MATH60026/MATH70026
Methods for Data Science
Lecture 3

Barbara Bravi, Imperial College London

Department of Mathematics, Academic year 2024-2025

IMPERIAL

Naive Bayes

One of the simplest multi-class classifier. It essentially relies upon:

Bayes' theorem + *naive* assumption

Naive Bayes

One of the simplest multi-class classifier. It essentially relies upon:

Bayes' theorem + *naive* assumption

Setup:

$$\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_p^{(i)}) \qquad i = 1, \ldots, N$$

$$y^{(i)} \in \mathcal{C}_q = \{c_1, \ldots, c_Q\}$$

Bayes' theorem:

$$P(Y = c_q \mid X = \mathbf{x}) = \frac{\overbrace{P(X = \mathbf{x} \mid Y = c_q)}^{\spadesuit}\overbrace{P(Y = c_q)}^{\clubsuit}}{\underbrace{P(X = \mathbf{x})}_{\bigstar}}, \quad q = 1, \ldots, Q$$

Likelihood ♠ Prior ♣ Evidence (normalisation) ★

Key quantity: probability of belonging to a class conditional on the predictors (posterior)

Naive Bayes

Let's look each term:

'Prior' (♣): reflects a priori knowledge on the class distribution.
A simple, agnostic choice is a prior based on class frequency:

Indicator function

$$P(Y = c_q) = \frac{\sum_{i=1}^{N} I(y^{(i)} = c_q)}{N}$$

Fraction of training data in each class

Naive Bayes

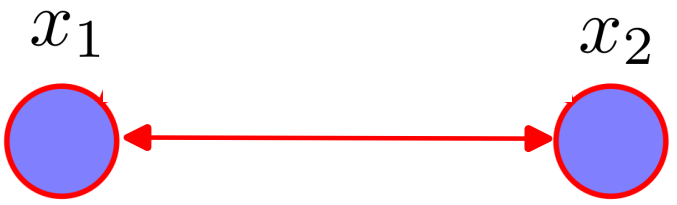
$\mathbf{`Likelihood'}$ $(\spadesuit)$: this is where the naive assumption enters.

Naive assumption = class-conditional independence of data features

Naive Bayes

$\mathbf{`Likelihood'}$ $(\spadesuit)$: this is where the naive assumption enters.

Naive assumption = class-conditional independence of data features

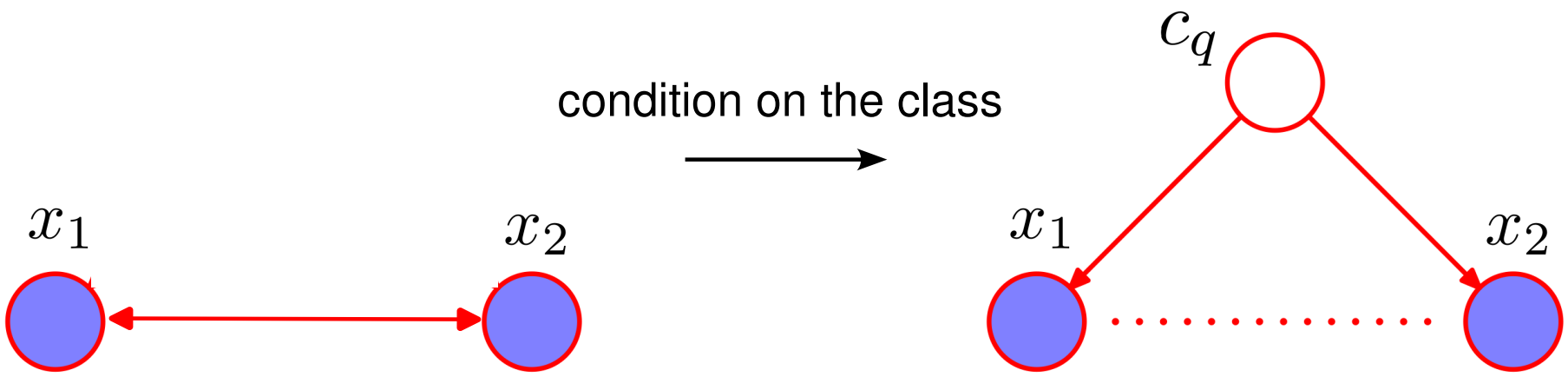


Two data features correlated

Naive Bayes

‘Likelihood’ (♠): this is where the naive assumption enters.

