SVM: Support Vector Machine

Diane Lingrand and many contributors

UNIVERSITÉ
CÔTE D'AZUR

SI4

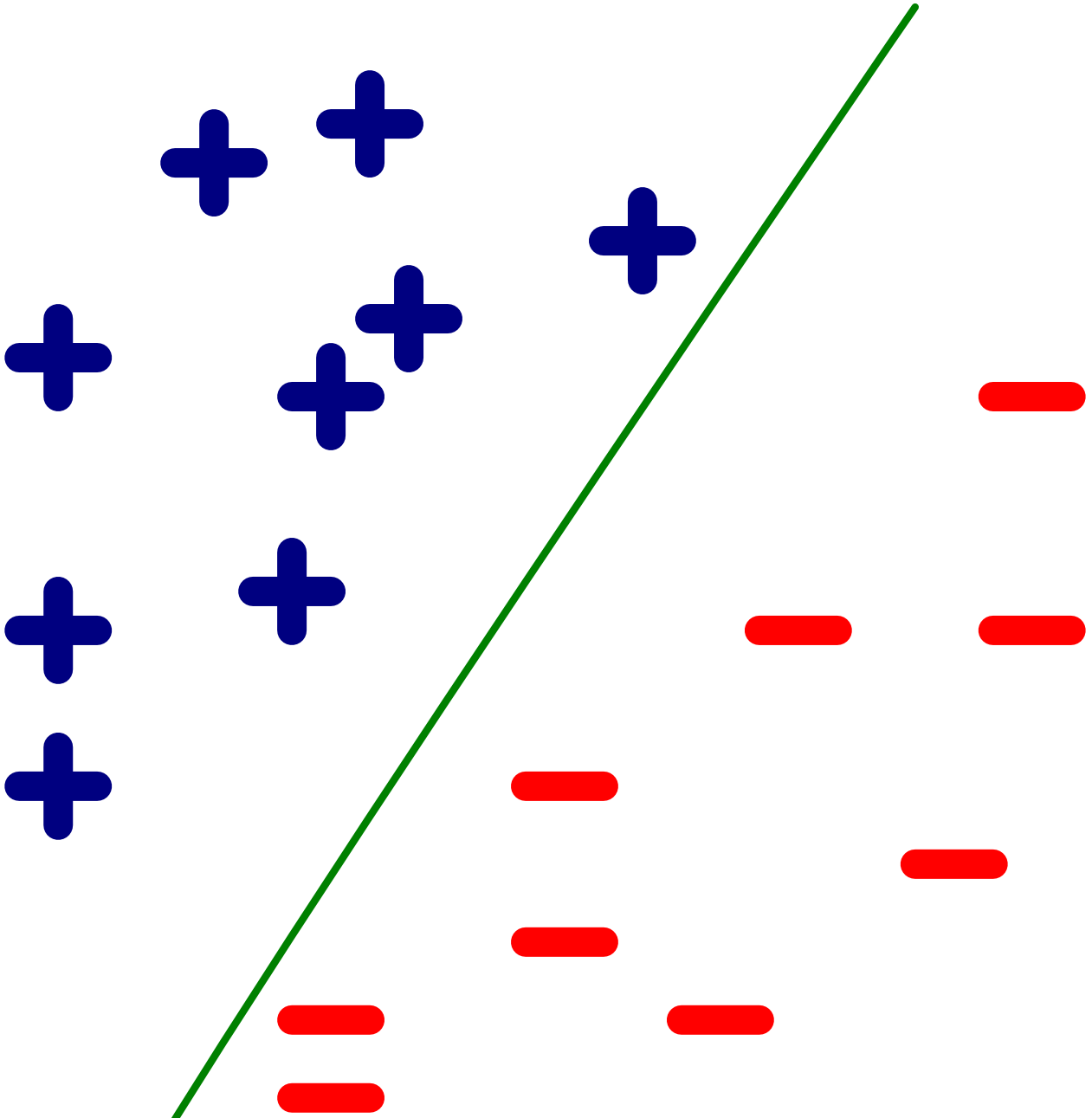
2022 - 2023

Outline

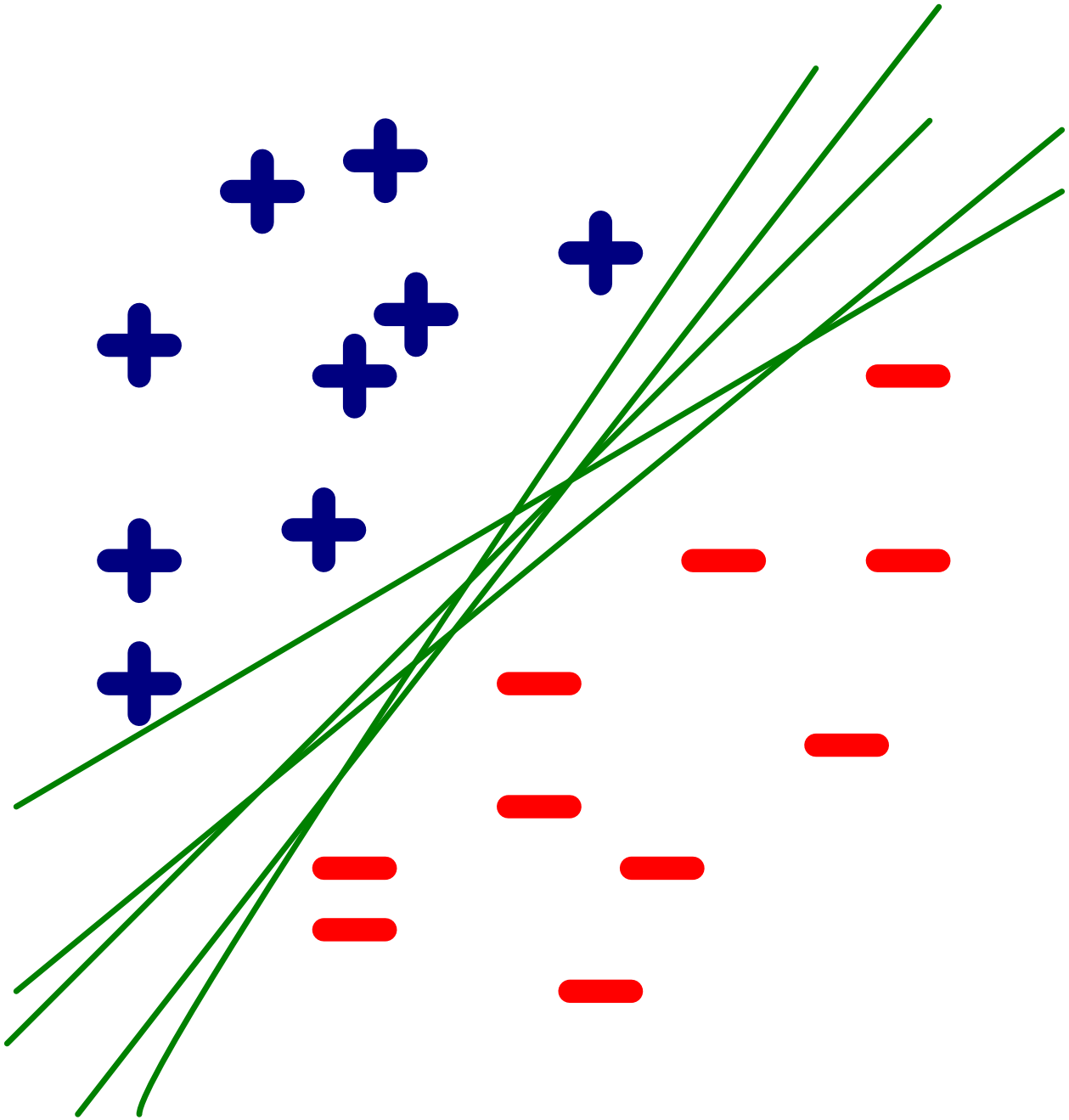
Introduction

- 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

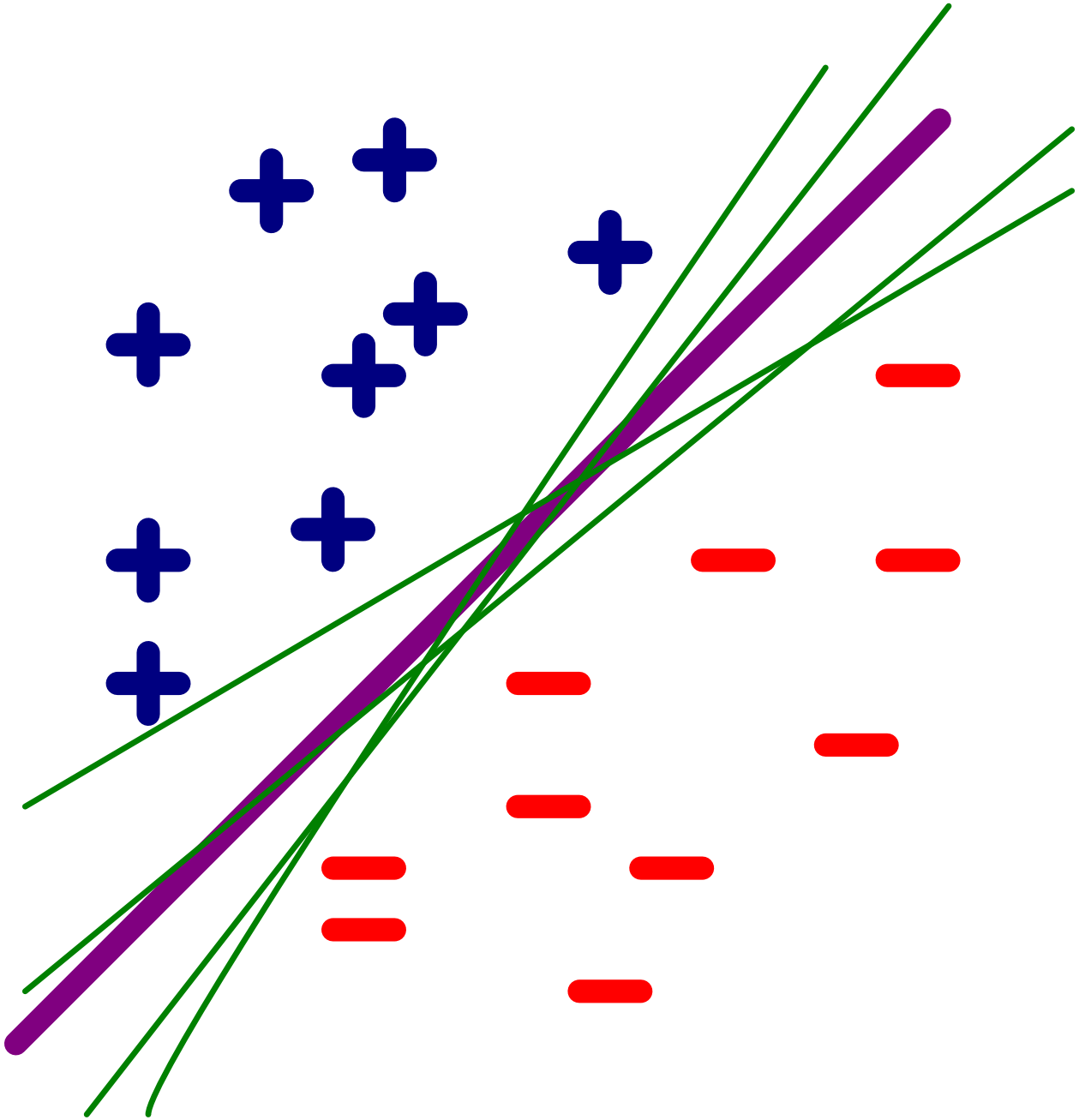
Principle : linear separation



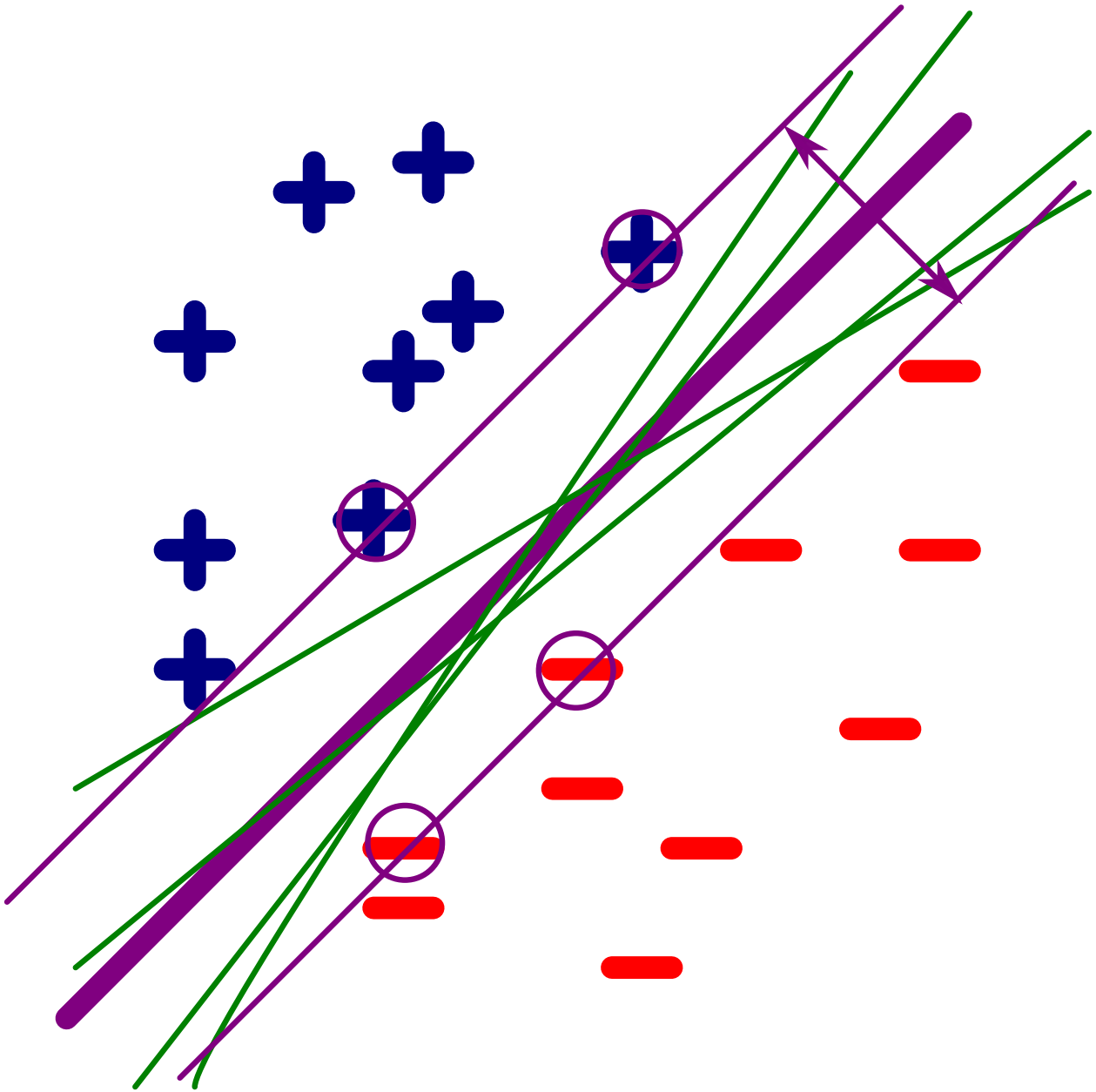
Principle : many solution



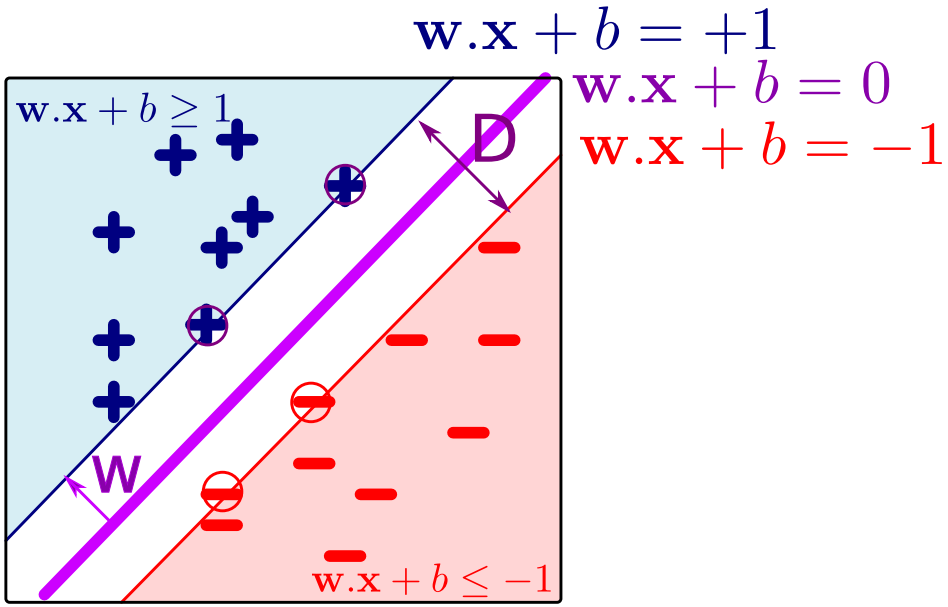
Principle : find the best one



Principle : maximising the margin



Problem formalisation



Prediction : $sign(\mathrm{w.x} + b)$

- Margin maximisation : $D =$
  -
- Labeled data :
  - positive samples : $y = +1$
    - $\mathrm{w.x} + b \geq 1$
  - negative samples : $y = -1$
    - $\mathrm{w.x} + b \leq -1$
  - thus : $y\,(\mathrm{w.x} + b) \geq 1$

Computing the margin (1)

$$A \in (w.x + b = +1) \quad \Rightarrow \quad w_1 x_1^A + w_2 x_2^A + b = +1 \tag{1}$$

$$B \in (w.x + b = -1) \quad \Rightarrow \quad w_1 x_1^B + w_2 x_2^B + b = -1 \tag{2}$$

$$(1) - (2) \quad \Rightarrow \quad w_1(x_1^A - x_1^B) + w_2(x_2^A - x_2^B) = 2 \tag{3}$$

$$\vec{AB} \parallel \vec{w} \quad \Rightarrow \quad \frac{x_2^B - x_2^A}{x_1^B - x_1^A} = \frac{w_2}{w_1} \tag{4}$$

$$\Rightarrow \quad x_2^B - x_2^A = \frac{w_2}{w_1}(x_1^B - x_1^A) \tag{5}$$

$$(5) \text{ in } (3) \quad \Rightarrow \quad w_1(x_1^A - x_1^B) + \frac{w_2^2}{w_1}(x_1^A - x_1^B) = 2 \tag{6}$$

$$\Rightarrow \quad \boxed{x_1^A - x_1^B = \frac{2w_1}{w_1^2 + w_2^2}} \tag{7}$$

$$(6) \text{ in } (5) \quad \Rightarrow \quad \boxed{x_2^A - x_2^B = \frac{2w_2}{w_1^2 + w_2^2}} \tag{8}$$

Computing the margin (2)

Remember :

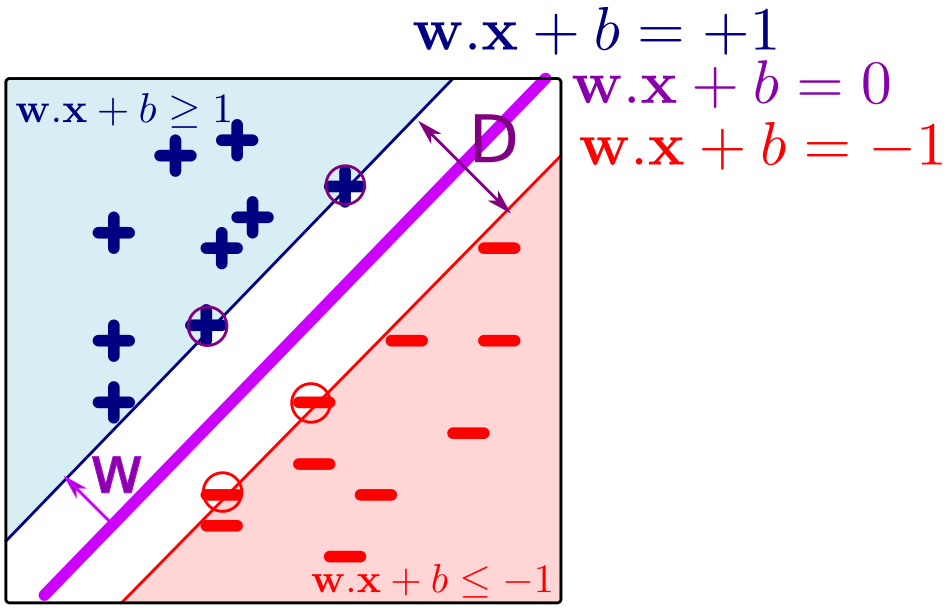
$$\boxed{x_1^A - x_1^B = \frac{2w_1}{w_1^2 + w_2^2}}$$

$$\boxed{x_2^A - x_2^B = \frac{2w_2}{w_1^2 + w_2^2}}$$

