Classification by Nearest Shrunken Centroids and Support Vector Machines

Florian Markowetz
florian.markowetz@molgen.mpg.de
Max Planck Institute for Molecular Genetics
Computational Diagnostics Group
Berlin, Germany

Practical Microarray Analysis 2006

Two roads to classification

Given: patient profiles already diagnosed by an expert.

Task: infer a general rule to diagnose new patients.

Basically, there are two ways to solve the task

1. model **class probabilities**
   → QDA, LDA, ...
2. model **class boundaries** directly
   → Optimal Separating Hyperplanes, SVM

What’s the problem?

In classification you have to trade off

- **overfitting** versus **underfitting**
- **bias** versus **variance**.

Curse of dimensionality! In 12’000 dimensions even linear methods are very complex → high variance!

Simplify your models

Discriminant analysis and gene selection

Comparing Gaussian likelihoods

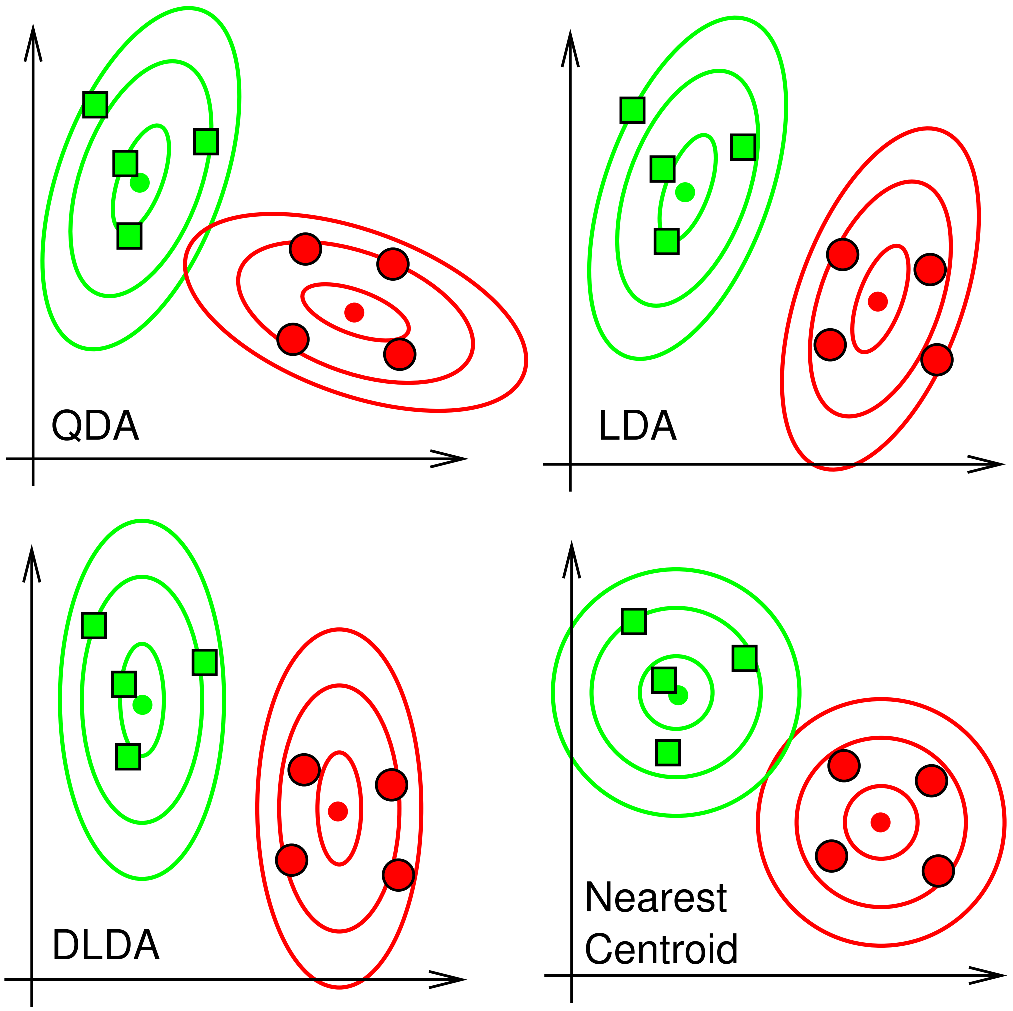
Assumption: each group of patients is well described by a Normal density.

Training: estimate **mean** and **covariance matrix** for each group.

Prediction: assign new patient to group with higher likelihood.

Constraints on covariance structure lead to different forms of discriminant analysis.

Disriminant analysis in a nutshell



Characterize each class by **mean** and **covariance structure**.

1. **Quadratic D.A.**
   different COVs
2. **Linear D.A.**
   requires same COVs.
3. **Diagonal linear D.A.**
   same diagonal COVs.
4. **Nearest centroids**
   forces COVs to $\sigma^2\mathbf{I}$.

Feature selection

Next simplification:

Base the classification only on a small number of genes.

Feature selection: Find the **most discriminative genes**.

We will see:
this task is different from testing for differential expression.

1. *Genes can be significantly differential expressed, but still useless for classification.*
2. *And predictive genes may not be differential.*