Naive assumption = class-conditional independence of data features



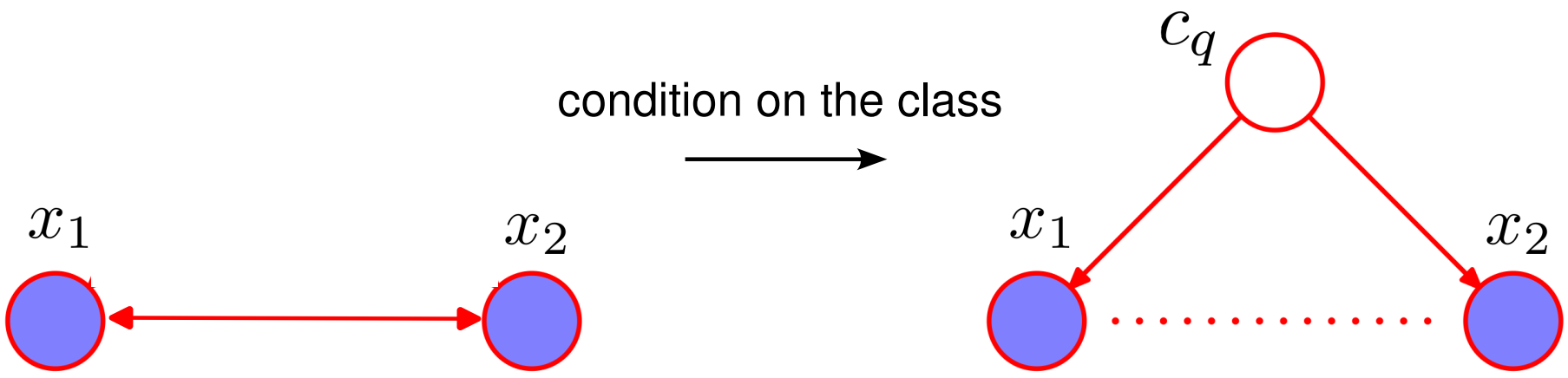
Two data features correlated

Assume they are independent when conditioned on the class

Naive Bayes

'Likelihood' (♠): this is where the naive assumption enters.

Naive assumption = class-conditional independence of data features



Two data features correlated

Assume they are independent when conditioned on the class

Naive assumption:
class-conditional feature probabilities factorise

$$P(X = \mathbf{x} \mid Y = c_q) = P(X_1 = x_1, X_2 = x_2, \ldots, X_p = x_p \mid Y = c_q)$$
$$= \prod_{j=1}^{p} P(X_j = x_j \mid Y = c_q)$$

Naive Bayes

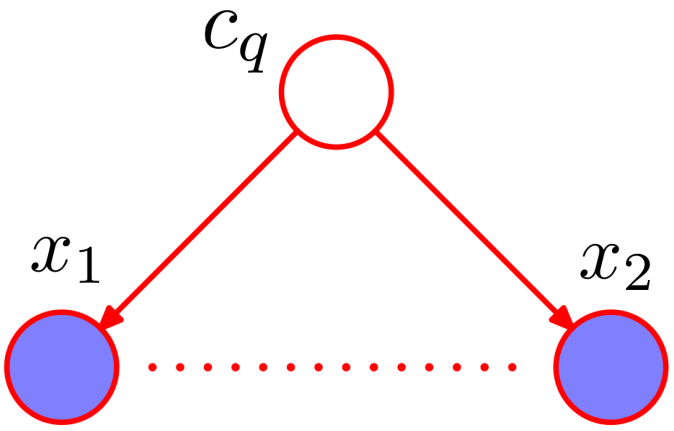
‘Likelihood’ (♠): this is where the naive assumption enters.

Naive assumption = class-conditional independence of data features

Advantages:

1. **Scalability**: estimation for each descriptor computed *separately* (fitting correlations is expensive and requires many data points).

2. Simple to deal with **mixed-type** data (continuous and discrete features).



Assume they are independent when conditioned on the class

Just an approximation!

Naive Bayes

'Likelihood' (♠): this is where the naive assumption enters.