$$\begin{aligned} D &= \sqrt{(x_1^A - x_1^B)^2 + (x_2^A - x_2^B)^2} \\ &= \sqrt{\frac{4w_1^2}{(w_1^2 + w_2^2)^2} + \frac{4w_2^2}{(w_1^2 + w_2^2)^2}} \\ &= 2\sqrt{\frac{w_1^2 + w_2^2}{(w_1^2 + w_2^2)^2}} = \frac{2}{\sqrt{w_1^2 + w_2^2}} \end{aligned}$$

$$\boxed{D = \frac{2}{\|w\|}}$$

Problem formalisation



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

SVM problem :

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

Prediction : $sign(\mathrm{w.x} + b)$

dual problem

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

dual problem

maximisation of $\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_k \alpha_i \alpha_k y_i y_k \mathrm{x_i} \mathrm{x_k}$ under the contraints $\forall i \ \alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$

SVM : a unique solution

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

$$\forall i \ \alpha_i (y_i \, (\mathrm{w}.\mathrm{x}_i + b) - 1) = 0$$

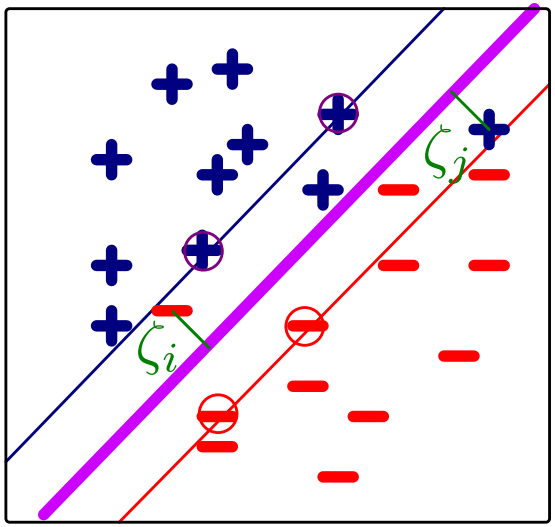
there exists an optimal solution

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

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



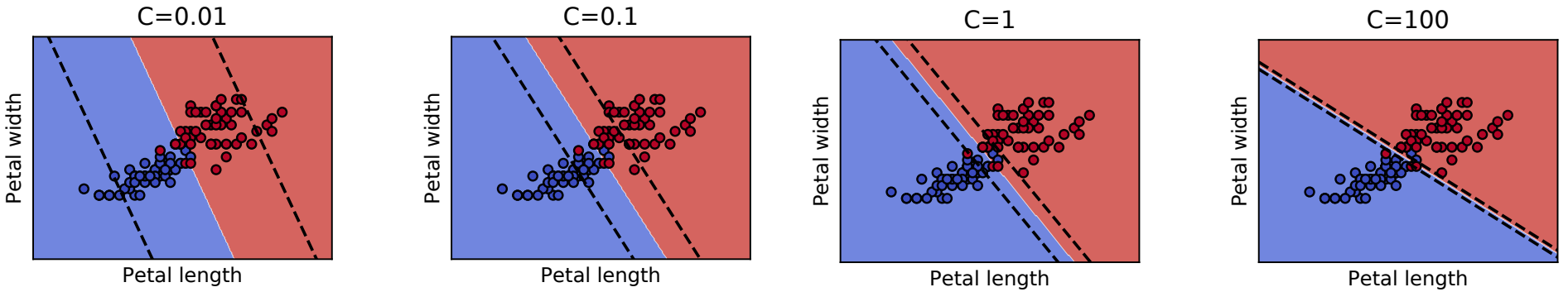
Soft margin

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

Dual formulation

maximisation of $\sum_i \alpha_i - \frac{1}{2}\sum_i \sum_k \alpha_i \alpha_k y_i y_k \mathrm{x_i x_k}$ under the contraints
$\forall i\ C \geq \alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$

Soft margin : parameter C

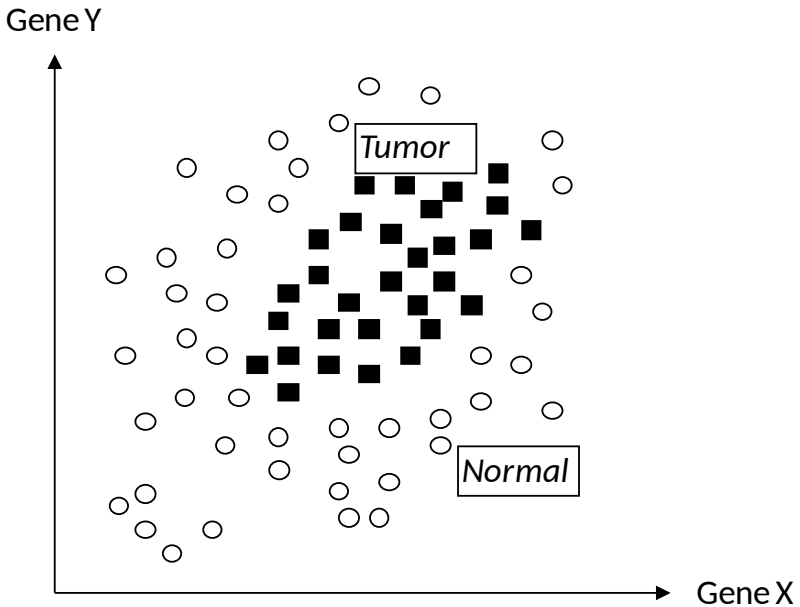


- A small C value will give a wider margin, at the cost of some misclassifications.
- A huge C value will give the hard margin classifier and tolerates zero constraint violation.
- Find the C value such that noisy data does not impact the solution too much.

