

SVM: Support Vector Machine

Diane Lingrand and many contributors



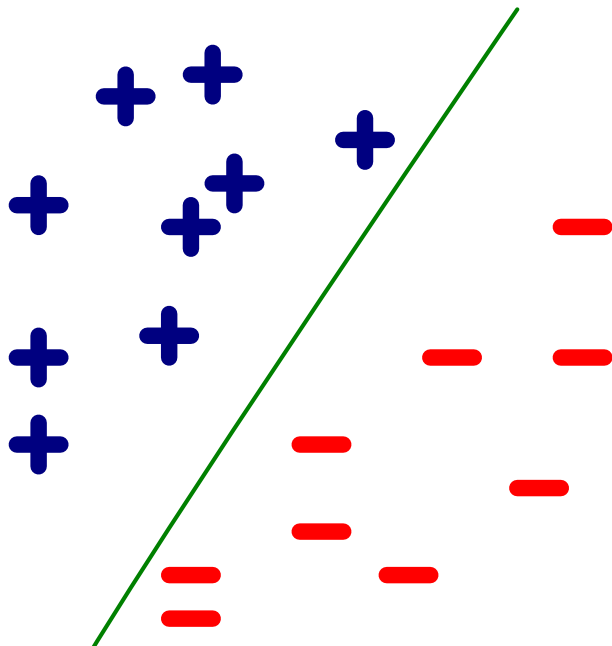
Membre de UNIVERSITÉ CÔTE D'AZUR

SI 4

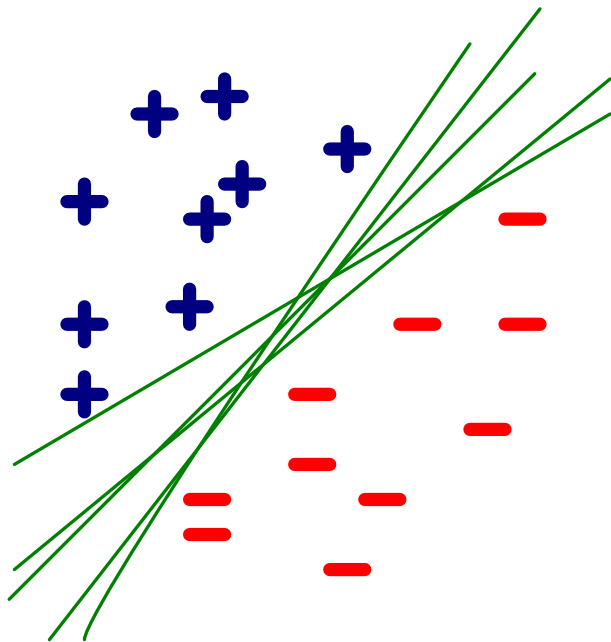
2019 - 2020

- 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

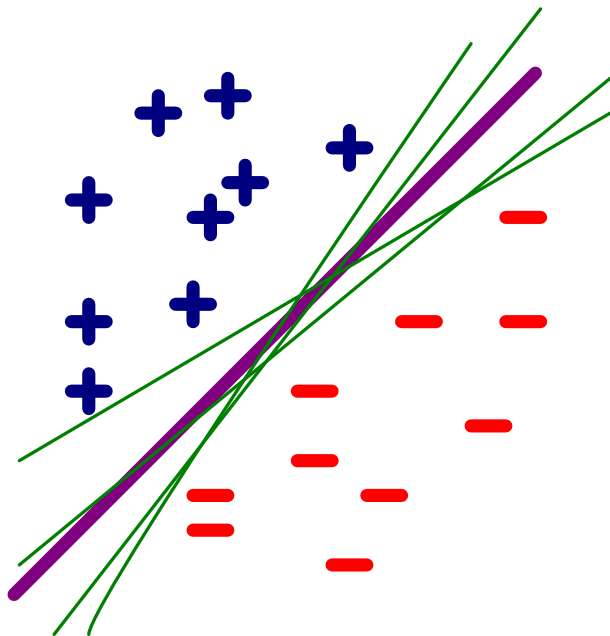
Principle : linear separation



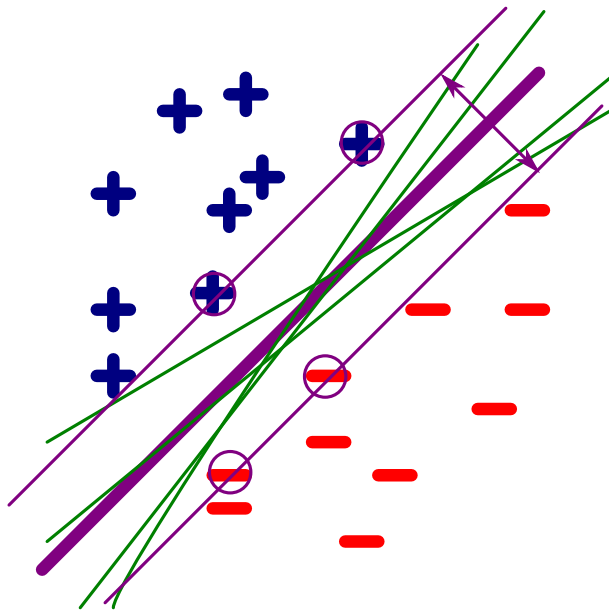
Principle : many solution

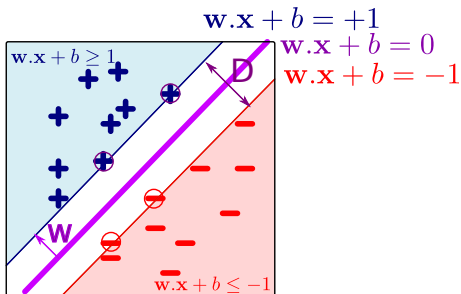


Principle : find the best one



Principle : maximising the margin





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

SVM problem :

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

Prediction : $\text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$

- 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 $\sum_i \alpha_i y_i = 0$

