Machine Learning

Bernd Fischer
Wolfgang Huber

EMBL

Example: Cancer Subtype Prediction

Differential Expression Analysis:

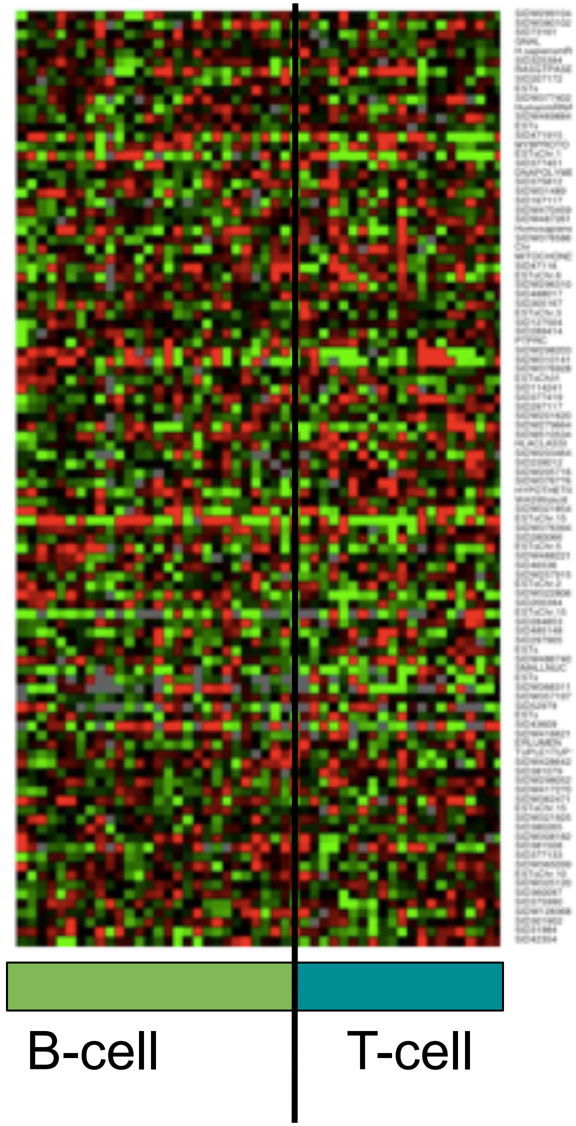
Which genes are differentially expressed between cancer subtypes?

Output:
p-values or q-values per gene or gene set.

Classification:

Which cancer subtype does a patient have, given his/her expression profile?

Output:
The cancer subtype of a new patient.



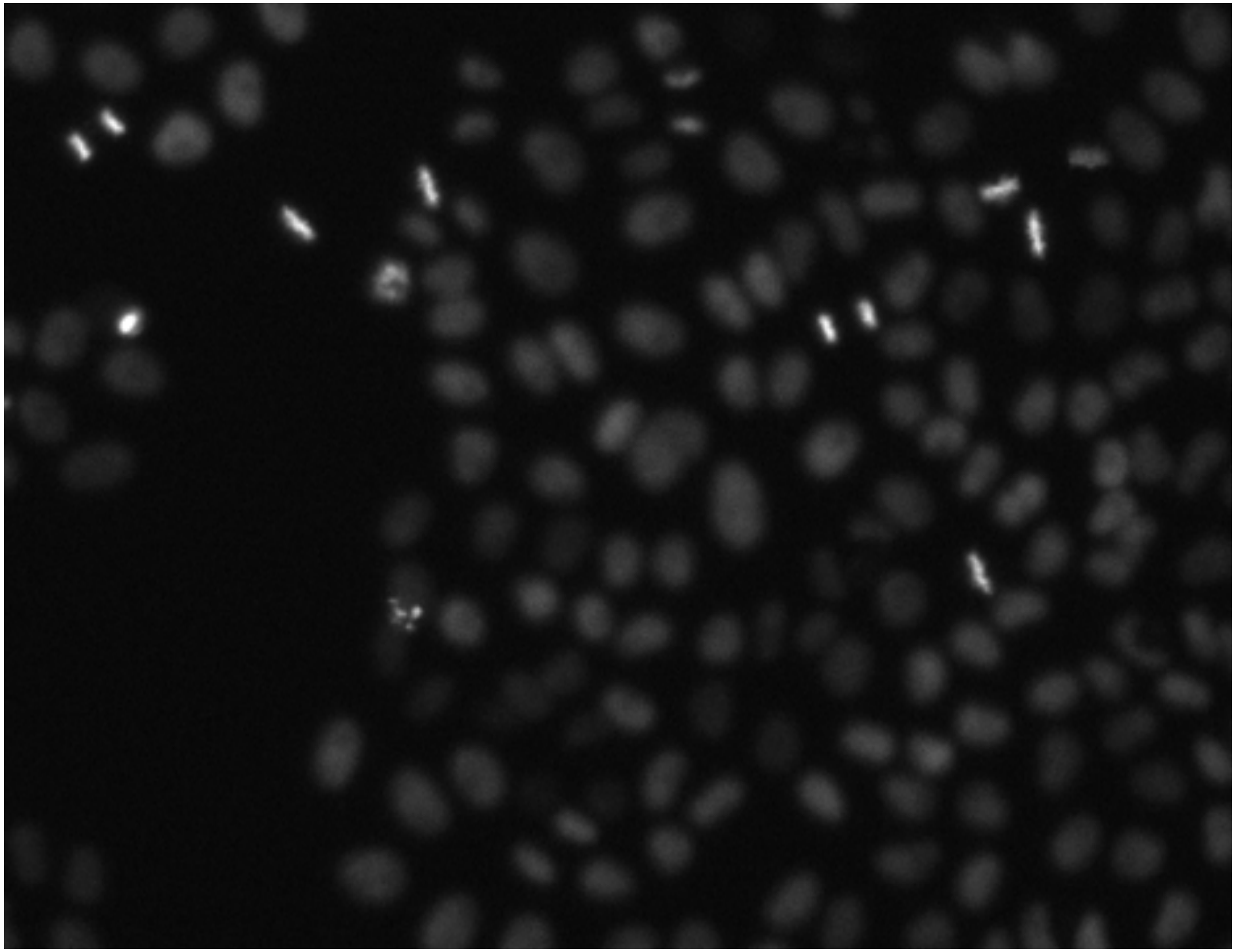
acute lymphoblastic leukemia (ALL)

Evidence based medicine

- Disease (e.g. HIV, diabetes, …)
- Multiple therapies available:
  - different drugs
    - targeting different processes
    - different side effects
  - surgical intervention
- Course of disease known for a number of patients and therapies
- Which combination of therapies/drugs has the highest success rate for a **new** patient?
- Decide based on
    - clinical factors, other low-dim. biochemical measures
    - expression profiles
    - genotypes
    - ...

Morphological Phenotyping I

- Image screen with a large number of images



e.g. *D. melanogaster*
full genome
knock-down screen:

~15 000 knock-downs
x 3 replicates
= 45 000 images

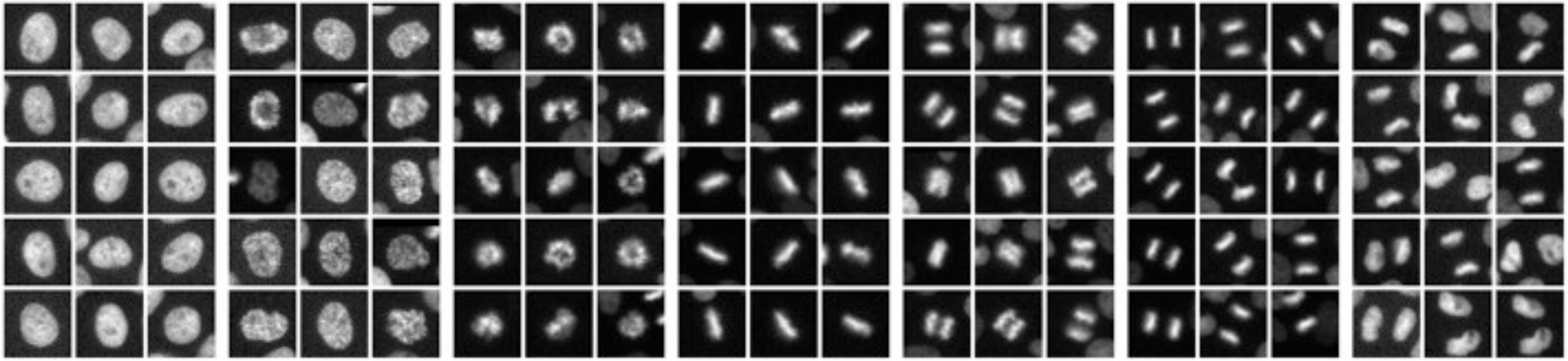
x 1000 cells per image
= 45 000 000 cells

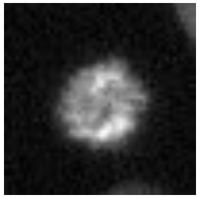
- Can we automatically annotate the cell cycle state of each cell?

EMBL

Morphological Phenotyping II

- Provide Human Annotation to a small set of cells:

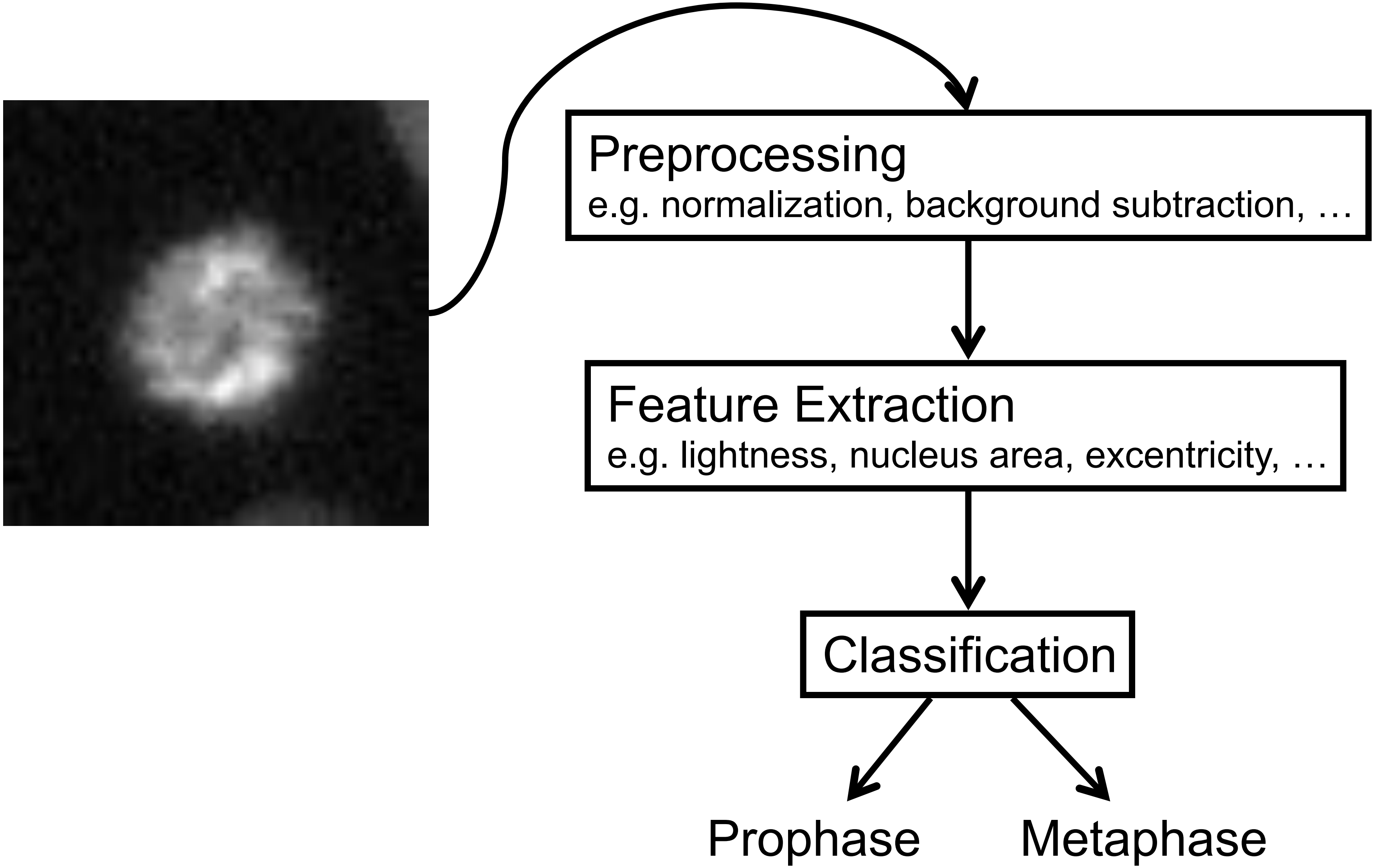




Which mitotic phase?
(Annotate automatically!)