Unbalanced data

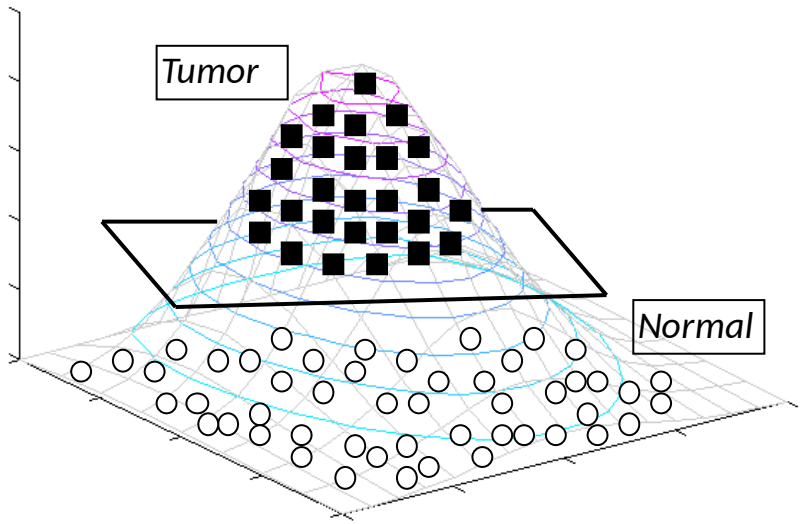
- the idea is to increase the cost of bad classification for the smallest class
- C is replaced by C^+ for positive data and C^- for negative data.
- implementation in `sklearn` (from the documentation)
  - SVC implements a keyword `class_weight` in the fit method. It's a dictionary of the form `class_label : 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



kernel

Φ



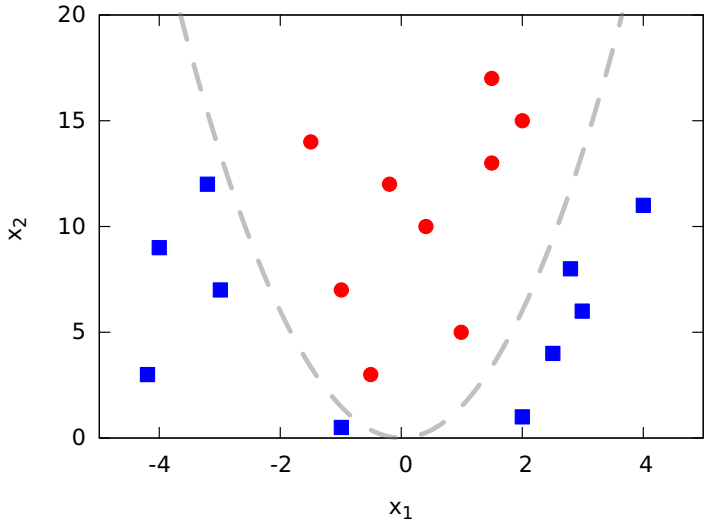
Data is not linearly separable in the input space

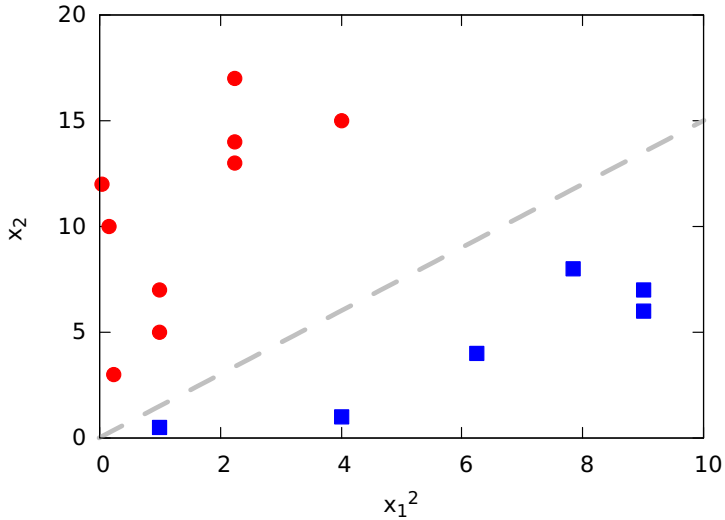
Data is linearly separable in the feature space obtained by a kernel

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

kernel trick

- Here, we define Φ explicitely
  - 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)





kernel trick

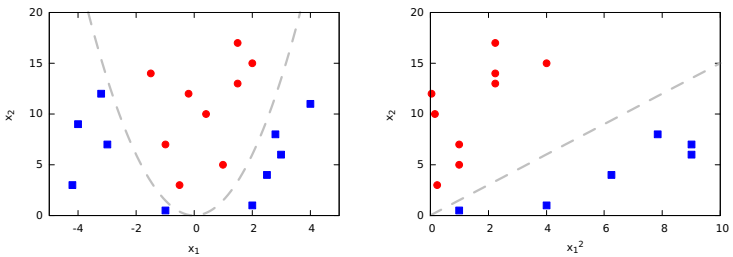
- SVM solution in the induced space (feature space) :

$$f(x) = \sum_{isupp.vect} \alpha_i (y_i \Phi(x_i).\Phi(x) + b)$$

- Instead of explicitely defining Φ, we rather define K :

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

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



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

Kernels

Dual formulation

maximisation of $\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_k \alpha_i \alpha_k y_i y_k K(\mathrm{x_i}, \mathrm{x_k})$ under the contraints

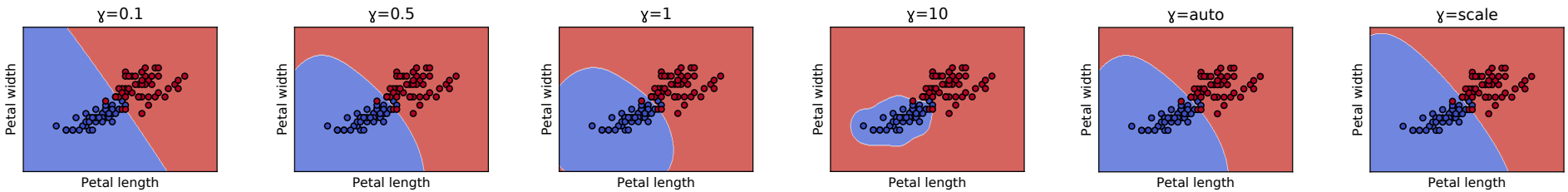
$\forall i \ C \geq \alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$

Prediction

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

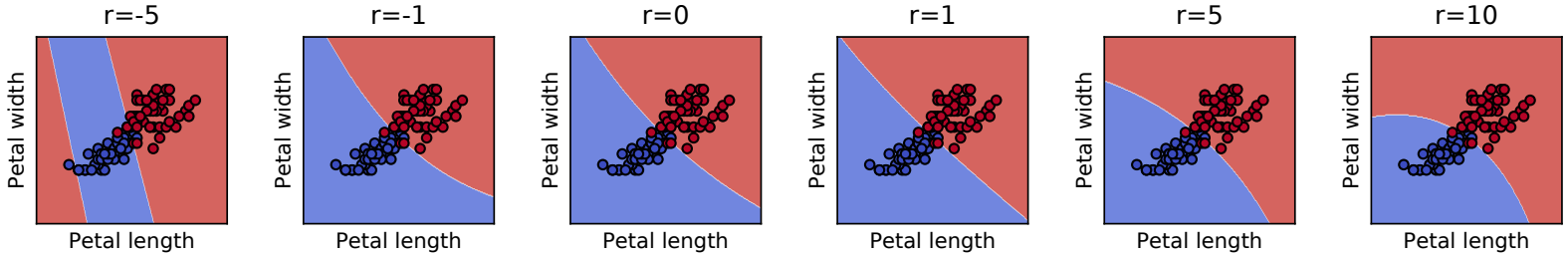
Predefined kernels (1)

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

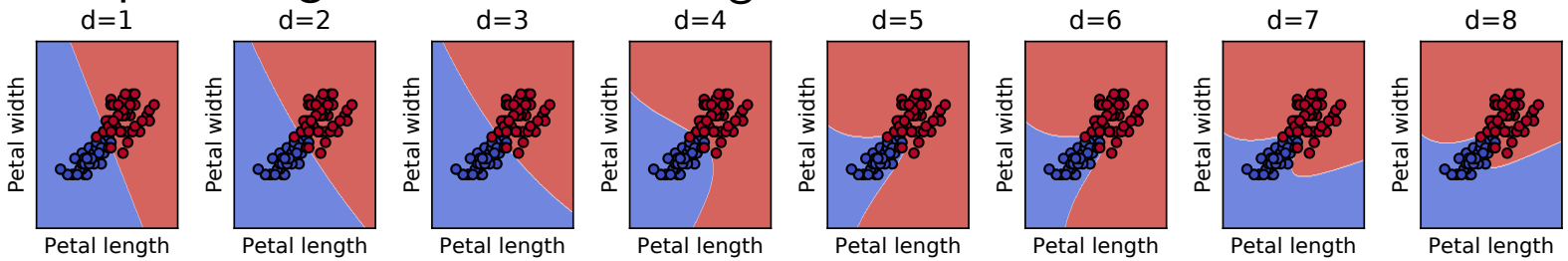


Predefined kernels (2)

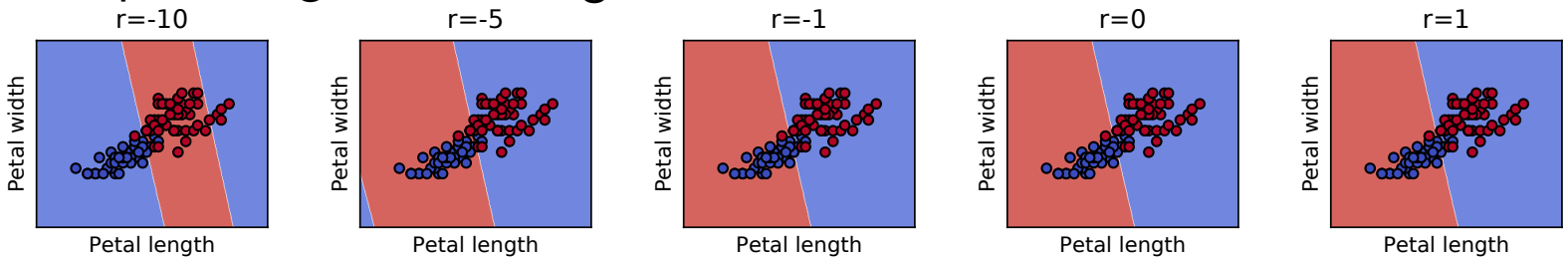
- 'polynomial' :
  - $(\gamma < x, x' > +r)^d$ (degree for d ; coef0 for r)
    - examples using degree 3, $C = 1$ and gamma = 'scale'



    - examples using $r = 0$, $C = 1$ and gamma = 'scale'



- 'sigmoid'
  - $tanh(\gamma < x, x' > +r)$
    - examples using $C = 1$ and gamma = 'scale'



Kernels in practice

- need to find the best kernel for your problem
- compare kernels with best parameters
  - C, γ, ...
- 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