What distributions? Standard options:

Discrete predictors: Count-based estimate

Boolean 'and'

$$P(X_j = x_j \mid Y = c_q) \approx \frac{\sum_{i=1}^{N} I(x_j^{(i)} = x_j \wedge y^{(i)} = c_q)}{\sum_{i=1}^{N} I(y^{(i)} = c_q)}, \quad j = 1, \ldots, p$$

Naive Bayes

'Likelihood' (♠): this is where the naive assumption enters.

What distributions? Standard options:

Discrete predictors: Count-based estimate

Boolean 'and'

$$P(X_j = x_j \mid Y = c_q) \approx \frac{\sum_{i=1}^{N} I(x_j^{(i)} = x_j \wedge y^{(i)} = c_q)}{\sum_{i=1}^{N} I(y^{(i)} = c_q)}, \quad j = 1, \ldots, p$$

Problem of empty entries:
Laplace smoothing, see notebook

Naive Bayes

'Likelihood' (♠): this is where the naive assumption enters.

What distributions? Standard options:

Discrete predictors: Count-based estimate

$$P(X_j = x_j \mid Y = c_q) \approx \frac{\sum_{i=1}^{N} I(x_j^{(i)} = x_j \wedge y^{(i)} = c_q)}{\sum_{i=1}^{N} I(y^{(i)} = c_q)}, \quad j = 1, \ldots, p$$

Boolean 'and'

Problem of empty entries:
Laplace smoothing, see notebook

Continuous predictors: Gaussian distribution

$$P(X_j = x_j \mid Y = c_q) \approx \frac{1}{\sqrt{2\pi\sigma_{j,q}^2}} e^{\frac{-(x_j - \mu_{j,q})^2}{2\sigma_{j,q}^2}}$$

$$\mu_{j,q} = \frac{1}{N_q} \sum_{y^{(i)} \in c_q} x_j^{(i)}$$

$$\sigma_{j,q}^2 = \frac{1}{N_q} \sum_{y^{(i)} \in c_q} (x_j^{(i)} - \mu_{j,q})^2$$

Sample estimator of mean
and variance in each class

Naive Bayes

Normalisation factor ($\bigstar$): sum of previous terms over classes.

$$P(X = \mathbf{x}) = \sum_{q=1}^{Q} P(X = \mathbf{x} \mid Y = c_q) P(Y = c_q)$$

$$= \sum_{q=1}^{Q} [\clubsuit \cdot \spadesuit]_q$$

Naive Bayes

Normalisation factor ($\bigstar$): sum of previous terms over classes.

$$P(X = \mathbf{x}) = \sum_{q=1}^{Q} P(X = \mathbf{x} \mid Y = c_q) P(Y = c_q)$$

$$= \sum_{q=1}^{Q} [\clubsuit \cdot \spadesuit]_q$$

Final Naive Bayes classifier prediction:

$$P(Y = c_q \mid X = \mathbf{x}) = \frac{P(Y = c_q) \prod_{j=1}^{p} P(X_j = x_j \mid Y = c_q)}{\sum_{q=1}^{Q} [\clubsuit \cdot \spadesuit]_q} = \frac{[\clubsuit \cdot \spadesuit]_q}{\sum_{q=1}^{Q} [\clubsuit \cdot \spadesuit]_q}$$

Naive Bayes

The prediction is a vector of probabilistic class assignments (soft classification):

$$\boldsymbol{\pi}^{(\mathrm{NB})} = \begin{bmatrix} \pi_1^{(\mathrm{NB})} \\ \vdots \\ \pi_Q^{(\mathrm{NB})} \end{bmatrix}, \text{ where each component is:}$$

$$\pi_q^{(\mathrm{NB})} = P(Y = c_q \mid X = \mathbf{x}^{\mathrm{in}}) = \frac{[\clubsuit \cdot \spadesuit]_q}{\sum_{q=1}^{Q} [\clubsuit \cdot \spadesuit]_q}$$

New sample to classify

Normalised by construction:

$$\sum_{q=1}^{Q} \pi_q^{(\mathrm{NB})} = 1$$

Naive Bayes

The prediction is a vector of probabilistic class assignments (soft classification):

$$\boldsymbol{\pi}^{(\mathrm{NB})} = \begin{bmatrix} \pi_1^{(\mathrm{NB})} \\ \vdots \\ \pi_Q^{(\mathrm{NB})} \end{bmatrix}, \text{ where each component is:}$$

$$\pi_q^{(\mathrm{NB})} = P(Y = c_q \mid X = \mathbf{x}^{\mathrm{in}}) = \frac{[\clubsuit \cdot \spadesuit]_q}{\sum_{q=1}^{Q} [\clubsuit \cdot \spadesuit]_q} \qquad \sum_{q=1}^{Q} \pi_q^{(\mathrm{NB})} = 1$$

As usual, the hard classification version is obtained by:

$$\hat{y}^{(\mathrm{NB})} = \underset{q}{\operatorname{argmax}}\, \boldsymbol{\pi}^{(\mathrm{NB})}$$

Summary

1. Naive Bayes classifier's target is to predict the probability of belonging to a class conditional on the data features.

