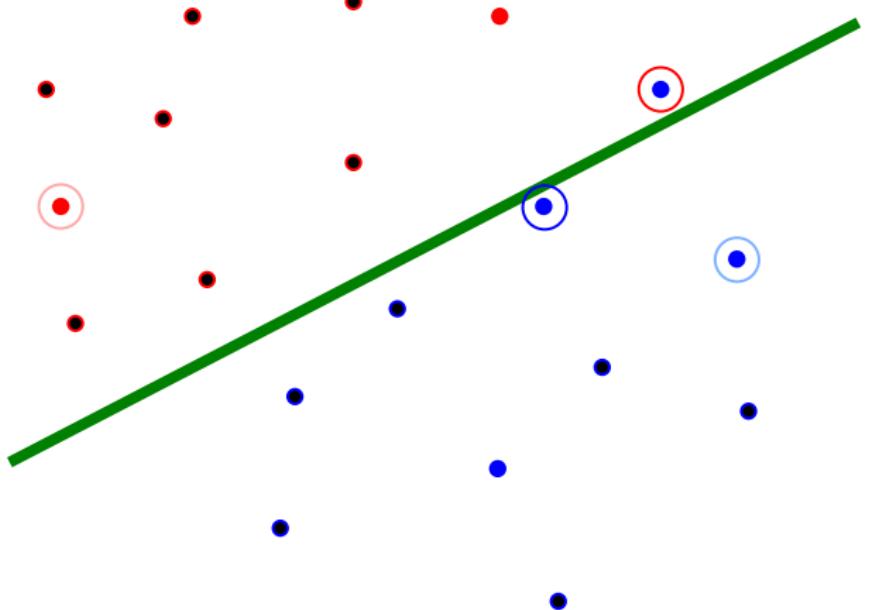
# Active learning example : recompute linear SVM and labels



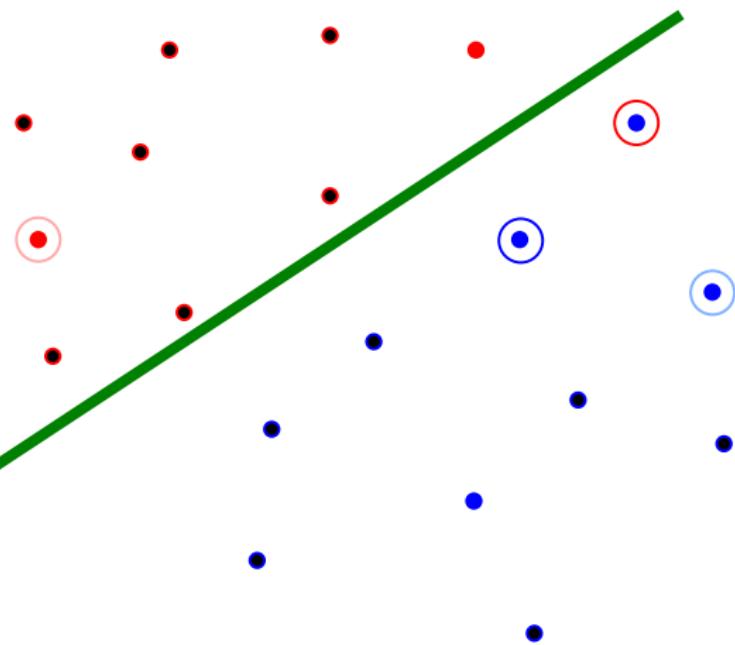
# Active learning example : ask for labels of the 2 closest



## Active learning example : correct one label

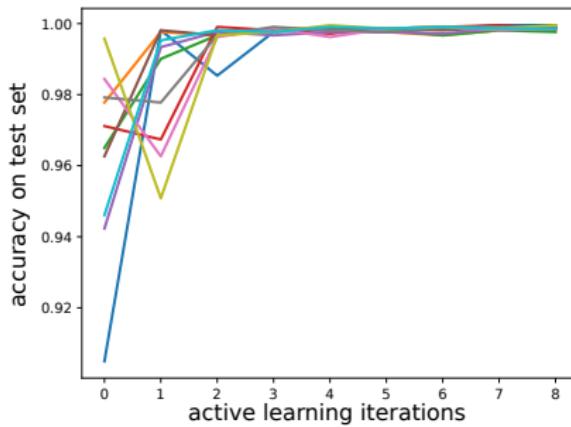


# Active learning example : recompute linear SVM



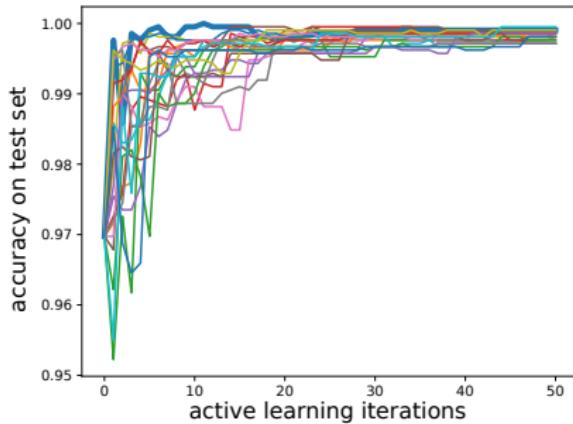
## Active learning example : MNIST classes '0' and '1'

- Start randomly with 2 samples from class '0' and 2 samples from class '1'.
  - Each curve corresponds to a new random start (10 curves).
- Add 4 new annotated samples at each iteration. The new annotated samples are chosen as the closest to the boundary.
  - Fast convergence even for the worse starts
  - With all data : 0.99905



## Active learning example : MNIST classes '0' and '1'

- Start with 2 samples from class '0' and 2 samples from class '1'
  - the same start for all the curves
- Add 4 new annotated samples at each iteration.
- In bold : the new annotated samples are chosen as the closest to the boundary.
- Other curves : new annotated samples are randomly chosen (each curve corresponds to different trials). **Longer to converge !**