Custom kernels

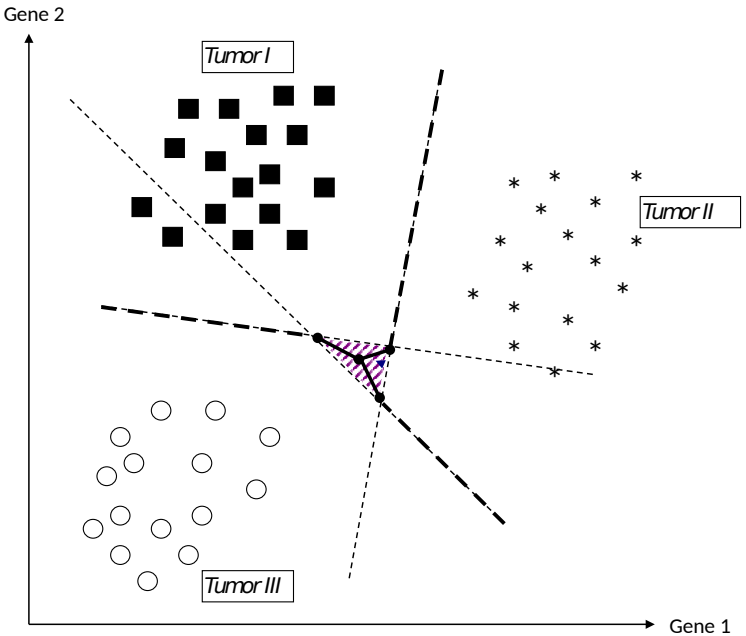
hand make kernel

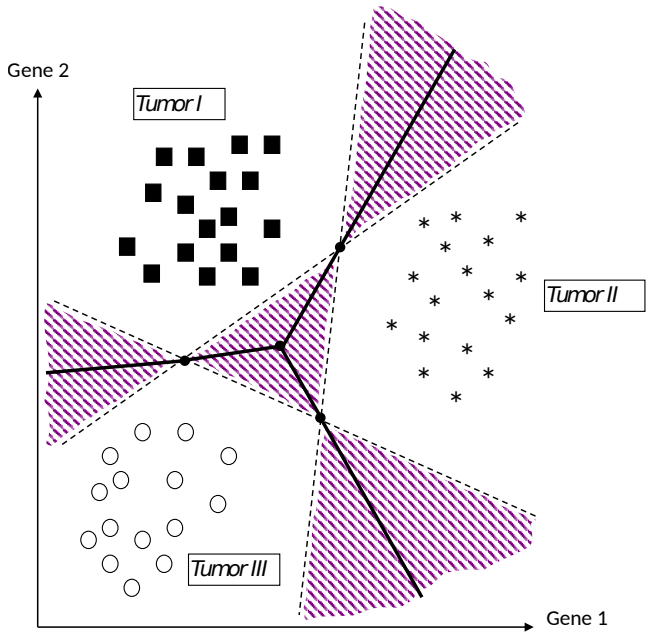
```
def my_kernel(X, Y):
    return np.dot(X, Y)
classif = svm.SVC(kernel=my_kernel)
```

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

Multiclass SVM

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



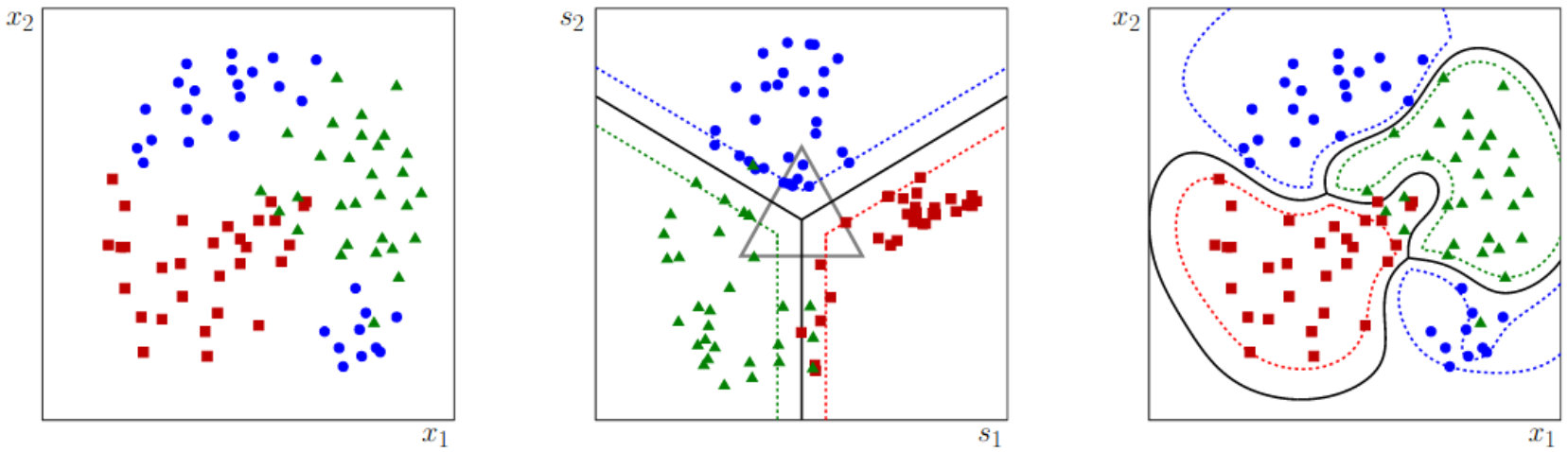


- one versus one (`'ovo'`)
  - build $n(n-1)/2$ SVM classifiers
  - maximum vote from classifiers
- one versus the rest (`'ovr'`)
  - build n SVM classifiers
  - unbalanced classes

Multicass SVM : GenSVM (2016)

- True multiclass (k classes) :
  - simplex encoding in dimension $k-1$
    - no region of ambiguity in the prediction space
  - loss function with different weights for classification errors (hinge loss)
    - convex loss function - IM (Iterative Majorization) algorithm for global minimum
  - take into account all kernels
  - warm re-starts
    - reduce computations for cross-validation, active learning ...
- references
  - paper : `https://jmlr.org/papers/volume17/14-526/14-526.pdf`
  - implementation : `https://gensvm.readthedocs.io/en/latest/cls_gensvm.html`

GenSVM (2)



(a) Input space

(b) Simplex space

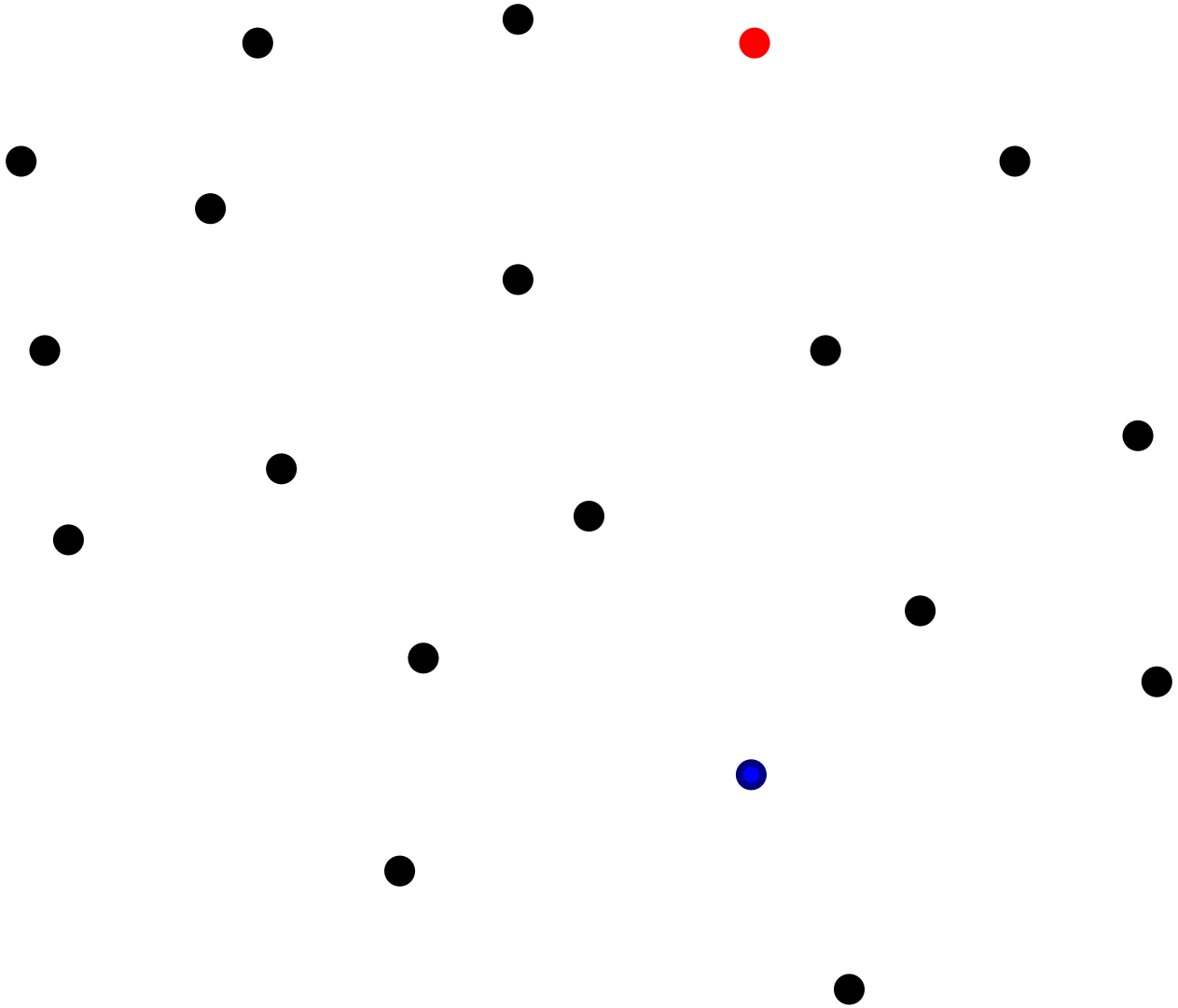
(c) Input space with boundaries

Figure 2: Illustration of GenSVM for a 2D data set with $K = 3$ classes. In (a) the original data is shown, with different symbols denoting different classes. Figure (b) shows the mapping of the data to the $(K-1)$-dimensional simplex space, after an additional RBF kernel mapping has been applied and the optimal solution has been determined. The decision boundaries in this space are fixed as the perpendicular bisectors of the faces of the simplex, which is shown as the gray triangle. Figure (c) shows the resulting boundaries mapped back to the original input space, as can be seen by comparing with (a). In Figures (b) and (c) the dashed lines show the margins of the SVM solution.