Feature selection

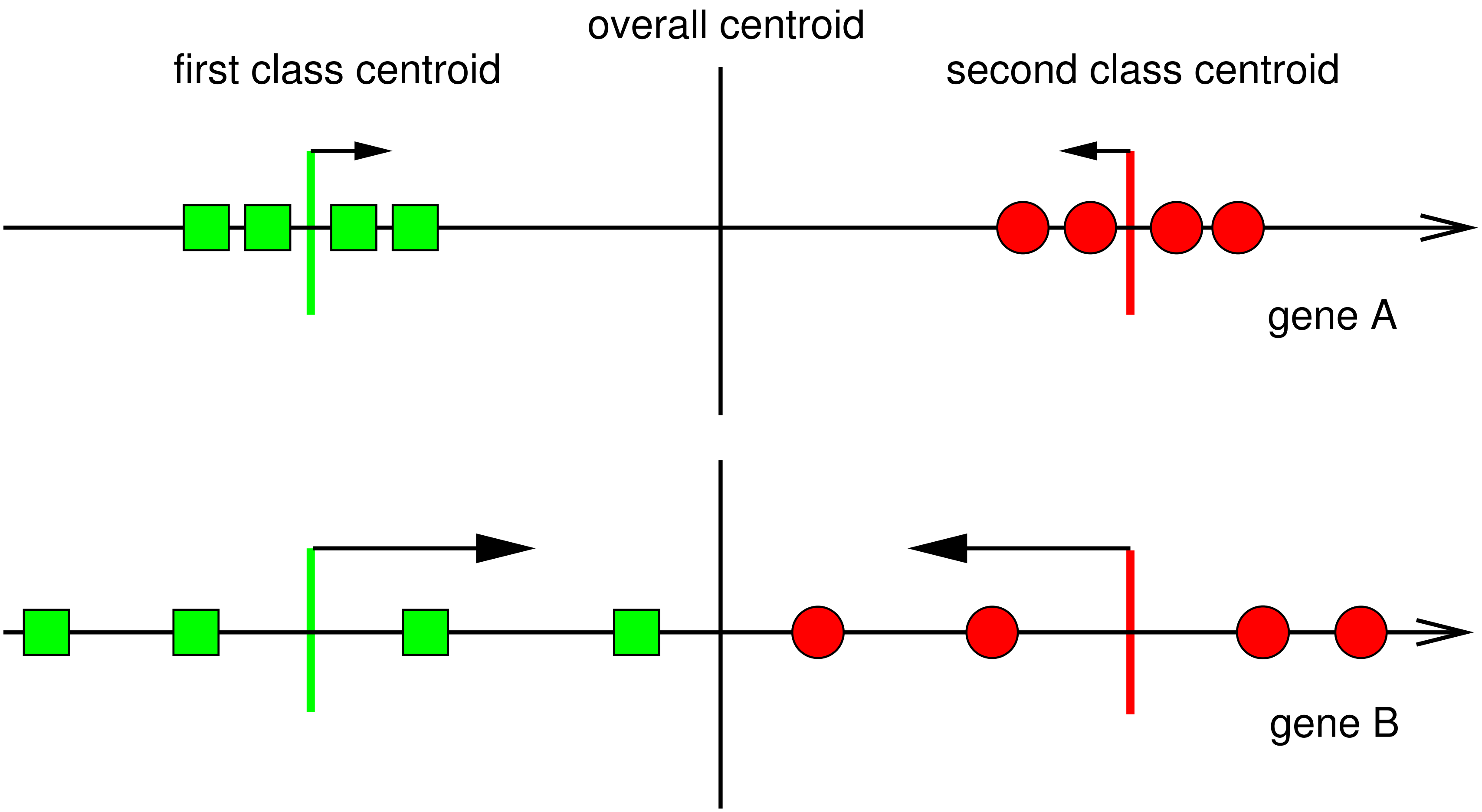
1. **Filter:**

   - Rank genes according to discriminative power by t-statistic, Wilcoxon, ...
   - Use only the first k for classification.
   - Discrete, hard thresholding.

2. **Shrinkage:**

   - Continously shrink genes until only a few have influence on classification.
   - Example: Nearest Shrunken Centroids.

Shrunken Centroids



Nearest Shrunken Centroids *cont'd*

The group centroid $\bar{x}_{gk}$ for gene g and class k is compared to the overall centroid $\bar{x}_g$ by

$$\bar{x}_{gk} \; = \; \bar{x}_g + m_k(s_g + s_0)\, d_{gk} \ ,$$

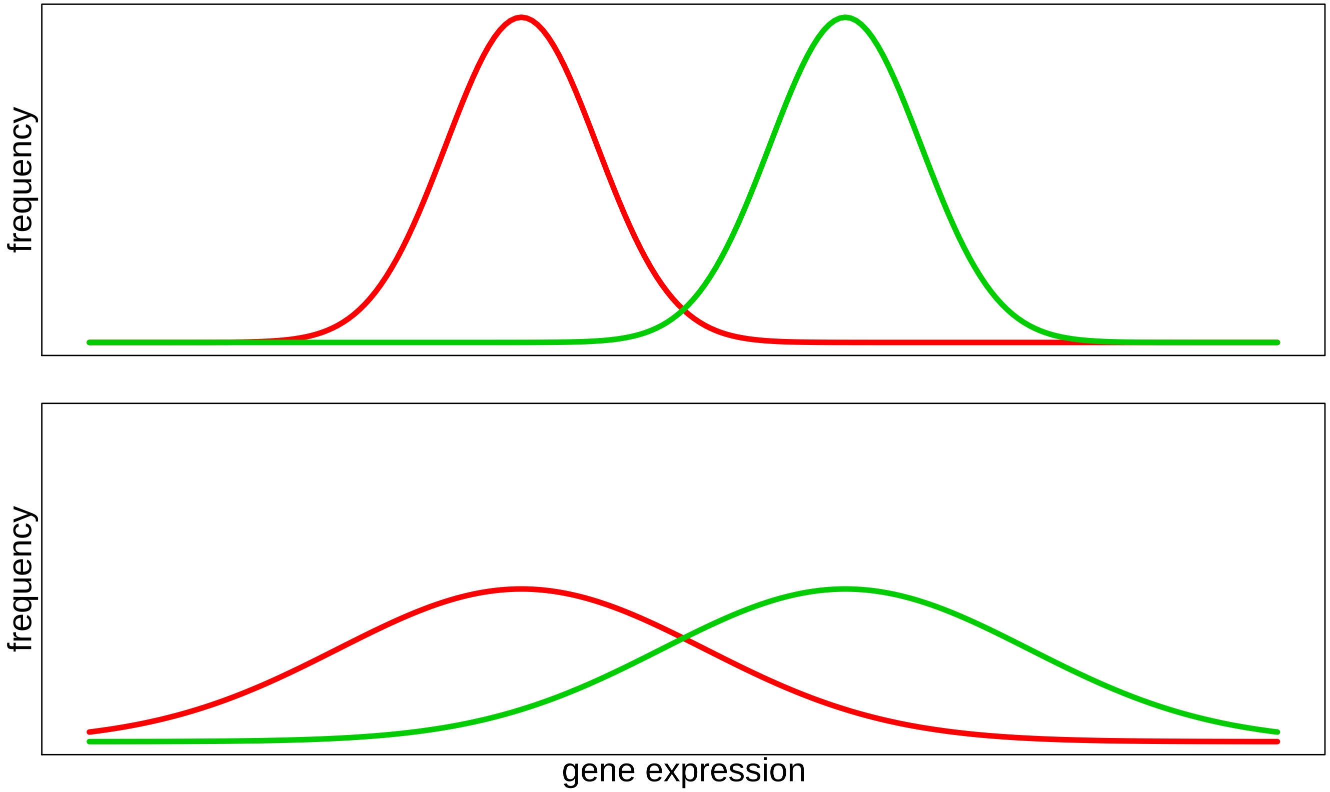
where s_g is the pooled within-class standard deviation of gene g and s_0 is an offset to guard against genes with low expression levels.

