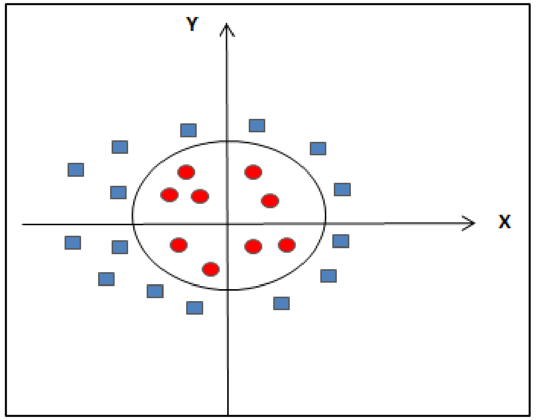
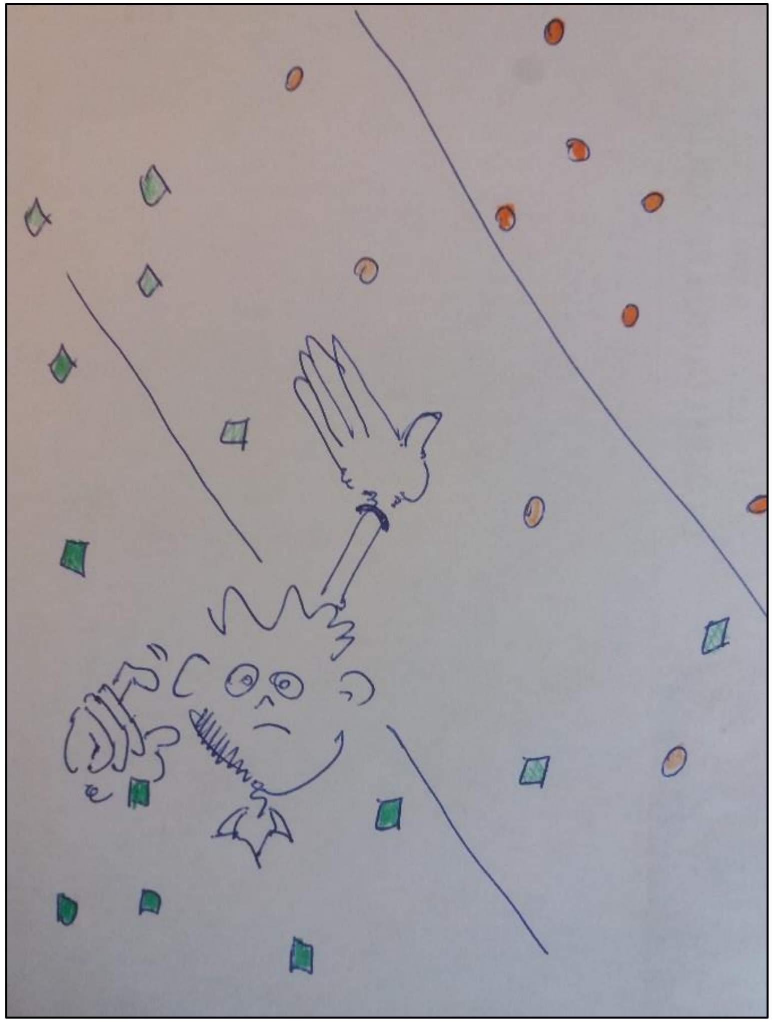
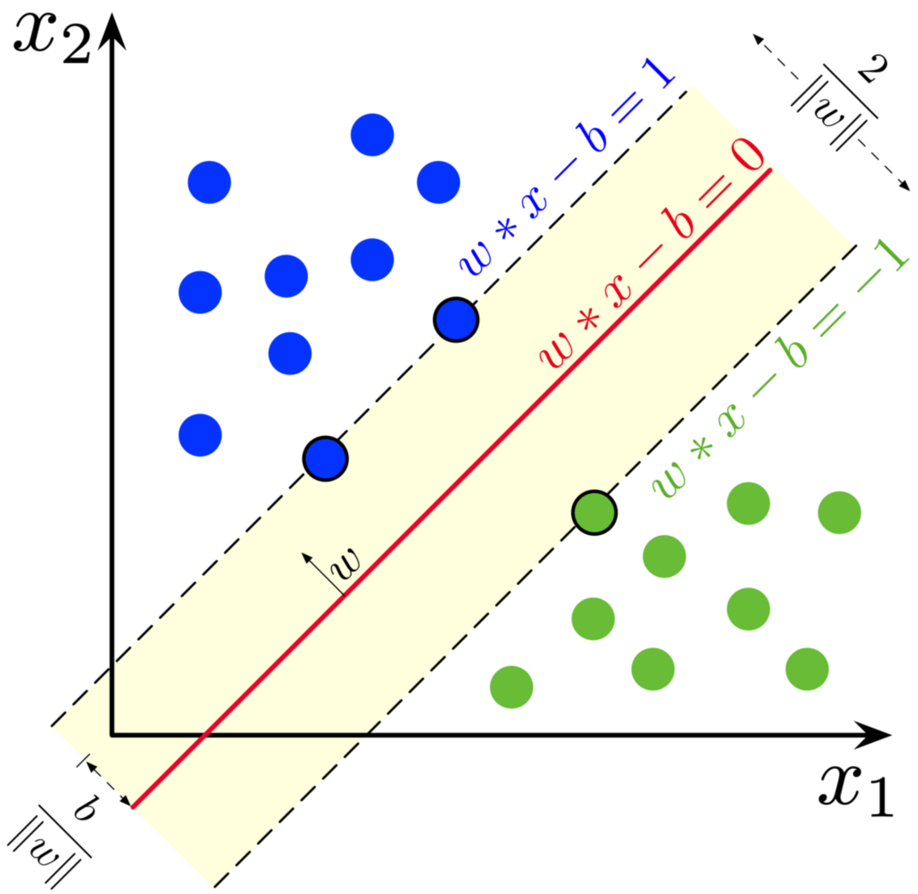
Large margins and SVMs







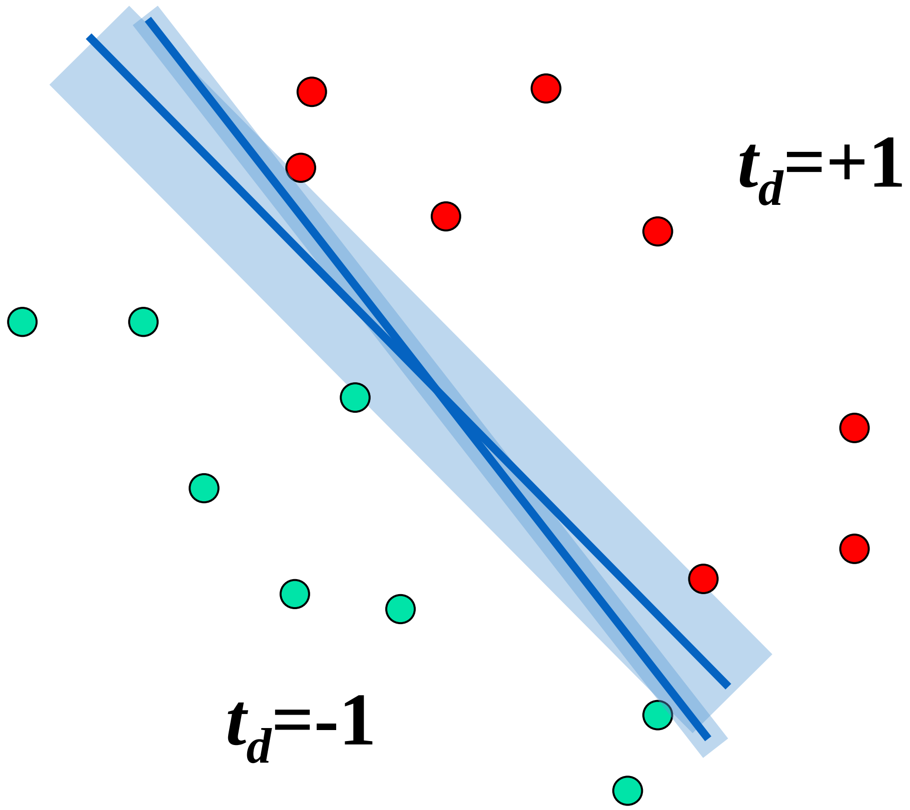
Ariel Shamir
Zohar Yakhini

IDC HERZLIYA

In previous chapters ….