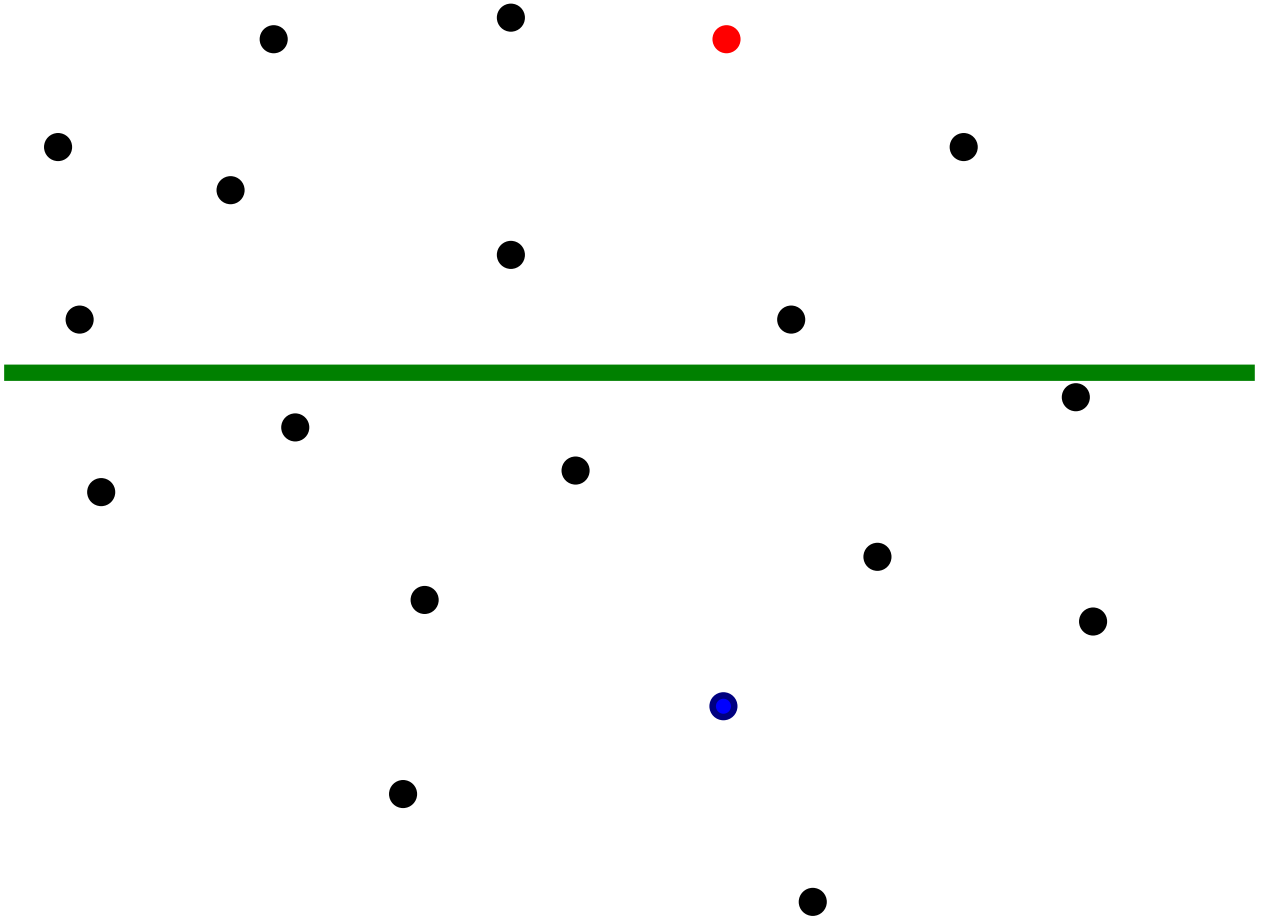
Active learning with SVM

- 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

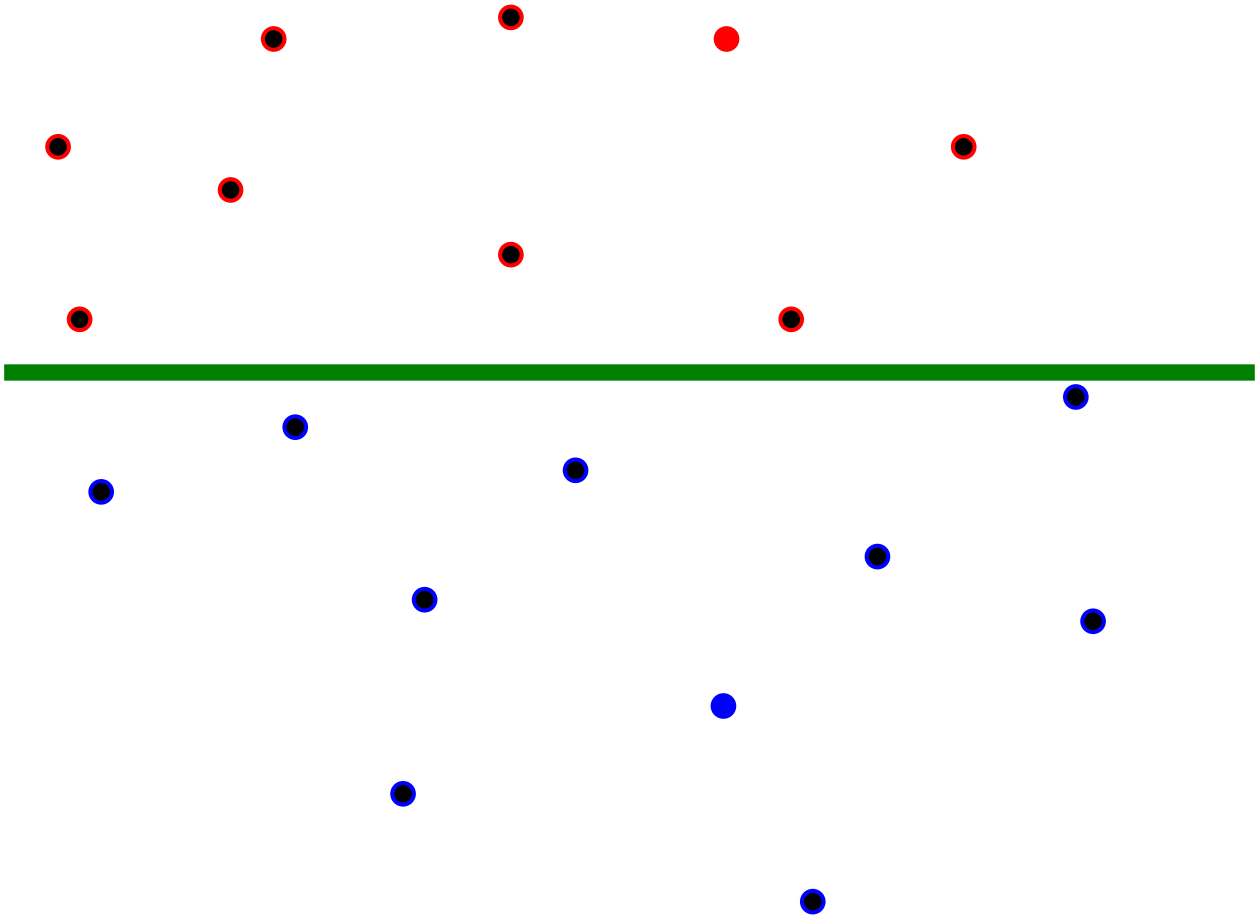
Active learning example : start with 2 annoted samples



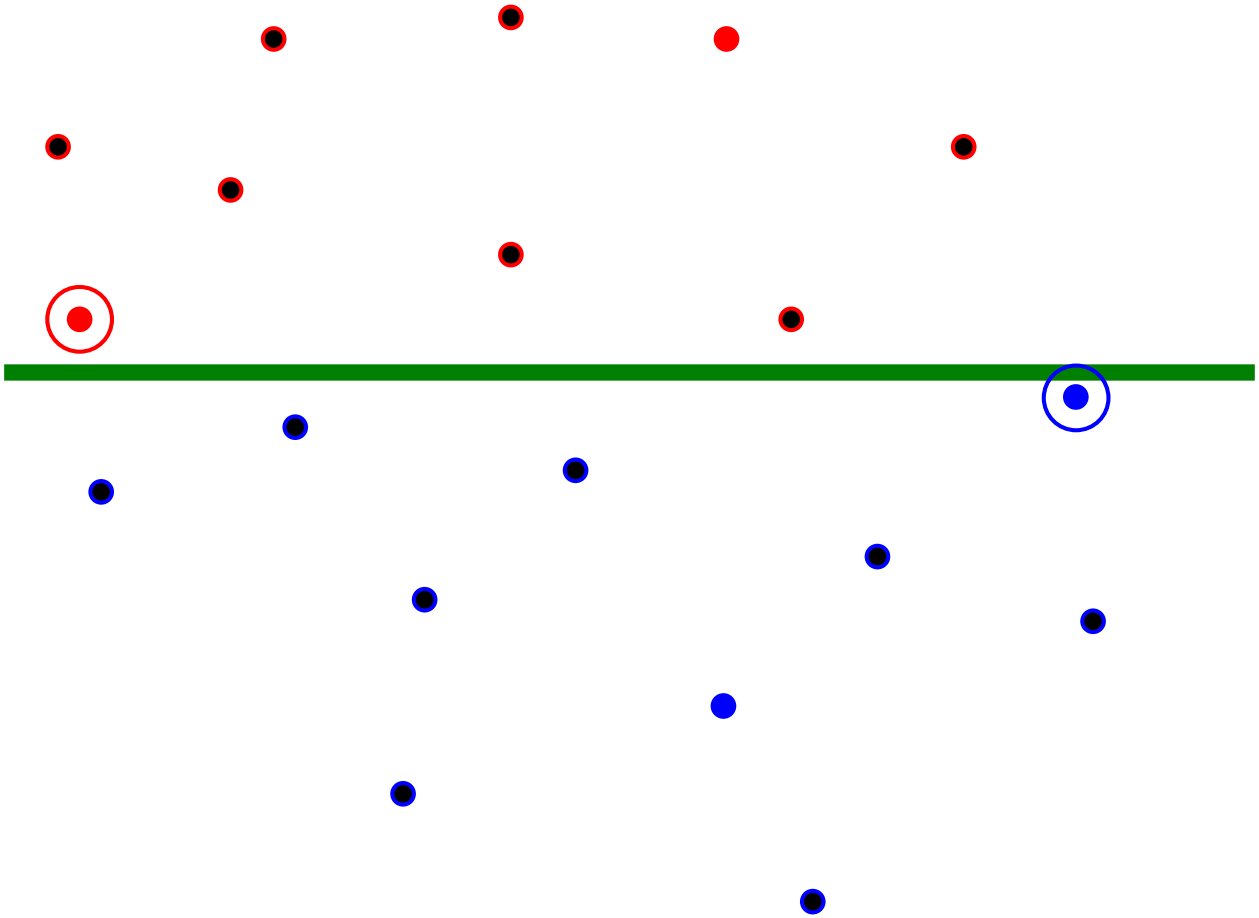
Active learning example : linear SVM classification



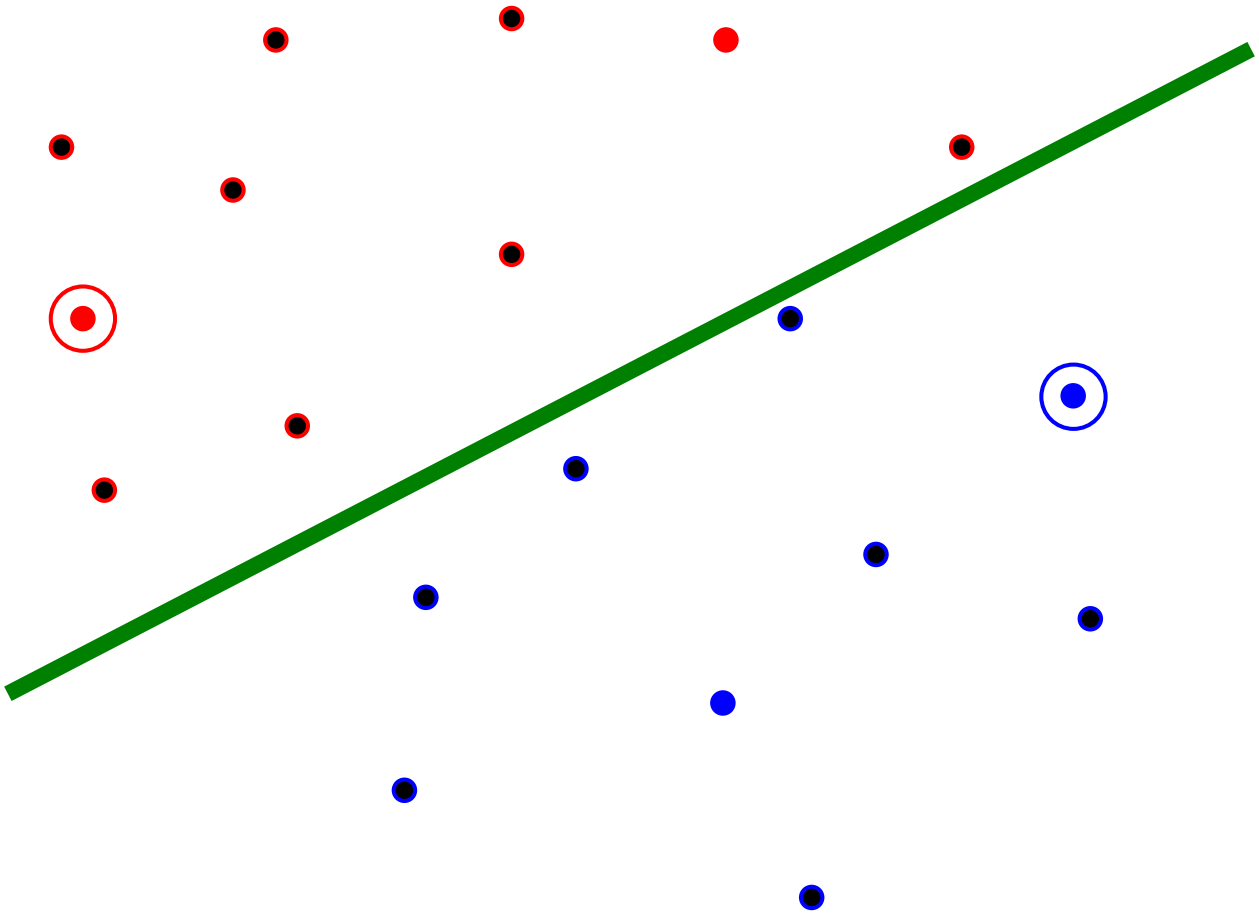
Active learning example : label all data