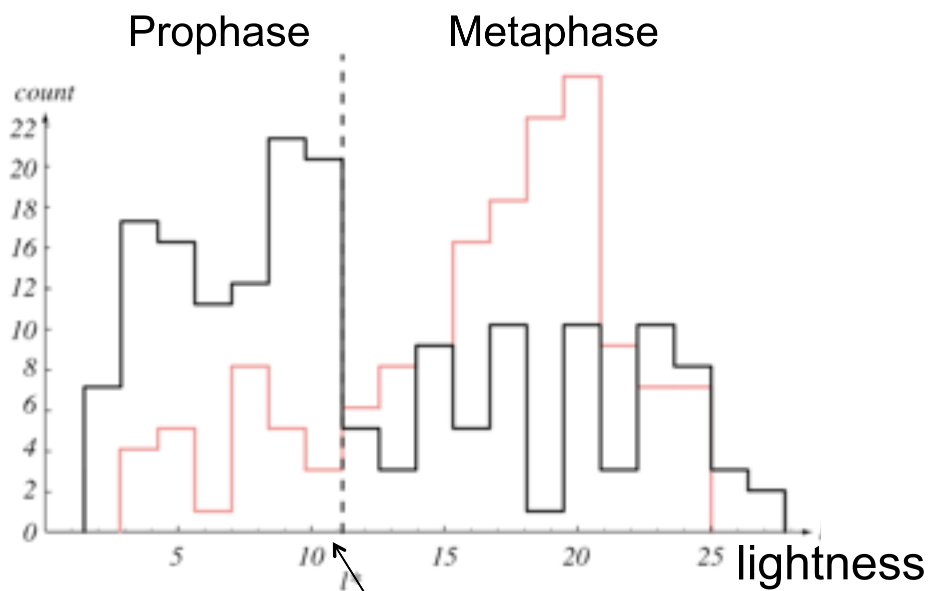
EMBL

Automatic Classification Workflow

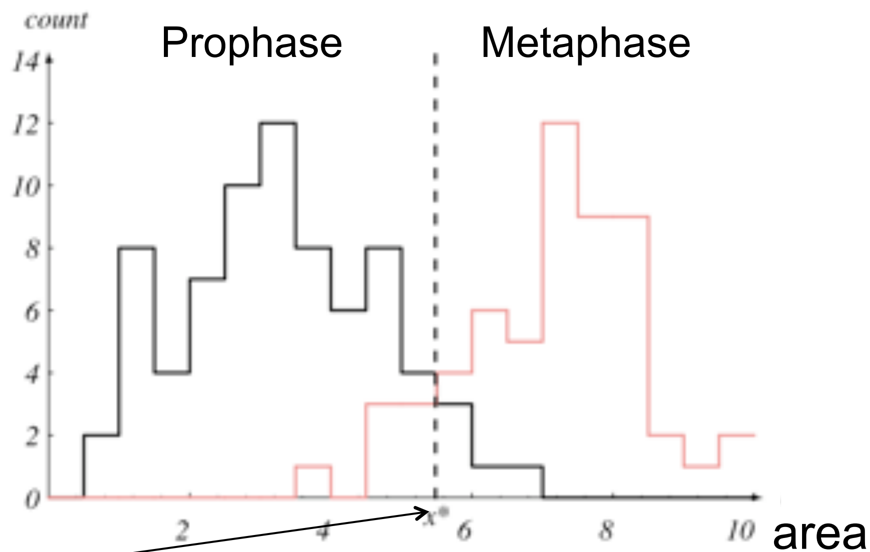


Prophase/ Metaphase Classification

Predict mitotic state based on lightness



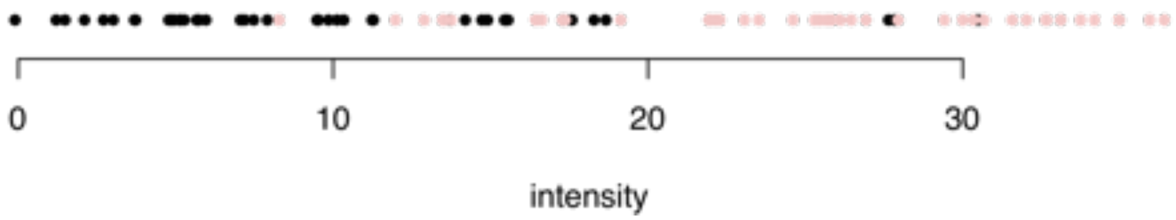
Predict mitotic state based on nucleus area



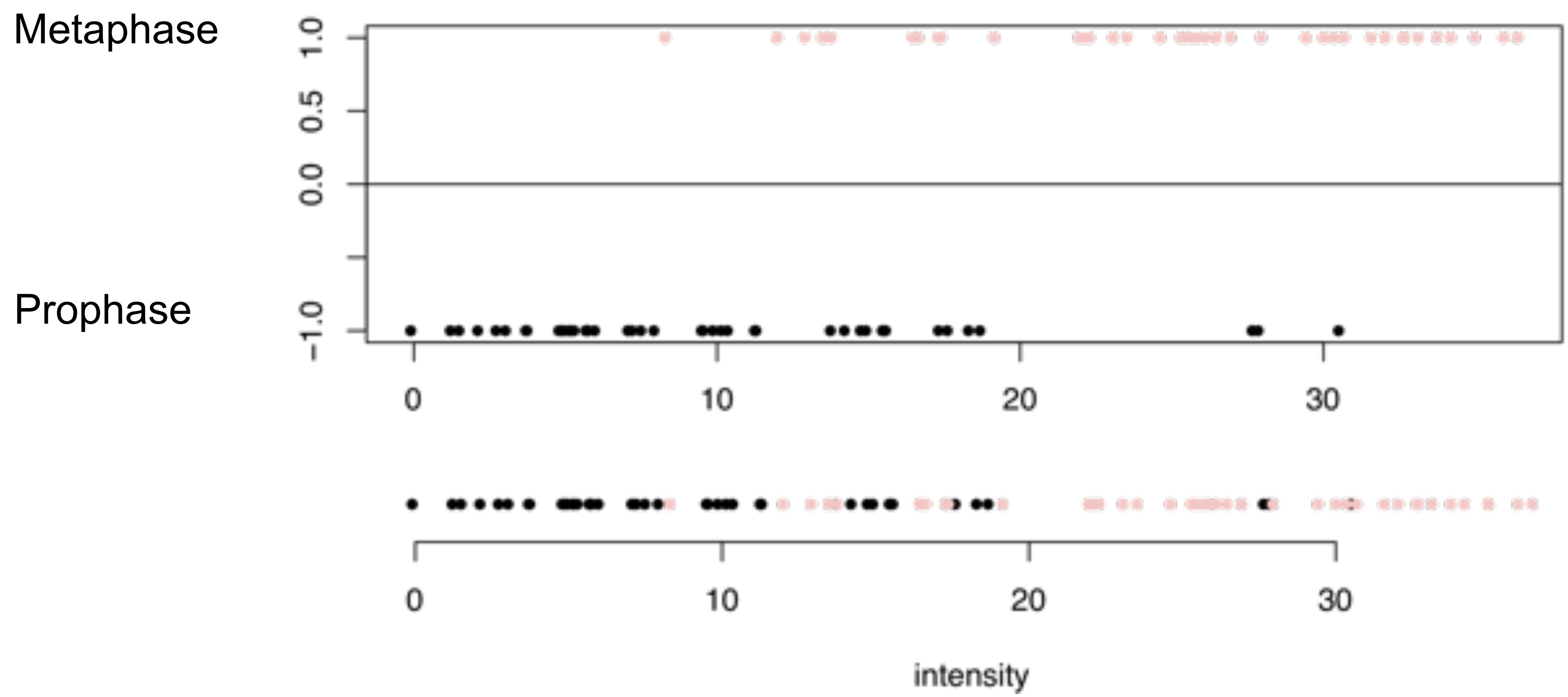
Decision boundary with lowest prediction error