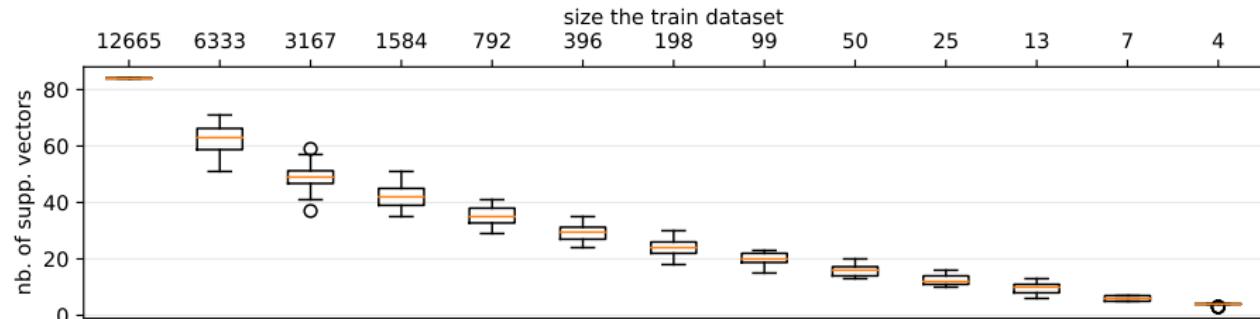
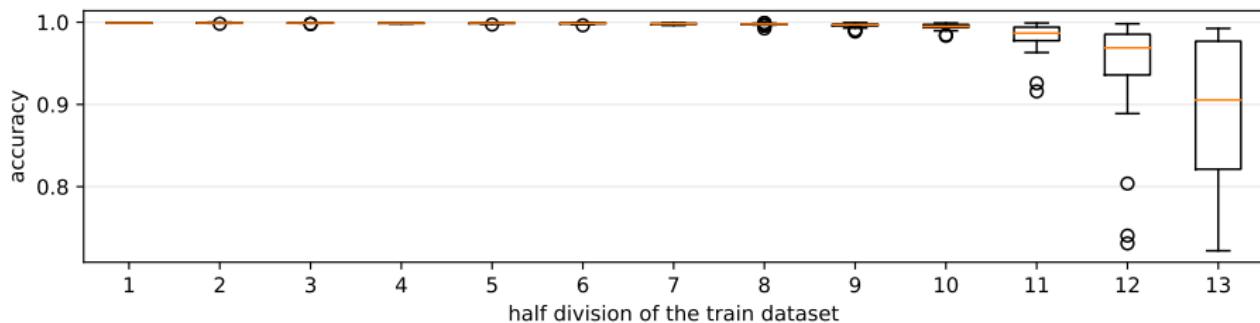
# Support Vector Machines

- SVMs are important because of theoretical reasons :
  - Robust to very large number of variables
  - Produce sparse models that are defined only by a small subset of training points (“support vectors”)
  - Can learn both simple and highly complex classification models (by using the “kernel trick”)
  - Do not require direct access to data and can work with only dot-products of data points/
  - Employ sophisticated mathematical principles to avoid overfitting with internal capacity control (regularization)

- SVMs are important because of superior empirical results
  - Do not have more free parameters than the number of support vectors, irrespective of the number of variables in the dataset
  - Require solution of a convex QP optimization problem that has a global minimum and can be solved efficiently. Thus optimizing the SVM model parameters is not subject to heuristic optimization failures that plague other machine learning methods (e.g. neural networks, decision trees, ...)

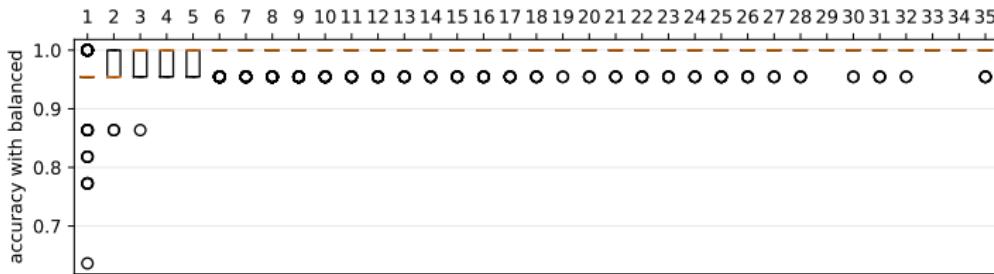
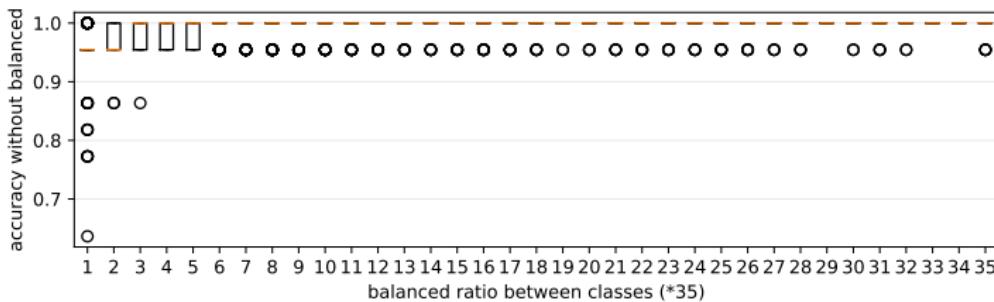
# Experimenting the size reduction of train data

- MNIST data set (2 classes : '0' and '1', 12665 data, 784 features)
- linear SVM using default parameter ( $C=1$ )
- recursive division by 2 of the train dataset (not the test !)
- 24 trials for each set of parameters (data shuffled)



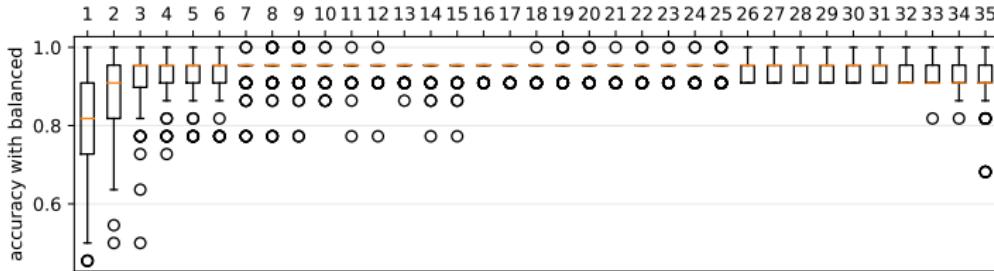
# Unbalanced data

- Iris dataset (3 classes, 150 data, 4 features).
- linear SVM using default parameters. 2nd experiment using balanced weights.
- constant size of train dataset :35. Varying ratio of class sizes.
- 100 trials for each set of parameters (data shuffled)



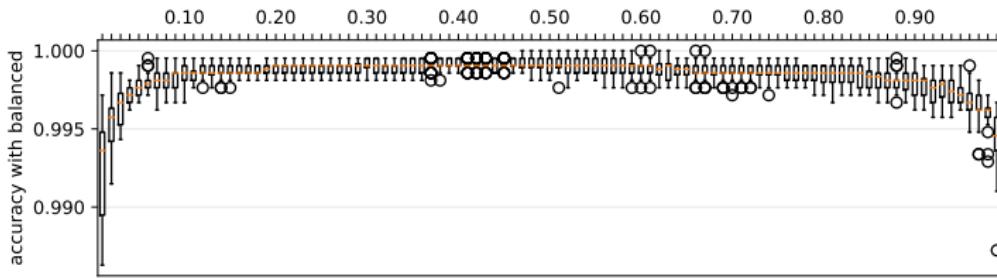
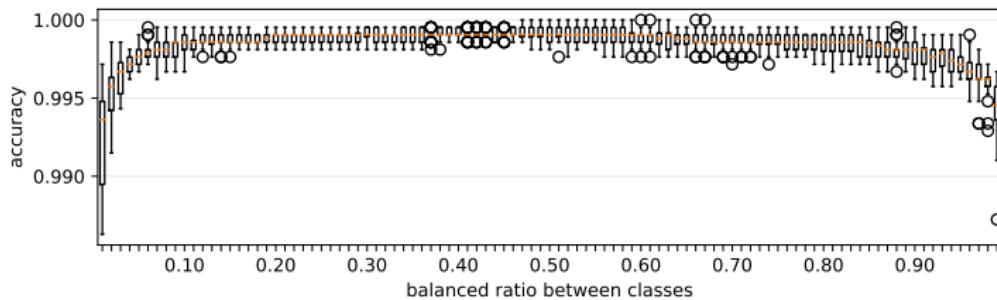
# Unbalanced data