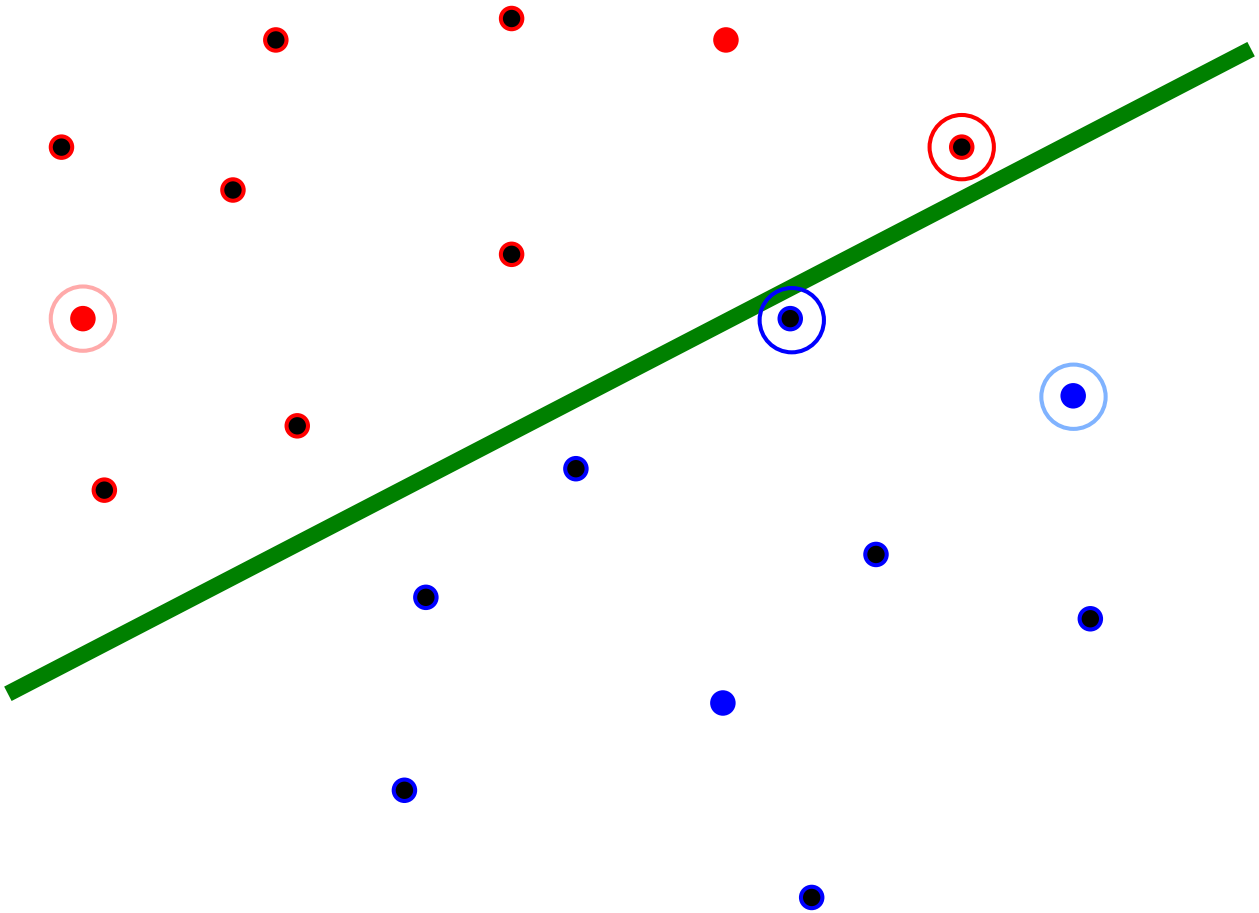
Active learning example : ask for labels of the 2 closest



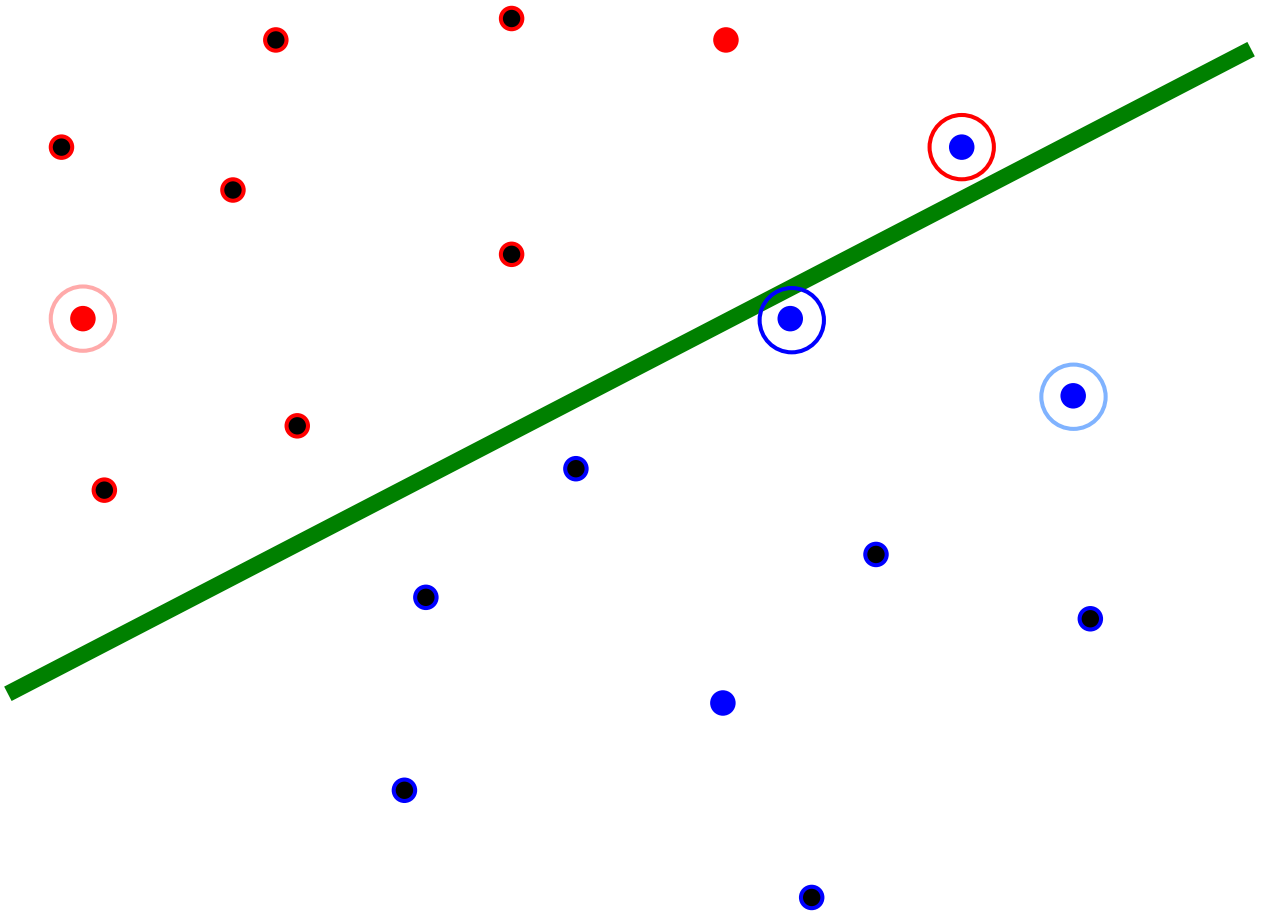
Active learning example : recompute linear SVM and labels



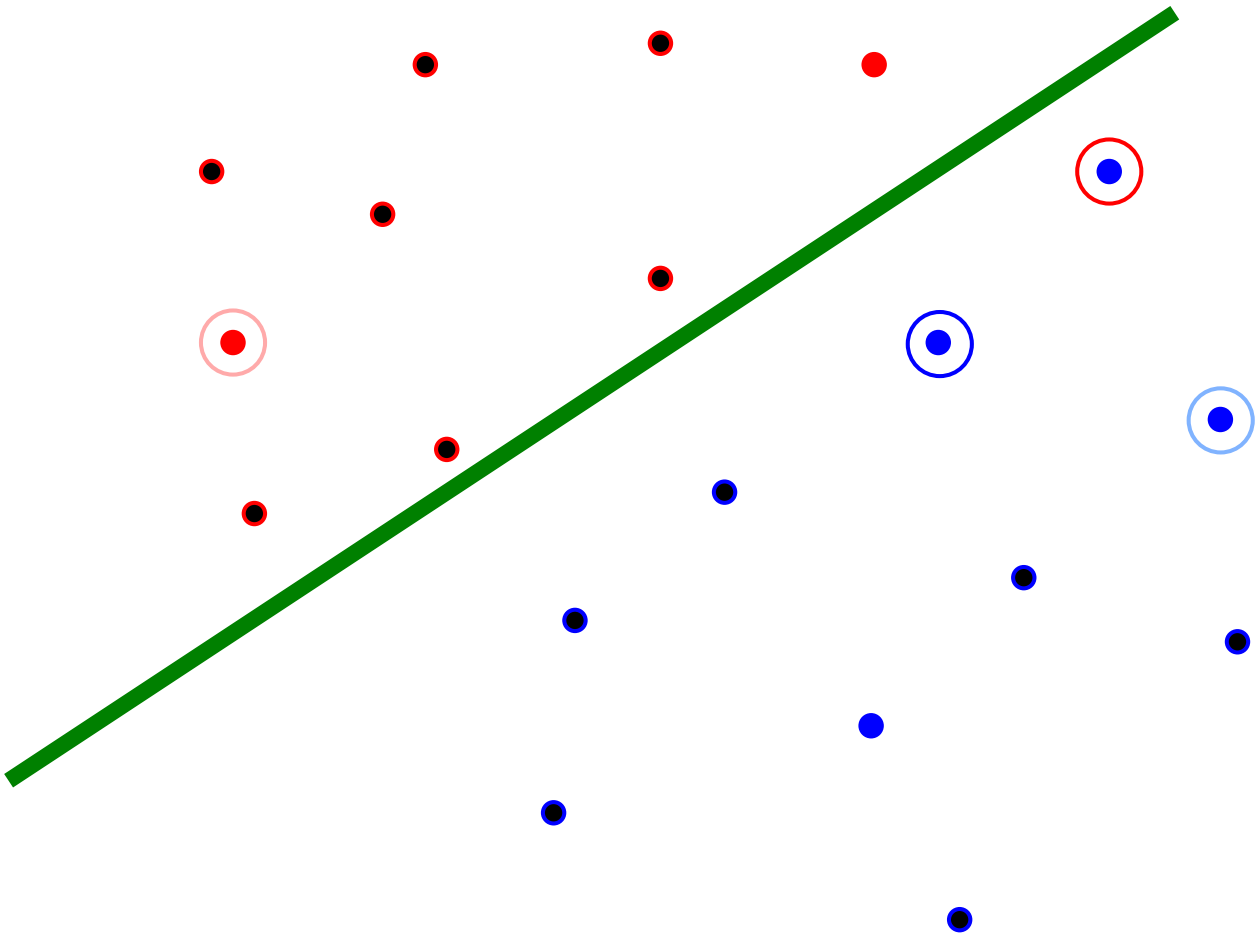
Active learning example : ask for labels of the 2 closest



Active learning example : correct one label

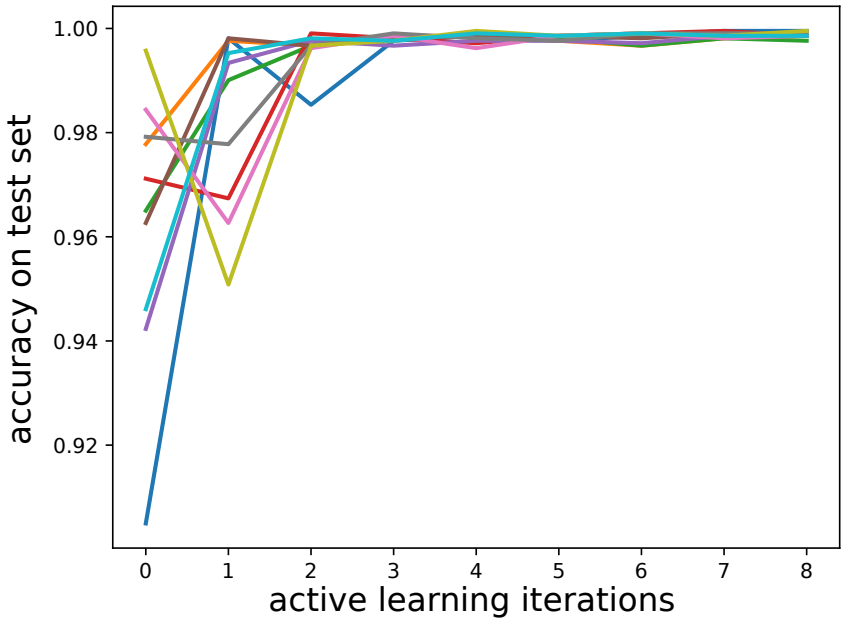


Active learning example : recompute linear SVM



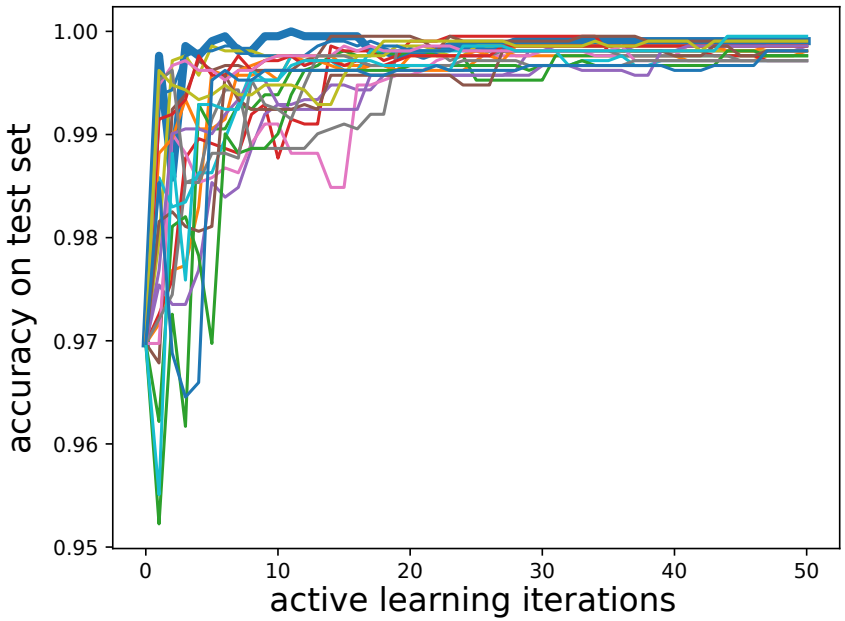
Active learning example : MNIST classes '0' and '1'