None of the two features individually has a good predictive power

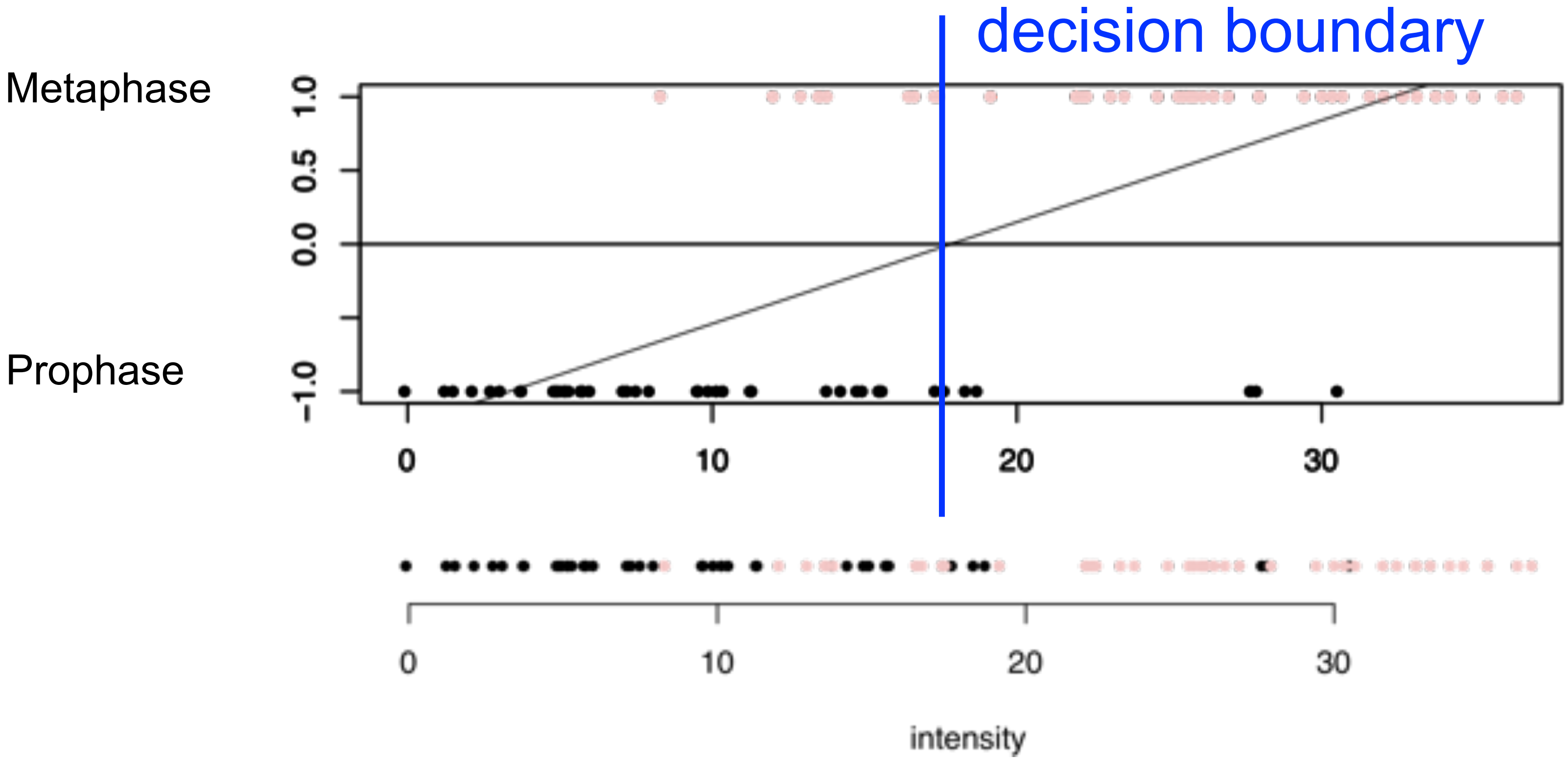
A Simple Least Squares Classifier: d=1



A Simple Least Squares Classifier: d=1

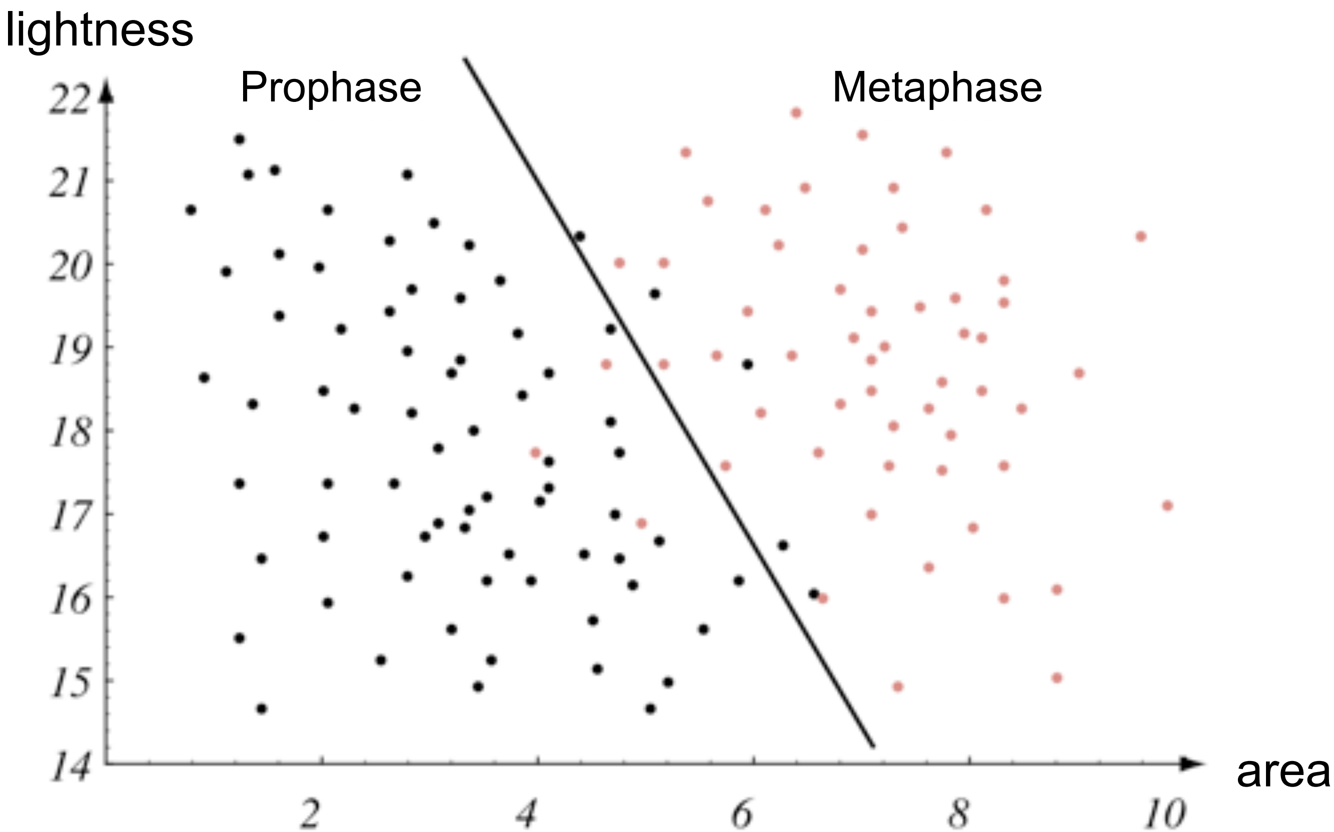


A Simple Least Squares Classifier: d=1




```
y[i]=-1 for pro phase
y[i]=+1 for meta
X[i,]=c(area[i],intensity[i])
model <- lm(y ~ X)
ynew <- predict(model,newdata=Xnew)
ifelse(ynew < 0,-1,1)
```

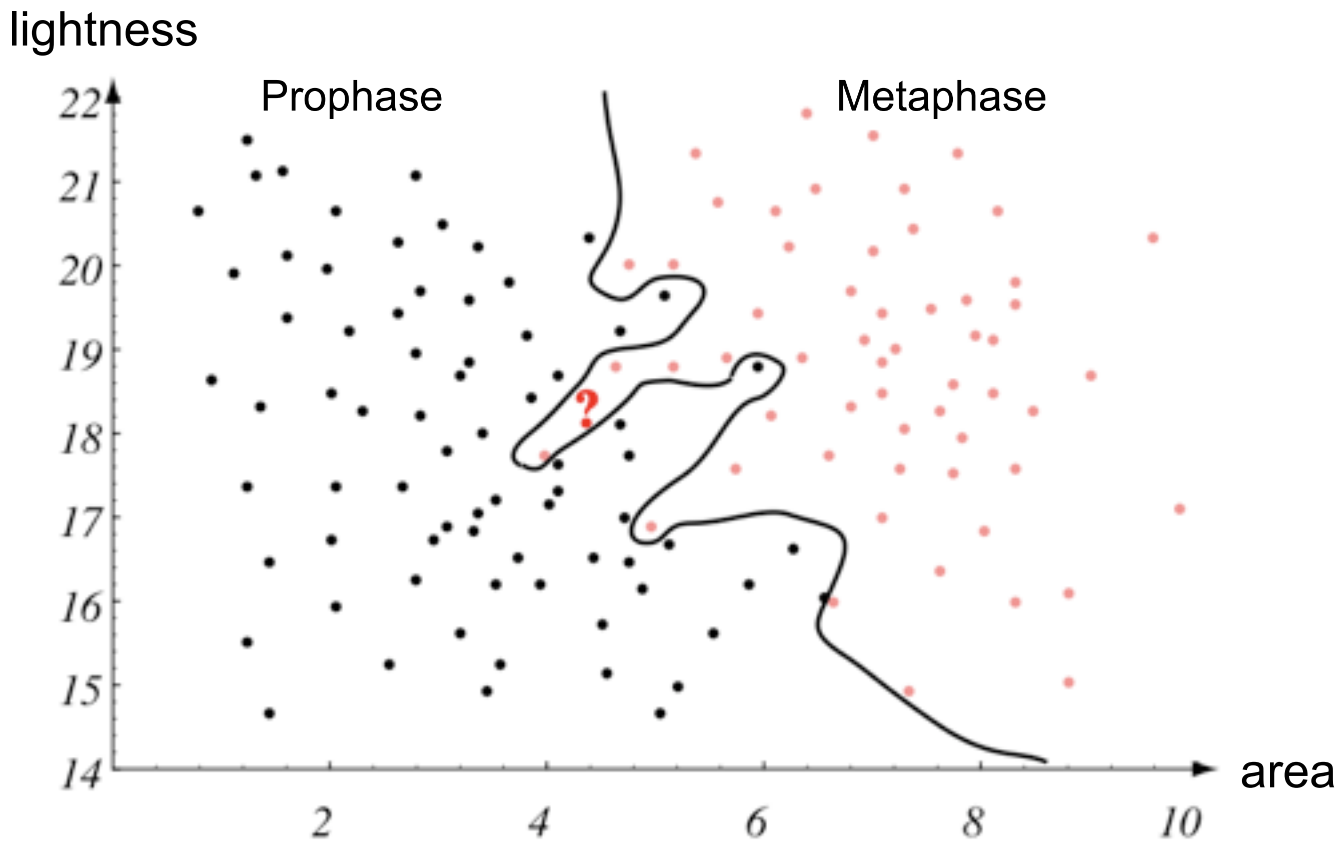
A Simple Least Squares Classifier: d=2



Fit a least squares linear regression model to the data. Black line shows decision boundary

```
y[i]=+1 for prophase
y[i]=-1 for metaphase
X[i,]=(area[i],lightness[i])
model <- lm.fit(X,y)
ynew <- predict(model,Xnew)
             $fitted.values
ifelse(ynew < 0,-1,1)
```

k-Nearest-Neighbor Classifier



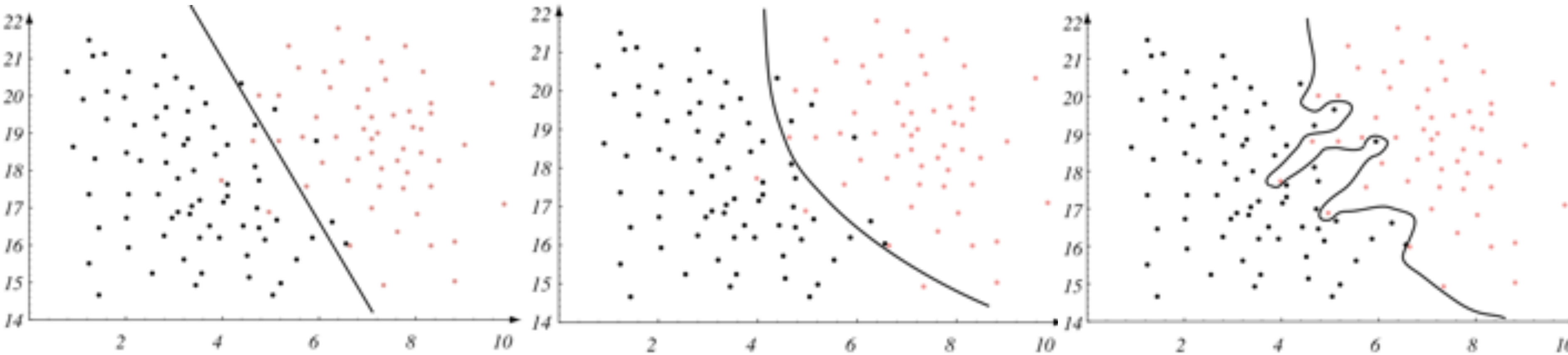
Assign each new cell to the class of its nearest neighbor.
Black line shows decision boundary

```
y[i]=+1 for pro phase
y[i]=-1 for meta phase
X[i,]=(area[i],lightness[i])
library(class)
d = knn(X,Xnew,y,k=1)
```

Which Decision Boundary?

High bias
Low variance

Low bias
High variance

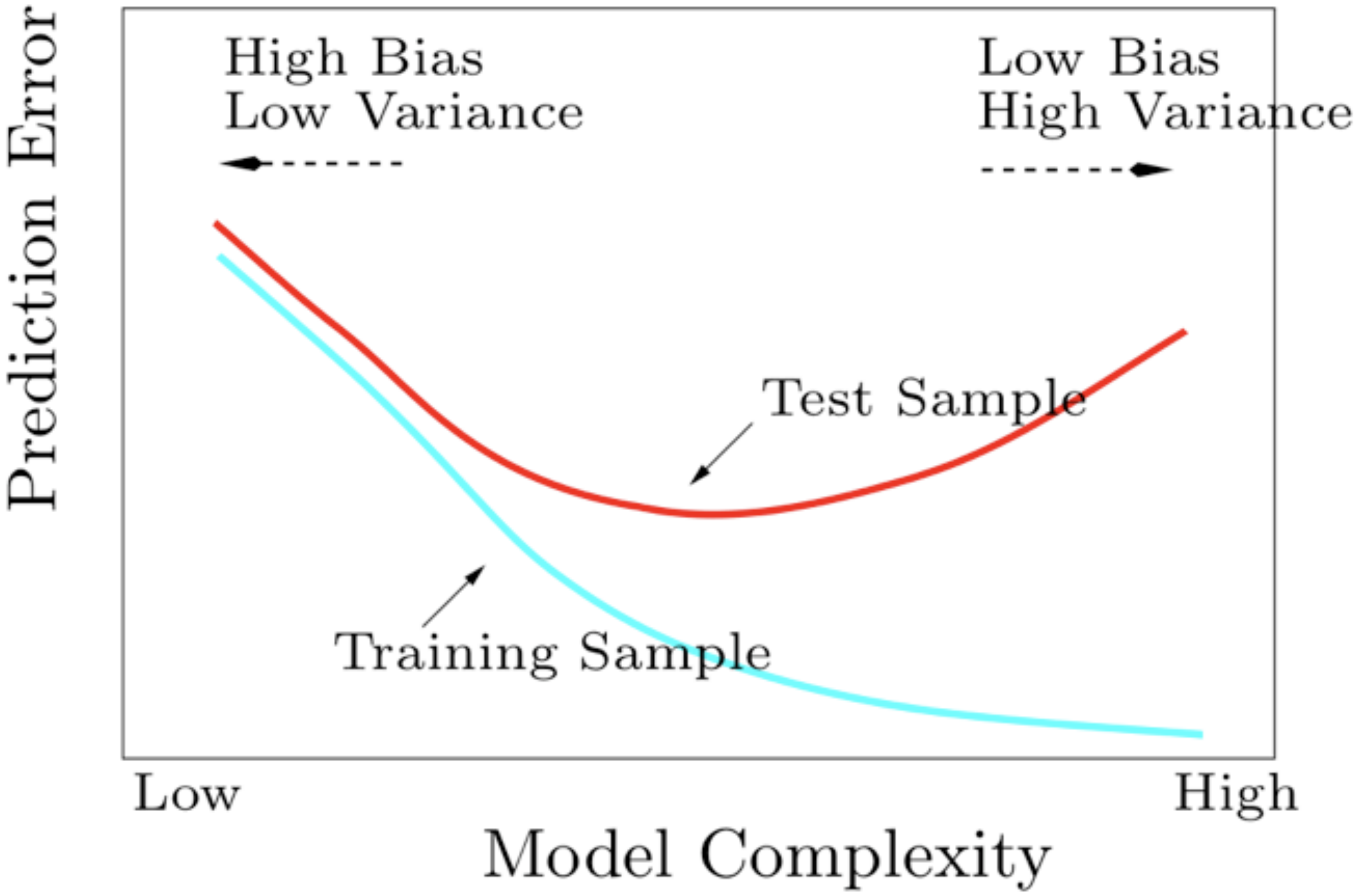


low model complexity
(needs 2 parameters to describe the decision boundary)

high model complexity
(needs hundreds of parameter to describe the decision boundary)

Which decision boundary has the lowest **prediction error**?

