Model Selection and Cross Validation

Objective

- Cross validation is a method which tries to select the best model from a given set of models.
- Assumption: Quality measure is predictive performance.
- "Set of models" can simply mean "set of different parameter values".

Terminology

The problem of choosing a good model is called **model selection**.

Model selection problem for SVM

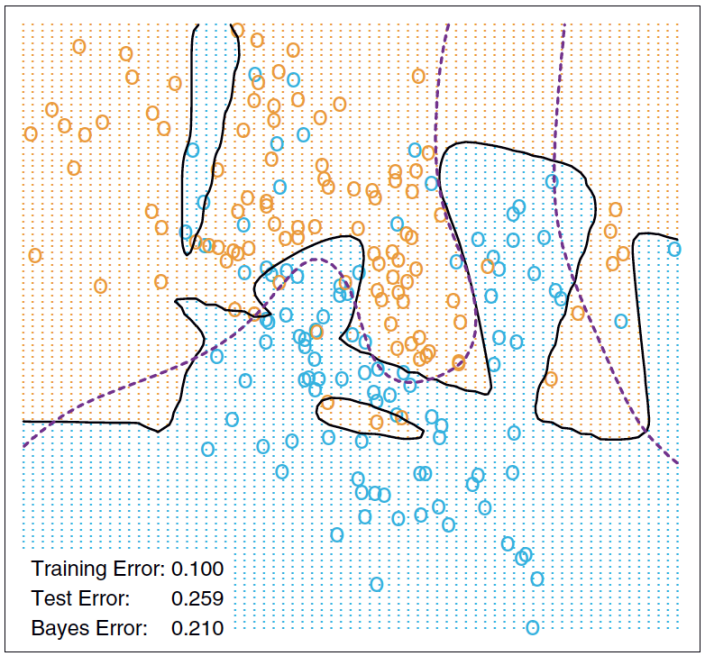
- The SVM is a *family* of models indexed by the margin parameter γ and the kernel parameter(s) σ.
- Our goal is to find a value of (γ, σ) for which we can expect small generalization error.

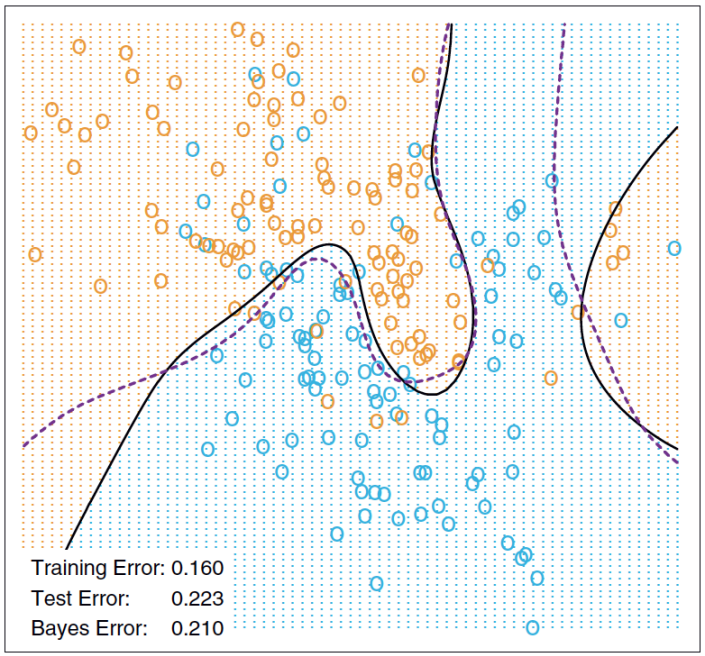
Naive approach

- We could include (γ, σ) into the optimization problem, i.e. train by minimizing over α *and* (γ, σ).
- This leads to a phenomenon called **overfitting**: The classifier adapts too closely to specific properties of the training data, rather than the underlying distribution.