Shrinkage: Each d_{gk} is reduced by Δ in absolute value, until it reaches zero. Genes with $d_{gk} = 0$ for all classes do not contribute to the classification. (Tibshirani *et al.*, 2002)

Shortcomings of filter and shrinkage methods

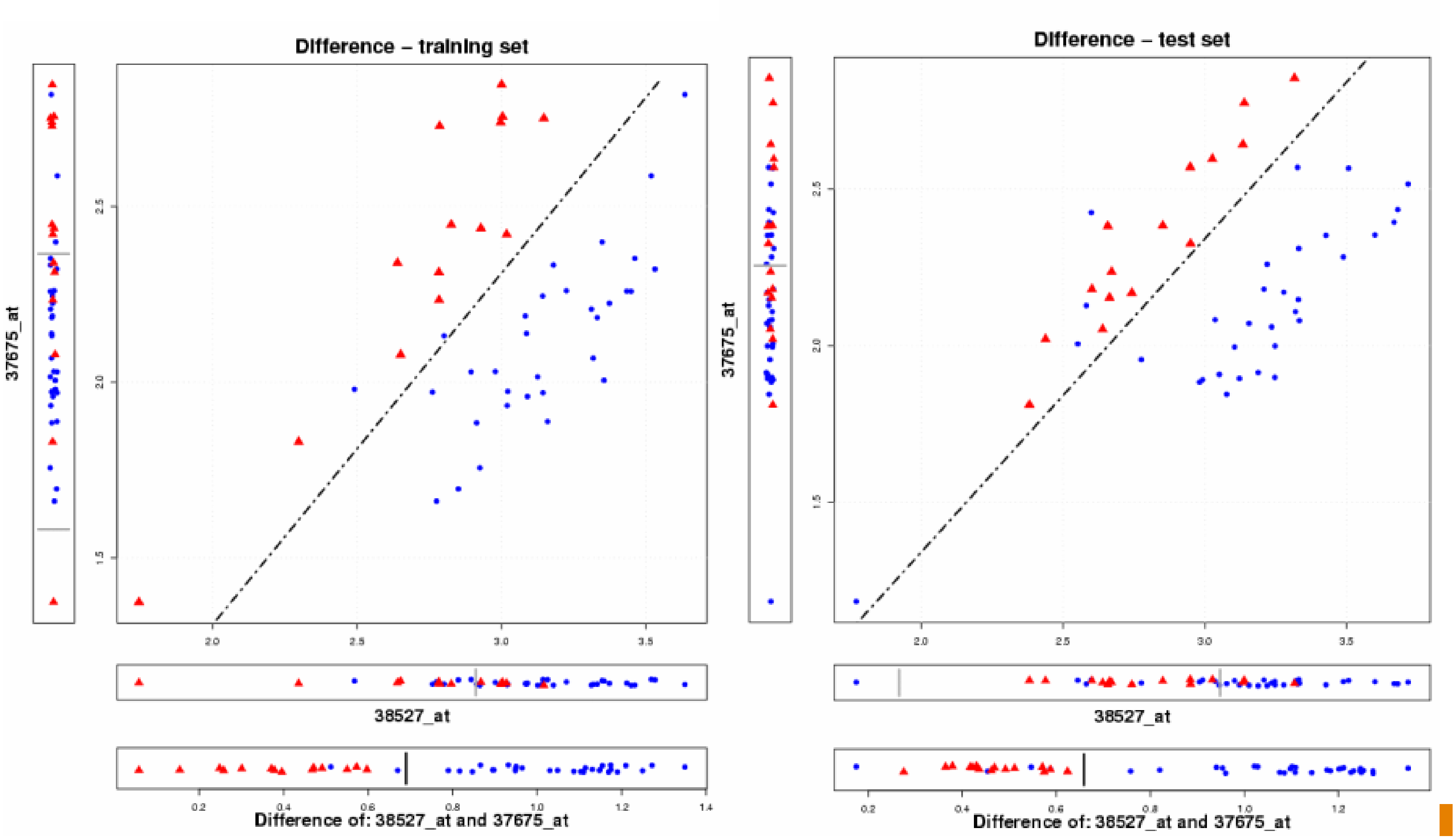
1. Highly correlated genes get similar score but offer no new information.
   But see (Jaeger *et al.*, 2003) for a cure.
2. Filter and Shrinkage work only on single genes.
   They don’t find interactions between groups of genes.
3. Filter and Shrinkage methods are only heuristics.
   Search for *best subset* is infeasible for more than 30 genes.

Differential genes may not be predictive!



The upper one is differential and predictive, the lower one is also differential, but not predictive.