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

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

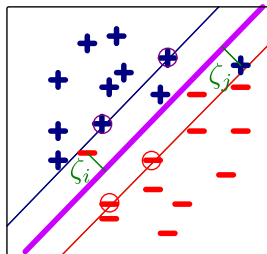
there exists an optimal solution

- 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} : $\text{sign}(\mathbf{w} \cdot \mathbf{x} + b) = \text{sign}(\sum_{i, \alpha_i > 0} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x})$

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



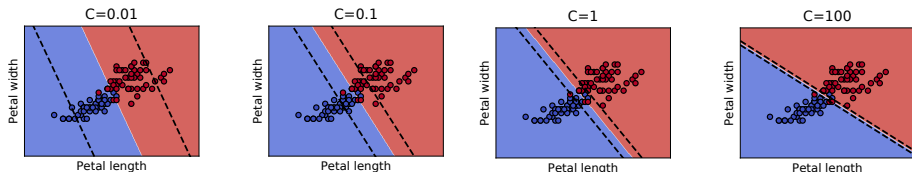
Soft margin

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 $\sum_i \alpha_i y_i = 0$

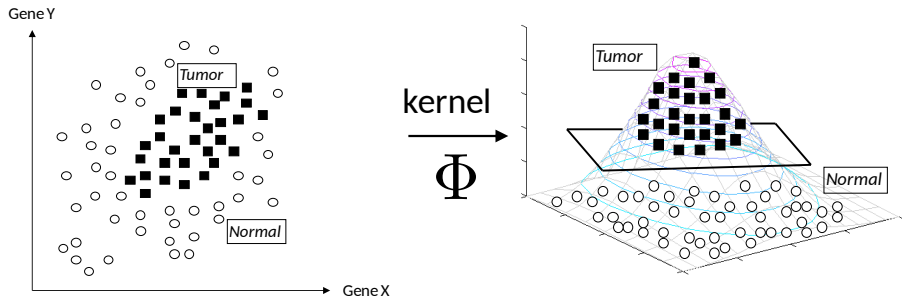
Soft margin : parameter C



- 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 $C * \text{value}$.
 - 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 $C * \text{sample_weight}[i]$.