- Iris dataset (3 classes, 150 data, 2 features).
- linear SVM using default parameters. 2nd experiment using balanced weights.
- constant size of train dataset :35. Varying ratio of class sizes.
- 100 trials for each set of parameters (data shuffled)



# Unbalanced data

- MNIST data set (2 classes : '0' and '1', 12665 data, 784 features)
- linear SVM using default parameters. 2nd experiment using balanced weights.
- constant size of train dataset : 5923. Varying ratio of class sizes.
- 24 trials for each set of parameters (data shuffled)



# Outline

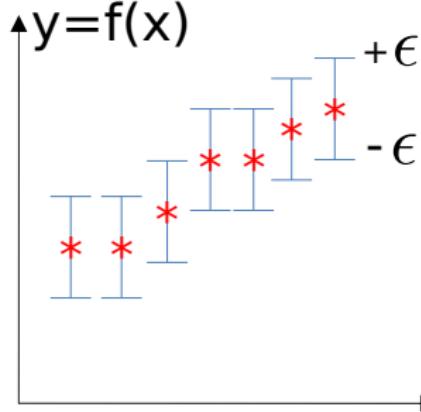
1 SVM classification

2 SVM regression

3 SVM one class

# $\epsilon$ Support Vector Regression ( $\epsilon$ -SVR)

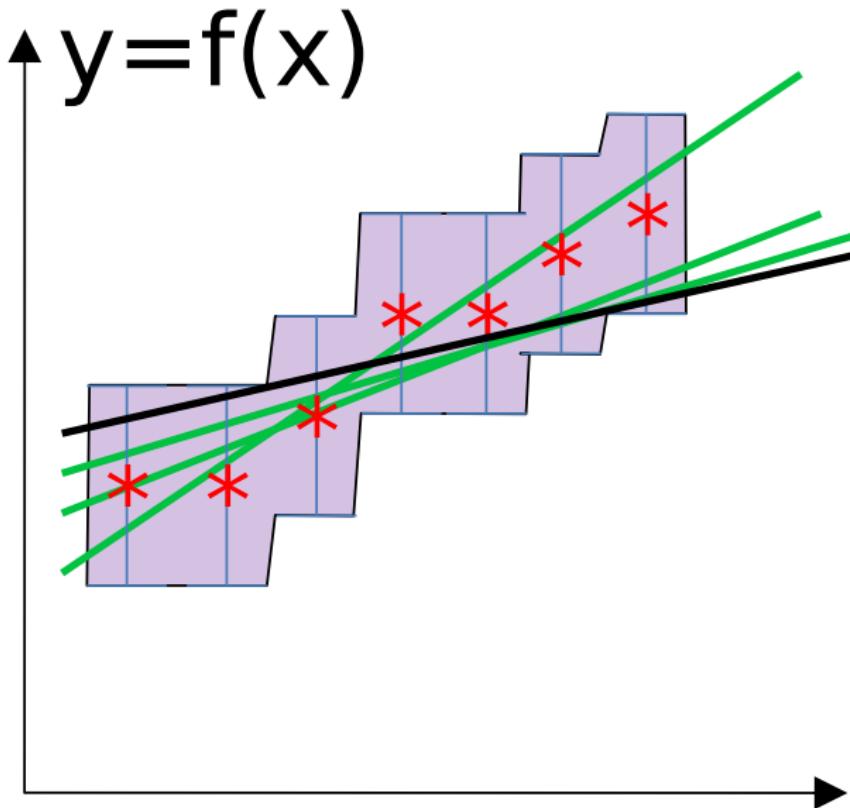
Given training data :  $(x_0, x_1, \dots) \in R^n$  and  $(y_0, y_1, \dots) \in R$



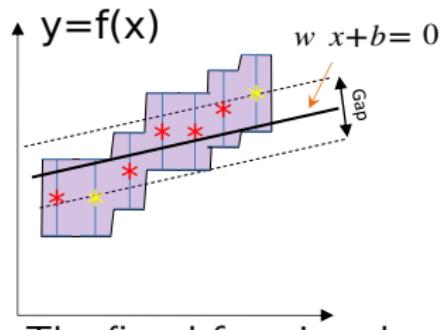
Find a function  $f(x) = w \cdot x + b$  that approximates  $y$  :

- $f$  has at most  $\epsilon$  derivation from the true values  $y_i$ ;
- it is as “flat” as possible (in order to avoid overfitting)

# $\epsilon$ Support Vector Regression



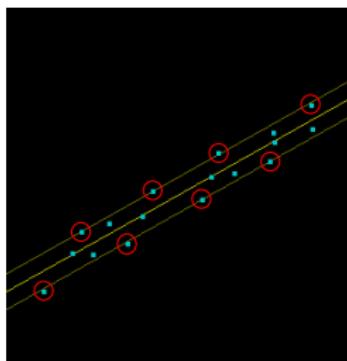
# Formulation of hard-margin $\epsilon$ -SVR



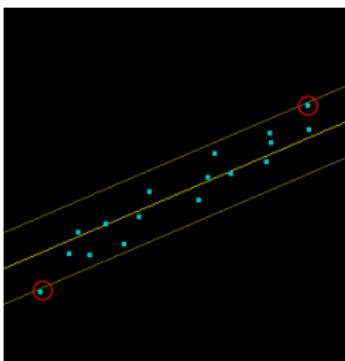
Find  $f(x) = w \cdot x + b$  by minimisation  
of  $\frac{1}{2} \|w\|^2$  subject to constraints :  
 $|y_i - (w \cdot x_i + b)| \leq \epsilon$

The fitted function should go through  $\epsilon$  neighborhood of all points.