Overfitting: Illustration

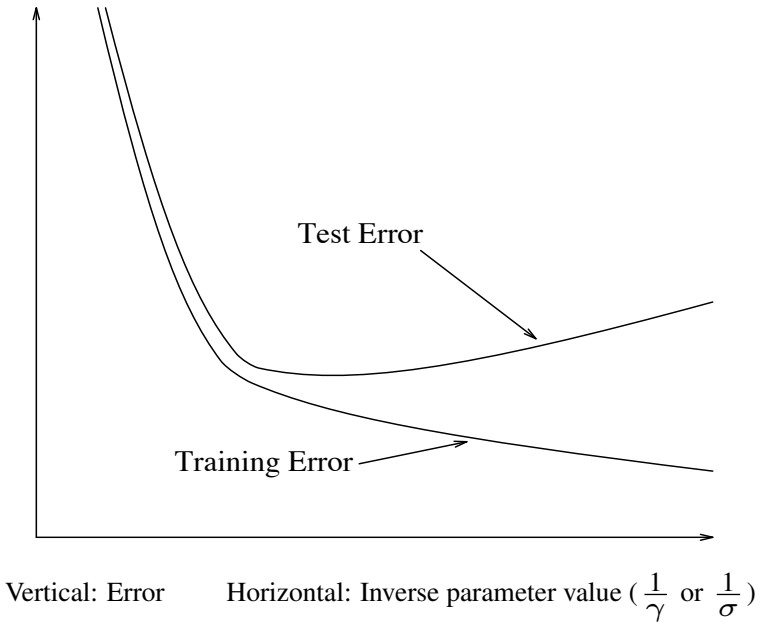
Overfitting is best illustrated with a *nonlinear* classifier.





- The classifier in this example only has a "bandwidth" parameter σ, similar to the parameter σ of the RBF kernel.
- Small σ permits curve with sharp bends; large σ: Smooth curve.

Conceptual illustration



- If classifier can adapt (too) well to data: Small training error, but possibly large test error.
- If classifier can hardly adapt at all: Large training and test error.
- Somewhere in between, there is a sweet spot.
- Trade-off is controlled by the parameter.

Model Selection by Cross Validation

(From now on, we just write γ to denote the entire set of model parameters.)

Cross Validation: Procedure

Model selection:

1. Randomly split data into three sets: training, validation and test data.

2. Train classifier on training data for different values of γ.
3. Evaluate each trained classifier on validation data (ie compute error rate).
4. Select the value of γ with lowest error rate.

Model assessment:

5. Finally: Estimate the error rate of the selected classifier on test data.

Meaning

- The quality measure by which we are comparing different classifiers $f(\,.\,;\gamma)$ (for different parameter values γ) is the risk

$$R(f(\,.\,;\gamma)) = \mathbb{E}[L(y, f(x;\gamma))]\,.$$

- Since we do not know the true risk, we estimate it from data as $\hat{R}(f(\,.\,;\gamma))$.

Importance of model assessment step

- We always have to assume: Classifier is better adapted to *any* data used to select it than to actual data distribution.
- Model selection: Adapts classifier to *both* training and validation data.
- If we estimate error rate on this data, we will in general underestimate it.

Procedure in detail

We consider possible parameter values $\gamma_1, \ldots, \gamma_m$.

1. For each value γ_j, train a classifier $f(\,.\,;\gamma_j)$ on the training set.
2. Use the validation set to estimate $R(f(\,.\,;\gamma_j))$ as the empirical risk

$$\hat{R}(f(x;\gamma_j)) = \frac{1}{n_v}\sum_{i=1}^{n_v} L(\tilde{y}_i, f(\tilde{\mathbf{x}}_i, \gamma_j)) \ .$$

   n_v is the size of the validation set.
3. Select the value γ^* which achieves the smallest estimated error.
4. Re-train the classifier with parameter γ^* on all data except the test set (i.e. on training + validation data).
5. Report error estimate $\hat{R}(f(\,.\,;\gamma^*))$ computed on the *test* set.