Non linearly separable data : the kernel trick



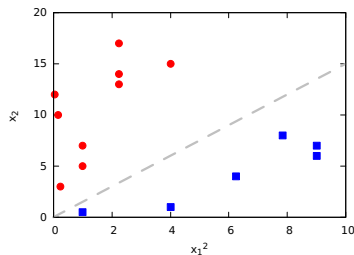
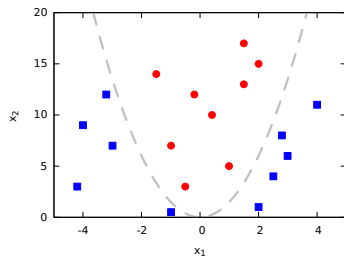
Data is not linearly separable
in the input space

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

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

- Here, we define Φ explicitly

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



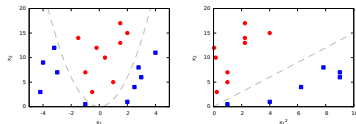
- SVM solution in the induced space (feature space) :

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

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

$$\mathbf{K}(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$$

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



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

Dual formulation

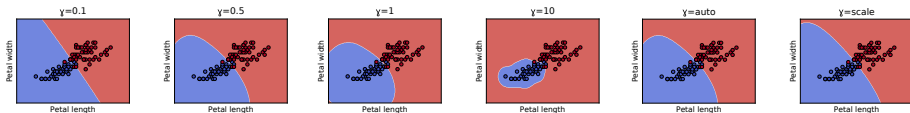
maximisation of $\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_k \alpha_i \alpha_k y_i y_k K(\mathbf{x}_i, \mathbf{x}_k)$ under the constraints
 $\forall i \ C \geq \alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$

Prediction

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

Predefined kernels (1)

- 'linear'
- 'rbf' (radial basis function) :
 - $\exp(-\gamma\|x - x'\|^2)$ (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 $C = 1$:

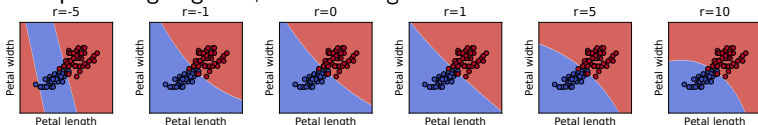


Predefined kernels (2)

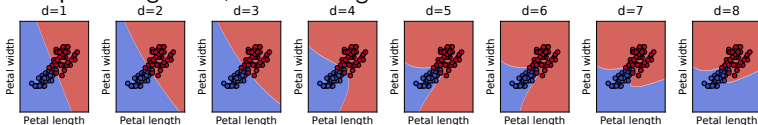
- 'polynomial' :

- $(\gamma < x, x' > + r)^d$ (degree for d ; coef0 for r)

- examples using degree 3, $C = 1$ and gamma = 'scale'



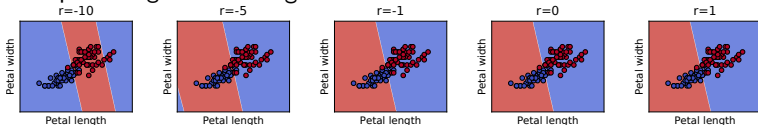
- examples using $r = 0$, $C = 1$ and gamma = 'scale'



- 'sigmoid'

- $\tanh(\gamma < x, x' > + r)$

- examples using $C = 1$ and gamma = 'scale'



For text, genes,

hand make kernel

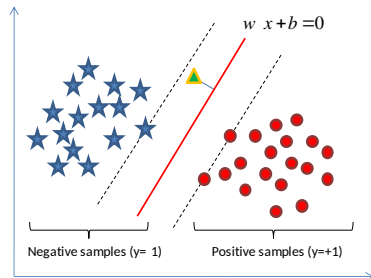
```
def my_kernel(X, Y):  
    return np.dot(X, Y)  
classif = svm.SVC(kernel=my_kernel)
```

See https://www.tensorflow.org/tutorials/representation/kernel_methods for examples.

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

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

- 1 SVMs output a class label (positive or negative) for each sample $\text{sign}(w \cdot x + b)$
- 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.