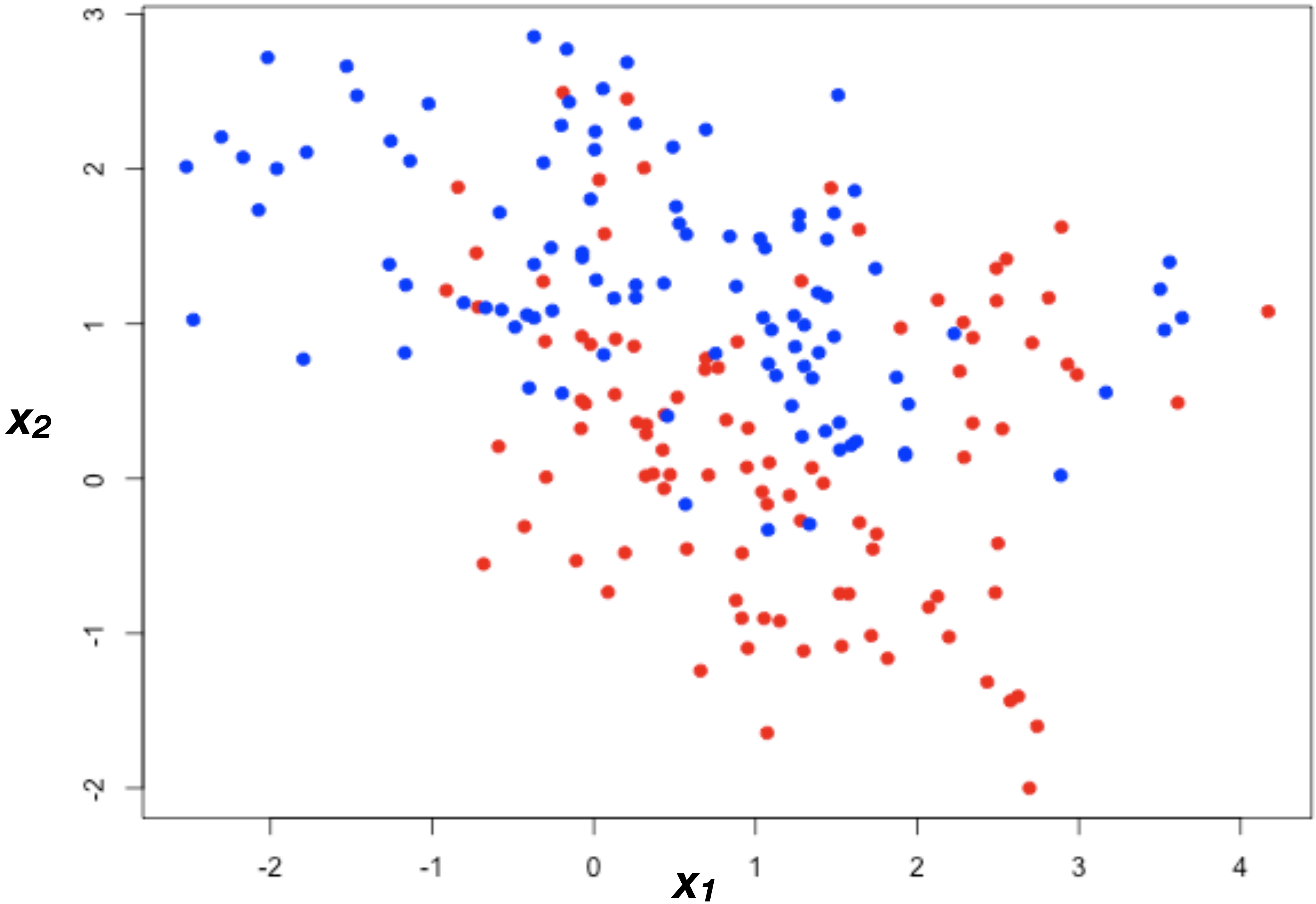
Bias-Variance-Dilemma



Cross-Validation

- cross validation is an easy & useful method to estimate the prediction error.
- The data consist of *n* samples with *d* features and a known class label
- Method (*m*-fold cross-validation):
  - Split the data into *m* approximately equally sized subsets
  - Train the classifier on (*m*-1) subsets
  - Test the classifier on the remaining subset. Estimate the prediction error by comparing the predicted class label with the true class labels.
  - Repeat the last two steps *m* times (use each subset once as test set)

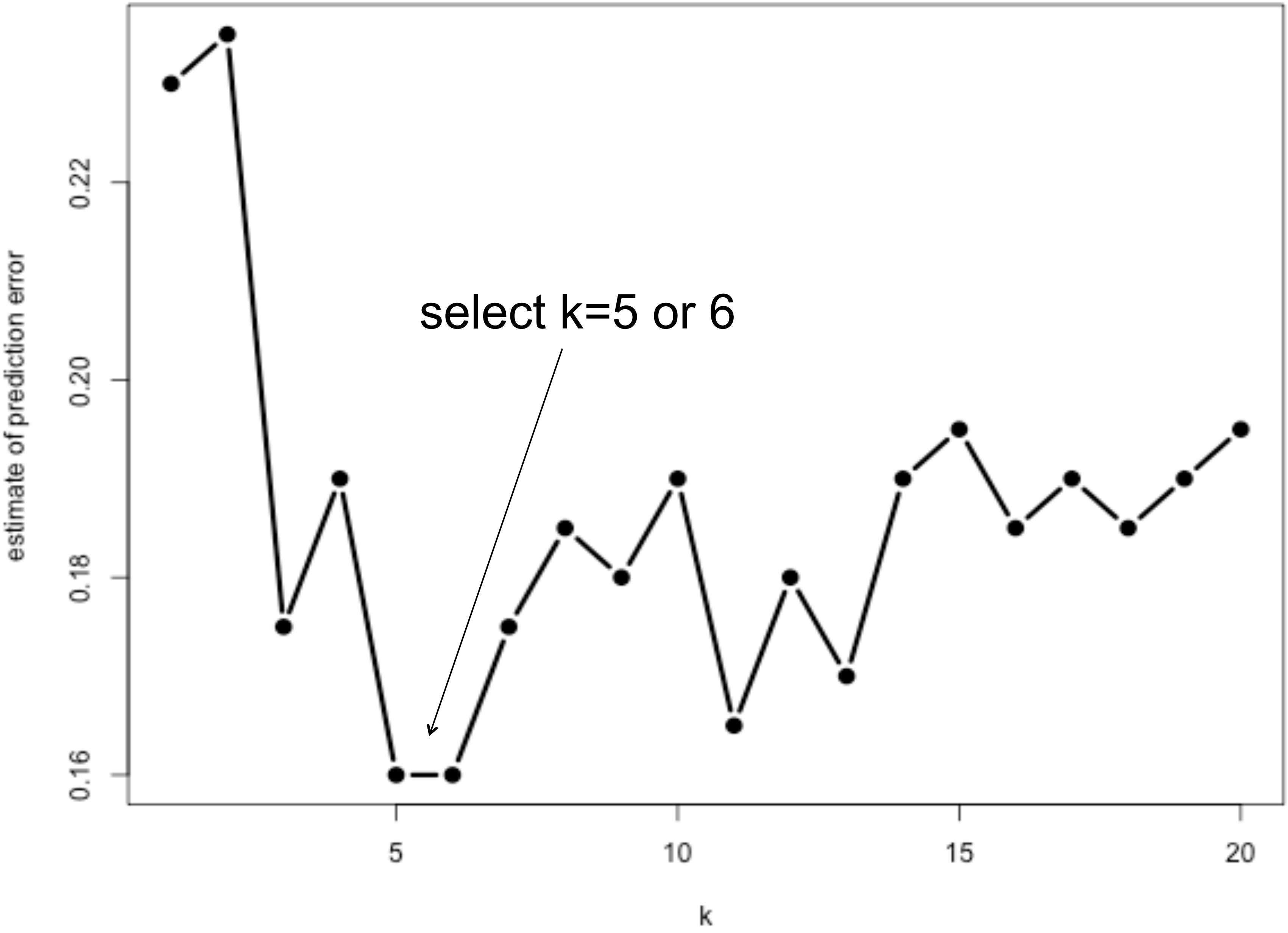
Example: Two classes, two variables, 200 objects



20-fold cross-validation for *k*-nearest neighbours

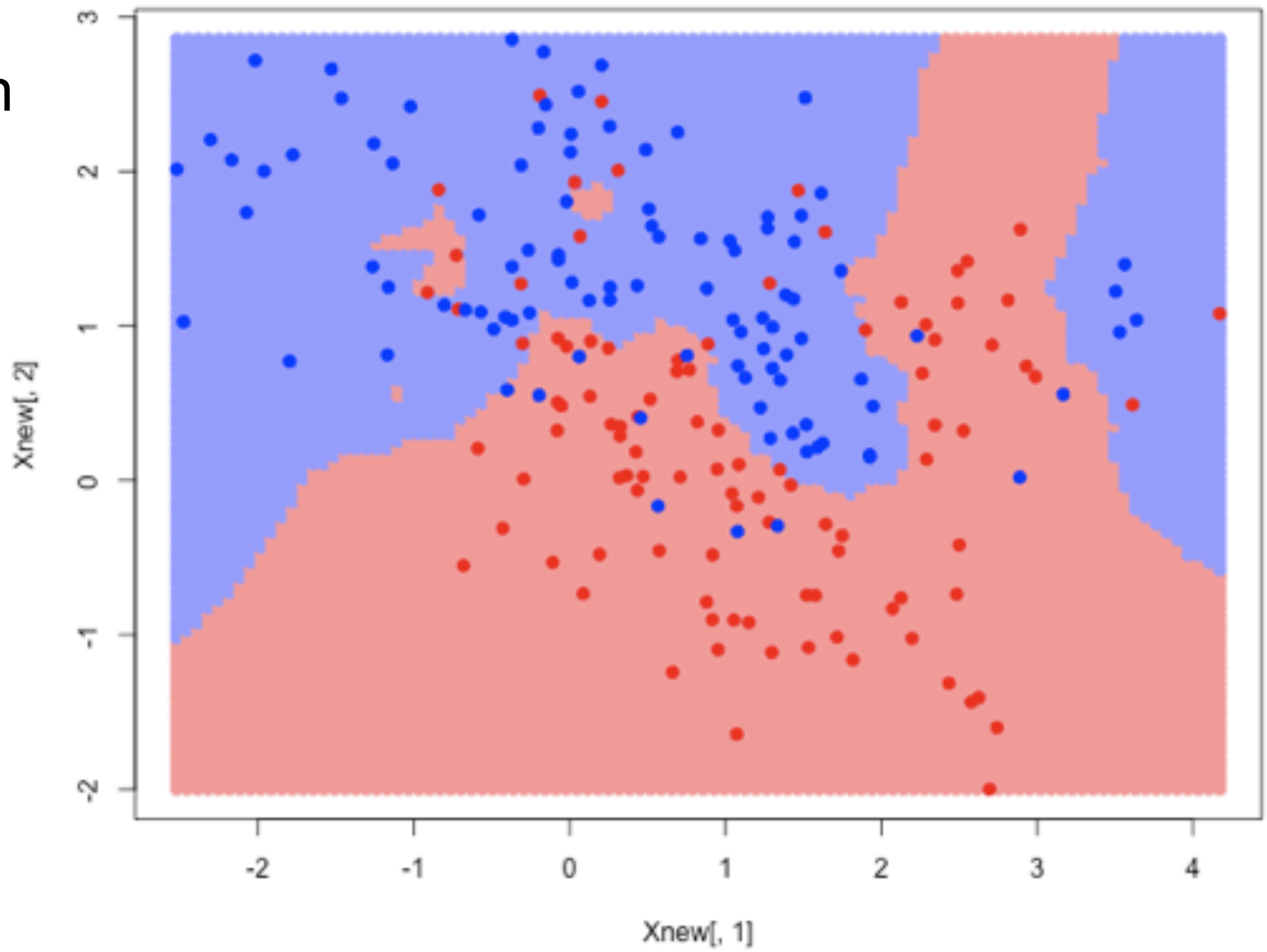
```
> S <- rep(1:10,length.out=200)
 [1]  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5  6  7  8  9 10  1  2  3  4  5  6  7  8 …
> S <- sample(S)  # Random permutation
> Err <- matrix(NA_real_, nrow=20, ncol=10)
> for (k in 1:20) { # Test all k from 1 to 20
+   for (s in 1:10) { # Perform 10-fold cross-validation
+     Xtrain = X[S != s,]
+     ytrain = y[S != s]
+     Xtest = X[S == s,]
+     ytest = y[S == s]
+     ypred = knn(Xtrain,Xtest,ytrain,k)
+     Err[k,s] = sum(ypred != ytest) / length(ytest)
+   }
+ }
> plot(apply(Err,1,mean),xlab="k",ylab="estimate of
  prediction error")
```

cross-Validation for *k*-nearest neighbours



Demo: Cross-Validation for k-nearest neighbours

Classification result (k=5)



The k-nearest neighbour classifier works well with low-dimensional data - but what if the data are high dimensional?