K-FOLD CROSS VALIDATION

Idea

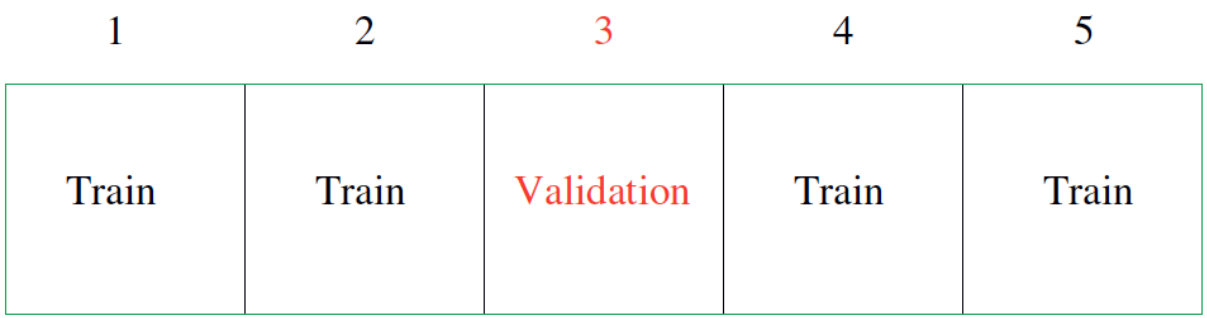
Each of the error estimates computed on validation set is computed from a single example of a trained classifier. Can we improve the estimate?

Strategy

- Set aside the test set.
- Split the remaining data into K blocks (called "folds").
- Use each fold in turn as validation set. Perform cross validation and average the results over all K combinations.

This method is called **K-fold cross validation**.

Example: K=5, step k=3



K-FOLD CROSS VALIDATION: PROCEDURE

Risk estimation

To estimate the risk of a classifier $f(\,.\,,\gamma_j)$:

1. Split data into K equally sized parts.
2. Train an instance $f_k(\,.\,,\gamma_j)$ of the classifier, using all folds except fold k as training data.
3. Compute the cross validation estimate

$$\hat{R}_{\mathrm{CV}}(f(\,.\,,\gamma_j)) := \frac{1}{K}\sum_{k=1}^{K}\frac{1}{|\text{fold } k|}\sum_{(\tilde{\mathbf{x}},\tilde{y})\in \text{ fold } k} L(\tilde{y},f_k(\tilde{\mathbf{x}},\gamma_j))$$

Repeat this for all parameter values $\gamma_1,\ldots,\gamma_m$.

Selecting a model

Choose the parameter value γ^* for which estimated risk is minimal.

Model assessment

Report risk estimate for $f(\,.\,,\gamma^*)$ computed on *test* data.

How to Choose K?

Extremal cases

- $K = n$, called **leave one out cross validation** (loocv)
- $K = 2$

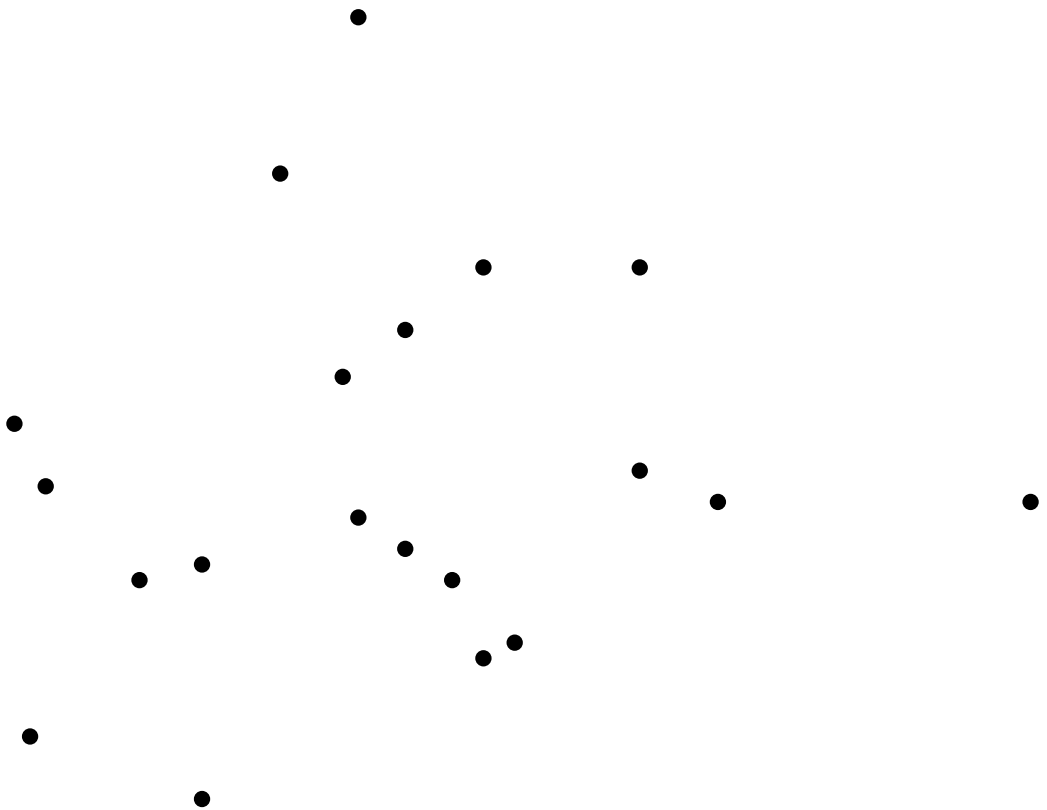
An often-cited problem with loocv is that we have to train many (= n) classifiers, but there is also a deeper problem.