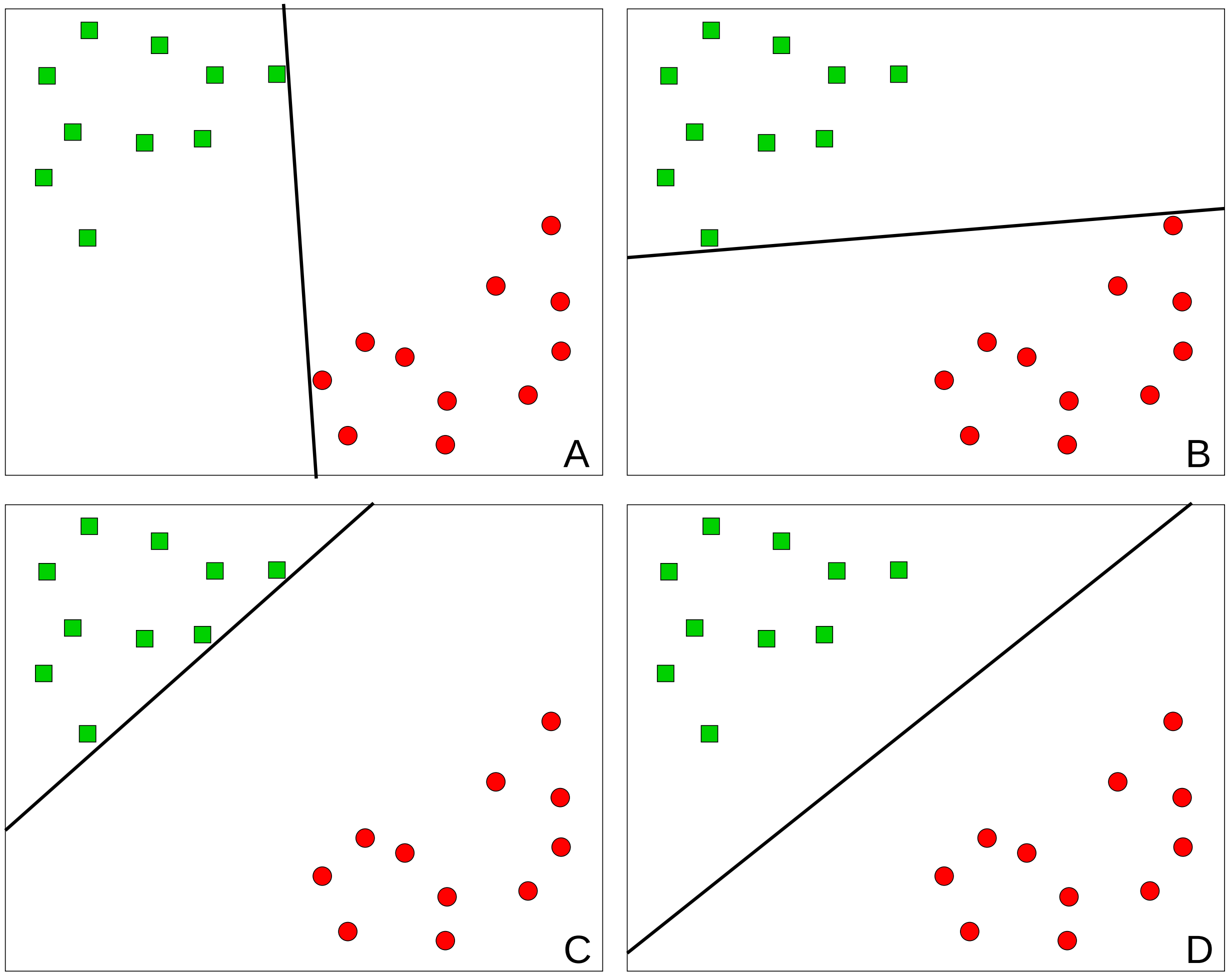
Predictive genes may not be differential!