Summary

1. Naive Bayes classifier's target is to predict the probability of belonging to a class conditional on the data features.

2. It uses the Bayes' theorem + naive assumption (it assumes the independence of data features conditional on the class).

Summary

1. Naive Bayes classifier's target is to predict the probability of belonging to a class conditional on the data features.

2. It uses the Bayes' theorem + naive assumption (it assumes the independence of data features conditional on the class).

3. It handles discrete and continuous predictors, multiple classes.

Summary

1. Naive Bayes classifier's target is to predict the probability of belonging to a class conditional on the data features.

2. It uses the Bayes' theorem + naive assumption (it assumes the independence of data features conditional on the class).

3. It handles discrete and continuous predictors, multiple classes.

4. The naive assumption leads to a drastic simplification of model construction, making it scalable and amenable to mixed-type data.

Summary

1. Naive Bayes classifier's target is to predict the probability of belonging to a class conditional on the data features.

2. It uses the Bayes' theorem + naive assumption (it assumes the independence of data features conditional on the class).

3. It handles discrete and continuous predictors, multiple classes.

4. The naive assumption leads to a drastic simplification of model construction, making it scalable and amenable to mixed-type data.

5. Standard choices for the factorised likelihood term (the count-based estimate for discrete, the gaussian distribution for continuous) help spare model calibration efforts (no learning by loss minimisation, no hyper-parameters to set).

Decision Trees

Building blocks of Random Forests, a powerful method used for classification and regression.

Let's start with classification: $\mathbf{x} = (x_1, ..., x_p) \rightarrow y \;\; y \in \mathcal{C}_q = \{c_1, \ldots, c_Q\}$

classes

Decision Trees

Building blocks of Random Forests, a powerful method used for classification and regression.

Let's start with classification: $\mathbf{x} = (x_1, ..., x_p) \rightarrow y \quad y \in \mathcal{C}_q = \{c_1, \ldots, c_Q\}$

classes

Key idea: partition the input space into regions separating samples of different classes into different regions through a series of *splits* (i.e., 'decisions' that generate a *tree* structure)

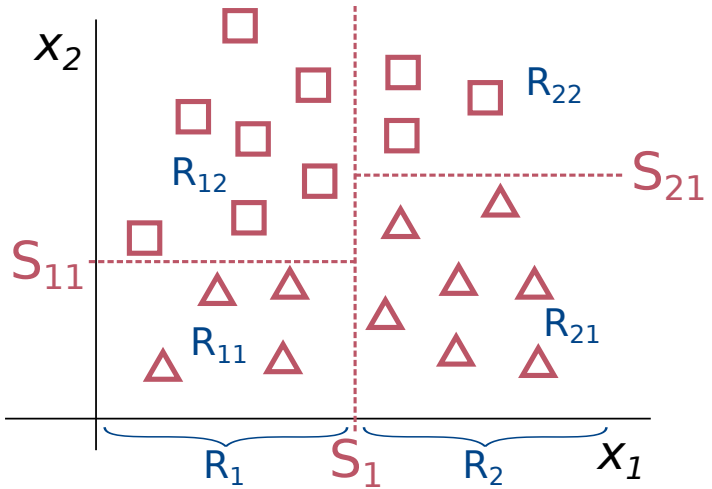
Decision Trees

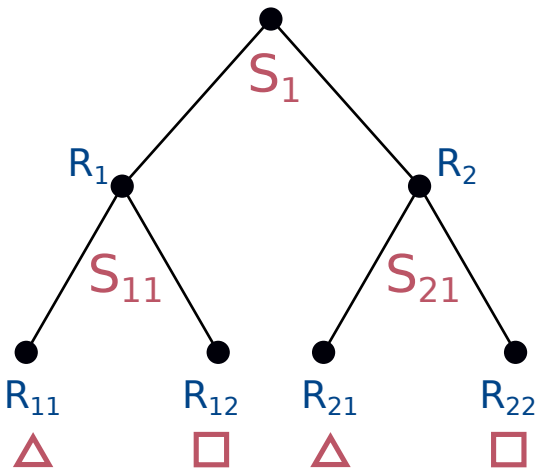
Building blocks of Random Forests, a powerful method used for classification and regression.