- Perceptron decision boundaries are linear
- To obtain linear separability, we can map instance space, in a non linear manner, into higher dimensional space
- Cover’s Thm: in higher dimensions, a higher fraction of dichotomies of K points are linearly separable
- We do not actually need to map to high-dim. We can use a kernel to actually learn a high dim linear decision boundary.
- We demonstrated how learning a perceptron can be cast as a task of learning weights for training instances: the dual perceptron and the kernel perceptron

Seeking large margin classifiers


t_d=+1
t_d=-1

Support Vector Machines

- Input: Data and a kernel function that (in effect) non-linearly maps to high dimensional space + related hyper parameters (kernel hyper parameters and slack coefficients)
- Output:
  - A subset of the training examples called "support vectors", denoted SV
  - A set of weights for these examples

- We classify +/- according to a simple "instance based" rule:

$$class(\vec{x}) = \operatorname{sgn}\left(\sum_{d \in SV} a_d\, t_d K(\vec{x}_d, \vec{x})\right)$$

Optimizing the Margin

- The margin of a separating surface $f(x)$=0 is defined as the minimum distance of an instance to the surface over all training samples:

$$\text{Margin} \equiv \min_{d \in D} dist(x_d, \mathrm{f}(x) = 0)$$

- Our goal, in SVM classification and learning, is to maximize that minimum distance
  - More stable
  - Less susceptible to noise
  - Less sensitive
  - Better generalization

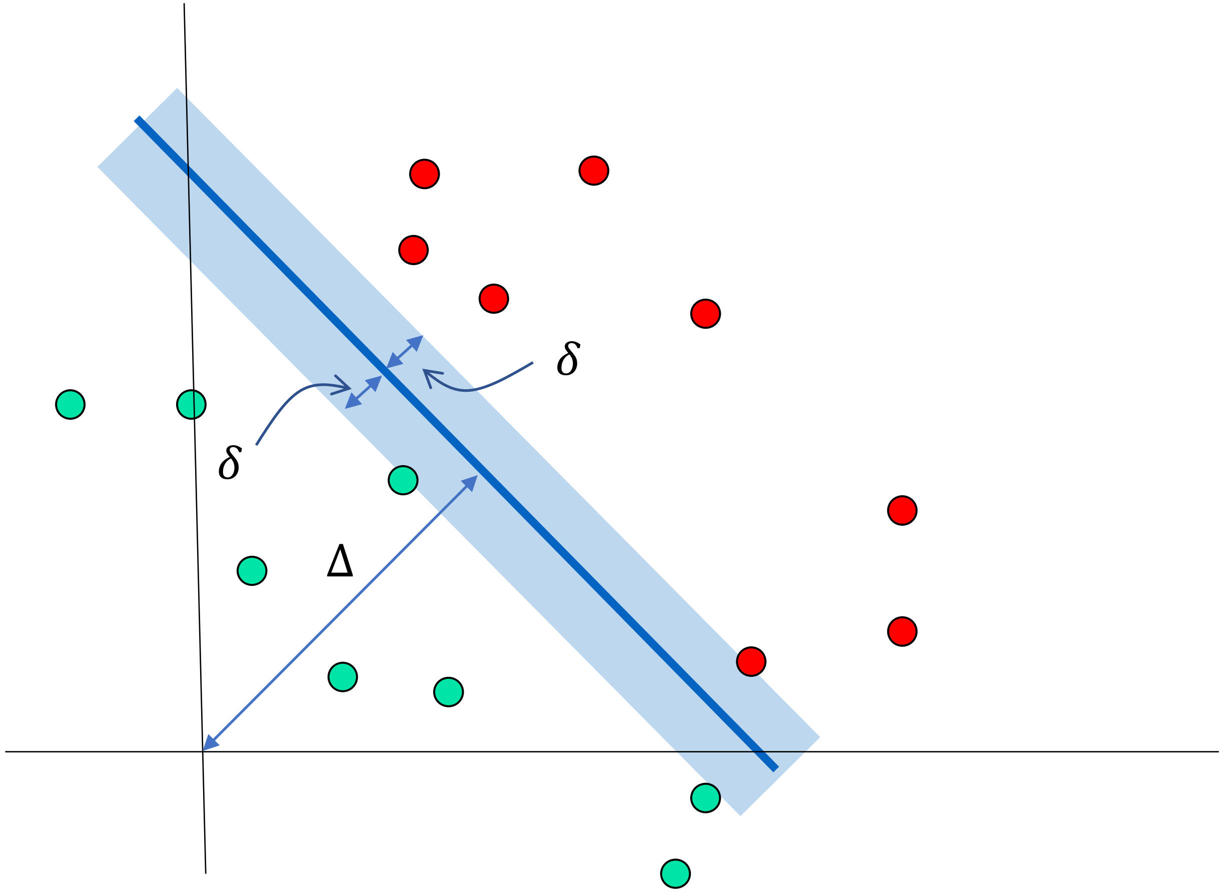
SVM Geometry

$$U_+ = \{\vec{x}: \vec{x} \cdot \vec{w} - (b + \delta\|\vec{w}\|) = 0\}$$

$$U = \{\vec{x}: \vec{x} \cdot \vec{w} - b = 0\}$$

$$U_- = \{\vec{x}: \vec{x} \cdot \vec{w} - (b - \delta\|\vec{w}\|) = 0\}$$

$$\Delta = \frac{b}{\|\vec{w}\|}$$



SVM Optimization

$$U_+ = \{\vec{x}:\ \vec{x}\cdot\vec{w} - (b + \delta\|\vec{w}\|) = 0\}$$

$$U = \{\vec{x}:\ \vec{x}\cdot\vec{w} - b = 0\}$$

$$U_- = \{\vec{x}:\ \vec{x}\cdot\vec{w} - (b - \delta\|\vec{w}\|) = 0\}$$

$$\Delta = \frac{b}{\|\vec{w}\|}$$

We want to find

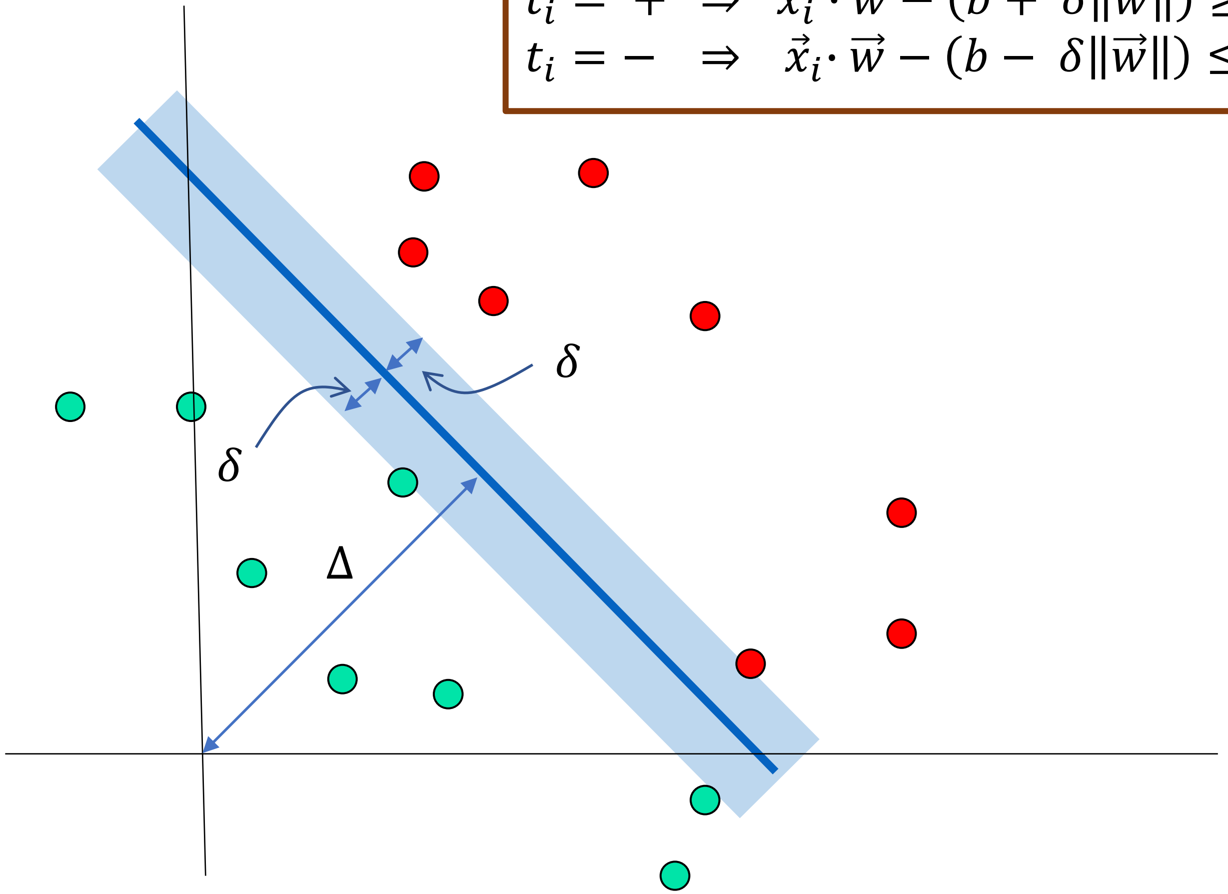
$$\max_{w,b,\delta} \delta$$

under the constraints:

$$\forall i$$

$$t_i = + \ \Rightarrow\ \vec{x}_i\cdot\vec{w} - (b + \delta\|\vec{w}\|) \geq 0$$

$$t_i = - \ \Rightarrow\ \vec{x}_i\cdot\vec{w} - (b - \delta\|\vec{w}\|) \leq 0$$