Argument 1: K should be small, e.g. $K = 2$

- Unless we have a lot of data, variance between two distinct training sets may be considerable.
- **Important concept:** By removing substantial parts of the sample in turn and at random, we can simulate this variance.
- By removing a single point (loocv), we cannot make this variance visible.

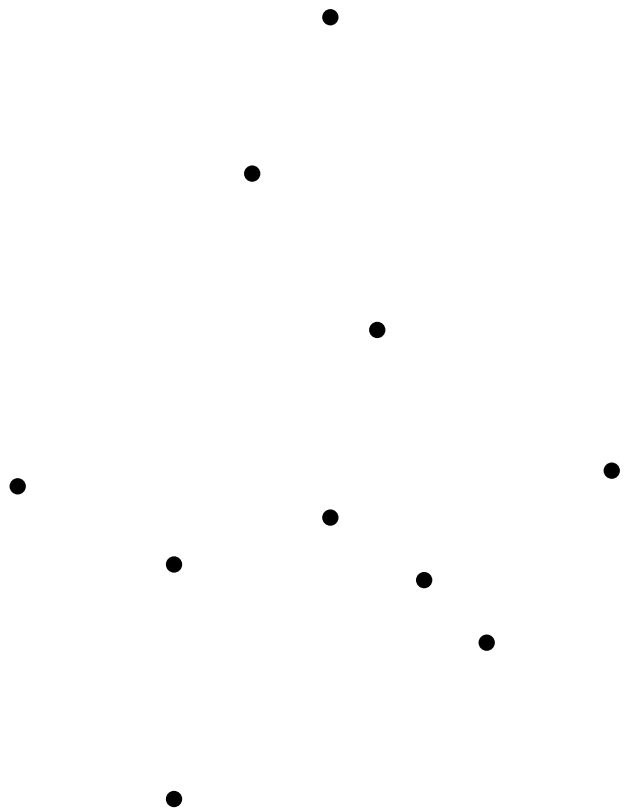
Illustration

$K = 2, n = 20$



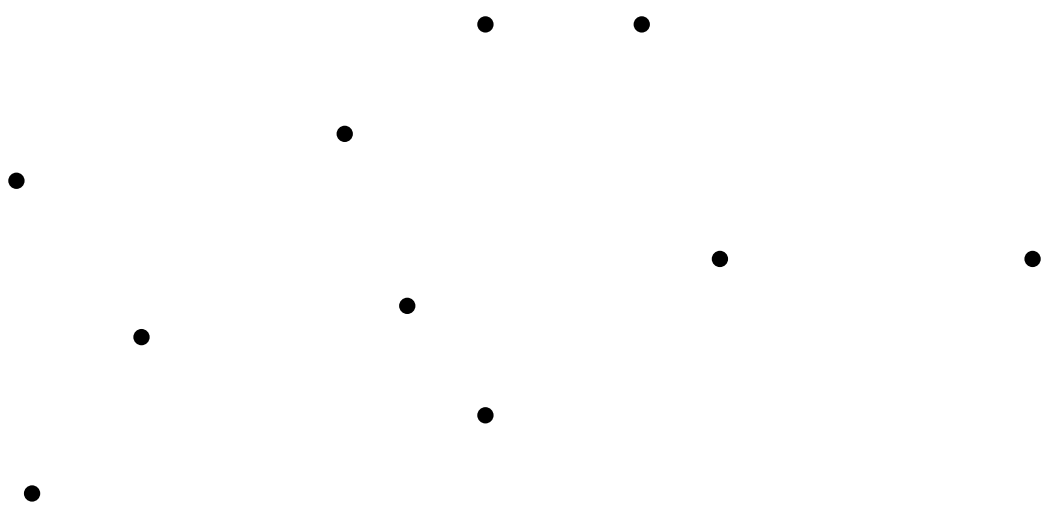
Illustration

$K = 2, n = 20$



Illustration

$K = 2, n = 20$



How to Choose K?