Let's start with classification: $\mathbf{x} = (x_1, ..., x_p) \rightarrow y \;\; y \in \mathcal{C}_q = \{c_1, \ldots, c_Q\}$

classes

Key idea: partition the input space into regions separating samples of different classes into different regions through a series of *splits* (i.e., 'decisions' that generate a *tree* structure)





Tree = set of splits

$$\text{split } (j,s) = \left\{ \begin{array}{l} \{x_j : x_j < s\} =: R_1(j,s) \} \\ \{x_j : x_j \geq s\} =: R_2(j,s) \} \end{array} \right\}$$

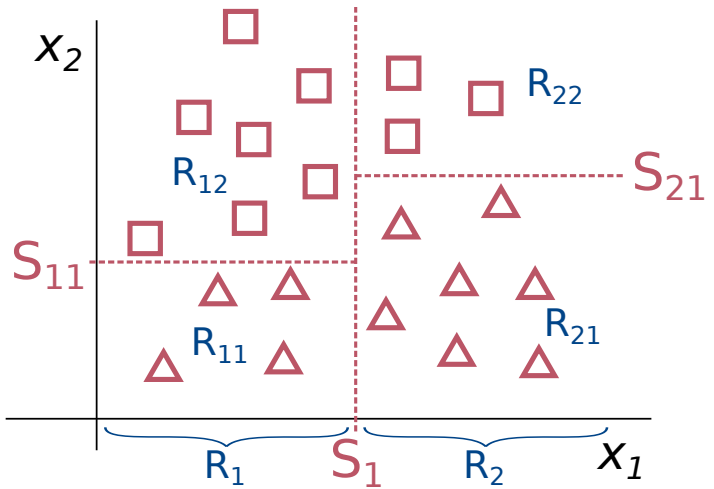
Decision Trees

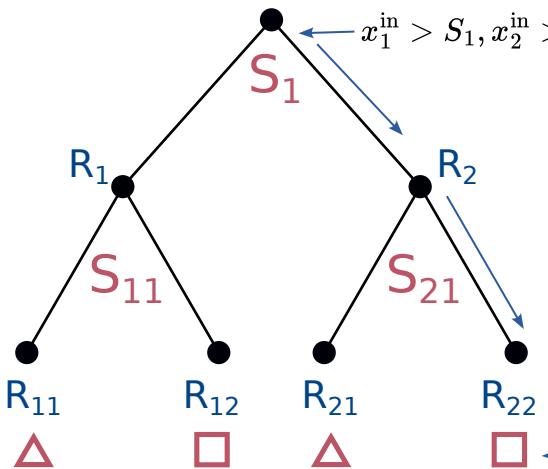
Building blocks of Random Forests, a powerful method used for classification and regression.

Let's start with classification: $\mathbf{x} = (x_1, ..., x_p) \rightarrow y \quad y \in \mathcal{C}_q = \{c_1, \ldots, c_Q\}$

classes

Key idea: partition the input space into regions separating samples of different classes into different regions through a series of *splits* (i.e., 'decisions' that generate a *tree* structure)





Tree = set of splits

$$\text{split } (j,s) = \left\{ \begin{array}{l} \{x_j : x_j < s\} =: R_1(j,s) \\ \{x_j : x_j \geq s\} =: R_2(j,s) \end{array} \right\}$$

Prediction = majority class in the region

Example: spam data

- Classifier of ‘spam’/’e-mail’ messages
- 57 continuous predictors (words, special characters, sequences of capital letters)
- Training set (3065) and Test set (1536)