SVM Optimization -cont

Now notice that if we have a triplet $\vec{w}, b, \delta$ that satisfies the constraints system then we can define:

$\vec{u} = \frac{\vec{w}}{\delta\|\vec{w}\|}$ and $c = \frac{b}{\delta\|\vec{w}\|}$

And then the triplet $\vec{u}, c, \delta$ also satisfies the constrained system

$$\max_{w,b,\delta} \delta$$

under the constraints:

$\|\vec{w}\| = \frac{1}{\delta}$,

$\forall i$

$t_i = + \quad \Rightarrow \vec{x}_i \cdot \vec{w} - (b+1) \geq 0$

$t_i = - \quad \Rightarrow \vec{x}_i \cdot \vec{w} - (b-1) \leq 0$

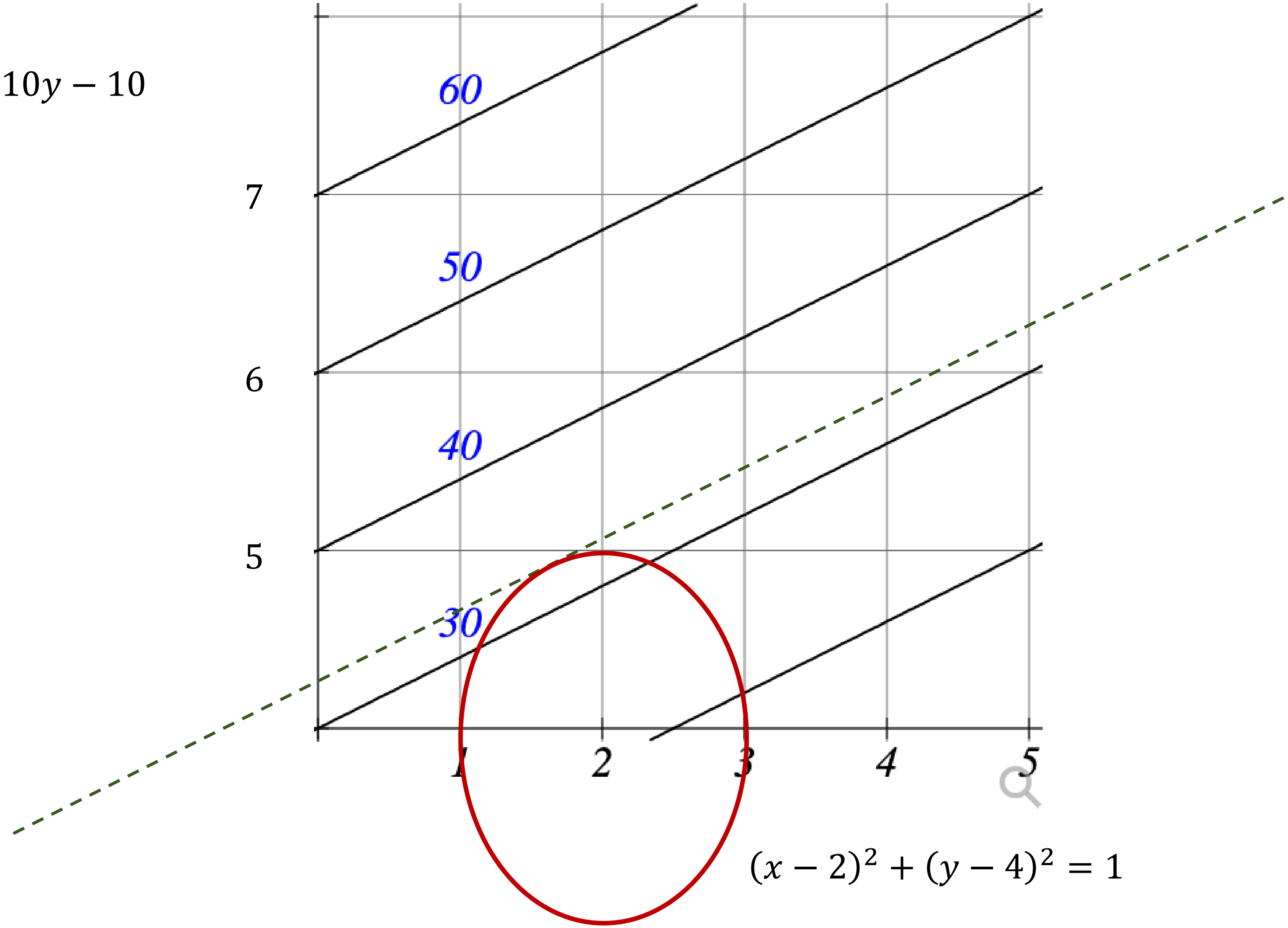
SVM Optimization – final

$$\min_{\vec{w},b} ||\vec{w}||^2$$

under the constraints:

$$\forall i \qquad t_i(\vec{x}_i \cdot \vec{w} \ - b) - 1 \geq 0$$

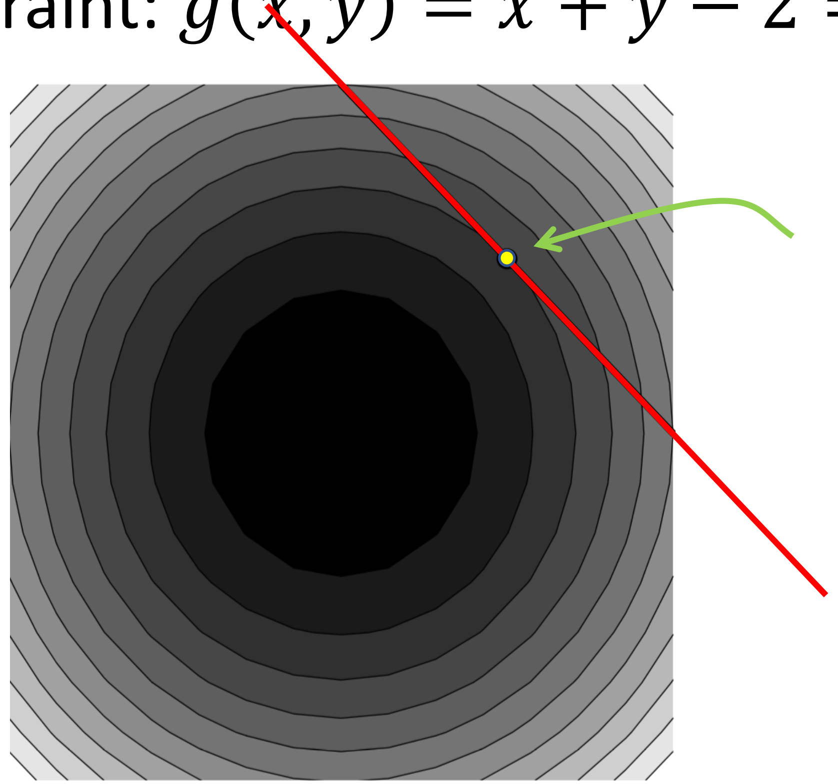
$f(x, y) = -4x + 10y - 10$



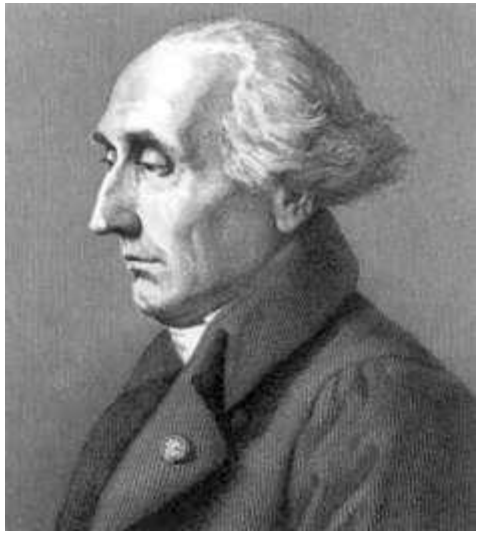
$(x-2)^2 + (y-4)^2 = 1$

Another Simple Example

- Miminize $f(x, y) = x^2 + y^2$
- Subject to the constraint: $g(x, y) = x + y - 2 = 0$