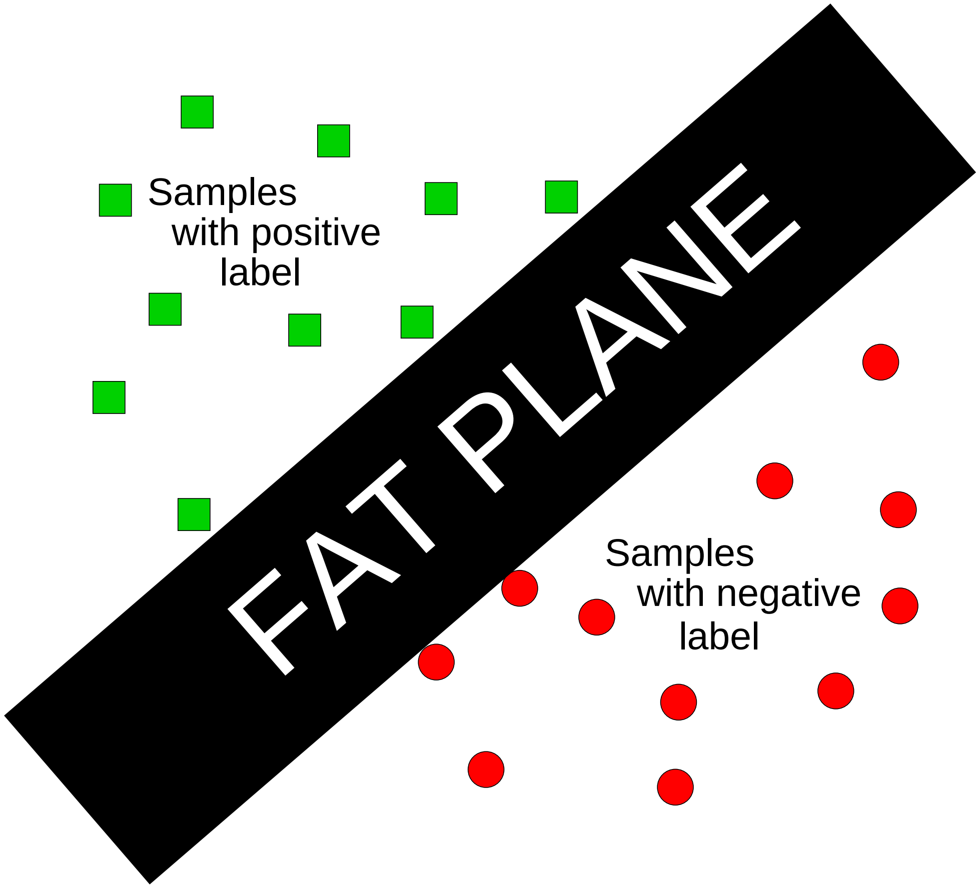
Support Vector Machines

— SVM —

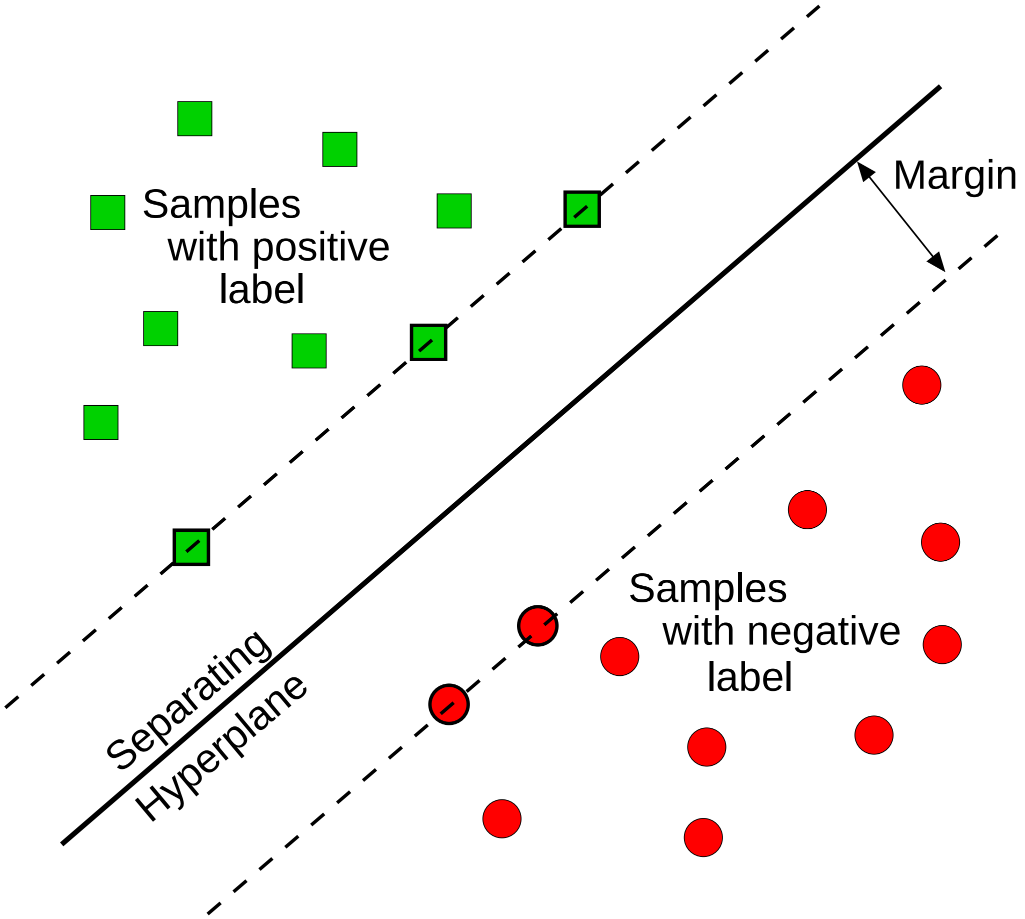
Which hyperplane is the best?



No sharp knive, but a fat plane



Separate the training set with maximal margin

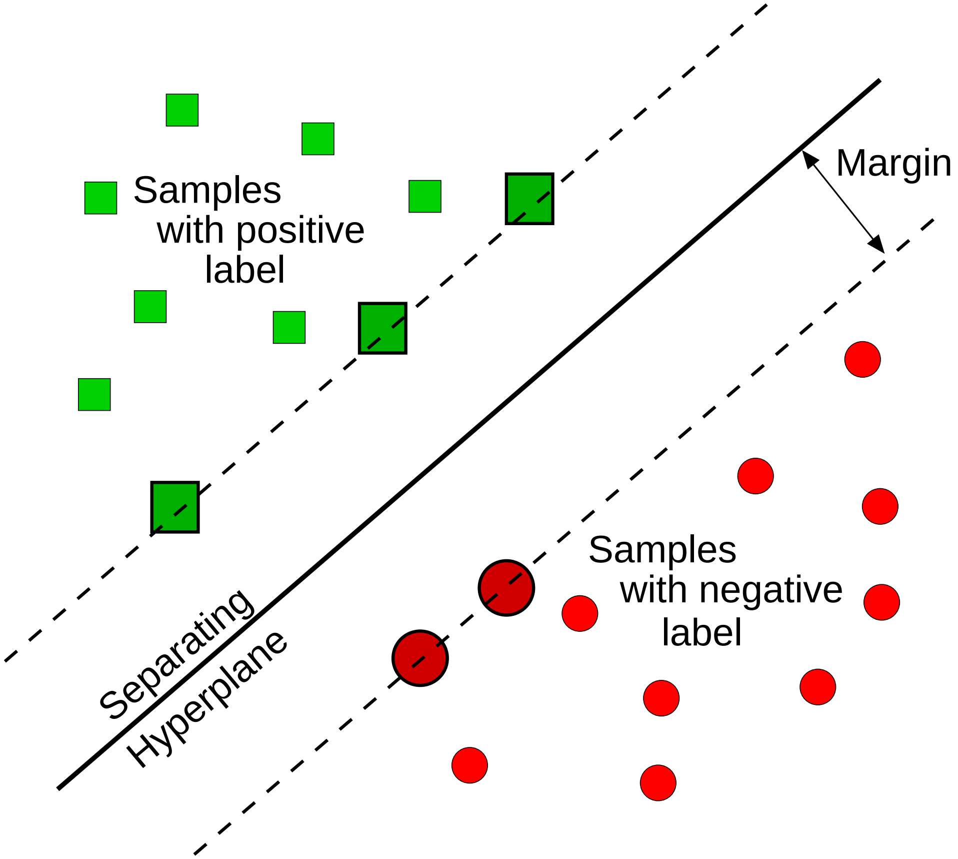


What are Support Vectors?