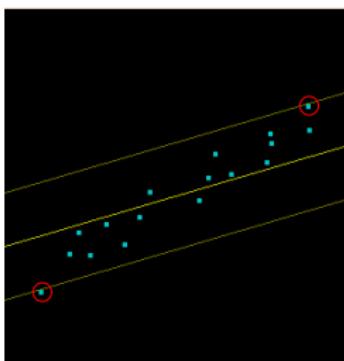
# Influence of parameter $\epsilon$



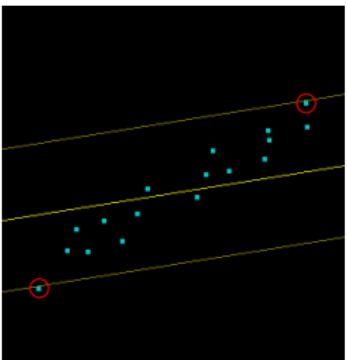
$\epsilon = 0.05$



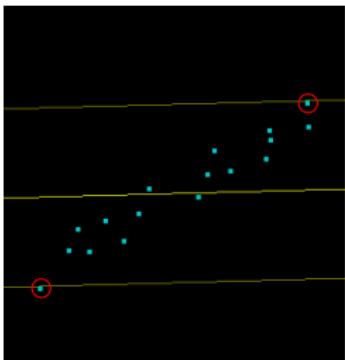
$\epsilon = 0.10$



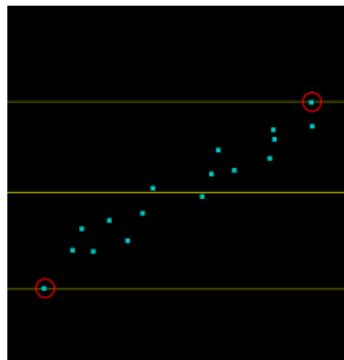
$\epsilon = 0.15$



$\epsilon = 0.20$

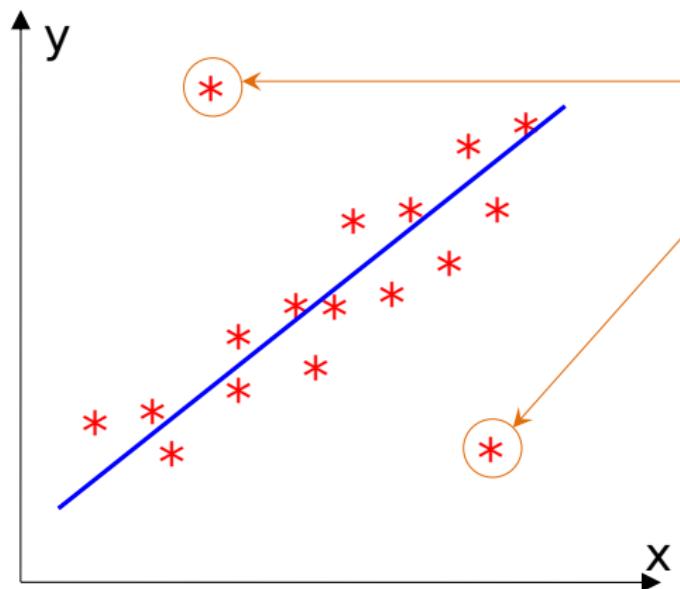


$\epsilon = 0.25$



$\epsilon = 0.30$

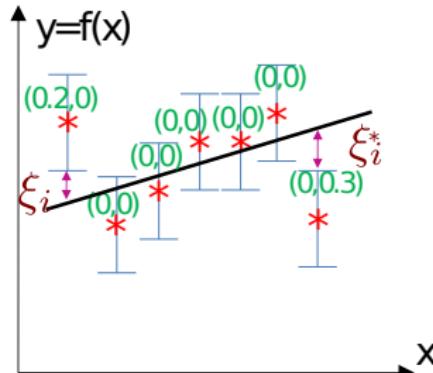
# Formulation of soft margin $\epsilon$ -SVR



If we have points like this (e.g., outliers or noise) we can either:

- a) increase  $\epsilon$  to ensure that these points are within the new  $\epsilon$  neighborhood, or
- b) assign a penalty ("slack" variable) to each of these points (as was done for "soft margin" SVMs)

# Formulation of soft margin $\epsilon$ -SVR

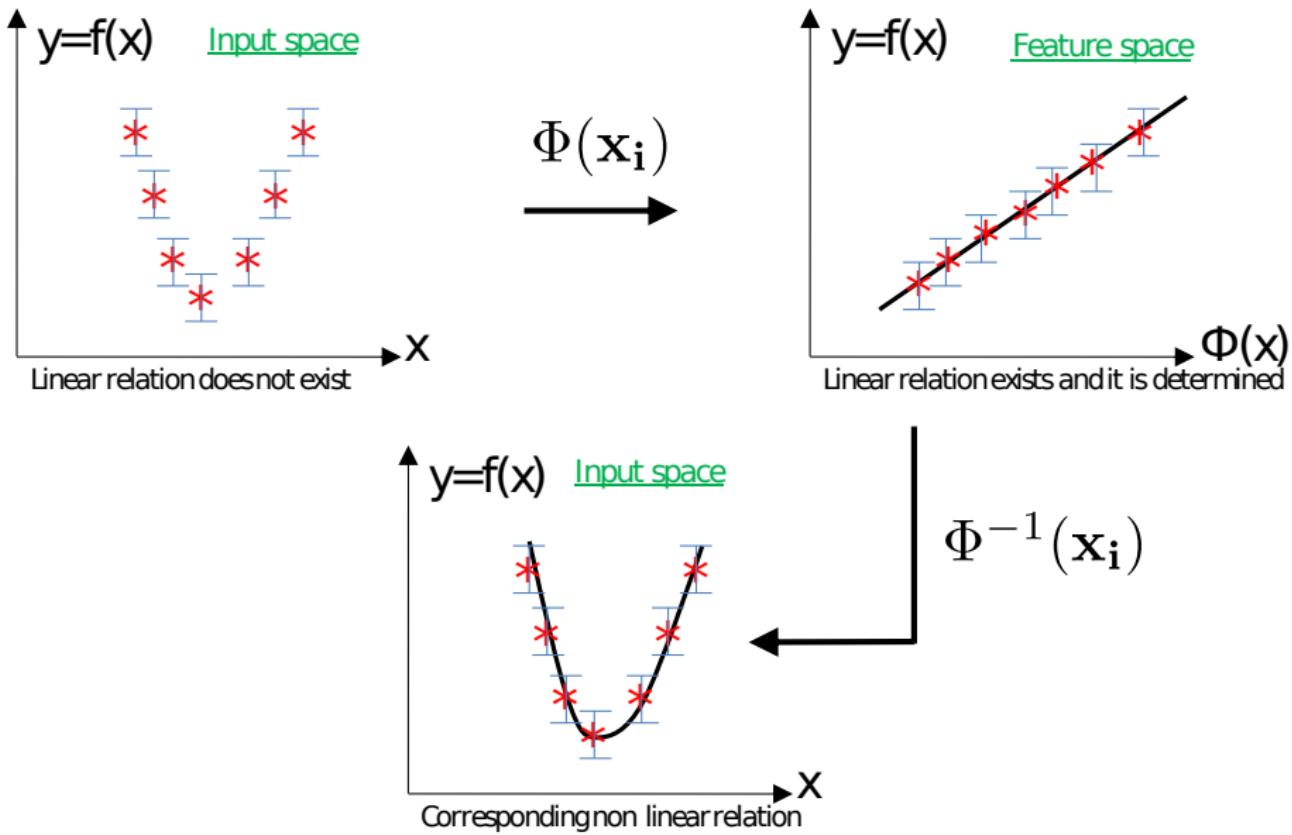


Find  $f(x) = w \cdot x + b$  by minimisation of  $\frac{1}{2} \|w\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*)$  subject to constraints :

$$y_i - (w \cdot x_i + b) \leq \text{epsilon} + \xi_i$$
$$y_i - (w \cdot x_i + b) \geq -\text{epsilon} - \xi_i^*$$
$$\xi_i, \xi_i^* \geq 0$$