Least Squares Classifier

- X: *n* x *d* matrix with *d*-dimensional features for *n* samples
- y: vector of length *n*.
    - y[i] = 0 for first class, and 1 for second class
- Fit a linear model by minimizing the squared error:

$$\hat{\beta} = \arg\min_{\beta} \|X\beta - y\|_2^2$$

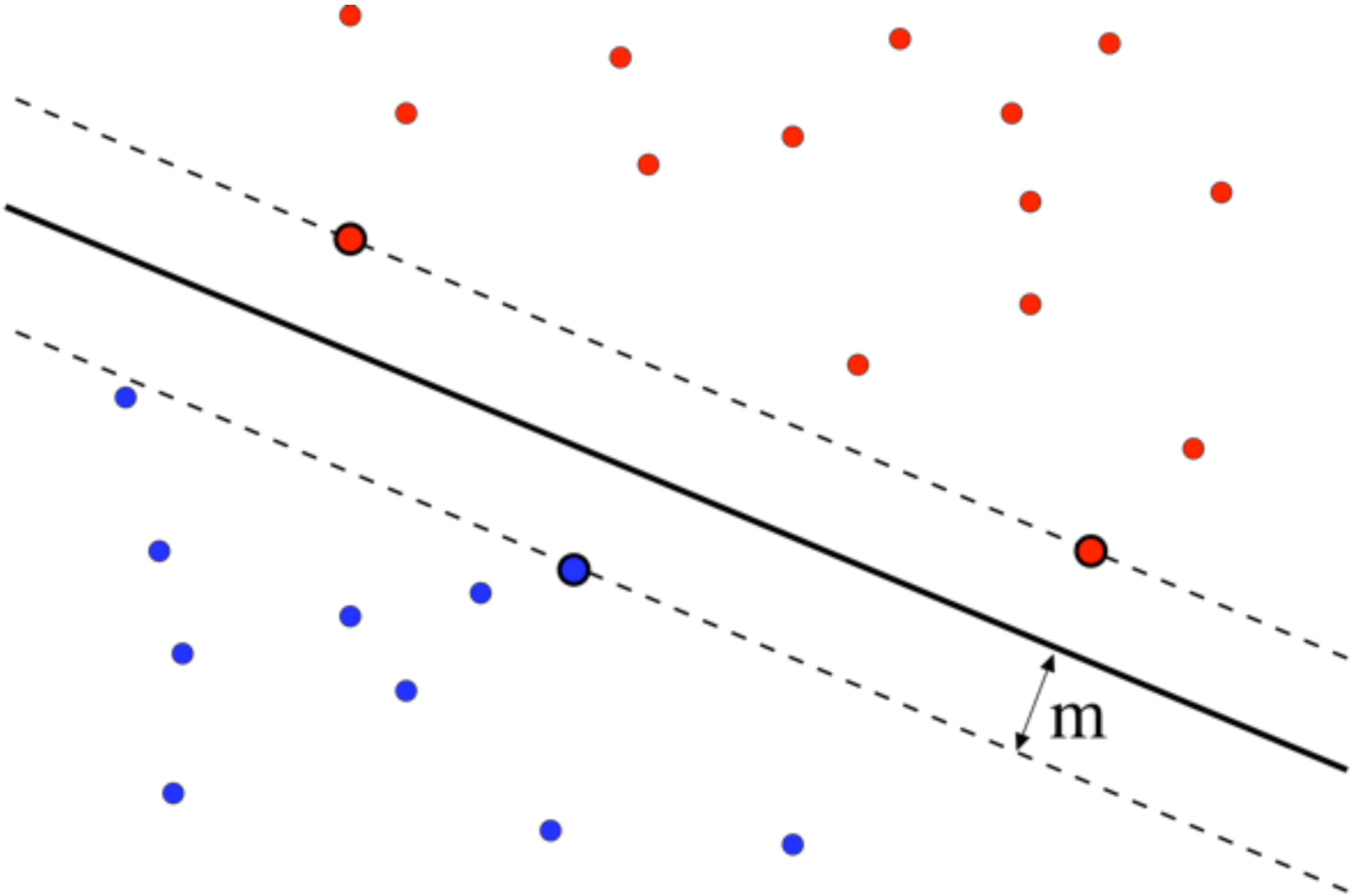
```
> model <- lm.fit(X,y)
> ynew <- predict(model,Xnew)$fitted.values
> ifelse(ynew < 0,-1,1)
```

- Extension to *k* classes (*k* > 2):
- Y is a *n* x *k* indicator matrix.
    - Each row contains exactly one “1” at column *j* if the sample belongs to class *j*. All other entries are zero.

In practice: **lda** (R-package MASS)

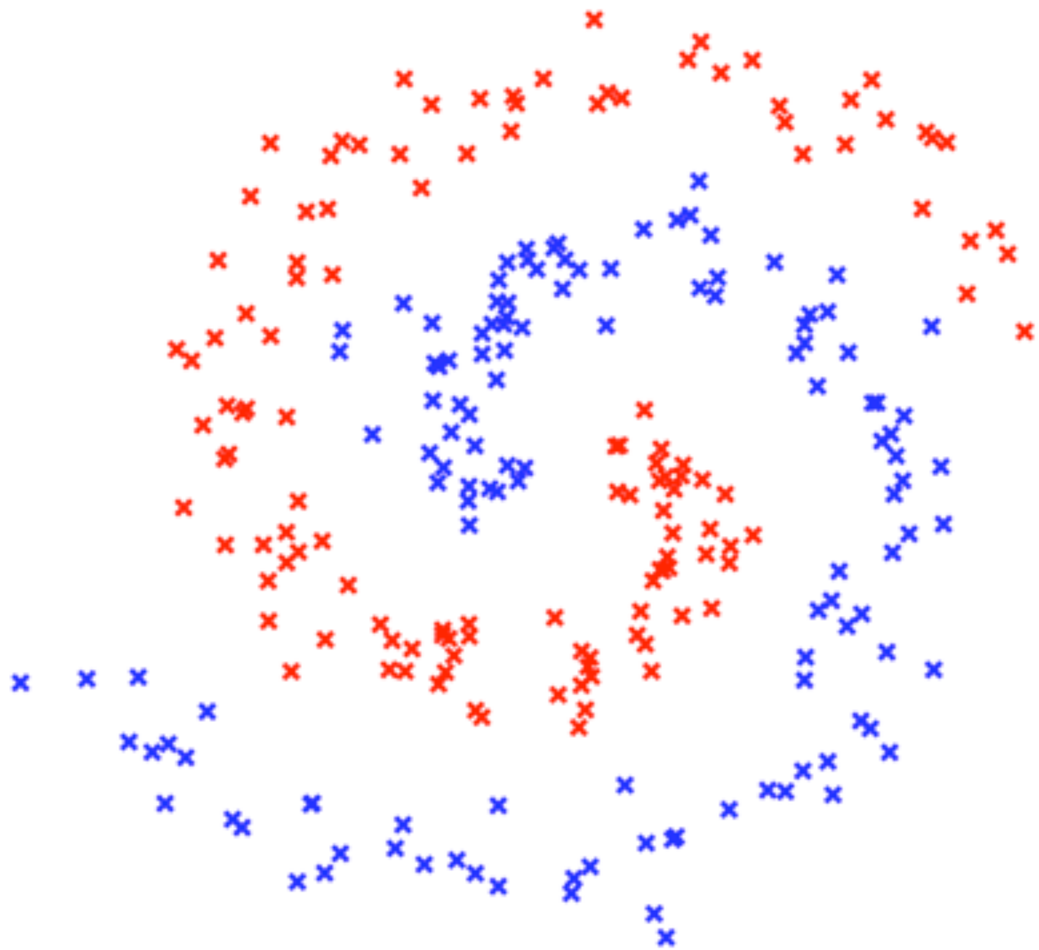
Support Vector Machine

- Find a separating hyperplane with maximal margin to the samples



Non-Linear Classifiers

These classes can not be separated by a linear hyperplane



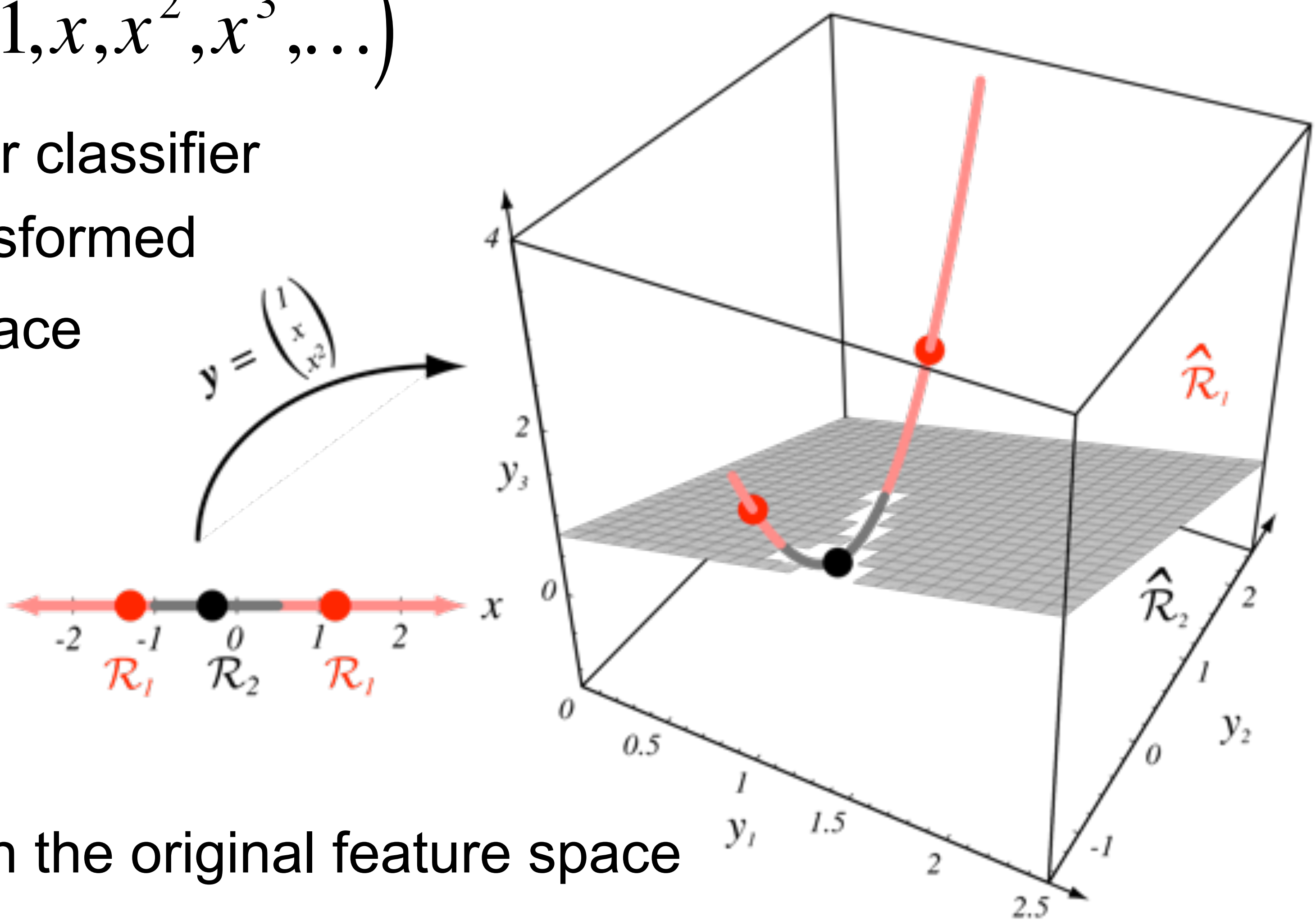
Feature Transformation

Transform the data with non-linear function, e.g.