Lagrange Multipliers



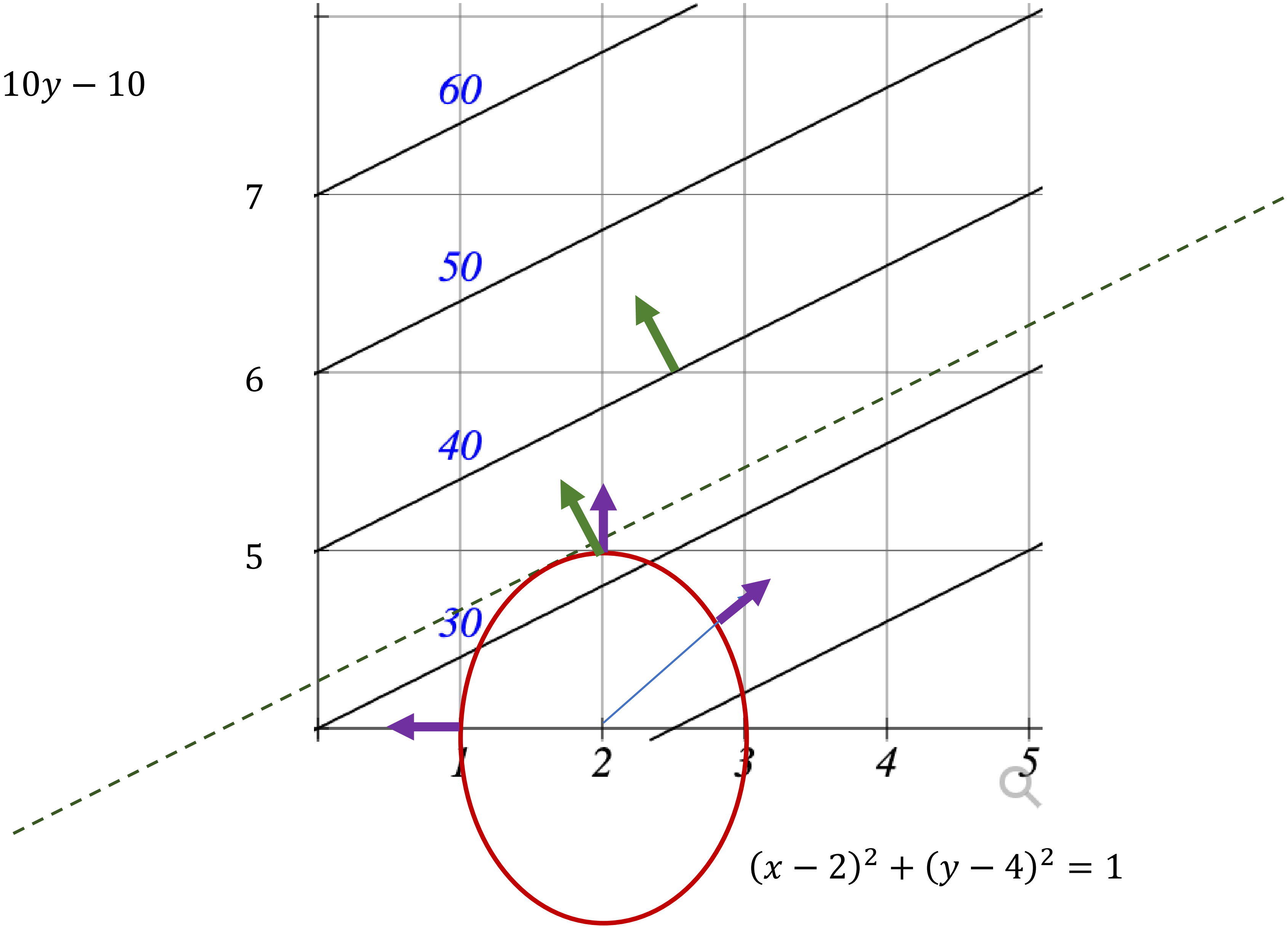
J.L. Lagrange
France
18th Century

Let f and g be continuously differentiable real valued functions.
Let c be a constant.

If $\vec{v}$ is an extremum of f on the constraint curve $g = c$ then

$$\exists\lambda \quad s.t \quad \nabla f(\vec{v}) + \lambda\nabla g(\vec{v}) = 0$$

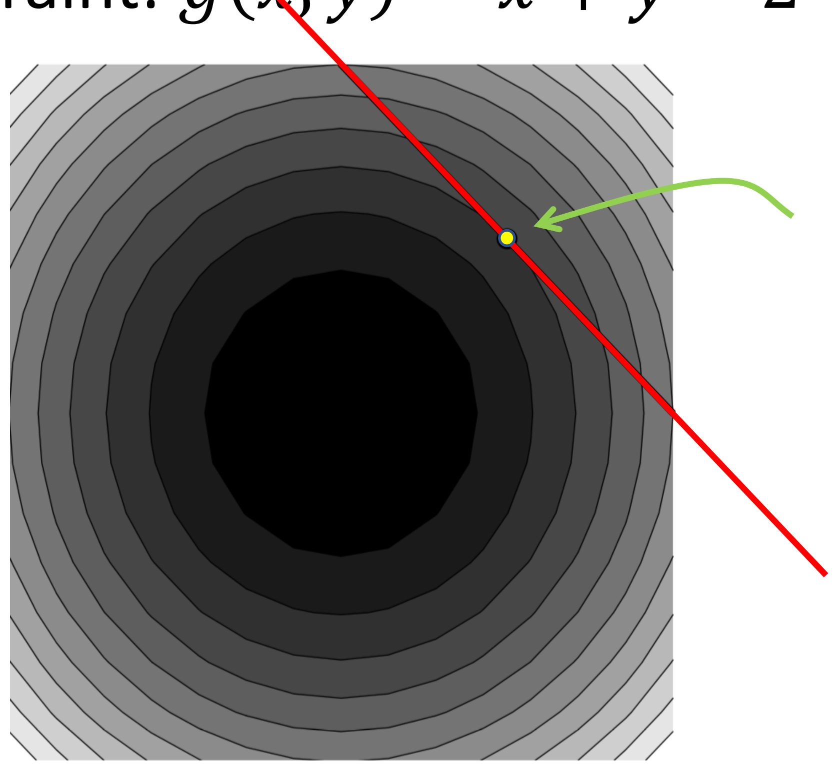
$f(x, y) = -4x + 10y - 10$



$(x-2)^2 + (y-4)^2 = 1$

Back to this:

- Miminize $f(x, y) = x^2 + y^2$
- Subject to the constraint: $g(x, y) = x + y - 2 = 0$



Lagrange Multipliers - Simple Example Solved

- $L(x, y) = x^2 + y^2 + \lambda(x + y - 2)$

- $\frac{\partial}{\partial x} L(x, y) = 2x + \lambda = 0$
- $\frac{\partial}{\partial y} L(x, y) = 2y + \lambda = 0$
- $\frac{\partial}{\partial \lambda} L(x, y) = x + y - 2 = 0$

- Subtracting the two first eqns we get $x = y$ and with the third we get a unique solution $x = 1, y = 1, \lambda = -2$

SVM Optimization – final

$$\min_{\vec{w},b} ||\vec{w}||^2$$

under the constraints:

$$\forall i \qquad t_i(\vec{x} \cdot \vec{w} \; - b) - 1 \geq 0$$

FJ Conditions

Fritz John
NYU applied math
1910-1994

Suppose that w^* solves the optimization task

$$\min_{w} f(w)$$

under the constraints: $1 \leq \forall i \ \leq m \quad g_i(w) \geq 0$, where f and all the g_i s are smooth real-valued functions.
Let $I(w^*) = \{1 \leq i \ \leq m : g_i(w^*) = 0\}$.