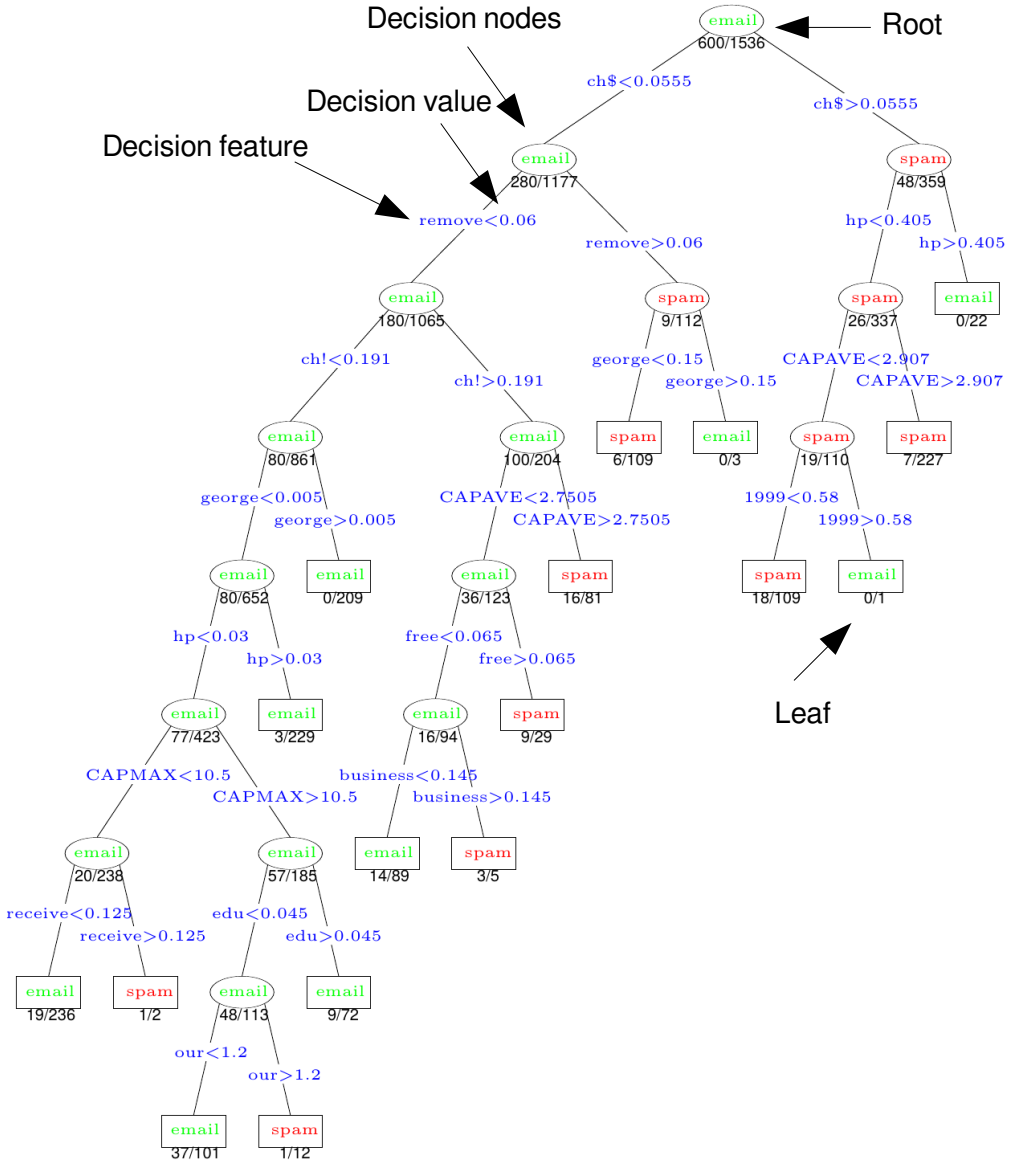
Class	Features				
	CAPTOT	ch$	remove	business	
email	0.11	0.053	0.21	0.003	...
spam	0.10	0.015	0.25	0.081	...
email	0.21	0.012	0.32	0.001	...
		...			

Example discussed in Hastie, Tibshirani, Friedman, Elements of statistical learning, chap. 9, 15

Example: spam data

- Classifier of ‘spam’/’e-mail’ messages
- 57 continuous predictors (words, special characters, sequences of capital letters)
- Training set (3065) and Test set (1536)

Class	Features				
	CAPTOT	ch$	remove	business	
email	0.11	0.053	0.21	0.003	...
spam	0.10	0.015	0.25	0.081	...
email	0.21	0.012	0.32	0.001	...
		...			



Example discussed in Hastie, Tibshirani, Friedman, Elements of statistical learning, chap. 9, 15

Decision Trees

Algorithm: it proceeds *greedily*, i.e., it computes the optimal split one level at a time.