- $\xi_i = 0$  if the linear function is above the lower boundary of the  $\epsilon$ -neighborhood ; otherwise it denotes distance from the linear function to the lower boundary.
- $\xi_i^* = 0$  if the linear function is below the upper boundary of the  $\epsilon$ -neighborhood ; otherwise it denotes distance from the linear function to the upper boundary

# Non linear $\epsilon$ -SVR



# Applying $\epsilon$ -SVR to real data

- In the absence of domain knowledge about decision functions, it is recommended to optimize the following parameters (e.g., by crossvalidation using gridsearch) :
  - parameter C
  - parameter  $\epsilon$
  - kernel parameters (e.g., degree of polynomial)
- Notice that parameter  $\epsilon$  depends on the ranges of variables in the dataset ; therefore it is recommended to normalize/rescale data prior to applying  $\epsilon$ SVR.

## Toy example in python

```
# pip3 install scikit-learn
from sklearn import svm
X = [[0, 0], [1, 1]]
y = [0,2]
reg = svm.SVR(kernel='linear')
reg.fit(X, y)
print('prediction for [2,2]', reg.predict([[2., 2.]]))
# 4
```

# Outline

1 SVM classification

2 SVM regression

3 SVM one class

**outlier** : inside the training dataset. Outlier detection assume that the inliers are concentrated in some regions. Samples outside these regions are considered as outliers.

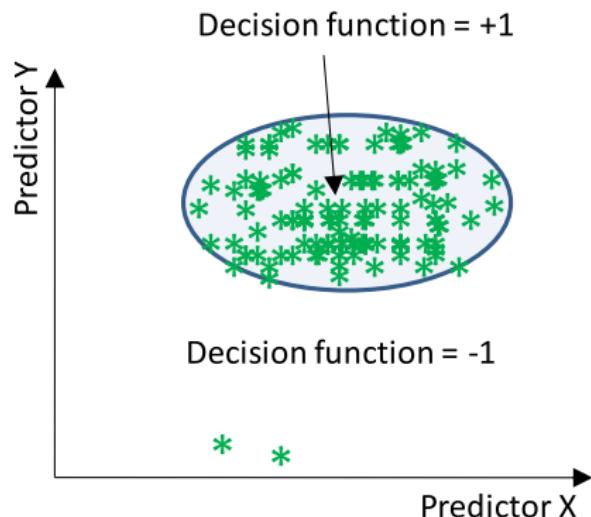
**novelty** : no outlier in the training dataset. If a new sample is detected as an outlier, it is also called a novelty.

# Key assumptions

- unsupervised learning :
  - We do not know classes/labels of samples (positive or negative) in the data available for learning
  - this is not a classification problem
- All positive samples are similar but each negative sample can be different in its own way
- Thus, do not need to collect data for negative samples !

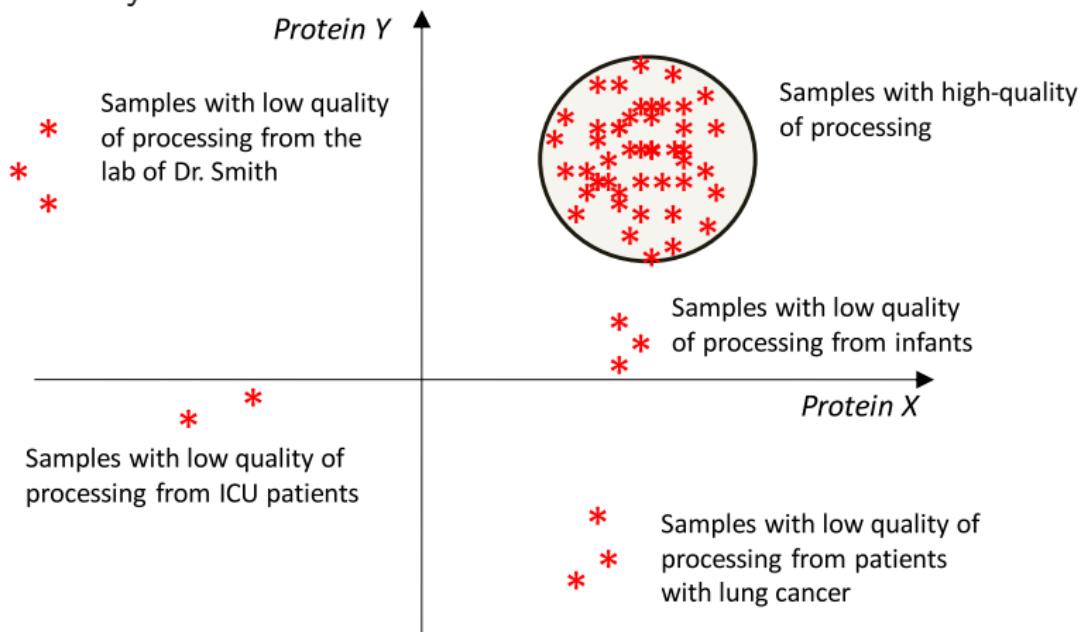
# Key ideas

- Find the simplest and most compact region in the space of predictors where the majority of data samples “live” (i.e., with the highest density of samples).
- Build a decision function that takes value +1 in this region and -1 elsewhere.
- Once we have such a decision function, we can identify novel or outlier samples in the data.