Then $\exists \alpha \in \mathbb{R}^m$ s.t

$$\nabla f(w^*) + \sum_{i \in I(w^*)} \alpha_i \ \cdot \nabla g_i(w^*) = 0$$

Applying the FJ conditions to the SVM optimization task

$$\min_{\vec{w},b} \|\vec{w}\|^2$$

under the constraints:

$$\forall i \qquad t_i(\vec{x}_i \cdot \vec{w} \ - b) - 1 \geq 0$$

$$f(w) = \ \|w\|^2 = w \cdot w,$$

$$\nabla f(w) = 2w$$

$$g_i(w) = t_i(\vec{x}_i \cdot \vec{w} \ - b) - 1,$$

$$\nabla g_i(w) = t_i \vec{x}_i$$

Applying the FJ conditions to the SVM optimization task - conclusion

$$\min_{\vec{w},b} ||\vec{w}||^2$$

under the constraints:

$$\forall i \qquad t_i(\vec{x}_i \cdot \vec{w} \; - b) - 1 \geq 0$$

Assume that w^* solves the SVM optimization task, then

$\exists \alpha \in \mathbb{R}^m$ s.t

$$w^* = \sum_{i \in I(w^*)} t_i \alpha_i \, \vec{x}_i$$

And: a solution for α can indeed be found by non-linear optimization techniques (KKT). **IF** it exists … , of course.

Support Vector Machines – Linear Decision Boundary

- <u>Input</u>: Training data
- <u>Output</u>:
  - A subset of the training examples called “support vectors”, denoted SV
  - A set of weights for these examples
- We classify +/- according to a simple “instance based” rule:

$$class(\vec{x}) = \operatorname{sgn}\left(\sum_{d \in SV} a_d \, t_d(\vec{x}_d, \vec{x})\right)$$

By solving the optimization question we learned coefficients a_d for which:

- This classifier has zero error on the training samples
- Margin is max
- Many coefficients are 0

Support Vector Machines – higher dimension

- Input: Data and a mapping function, φ , that non-linearly maps to high dimensional space
- Output:
  - A subset of the training examples called "support vectors", denoted SV
  - A set of weights for these examples (after the mapping)
- We classify +/- according to a simple "instance based" rule:

By solving the optimization question we learned coefficients a_d for which:

- This classifier has zero error on the training samples
- Margin is max (in the ambient space)
- Many coefficients are 0

$$class(\vec{x}) = \operatorname{sgn}\left(\sum_{d \in SV} a_d \, t_d\left(\varphi(\vec{x}_d), \varphi(\vec{x})\right)\right)$$

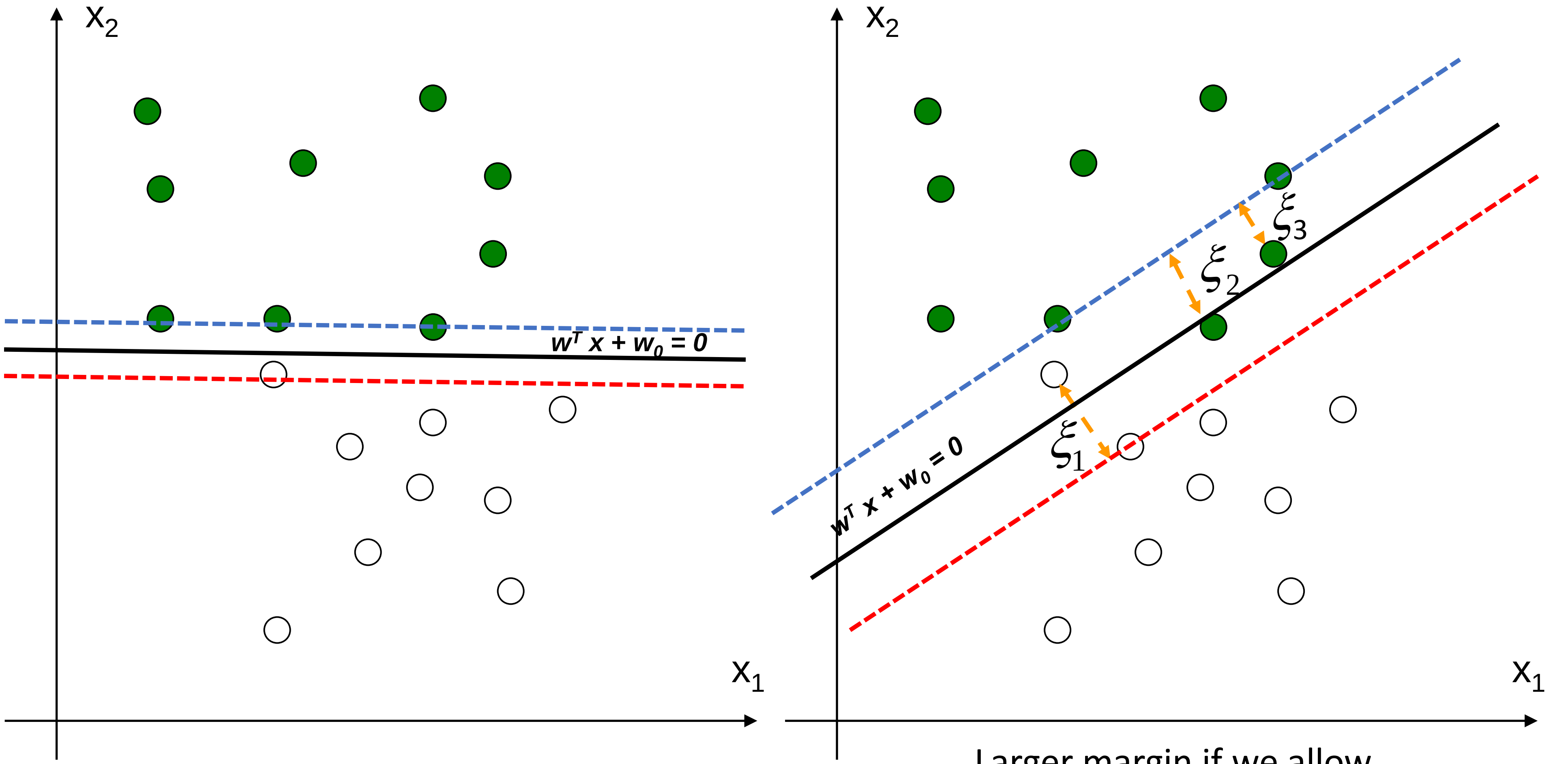
Support Vector Machines – Kernel

- Input: Data and a kernel function, K , that non-linearly maps to high dimensional space
- Output:
  - A subset of the training examples called "support vectors", denoted SV
  - A set of weights for these examples
- We classify +/- according to a simple "instance based" rule:

By solving the optimization question we learned coefficients a_d for which:

- This classifier has zero error on the training samples
- Margin is max (in the ambient space)
- Many coefficients are 0

$$class(\vec{x}) = \text{sgn}\left(\sum_{d \in SV} a_d\, t_d K(\vec{x}_d, \vec{x})\right)$$



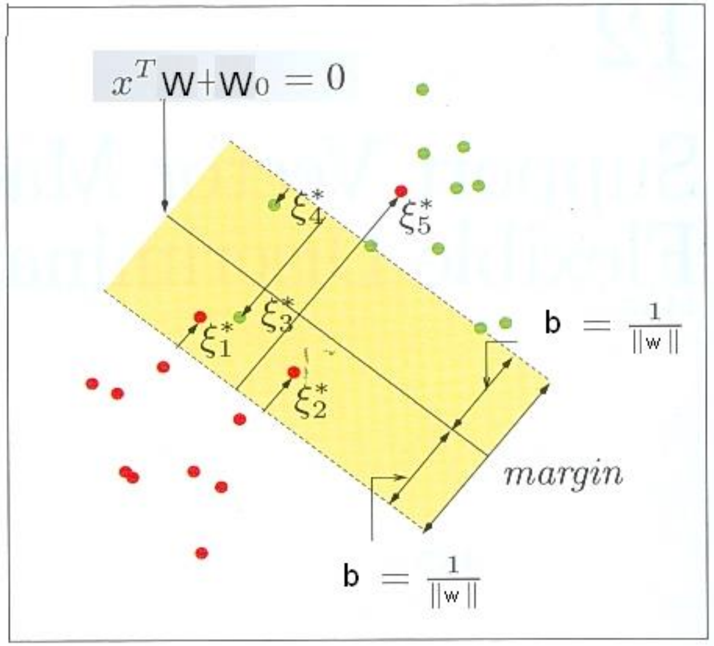
Larger margin if we allow
some error

Slack or Hinge variables

- Idea – we are willing to tolerate some wrongly classified training points or such that don't respect the margin. But - not too many …
- Introducing "slack" variables ξ_d

$$\text{Minimize}\ \|\mathbf{w}\|\ \text{subject to:}$$

$$\begin{cases} \forall d, t_d\left(\mathbf{w}\cdot\mathbf{x}^{(d)} + w_0\right) \geq 1 - \xi_d \\ \xi_d \geq 0 \\ \sum \xi_d \leq \text{Const} \end{cases}$$