The points nearest to the separating hyperplane are called Support Vectors.

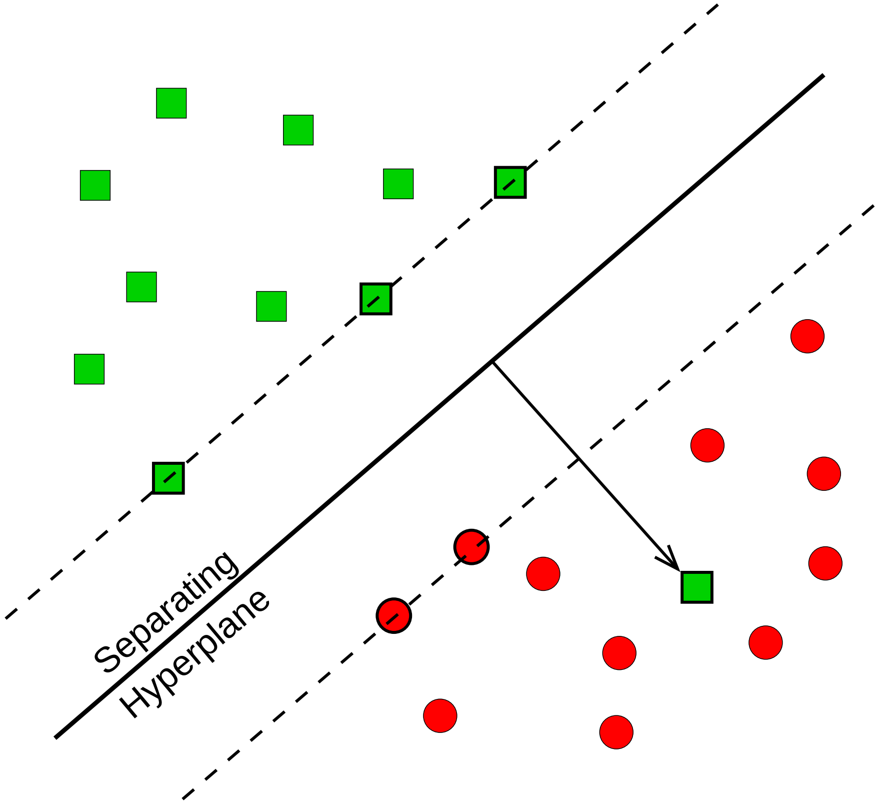
Only they determine the position of the hyperplane. **All other points have no influence!**

Mathematically: the weighted sum of the Support Vectors is the normal vector of the hyperplane.



Non-separable training sets

Use linear separation, but admit training errors.



Penalty of error: distance to hyperplane times *error cost* C.

The end?

The story of how to simplify your models is finished.

But for the sake of completeness:
How do we get from the simple linear Optimal Separating Hyperplane to a full-grown Support Vector Machine?

It’s a trick, a **kernel trick**.

Inner product

Expression profiles $p = (p_1, p_2, \ldots, p_g) \quad \in \mathbb{R}^g$
and $q = (q_1, q_2, \ldots, q_g) \quad \in \mathbb{R}^g$.

The **inner product** (aka skalar product) is defined as

$$\langle p, q \rangle \;=\; p_1 q_1 + p_2 q_2 + \ldots + p_g q_g$$

1. linear measure of similarity
2. related to covariance by $\langle p - \bar{p}, q - \bar{q} \rangle \;=\; g \cdot \mathrm{cov}(p, q)$
3. allows geometric constructions, *e.g.* **maximal margin hyperplanes**.

Kernel functions

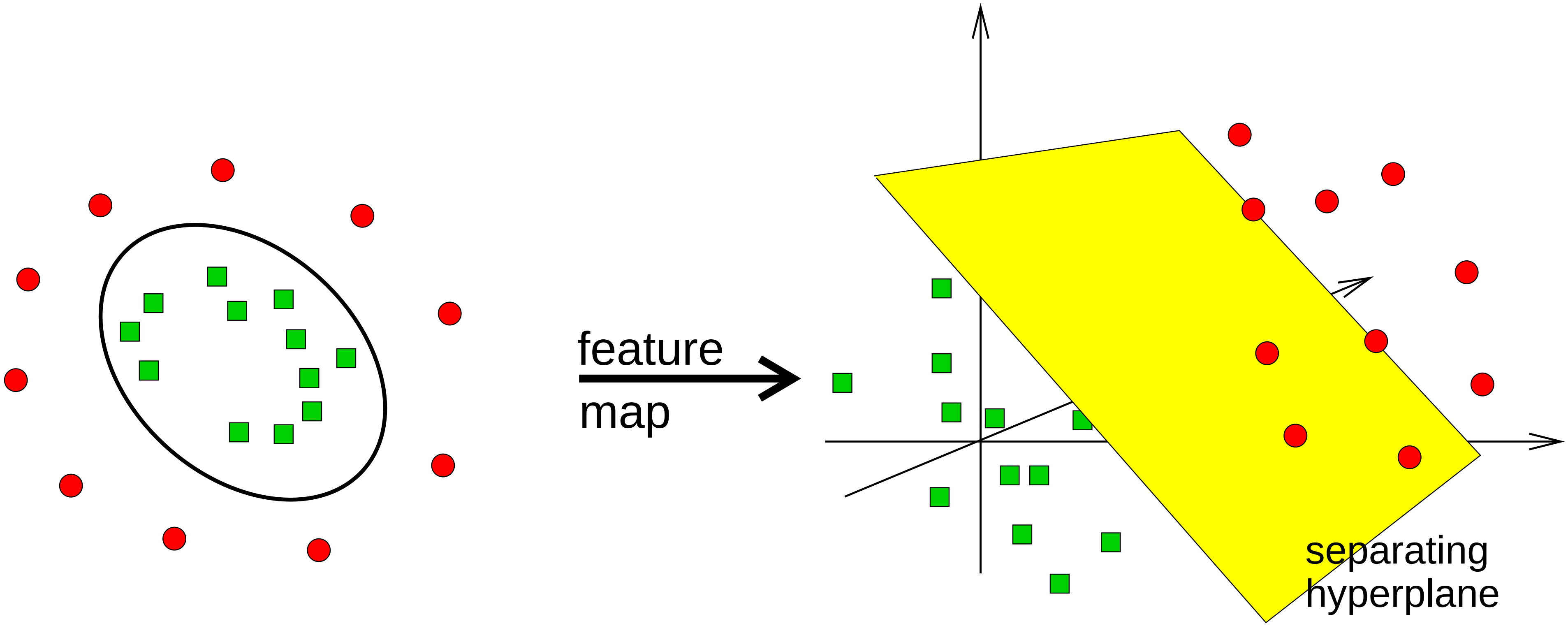
A kernel function is an inner product of profiles mapped to a (high-dimensional) **feature space**.