$$f(x) = \left(1, x, x^2, x^3, \ldots\right)$$

Train linear classifier in the transformed feature space

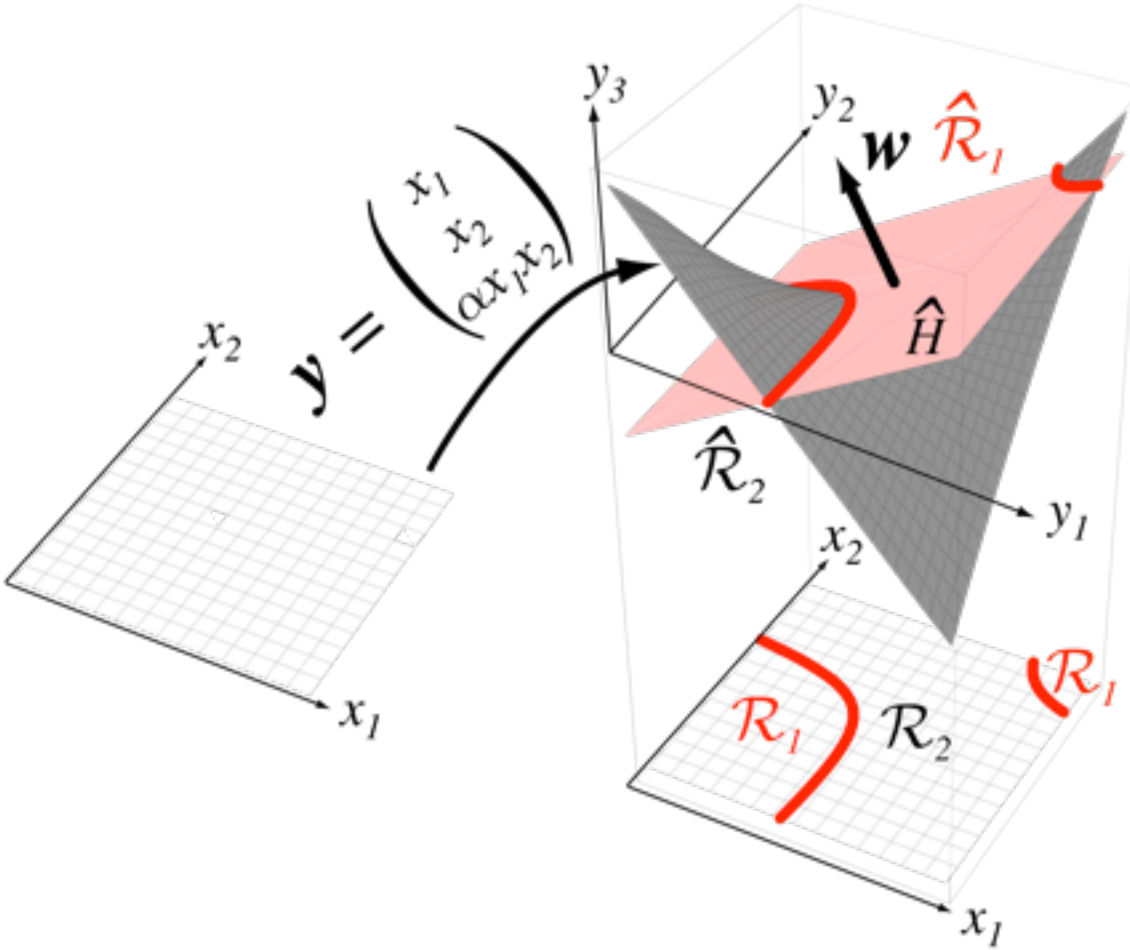
→ non-linear classifier in the original feature space



EMBL

Quadratic Extension

- Parabolic decision boundaries can be achieved by extending by the product $x_1 x_2$.



The Kernel Trick

Rewrite the model such that the features only appear within scalar products.

Example: least squares $$\hat{\beta} = \arg\min \|X\beta - y\|_2^2$$

It can be shown that there exists an α such that $\beta = X^t\alpha$

(Note β is *d*-dim.; α is *n*-dim.) $$\hat{\alpha} = \arg\min_{\alpha} \|XX'\alpha - y\|_2^2$$

The least squares problem can be reformulated as a scalar product.

The n x n matrix $S = XX^t$ contains all scalar products ($S_{ij} = x_i \cdot x_j$). Replace S_{ij} by $K_{ij} = K(x_i, x_j)$ that implicitly performs a feature transformation and the computation of the scalar product. The kernel matrix has to be positive semi-definite.

The Kernel Trick

Popular functions :

Linear kernel: $K(x_i, x_j) = x_i x_j$

Radial basis functions: $K(x_i, x_j) = \exp\left(-\frac{1}{2\sigma^2}\left\|x_i - x_j\right\|\right)$

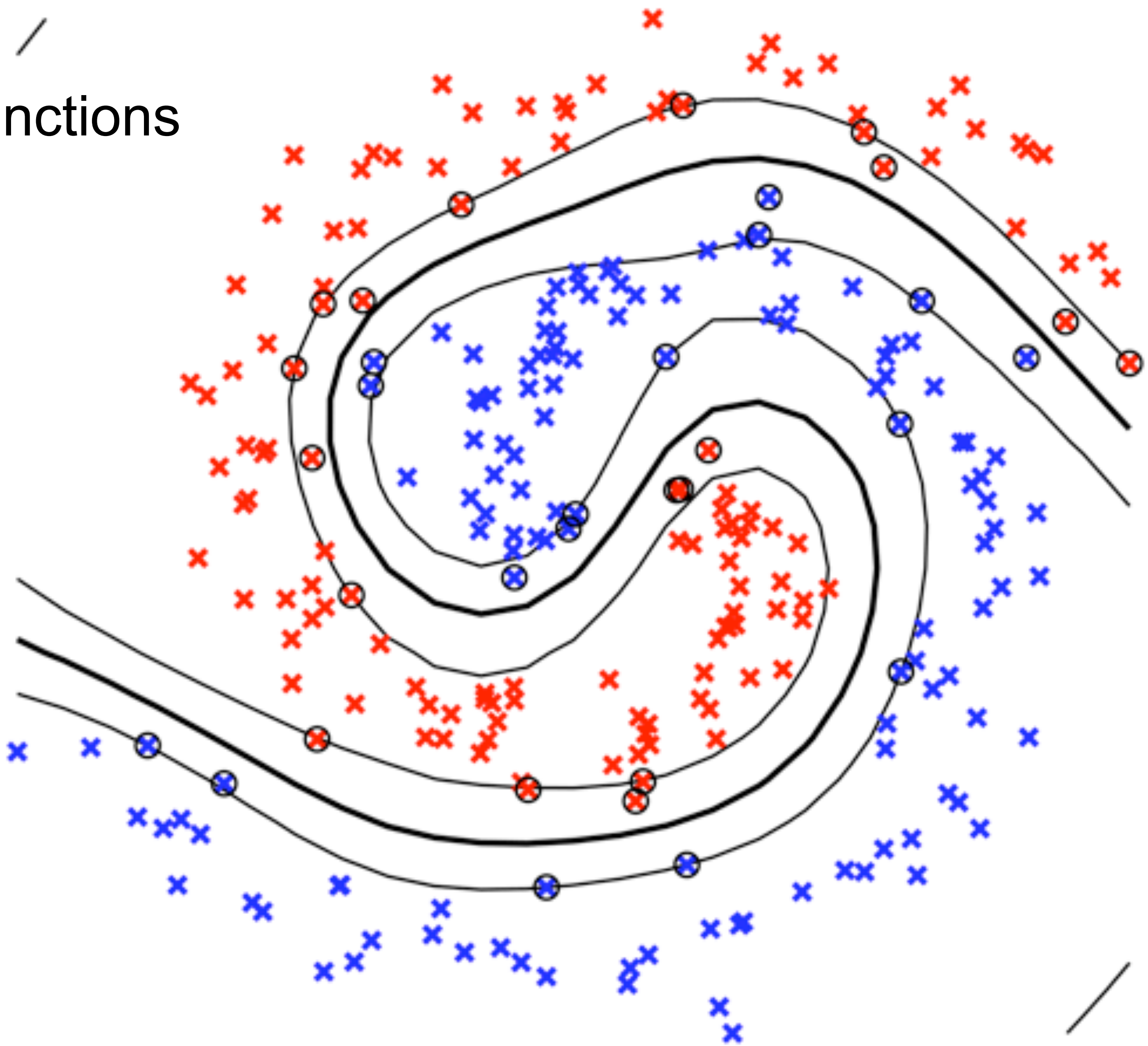
Polynomial kernel $K(x_i, x_j) = (x_i x_j + 1)^d$

Examples for SVM-Classification

SVM with

Radial Basis Functions

(RBF-kernel)



Thick line:

class separating

hyperplane

Thin line:

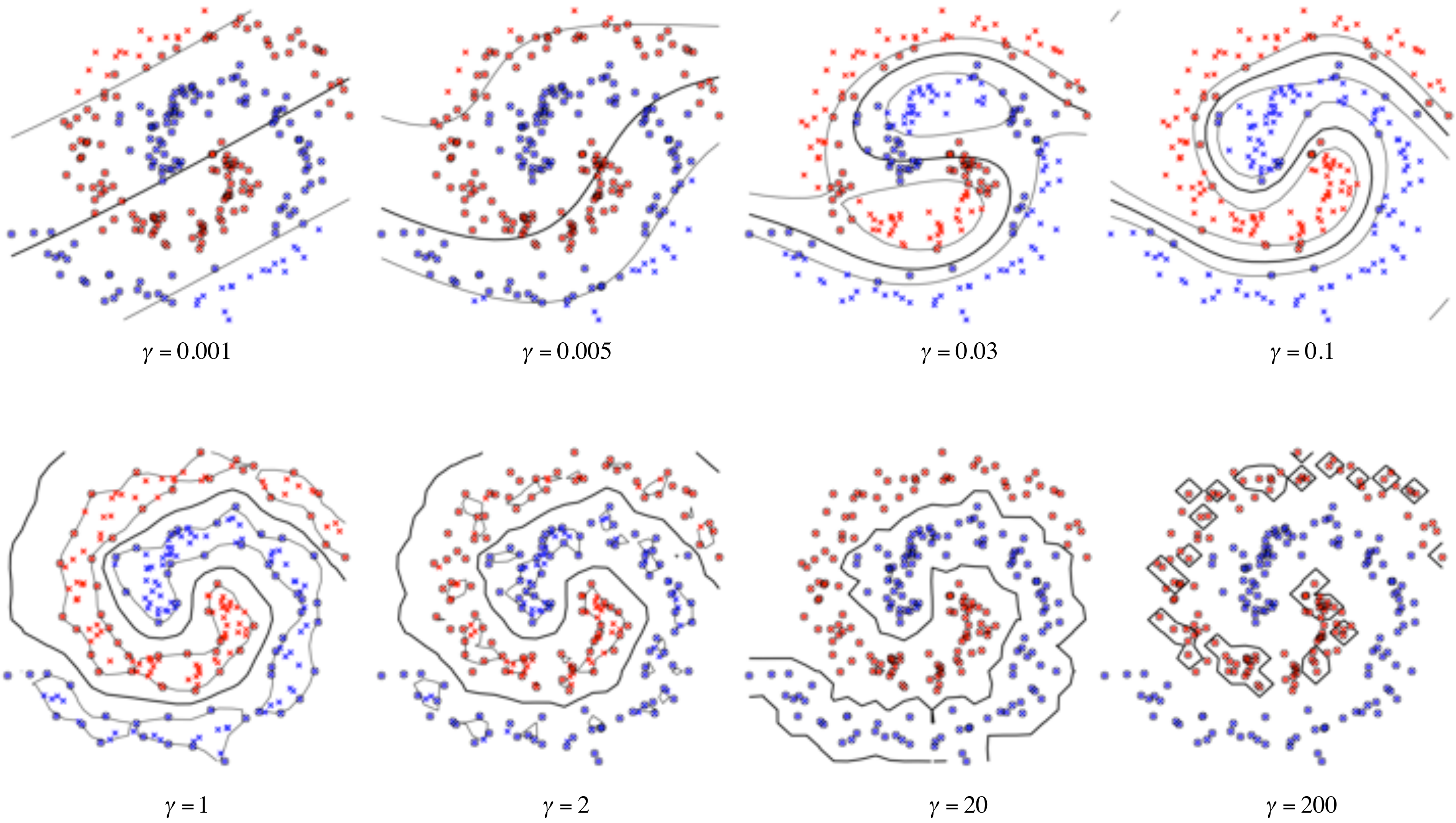
margin

Circles:

support vectors

EMBL

The Influence of the Kernel Parameter



$\gamma = \sigma^{-2}$, RBF

Curse of Dimensionality

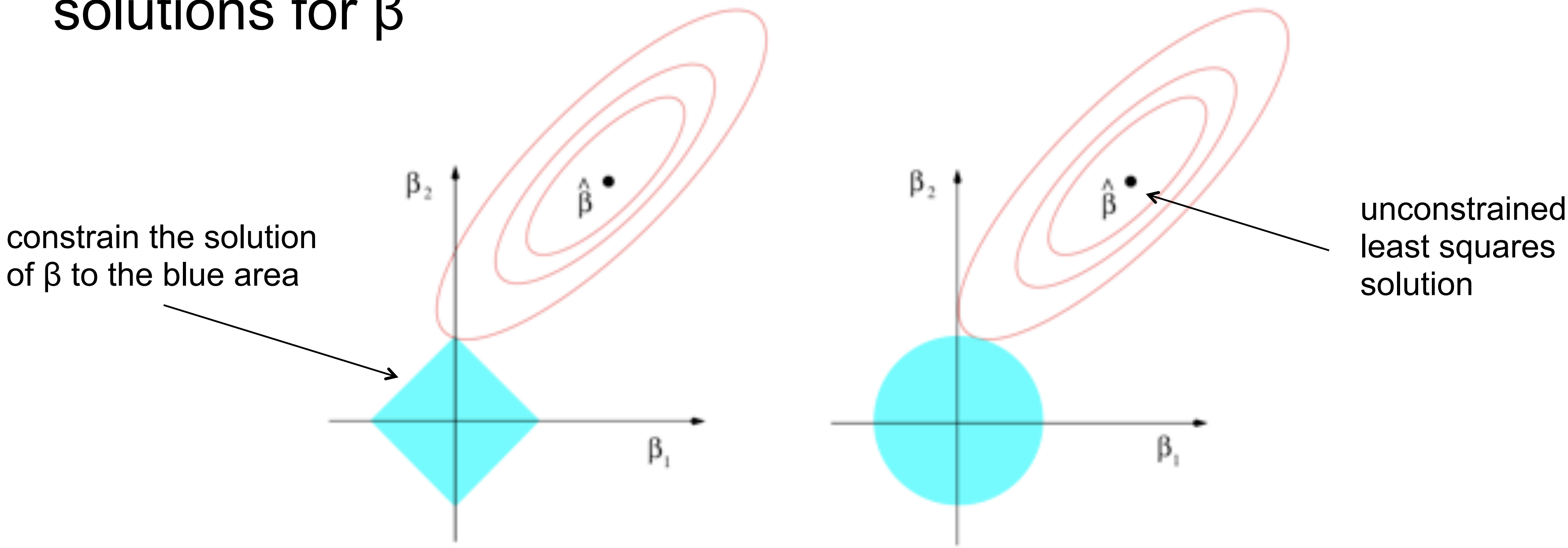
- Consider:
  - 10 samples per class
  - Each sample is characterised by several hundred features.
- Even a linear classifier will be (always) too complex: overfitting
- There is a need to lower the complexity even below that of the linear classifier

```
# file: demo-random.R
> X = matrix(rnorm(20*25000),
        nr=20,nc=25000)
> y = c(rep(-1,10),rep(1,10))
>
> # Fit a linear model
         by least squares
> model = lm.fit(X,y)
```

```
> # The two groups are perfectly separated!
> ynew = model$fit
> ynew = ifelse(ynew < 0, -1, 1)
> print("The predicted label of the training set")
[1] "The predicted label of the training set"
> print(ynew)
 [1] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1  1  1  1  1  1  1  1  1  1
> print("The true label of the training set")
[1] "The true label of the training set"
> print(y)
 [1] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  1  1  1  1  1  1  1  1  1  1
```

Regularization

- Reduce the complexity by reducing the space of permissible solutions for β



Lasso:

$$\hat{\beta} = \arg\min_{\beta} \|X\beta - y\|_2^2 + \lambda\|\beta\|_1^1$$

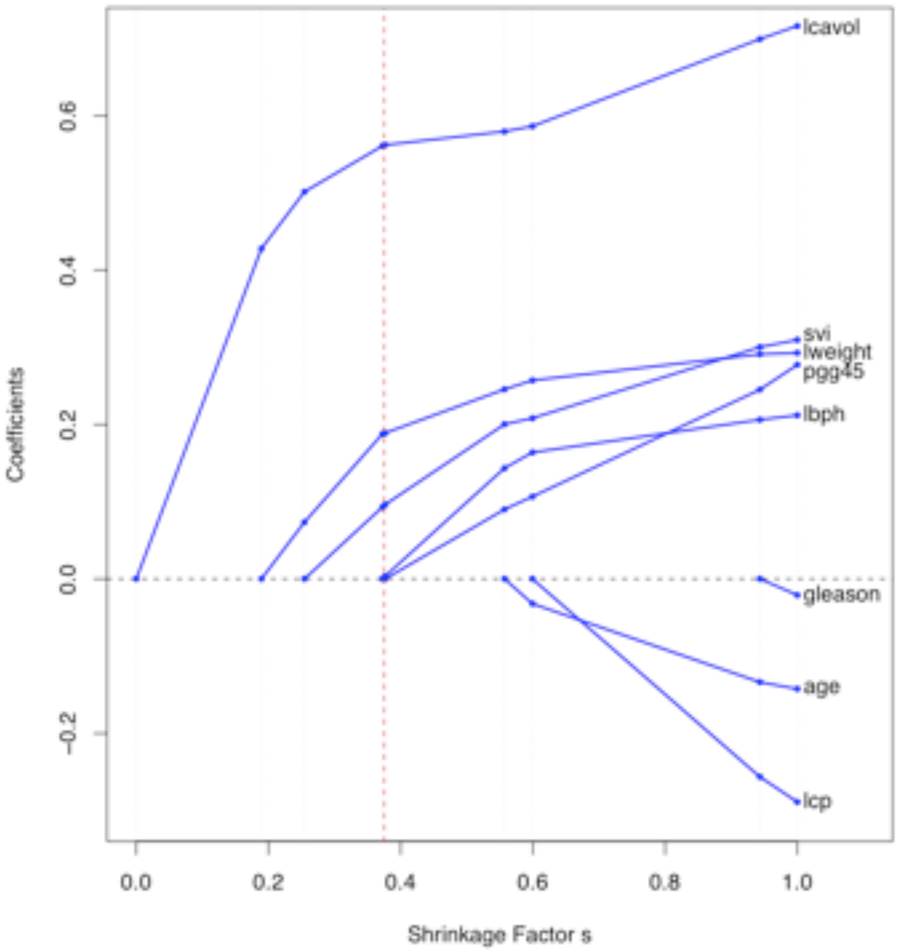
Ridge Regression

$$\hat{\beta} = \arg\min_{\beta} \|X\beta - y\|_2^2 + \lambda\|\beta\|_2^2$$

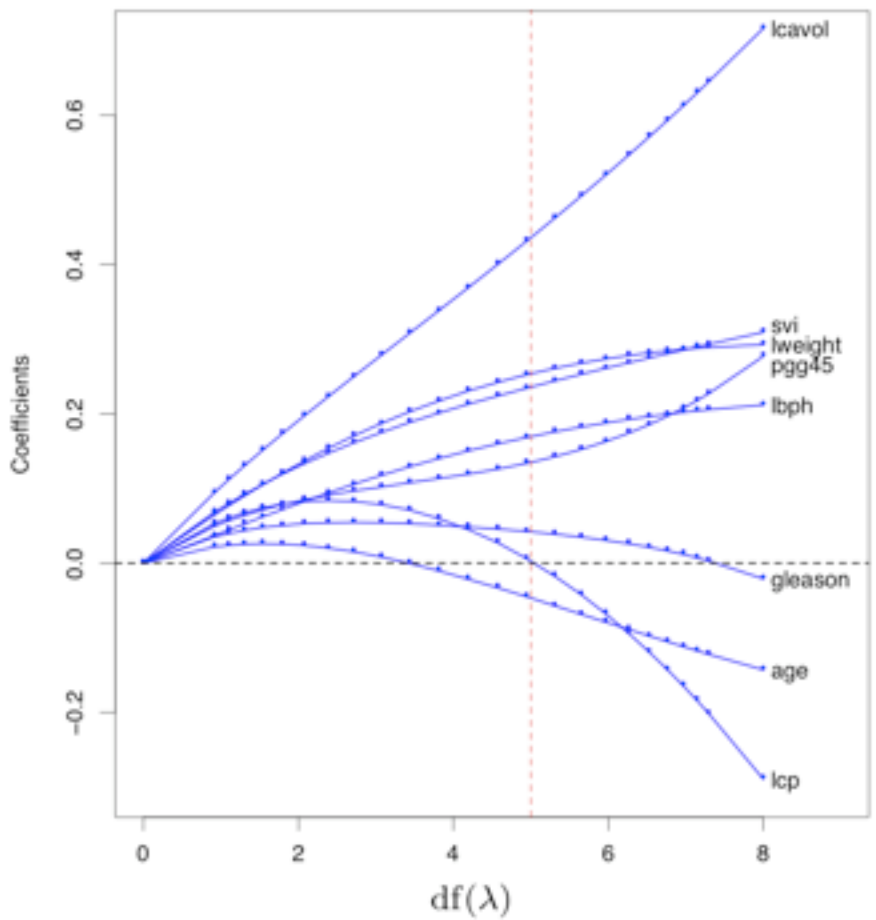
Lagrangian formulation of constrained optimization.
The blue area becomes larger, the smaller λ.
Lasso: sparse solution. Many coefficients βi become 0. Only a few coefficients are used for prediction. Implicitly **selects features**.