SVM Optimization (General)

Minimize $\frac{1}{2}\|w\|^2 + C\sum_{d\in D}\xi_d$

subject to

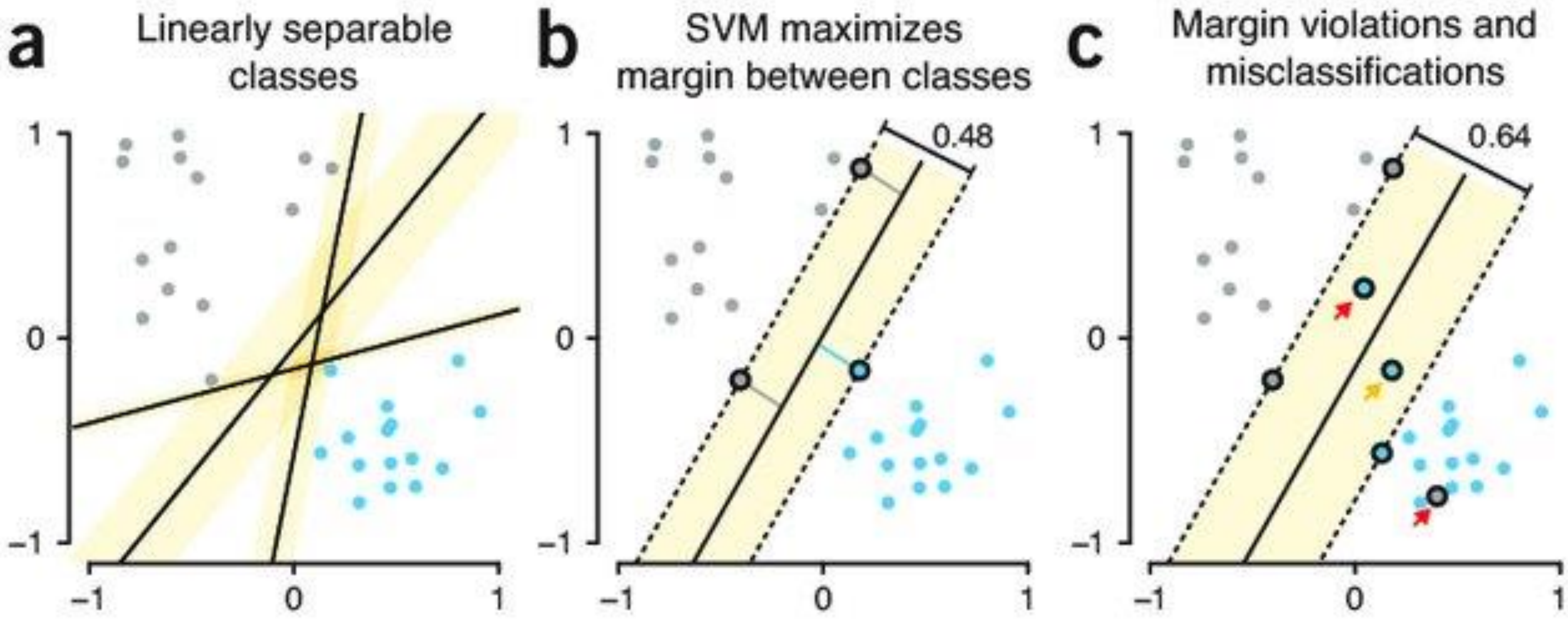
$\xi_d \geq 0\,,\ \forall d \quad t_d(w\cdot x_d + w_0) \geq 1\ - \xi_d$

Minimize $\frac{1}{2}\|\mathbf{w}\|^2$ subject to :

$t_d\left(\mathbf{w}\cdot\mathbf{x}^{(d)} + w_0\right) - 1 \geq 0 \quad \forall d$

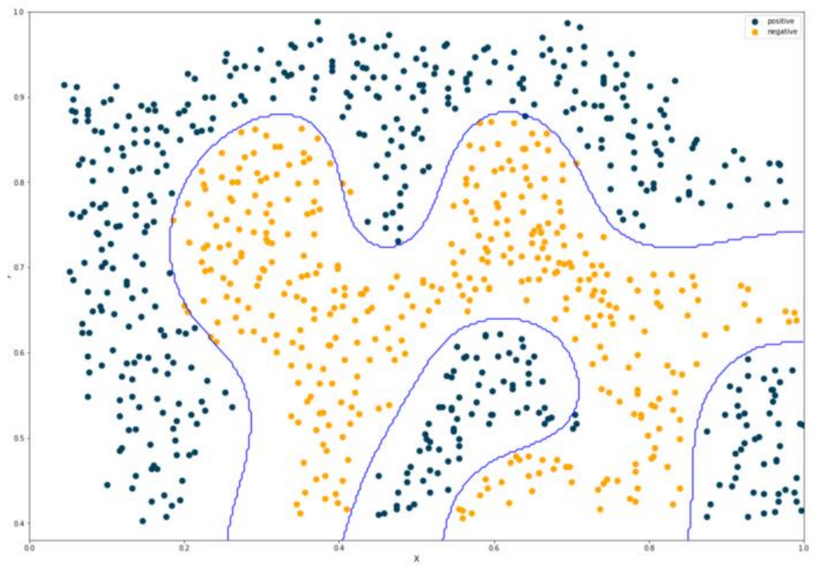
Without slack

Important: hinge variables allow for misclassifications



From Bzdok et al, Nat Methods 2018

Support Vector Machines – learning, in practice



- <u>Input</u>:
  Data instances and labels
- <u>Hyperparameters</u>:
  A kernel
  Control of the slack variables
- <u>Output</u>:
  A subset of training instances - the “support vectors”
  A set of weights for these data points
- We classify +/- according to

$$class(\vec{x}) = \operatorname{sgn}\left(\sum_{d \in SV} a_d \, t_d K(\vec{x}_d, \vec{x})\right)$$

Examples of practically running SVMs, including non-linear boundaries and using slack

In the recitation

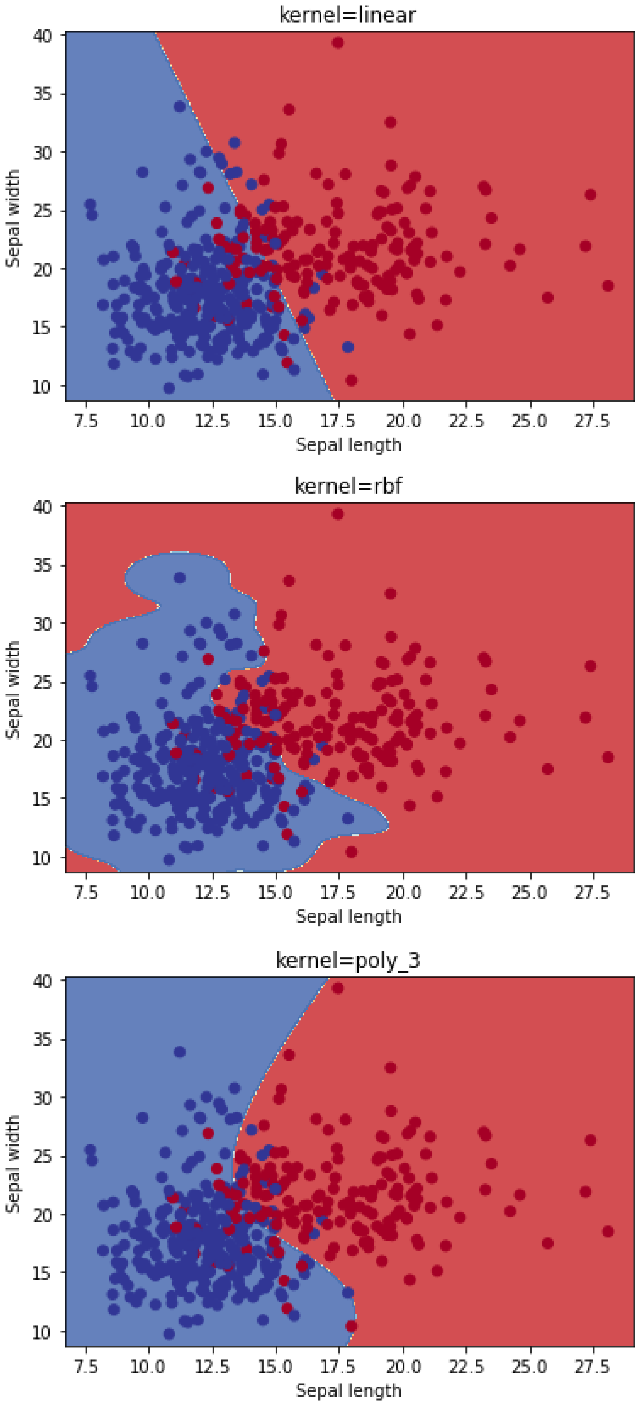
SVM Learning: Summary of Principles

1. Map instance space non linearly into a higher dimensional feature space (mapping space) using a mapping that affords an efficient kernel.
2. Optimize for linear separability in the higher dimensional space using dual formulation (FJ conditions). Similar to the kernel Perceptron.
3. This process finds the optimal margin linear separation in the ambient space (using quadratic programming).
4. We obtain a non-linear decision boundary in the original instance space.
5. Can accommodate some mis-classified training data points, to an extent controlled by a process hyperparameter.
   The process converges even if there is no perfect separation.
6. Output: SVs and weights, to be used for execution.