$$\mathcal{K}(p,q) \;=\; \langle \Phi(p), \Phi(q) \rangle$$

$$\Phi : \mathbb{R}^g \longrightarrow \mathcal{H}$$

1. **nonlinear** measure of similarity
2. allows geometric constructions in feature space
3. **Kernel trick**: substitute inner product $\langle p, q \rangle$ by kernel $\mathcal{K}(p,q)$.

Separation may be easier in higher dimensions



complex in low dimensions

simple in higher dimensions

Examples of Kernels

Standard kernels for classification are

linear $\mathcal{K}(p,q) = \langle p,q \rangle$

polynomial $\mathcal{K}(p,q) = (\langle p,q \rangle + 1)^d$

radial basis function $\mathcal{K}(p,q) = \exp\left(-\gamma \|p-q\|^2\right)$

In the exercises we will see: **linear kernels** are usually all you need for microarray datasets.

Why is it a trick?

We do not need to know,
how the feature space really looks like,
we just need the kernel function as a measure of similarity.

This is kind of black magic: we do not know what happens inside the kernel, we just get the output.

Still, we have the geometric interpretation of the maximal margin hyperplane, so SVMs are more transparent than e. g. Artificial Neural Networks.

Support Vector Machines

A Support Vector Machine is

a maximal margin hyperplane in feature space

built by using a kernel function in gene space.

Parameters of SVM

As a user: what do you need to care about?

Kernel Parameters	γ:	width of Gaussian kernel (rbf)
	d:	degree of polynomial
Error weight	C:	influence of training errors

SVM@work: low complexity

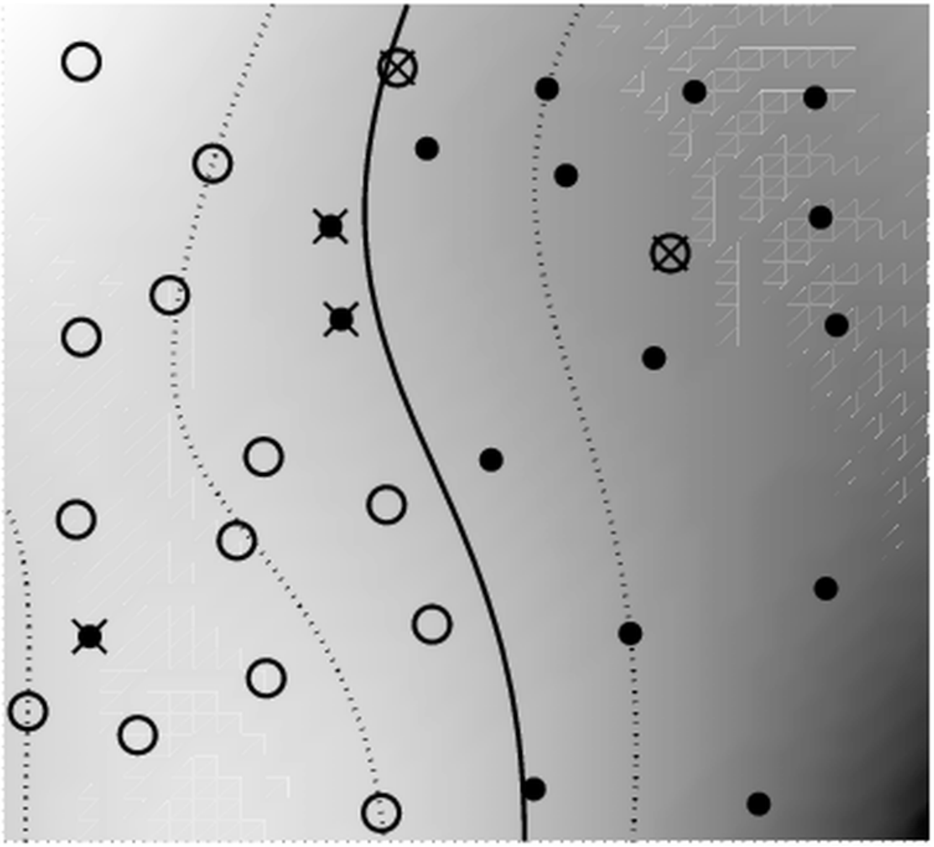


Figure taken from SCHÖLKOPF and SMOLA, *Learning with Kernels*, MIT Press 2002, p217