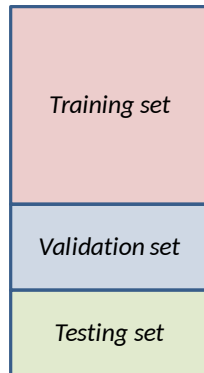
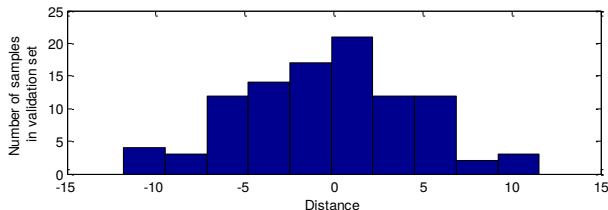
- Question : How can one use SVMs to estimate posterior class probabilities, i.e. $P(\text{positive class} \mid \text{sample})$?

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

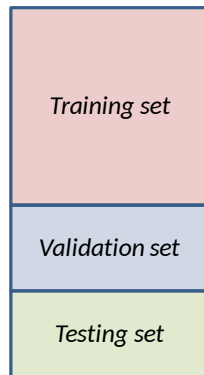
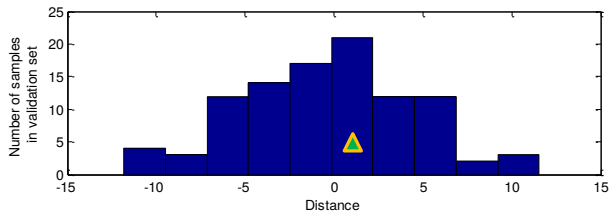
3. Create a histogram with Q (e.g., say 10) bins using the above distances. Each bin has an upper and lower value in terms of distance.



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 $[0, 2.5]$



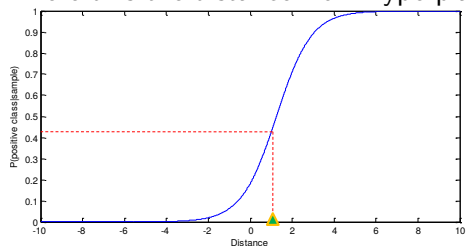
- Compute probability $P(\text{positive class} \mid \text{sample \#382})$ as a fraction of true positives in this bin.

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

- 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

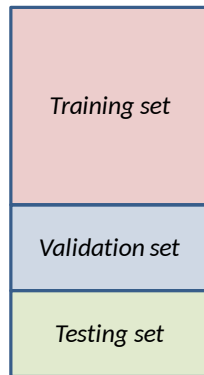


Platt's method

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

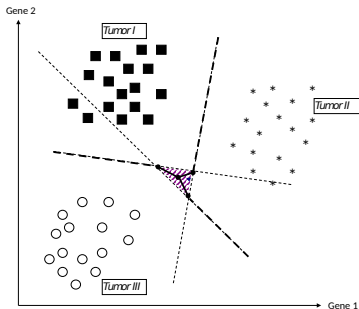
Sample #	1	2	3	4	5		98	99	100
Distance	2	1	8	3	4	...	2	0.3	0.8

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.

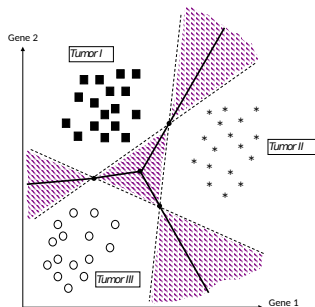


Multiclass SVM

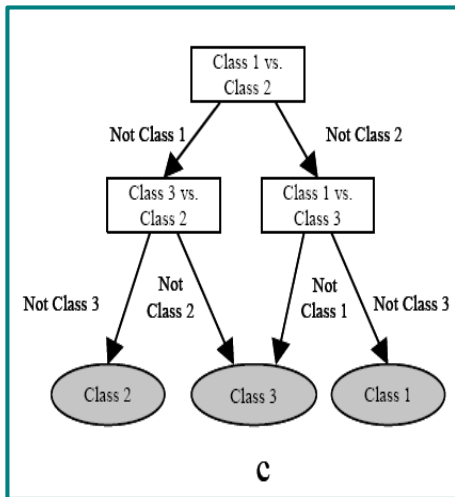
- n classes
- two methods (`decision_function_shape(...)`)