Argument 2: K should be large, e.g. $K = n$

- Classifiers generally perform better when trained on larger data sets.
- A small K means we substantially reduce the amount of training data used to train each f_k, so we may end up with weaker classifiers.
- This way, we will systematically overestimate the risk.

Common recommendation: $K = 5$ to $K = 10$

Intuition:

- $K = 10$ means number of samples removed from training is one order of magnitude below training sample size.
- This should not weaken the classifier considerably, but should be large enough to make measure variance effects.

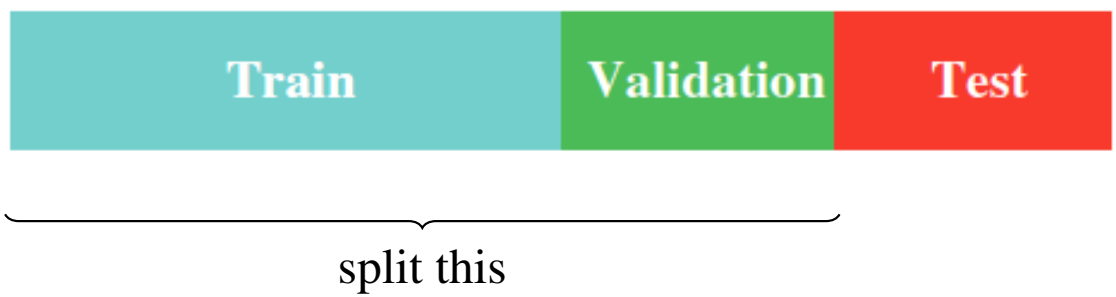
Summary: Cross Validation

Purpose

Estimates the risk $R(f) = \mathbb{E}[L(y, f(x))]$ of a classifier (or regression function) from data.

Application to parameter tuning

- Compute one cross validation estimate of $R(f)$ for each parameter value.
- Example above is margin parameter γ, but can be used for any parameter of a supervised learning algorithm.
- Note: Cross validation procedure does not involve the test data.



TREE CLASSIFIERS

Idea

- Recall: Classifiers classify according to location in $\mathbb{R}^d$
- Linear classifiers: Divide space into two halfspaces
- What if we are less sophisticated and divide space only along axes? We could classify e.g. according to

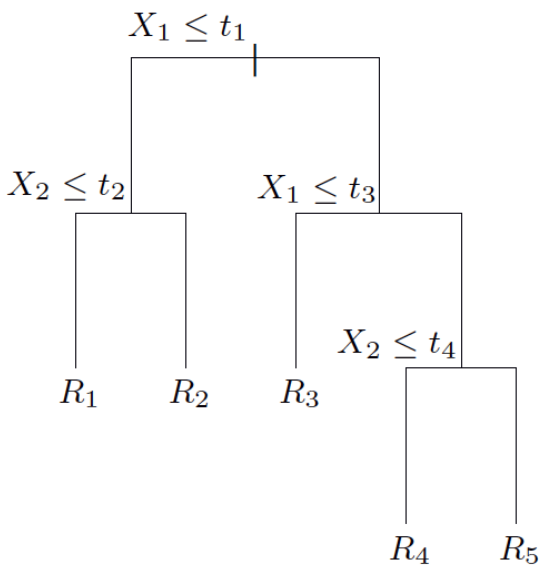
$$\mathbf{x} \in \begin{cases} \text{Class +} & \text{if } x_3 > 0.5 \\ \text{Class -} & \text{if } x_3 \leq 0.5 \end{cases}$$

- This decision would correspond to an affine hyperplane perpendicular to the x_3-axis, with offset 0.5.