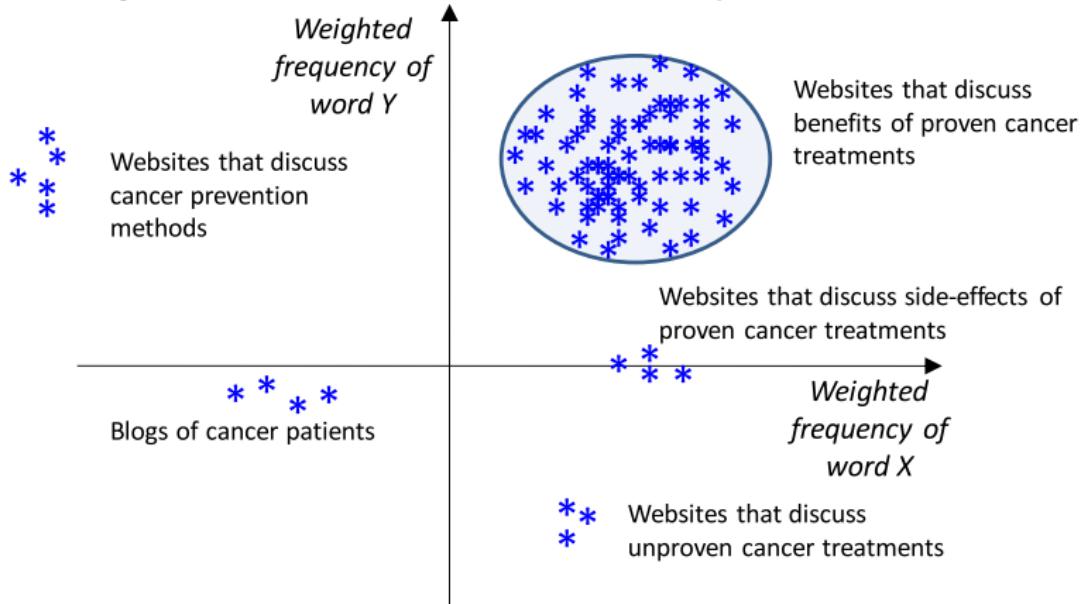
# Sample applications

Discover deviations in sample handling protocol when doing quality control of assays.



# Sample applications

Identify websites that discuss benefits of proven cancer treatments.

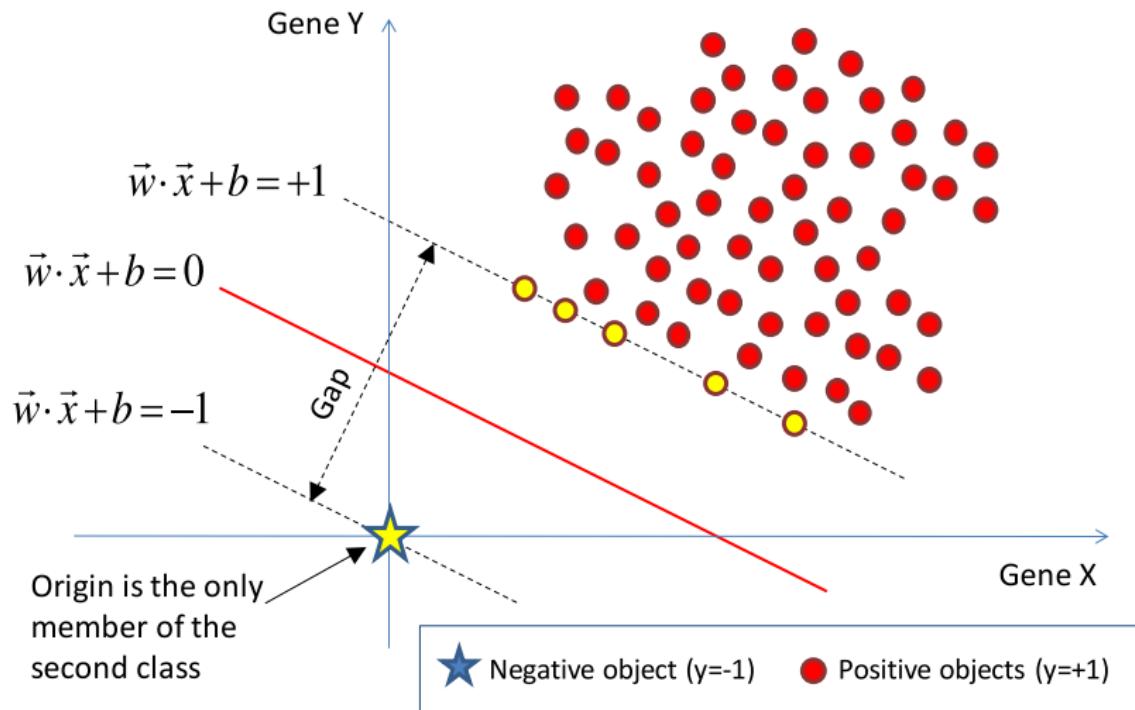


# On Presentation of one-class SVM...

- When we previously discussed SVM classification and regression, we first introduced the version of the method for linear data, then discussed the non-linear version.
- We will follow a similar course in the discussion of one-class SVMs but with one fundamental difference.
- Unlike SVMs and SVR, only non-linear one-class SVM is designed to be applied and makes practical sense.
- Furthermore, one-class SVM should be used with special types of kernel functions.
- So, even though we will describe linear one-class SVM, it is presented here only for educational purposes and its practical meaning may be limited.

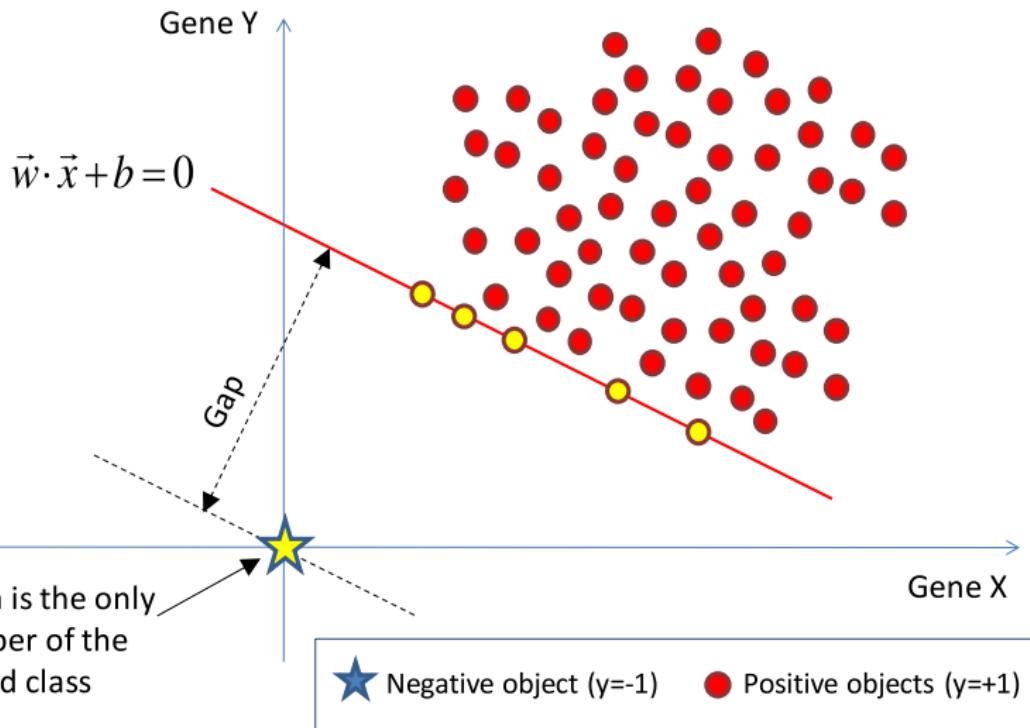
# Linear one class SVM

**Main idea:** Find the maximal gap hyperplane that separates data from the origin (i.e., the only member of the second class is the origin).