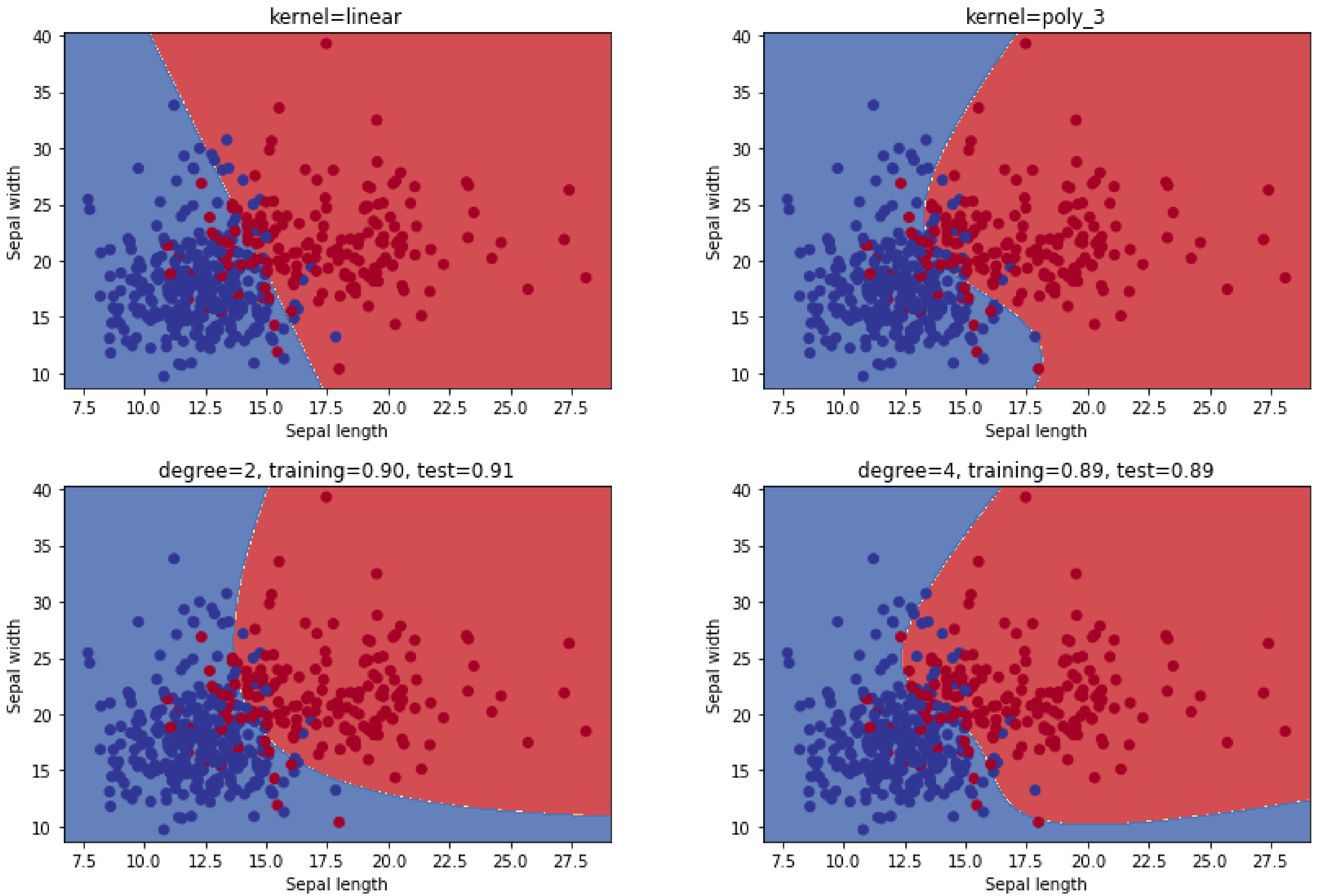
Epilogue: overfitting

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
%matplotlib inline
```

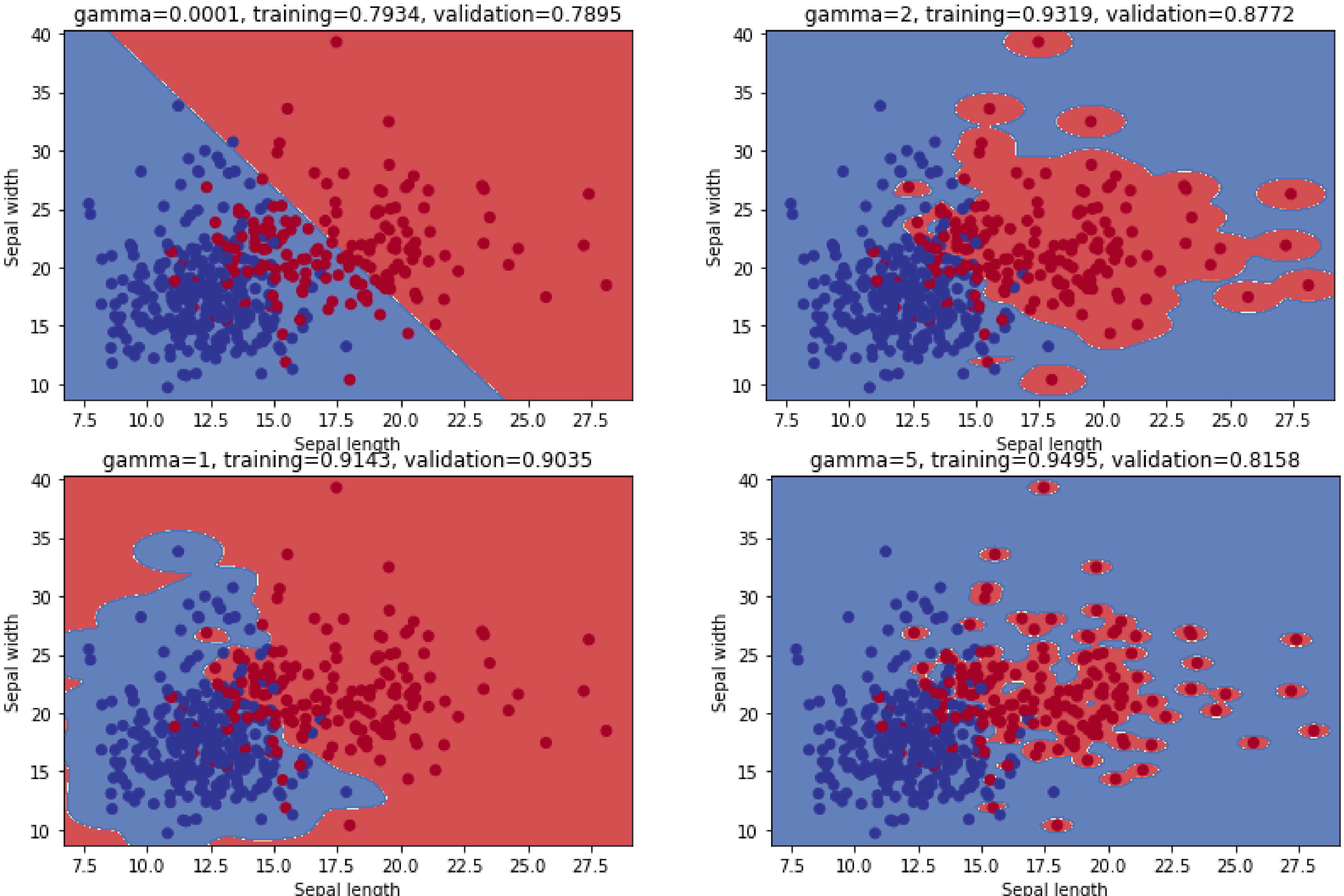
```
kernels = ['linear', 'rbf', 'poly']
for kernel in kernels:
    svc = svm.SVC(kernel=kernel).fit(X, y)
    if kernel == 'poly':
        kernel = 'poly_3'
    plotSVC('kernel=' + str(kernel))
```

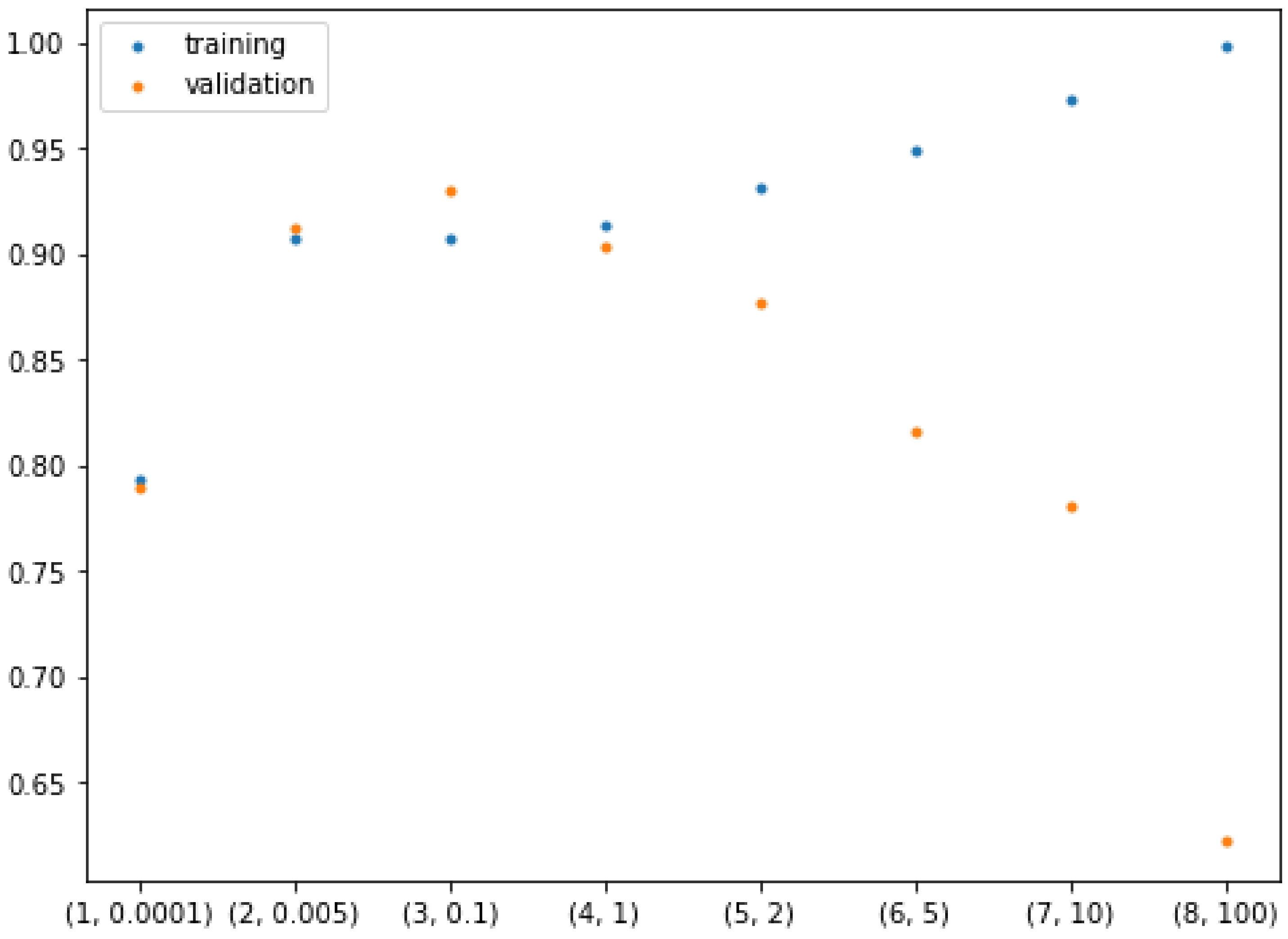


Polynomial kernels

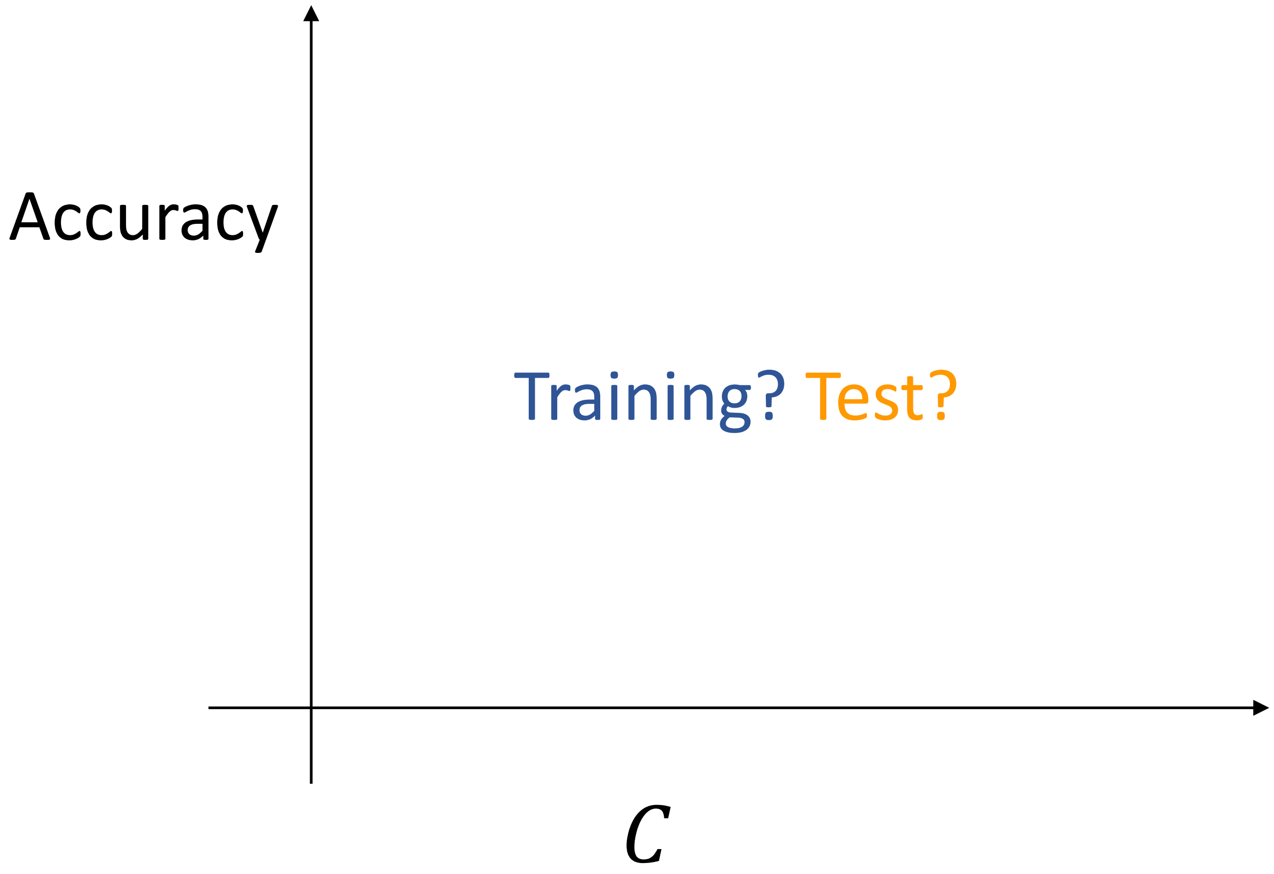


```
gammas = [0.0001, 0.005, 0.1, 1, 2, 5, 10, 100]
train_acc = []
val_acc = []
for gamma in gammas:
    svc = svm.SVC(kernel='rbf', gamma=gamma).fit(X, y)
    train_acc.append(svc.score(X, y))
    val_acc.append(svc.score(X_test, y_test))
    plotSVC('gamma={}, training={:.4f}, validation={:.4f}'.format(gamma, train_acc[-1], val_acc[-1]))
```


gamma=0.0001, training=0.7934, validation=0.7895
gamma=2, training=0.9319, validation=0.8772
gamma=1, training=0.9143, validation=0.9035
gamma=5, training=0.9495, validation=0.8158
Sepal width
Sepal length
40
35
30
25
20
15
10
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5



RBF overfitting



Effect of the slack regularization?