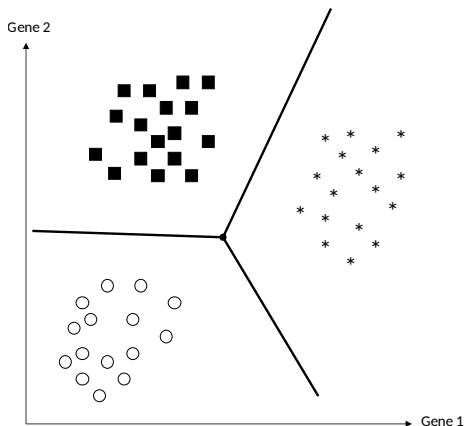
- one versus one ('ovo')
 - build $n(n-1)/2$ SVM classifiers
 - maximum vote from classifiers



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



- 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

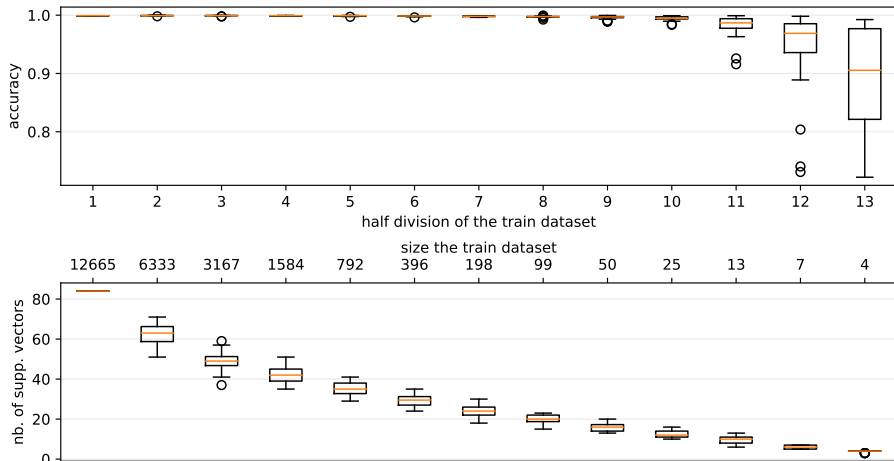
- hyperplane estimation using small set of labelled data
- estimation of distance for non-labelled data
- selection of data close to the hyperplane
 - ask the user for label
- refine the estimation

- 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, ...)

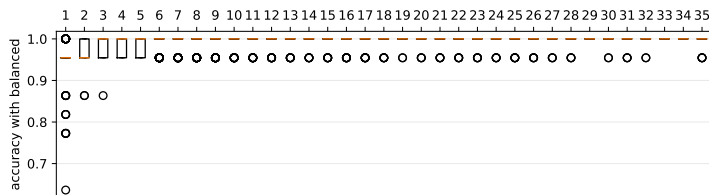
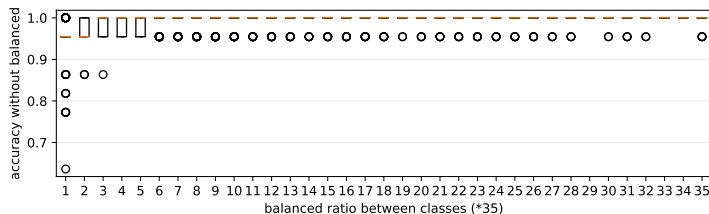
Experimenting the size reduction of train data

- 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)



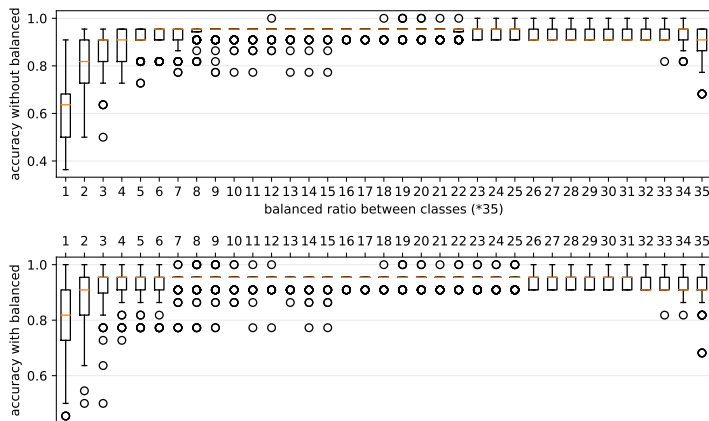
Unbalanced data

- 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)