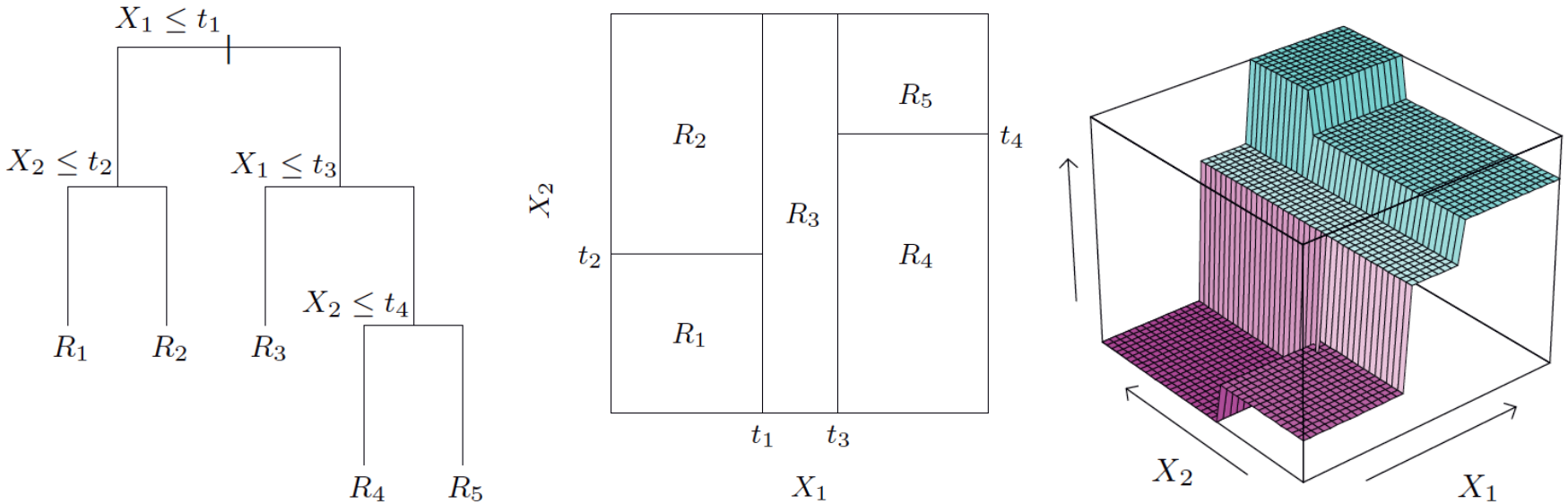
Tree classifier

A **tree classifier** is a binary tree in which

- Each inner node is a rule of the form $x_i > t_i$.
- The threshold values t_i are the parameters which specify the tree.
- Each leaf is a class label.



TREES



- Each leaf of the tree corresponds to a region R_m of $\mathbb{R}^d$.
- Classes $k \in \{1, \ldots, K\}$ (not restricted to two classes).
- Training: Each node is assigned class to which most points in R_m belong,

$$k(m) := \arg\max_k \#\{x_i \in R_m \text{ with } y_i = k\}$$

FINDING A SPLIT POINT

- In training algorithm, we have to fix a region R_m and split it along an axis j at a point t_j.
- The split results in two new regions R_m^1 and R_m^2.
- On each region, we obtain a new class assignment $k^1(m)$ and $k^2(m)$.
- Strategy is again: Define cost of split at t_j and minimize it to find t_j.

Cost of a split

$$Q(R_m, t_j) := \frac{\sum_{\tilde{\mathbf{x}}_i \in R_m^1} \mathbb{I}\{\tilde{y}_i \neq k^1(m)\} + \sum_{\tilde{\mathbf{x}}_i \in R_m^2} \mathbb{I}\{\tilde{y}_i \neq k^2(m)\}}{\#\{x_i \in R_m\}}$$

In words:

$$Q = \text{proportion of training points in } R_m \text{ that get misclassified if we choose to split at } t_j$$

Overall algorithm

- At each step: Current tree leaves define regions $R_1, \ldots, R_M$.
- For each R_m, find the best split.
- Continue splitting regions until tree has depth D (input parameter).

Step of training algorithm

At each step: Current tree leaves define regions $R_1, \ldots, R_M$.