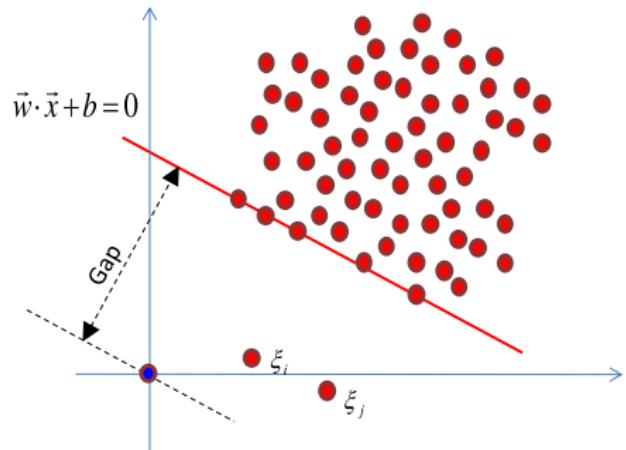
# Linear one class SVM

One-class SVM seeks the most compact region where the majority of data points live; so we are interested in another hyperplane:



# Formulation of linear one-class SVM

Given training data:  $\vec{x}_1, \vec{x}_2, \dots, \vec{x}_N \in R^n$



i.e., the decision function should be positive in all training samples except for small deviations

$$\text{Find } f(\vec{x}) = \text{sign}(\vec{w} \cdot \vec{x} + b)$$

$$\text{by minimizing } \frac{1}{2} \|\vec{w}\|^2 + \frac{1}{vN} \sum_{i=1}^N \xi_i + b$$

subject to constraints:

$$\vec{w} \cdot \vec{x}_i + b \geq -\xi_i$$

$$\xi_i \geq 0$$

for  $i = 1, \dots, N.$

upper bound on the fraction of outliers (i.e., points outside decision surface) allowed in the data

# Formulation of one-class SVM

## Linear case

Find  $f(\vec{x}) = \text{sign}(\vec{w} \cdot \vec{x} + b)$

by minimizing  $\frac{1}{2} \|\vec{w}\|^2 + \frac{1}{N} \sum_{i=1}^N \xi_i + b$

subject to constraints:

$$\vec{w} \cdot \vec{x} + b \geq -\xi_i$$

$$\xi_i \geq 0$$

for  $i = 1, \dots, N$ .

## Non-linear case (use “kernel trick”)

Find  $f(\vec{x}) = \text{sign}(\vec{w} \cdot \Phi(\vec{x}) + b)$

by minimizing  $\frac{1}{2} \|\vec{w}\|^2 + \frac{1}{N} \sum_{i=1}^N \xi_i + b$

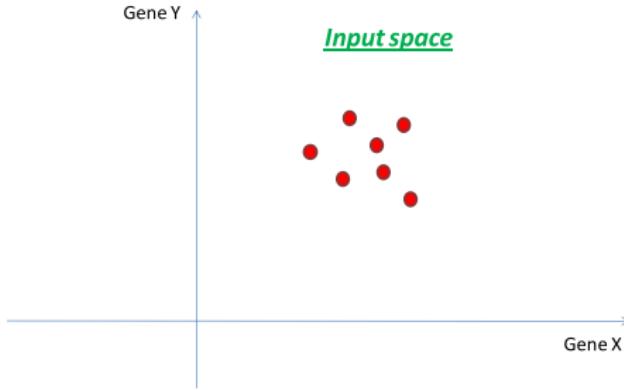
subject to constraints:

$$\vec{w} \cdot \Phi(\vec{x}) + b \geq -\xi_i$$

$$\xi_i \geq 0$$

for  $i = 1, \dots, N$ .

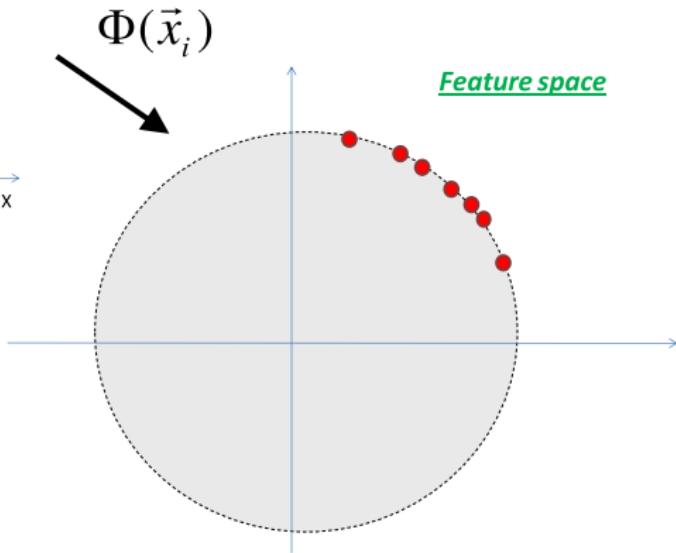
# Non linear one class SVM



Input space

We want to find the simplest and most compact region enclosing the majority of data points.