- 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

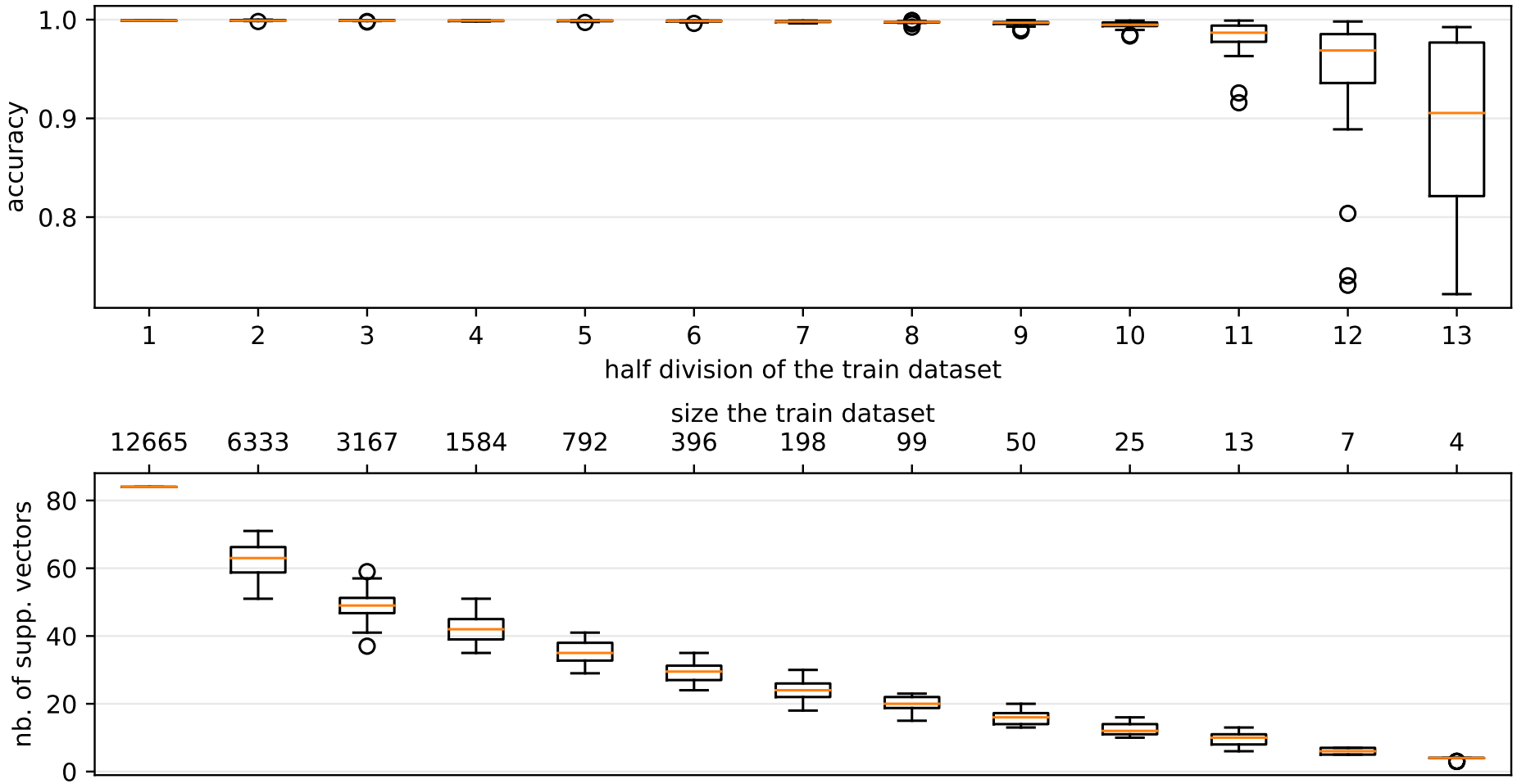
- SVMs are important because of theoretical reasons :
  - Robust to very large number of variables
  - Produce sparse models that are defined only by a small subset of training points ("support vectors")
  - Can learn both simple and highly complex classification models (by using the "kernel trick")
  - 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)

Support Vector Machines

- 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 networds, decision trees, ...)

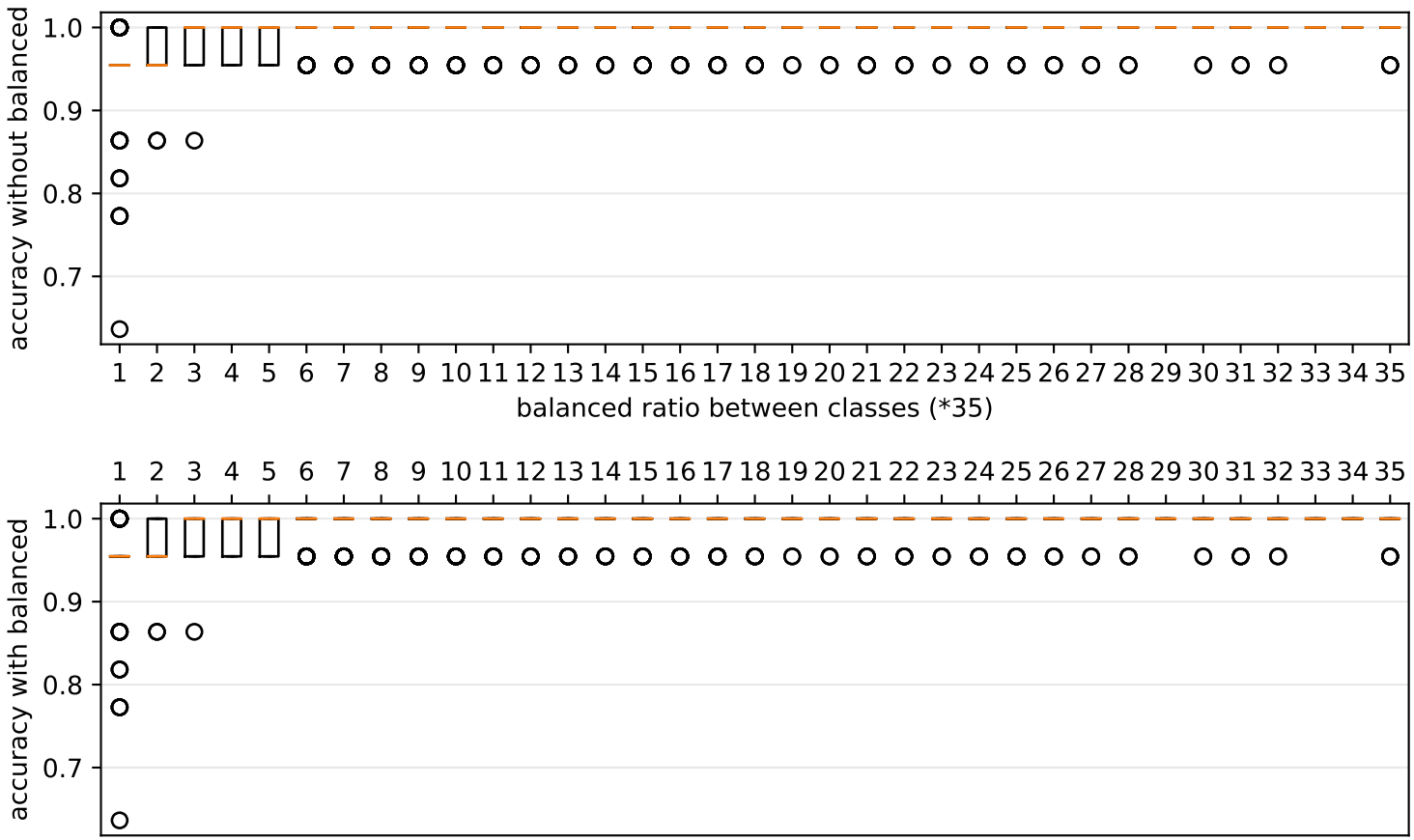
Experimenting the size reduction of train data

- 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)



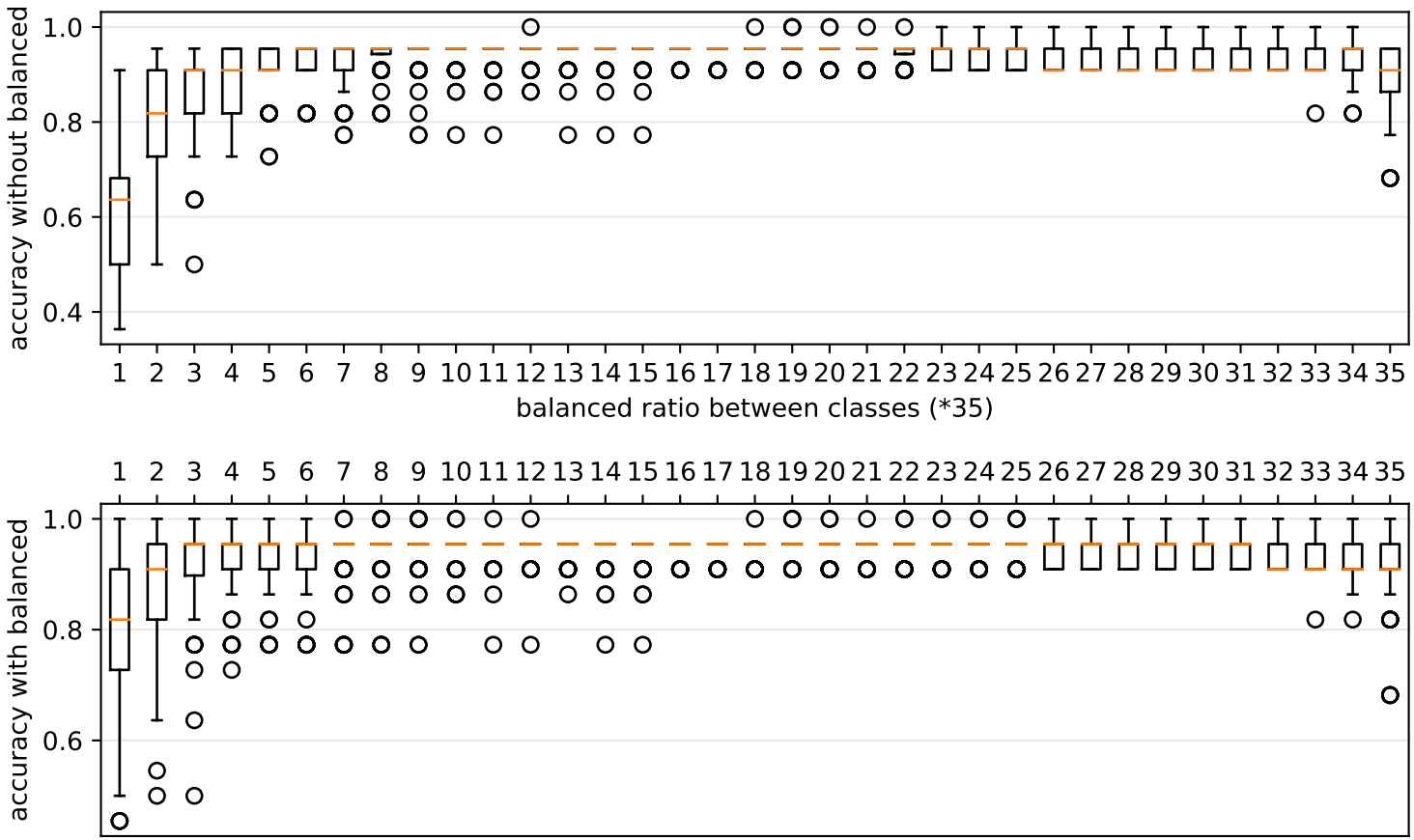
Unbalanced data

- 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)