SVM@work: medium complexity

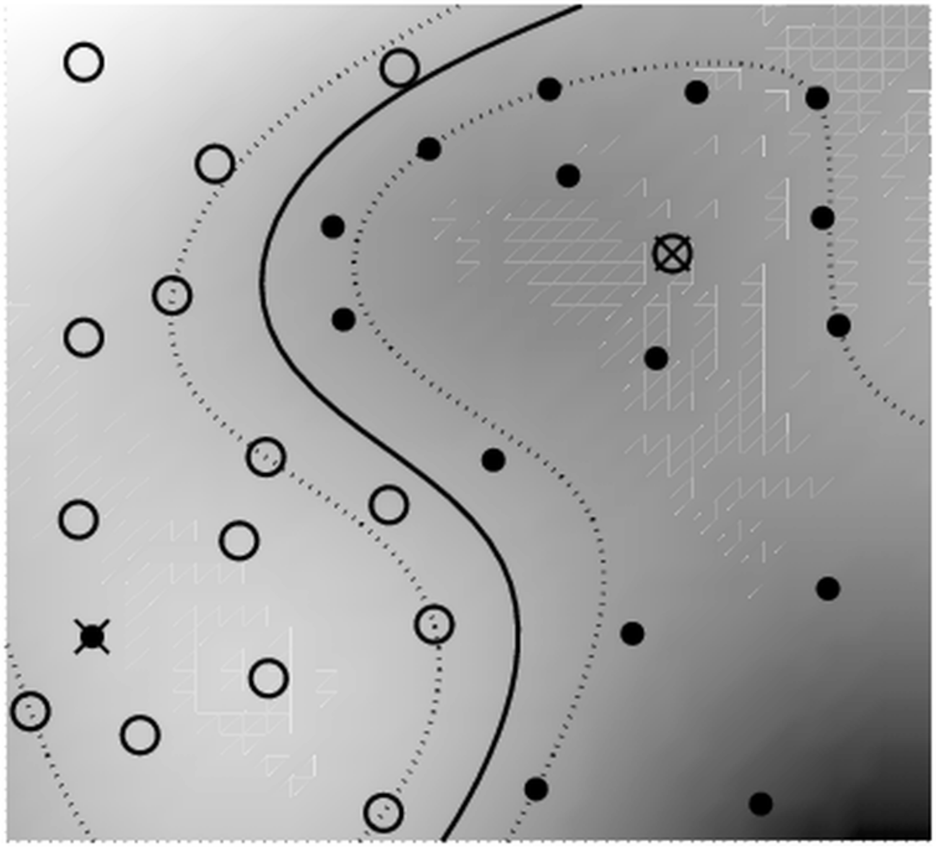


Figure taken from SCHÖLKOPF and SMOLA, *Learning with Kernels*, MIT Press 2002, p217

SVM@work: high complexity

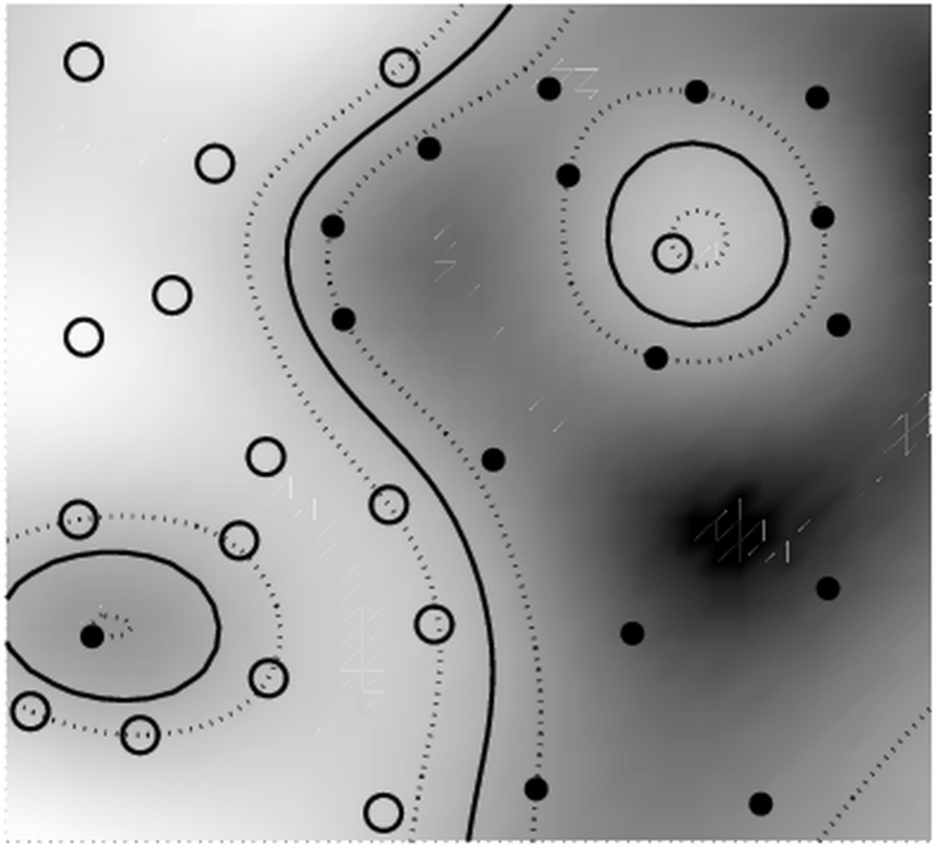


Figure taken from SCHÖLKOPF and SMOLA, *Learning with Kernels*, MIT Press 2002, p217

References

1. Trevor Hastie, Robert Tibshirani, Jerome Friedman
   The Elements of Statistical Learning. Springer 2001.

2. Bernhard Schölkopf and Alex Smola.
   Learning with Kernels. MIT Press, Cambridge, MA, 2002.

3. Robert Tibshirani, Trevor Hastie, Balasubramanian Narasimhan, Gilbert Chu
   Diagnosis of multiple cancer types by shrunken centroids of gene expression, PNAS, 99(10), 6567–6572, 2002.

4. Jochen Jäger, R. Sengupta and W.L. Ruzzo
   Improved Gene Selection for Classification of Microarrays, Proc. PSB 2003

Intro into practical session

Computational Diagnosis

TASK:

For 3 new patients in your hospital, decide whether they have a chromosomal translocation resulting in a BCR/ABL fusion gene or not.

IDEA:

Learn the difference between the cancer types from an archive of 76 expression profiles, which were analyzed and classified by an expert.

Training ... tuning ... testing

TRAINING:

```
model      <- svm(data       = "76 profiles",
                  labels     = "by an expert",
                  kernel     = "..",
                  parameters = "..")
```

TUNING:

```
svm.doctor <- tune.svm( data, labels,
                        all.parameter.values )
```

TESTING:

```
diagnosis <- predict(svm.doctor, new.patients)
```

Training ... tuning ... testing

TRAINING:

```
model      <- pamr.train( data , labels )
```

TUNING:

```
pamr.cv( data, labels )
```

TESTING:

```
diagnosis <- pamr.predict(new.patients,
                          best.treshold)
```