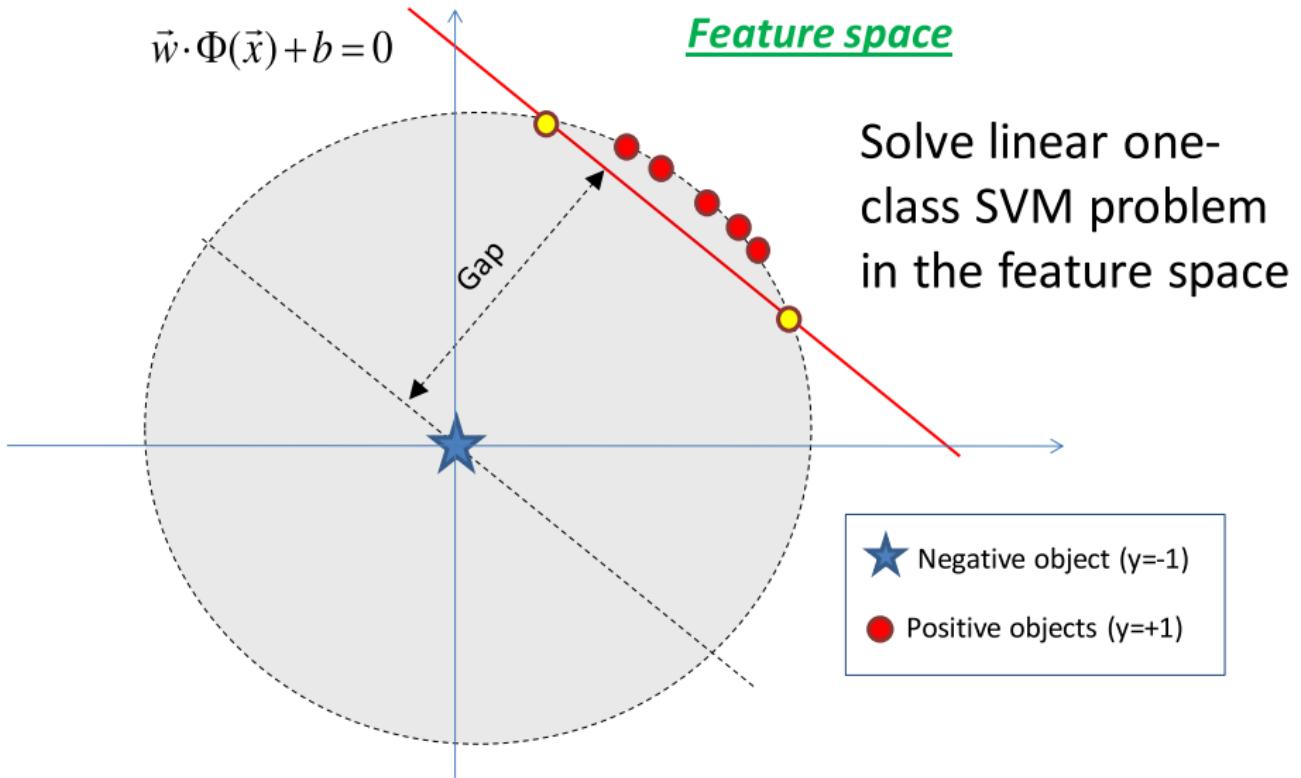
Use a kernel to map all points to a sphere with unit radius (e.g., use Gaussian kernel)



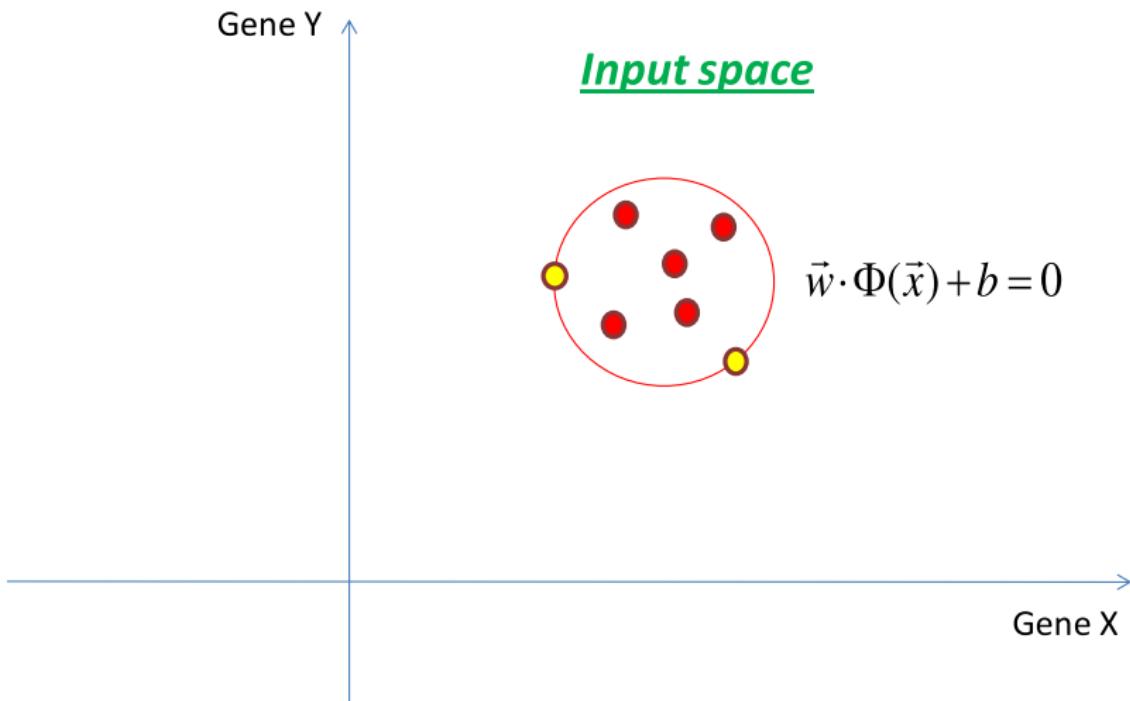
$$\Phi(\vec{x}_i)$$

Feature space

# Non linear one class SVM



## Non linear one class SVM

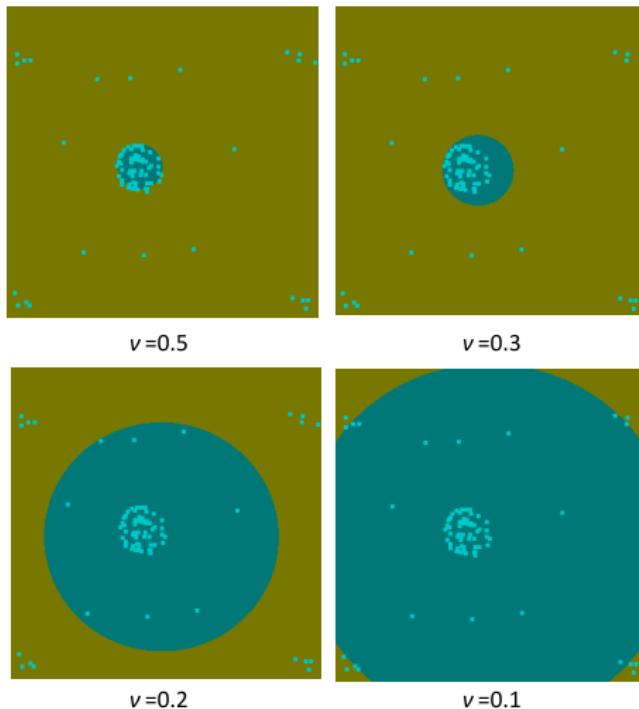


Consider the corresponding decision surface (image) in the input space – it is exactly what we were looking for!

# More about one-class SVM

- One-class SVMs inherit most of properties of SVMs for binary classification (e.g., “kernel trick”, sample efficiency, ease of finding of a solution by efficient optimization method, etc.) ;
- The choice of parameter  $\nu$  significantly affects the resulting decision surface.
- The choice of origin is arbitrary and also significantly affects the decision surface returned by the algorithm.

# Influence of the parameter $\nu$



- assume that outliers are in the neighborhood of the origin
- no use of partial information on outliers

## Sample python code

```
from sklearn.svm import OneClassSVM
svmOC = OneClassSVM(kernel='rbf', nu=0.1, gamma = 0.001)
svmOC.fit(inliers)
pred = svmOC.predict(test)
```