Regularization Path

The coefficients for varying regularization parameter λ

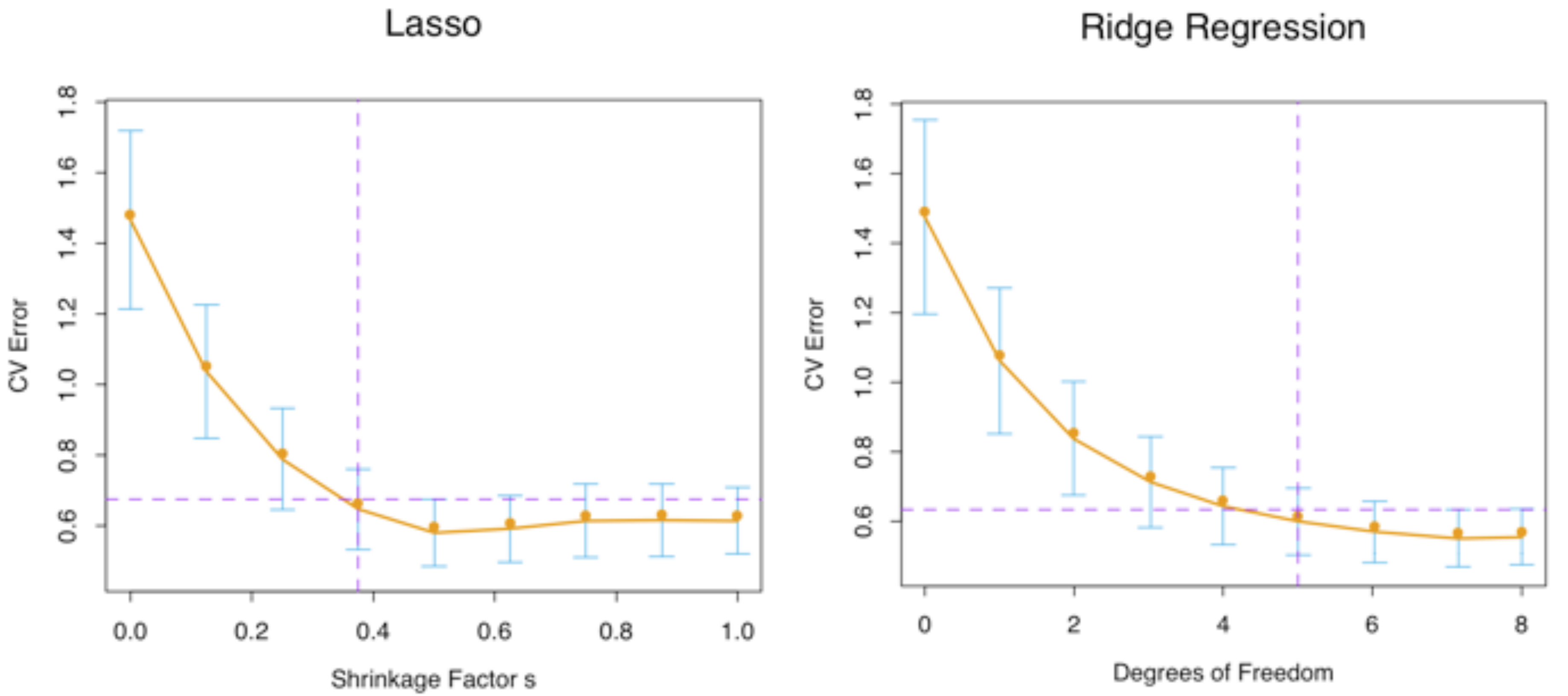


Lasso



Ridge Regression

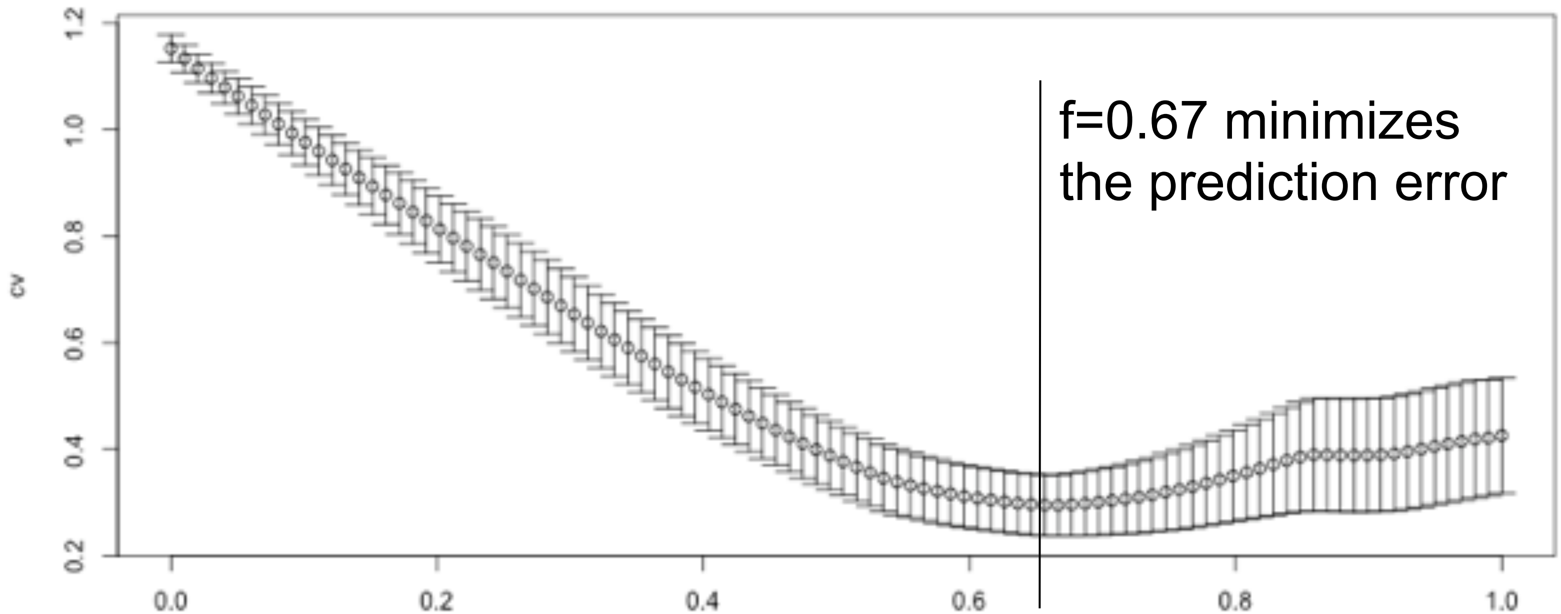
Cross-Validation for Regularized Regression



Demo Lasso I

- ALL cancer dataset: gene expression of 12000 genes
- Two classes B-cell ALL and T-cell ALL.
- Cross validation over a range of λ-values

```
# filename: demo-lars.R
>CV <- cv.lars(X,y,use.Gram=FALSE,trace=TRUE)
```



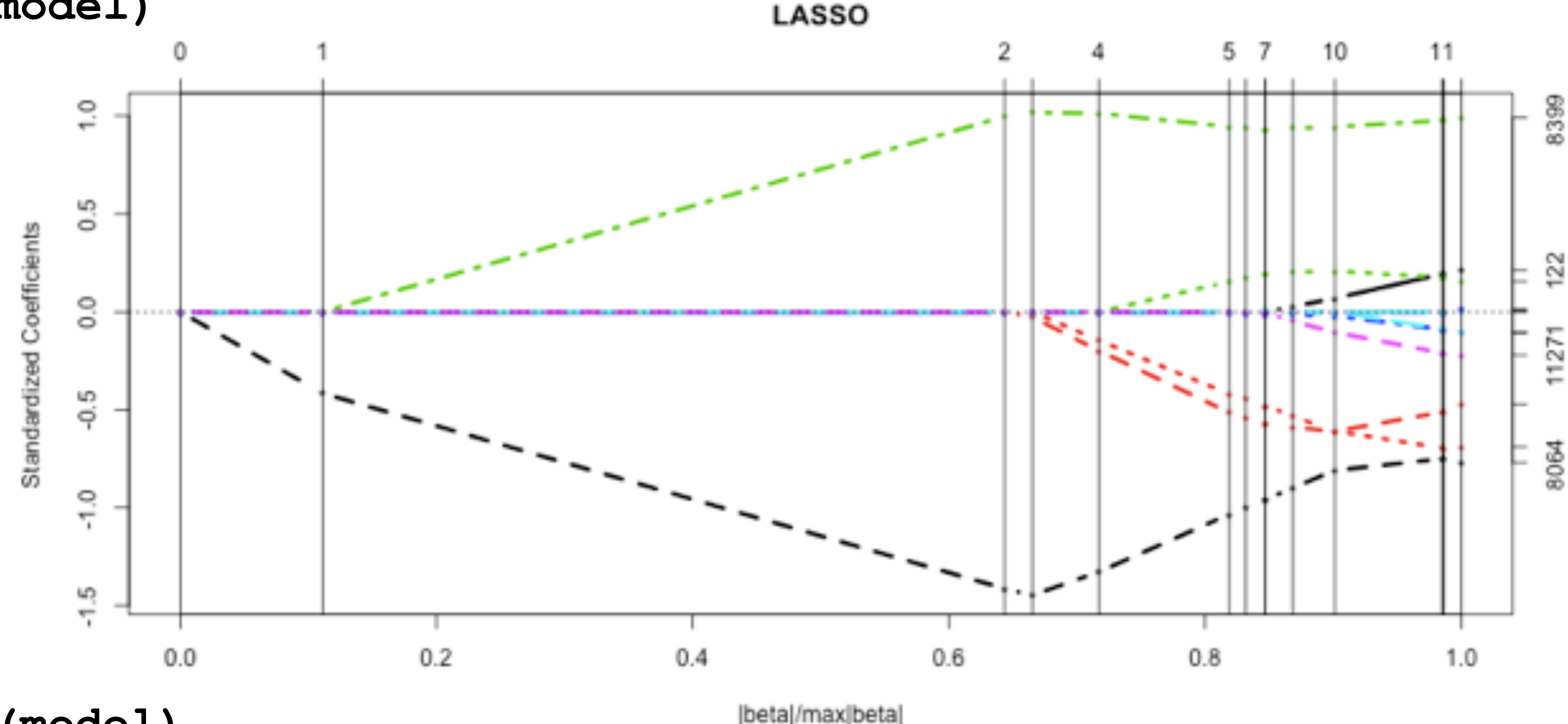
- Choose the fraction f of |β| that minimizes the prediction error

```
>f <- CV$fraction[which.min(CV$cv)]
```


EMBL

Demo Lasso II

```
> model <- lars(X,y,use.Gram=FALSE,trace=TRUE)
> plot(model)
```

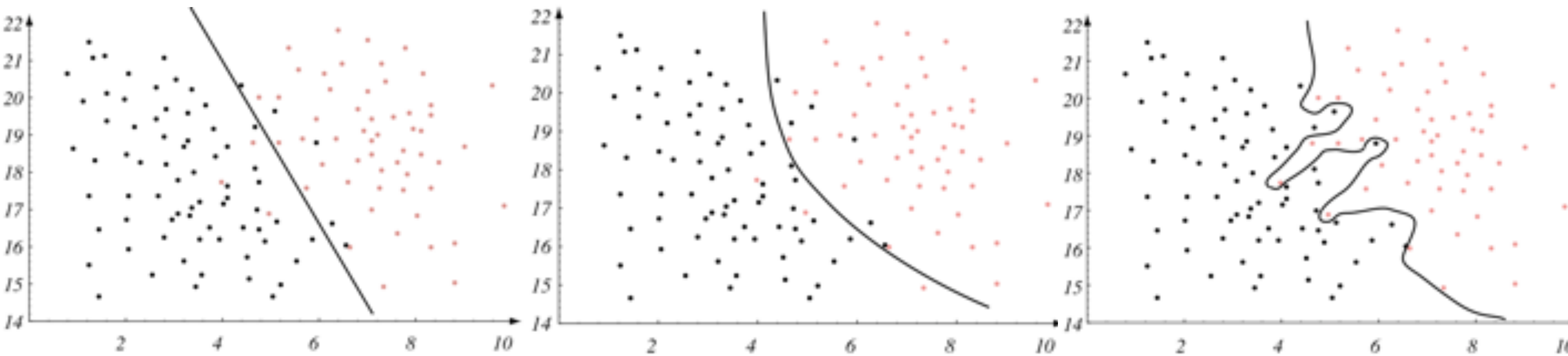



```
> print(model)
Sequence of LASSO moves:
     37988_at 38319_at 2031_s_at 38242_at 34908_at 35434_at …
Var      8064     8399      1144     8321     4955     5486 …
Step        1        2         3        4        5        6 …
```

Summary: It's all about adapting the complexity of the model to that of the data

High bias
Low variance

Low bias
High variance



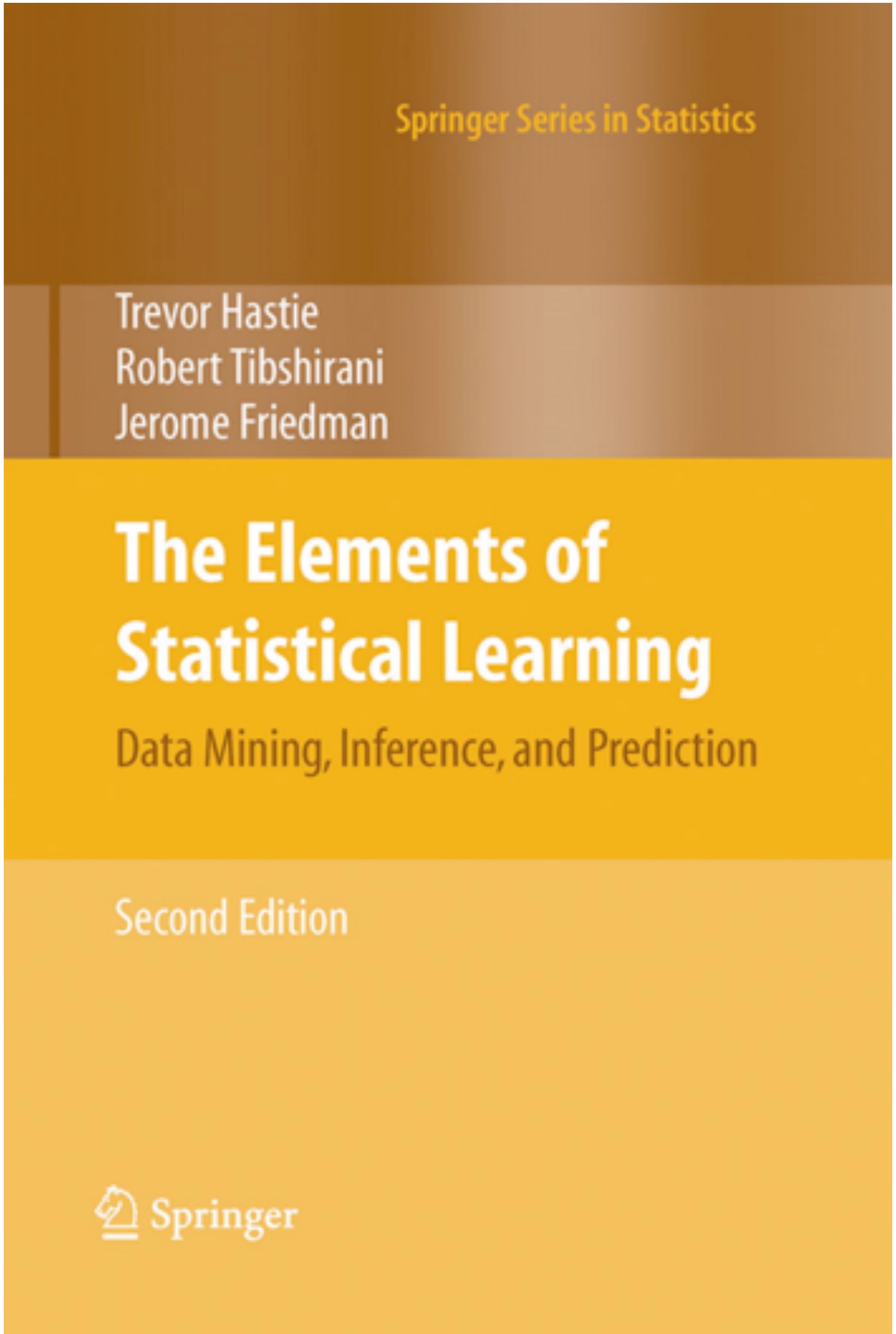
low model complexity
(2 parameters describe the decision boundary)

high model complexity
(hundreds of parameter to describe the decision boundary)

Reduce complexity by regularization (Lasso, ridge, …)

Increase complexity by feature transformation or kernel functions

Always assess classifiers by cross-validation


Springer Series in Statistics
Trevor Hastie
Robert Tibshirani
Jerome Friedman
The Elements of
Statistical Learning
Data Mining, Inference, and Prediction
Second Edition
Springer