For each region R_m:

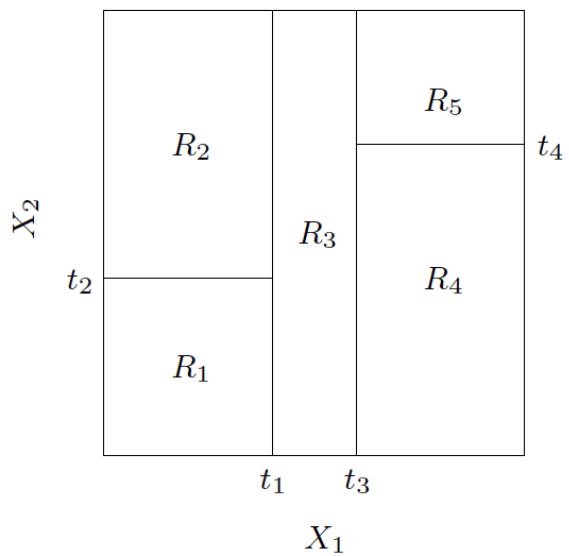
1. For each axis j: Compute best splitting point t_j as

$$t_j := \arg\min Q(R_m, t_j)$$

2. Select best splitting axis:

$$j := \arg\min_j Q(R_m, t_j)$$

3. Split R_m along axis j at t_j



EXAMPLE: SPAM FILTERING

Data

- 4601 email messages
- Classes: email, spam

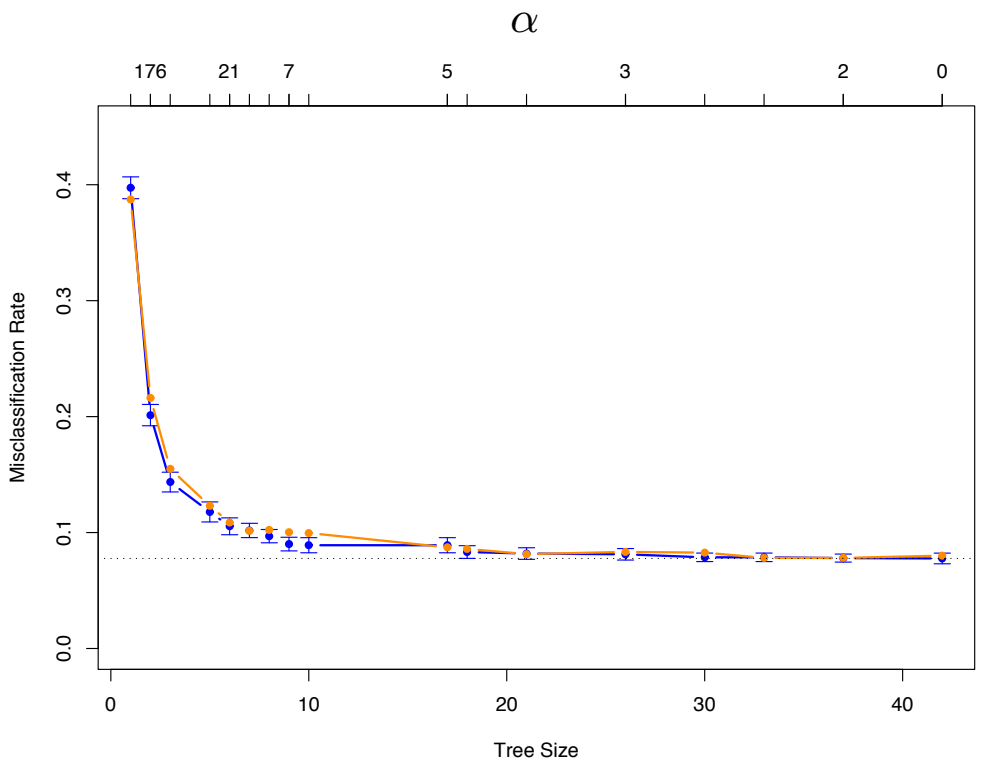
	george	you	your	hp	free	hpl	!	our	re	edu	remove
spam	0.00	2.26	1.38	0.02	0.52	0.01	0.51	0.51	0.13	0.01	0.28
email	1.27	1.27	0.44	0.90	0.07	0.43	0.11	0.18	0.42	0.29	0.01

Tree classifier

- 17 nodes
- Performance:

	Predicted	
True	Email	Spam
Email	57.3%	4.0%
Spam	5.3%	33.4%

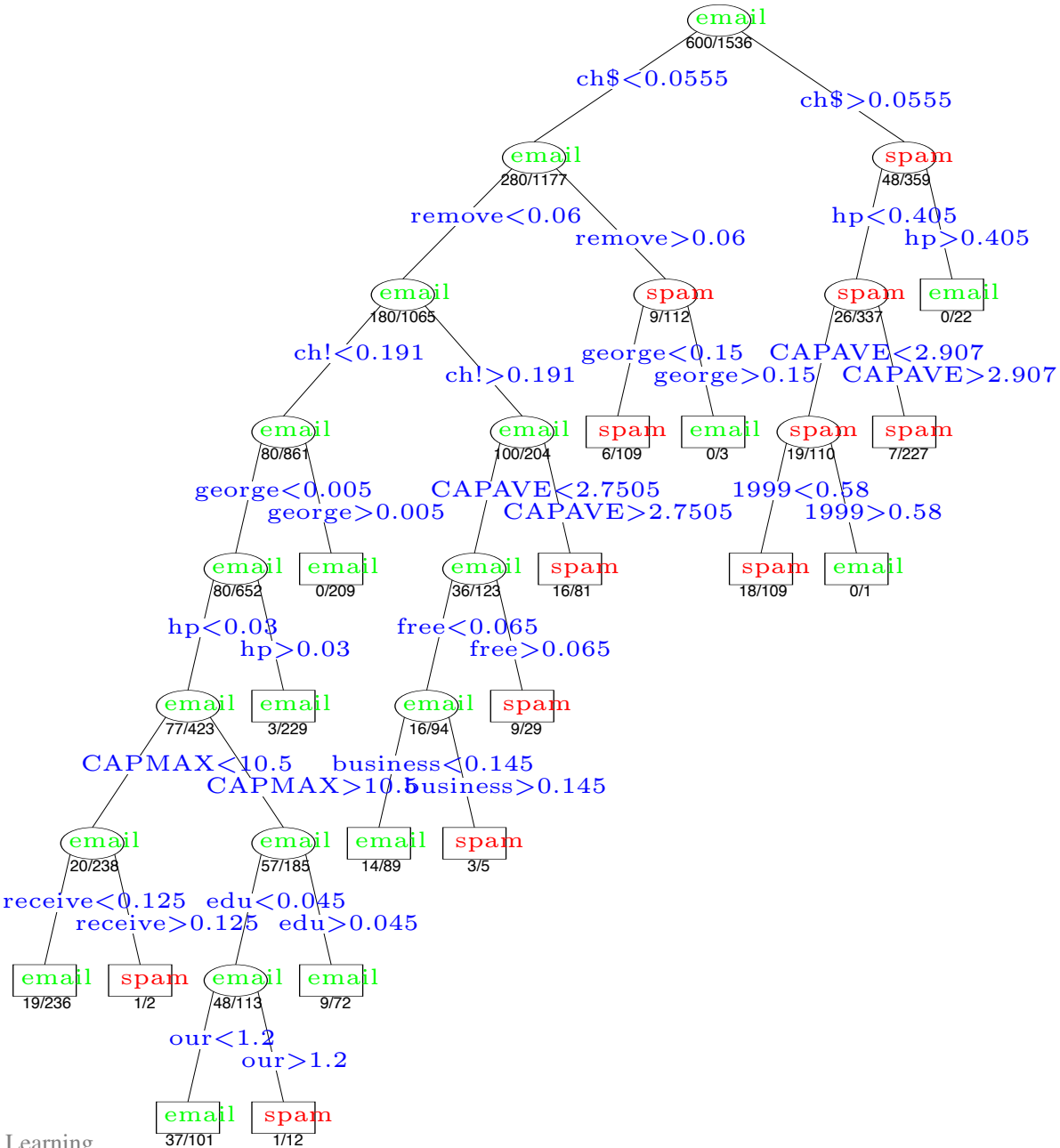
Influence of Tree Size



Tree Size

- Tree of height D defines 2^D regions.
- D too small: Insufficient accuracy. D too large: Overfitting.
- D can be determined by cross validation or more sophisticated methods ("complexity pruning" etc), which we will not discuss here.

SPAM FILTERING: TREE



Decision Stumps

- The simplest possible tree classifier is a tree of depth 1. Such a classifier is called a **decision stump**.
- A decision stump is parameterized by a pair (j, t_j) of an axis j and a splitting point t_j.
- Splits $\mathbb{R}^d$ into two regions.
- Decision boundary is an affine hyperplane which is perpendicular to axis j and intersects the axis at t_j.
- Often used in Boosting algorithms and other ensemble methods.