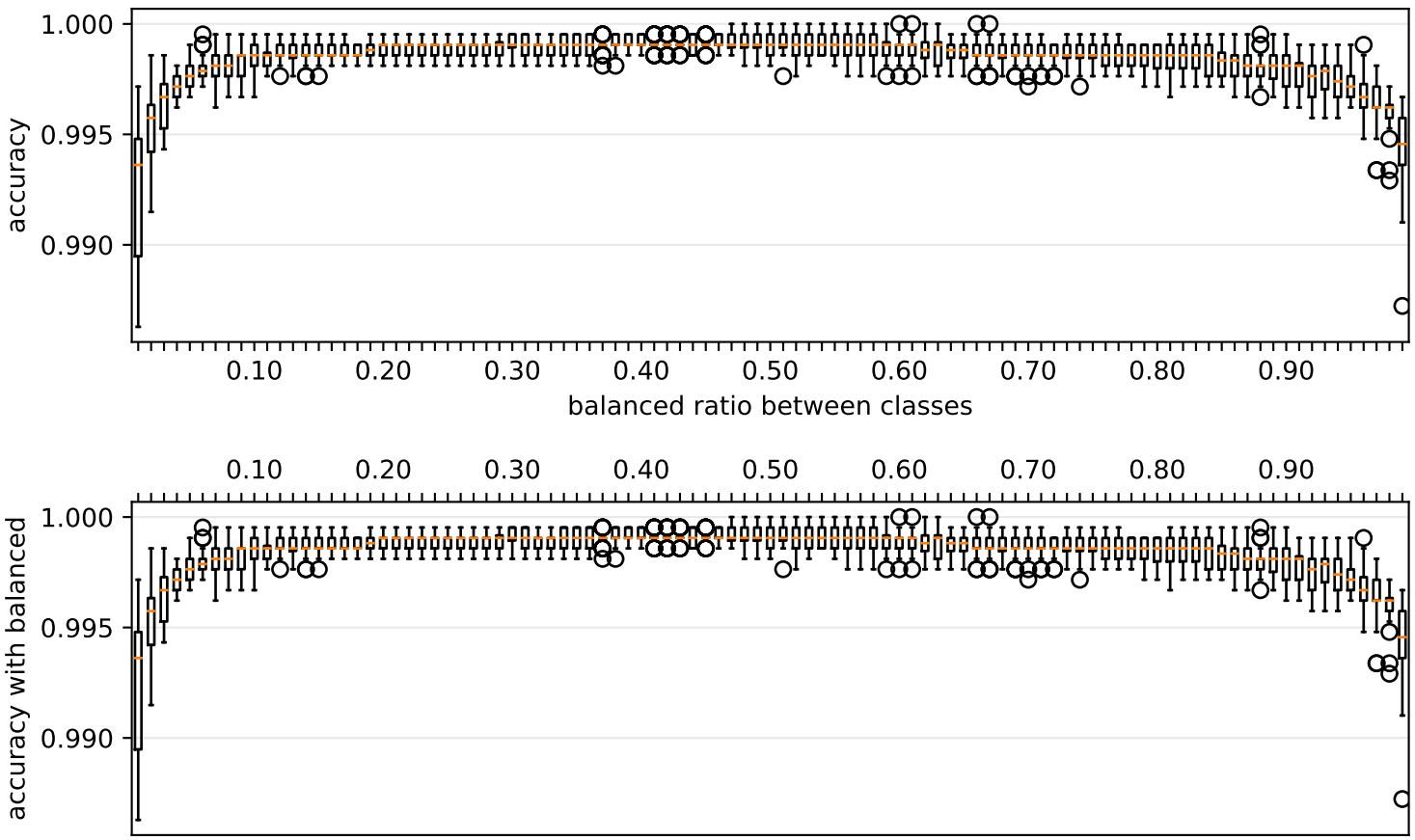
Unbalanced data

- 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)



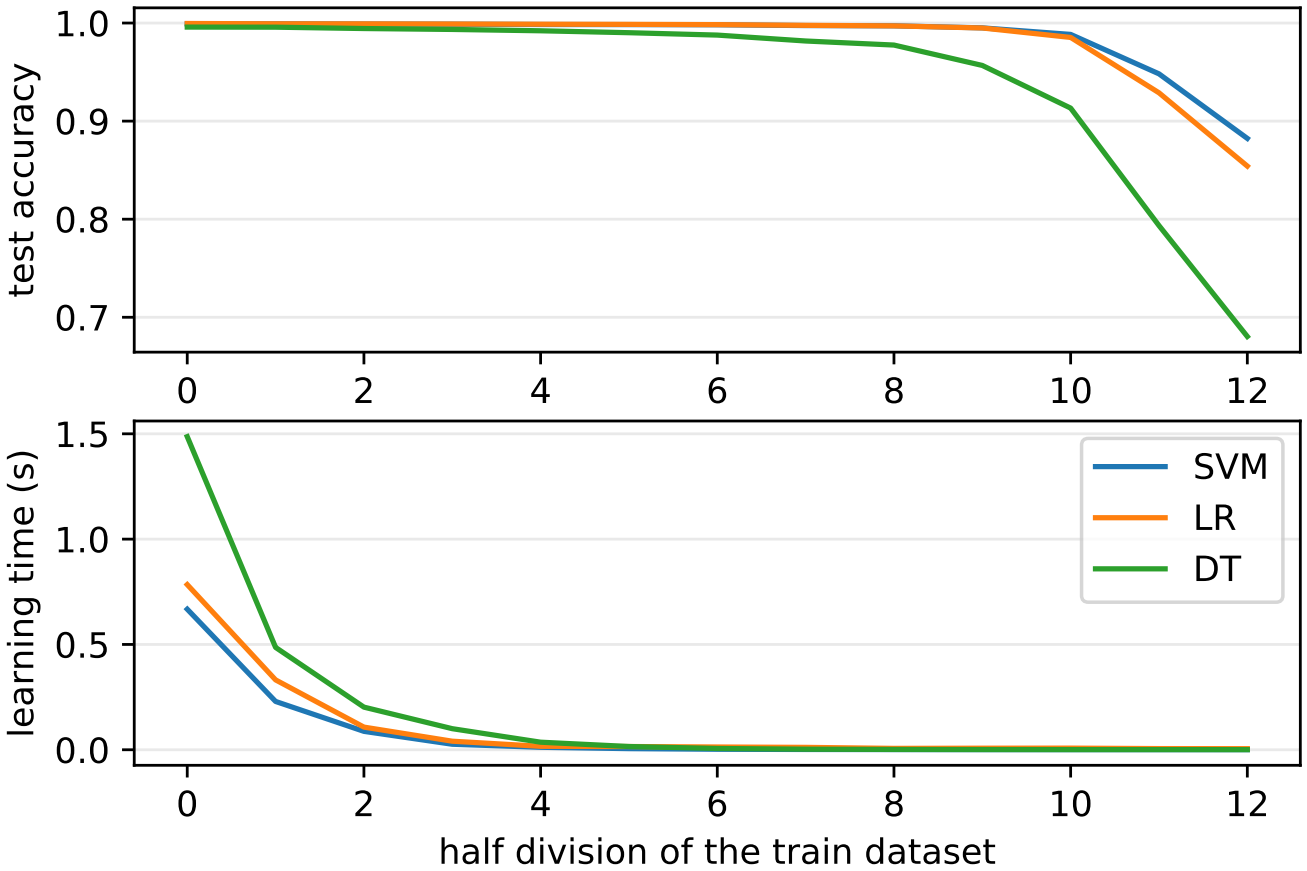
Unbalanced data

- 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)



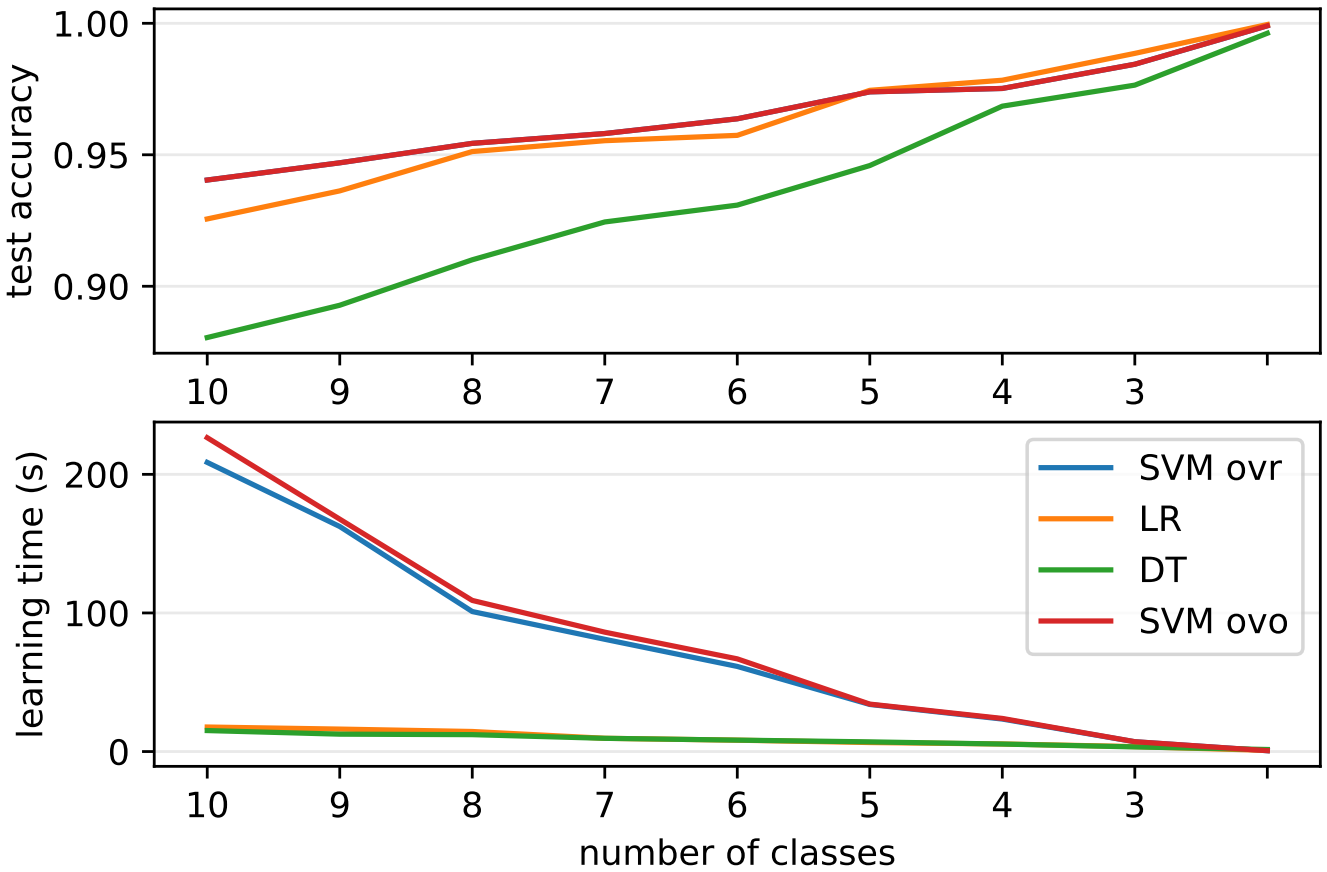
Comparison SVM/LR/DT

- MNIST data set (2 classes : '0' and '1', 12665 data, 784 features)
- linear SVM, LR and DT using default parameters
- Recursive divisions of the train dataset
- 24 trials for each set of parameters (data shuffled)



Comparison SVM/LR/DT

- MNIST data set :
  - at each step, the last class is removed
- linear SVM, LR and DT using default parameters



Practice

- dataset Iris :
  - binary classification :
    - train/test split
    - learning a SVM classification (start with linear kernel)
    - evaluate the performances and compare to previous method
    - try other hyper-parameters (C. . .) or other kernel (GridSearchCV)
  - 3 classes classification :
    - compare ovo and ovr options
- dataset digit
  - same questions
- dataset MNIST, fMNIST
  - same questions
  - don't forget to look carefully at the confusion matrix !
- Help : `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`

Outline

What else ?

- Regression SVM
- One class SVM
- Features reduction using SVM
- Probabilistic SVM
- SVM kernels for neural network layers