To obtain optimal splits, we a need **loss/cost function**: here taken from **information theory**.

Decision Trees

Algorithm: it proceeds *greedily*, i.e., it computes the optimal split one level at a time.

To obtain optimal splits, we a need **loss/cost function**: here taken from **information theory**.

Node impurity: proportion of data in each class:

$$\pi_q(R_{\boldsymbol{\alpha}}) = \frac{\sum_{i=1}^{N} I\left(\mathbf{x}^{(i)} \in R_{\boldsymbol{\alpha}} \wedge y^{(i)} \in c_q\right)}{\sum_{i=1}^{N} I(\mathbf{x}^{(i)} \in R_{\boldsymbol{\alpha}})}$$

Number of data in the region belonging to *q*

Number of data in the region

Decision Trees

Algorithm: it proceeds *greedily*, i.e., it computes the optimal split one level at a time.

To obtain optimal splits, we a need **loss/cost function**: here taken from **information theory**.

Node impurity: proportion of data in each class:

$$\pi_q(R_{\boldsymbol{\alpha}}) = \frac{\sum_{i=1}^{N} I\left(\mathbf{x}^{(i)} \in R_{\boldsymbol{\alpha}} \wedge y^{(i)} \in c_q\right)}{\sum_{i=1}^{N} I(\mathbf{x}^{(i)} \in R_{\boldsymbol{\alpha}})}$$

Number of data in the region belonging to *q*

Number of data in the region

Gini index

$$\mathrm{GI}\left[\boldsymbol{\pi}(R_{\boldsymbol{\alpha}})\right] = \sum_{q=1}^{Q} \pi_q(R_{\boldsymbol{\alpha}})\left(1 - \pi_q(R_{\boldsymbol{\alpha}})\right)$$

Cross-entropy

$$\mathrm{CE}\left[\boldsymbol{\pi}(R_{\boldsymbol{\alpha}})\right] = -\sum_{q=1}^{Q} \pi_q(R_{\boldsymbol{\alpha}}) \log \pi_q(R_{\boldsymbol{\alpha}})$$ *

They both measure the **deviation wrt to a homogeneous distribution** into classes, i.e. , they measure the **information** about classes carried by $\boldsymbol{\pi}(R_{\boldsymbol{\alpha}})$ at a given split

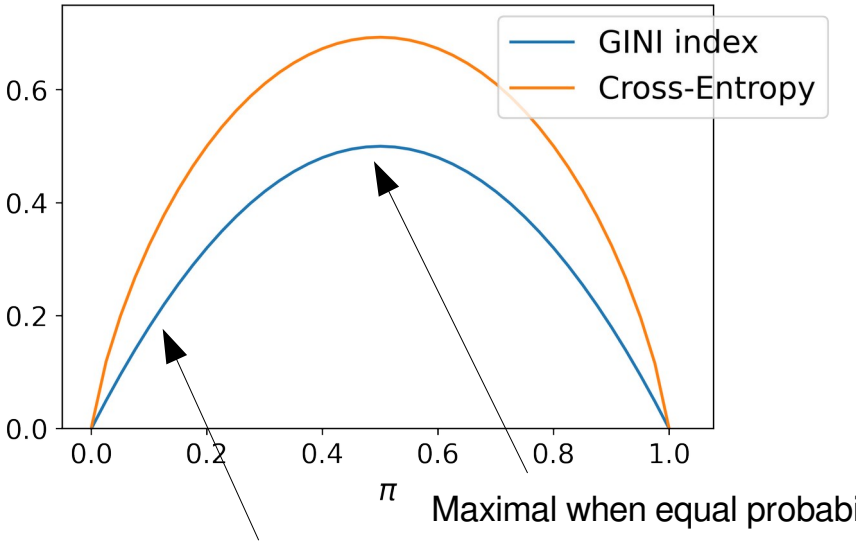
*We call it 'Cross-Entropy' because it can be derived as the cross-entropy between data and model-predicted distribution, see the Lecture Notes!

Decision Trees

Simple example: 1 splitting, 2 classes

$$\mathrm{GI}[\pi] = 2\pi(1-\pi)$$

$$\mathrm{CE}[\pi] = -\pi\log\pi - (1-\pi)\log(1-\pi)$$



Maximal when equal probability

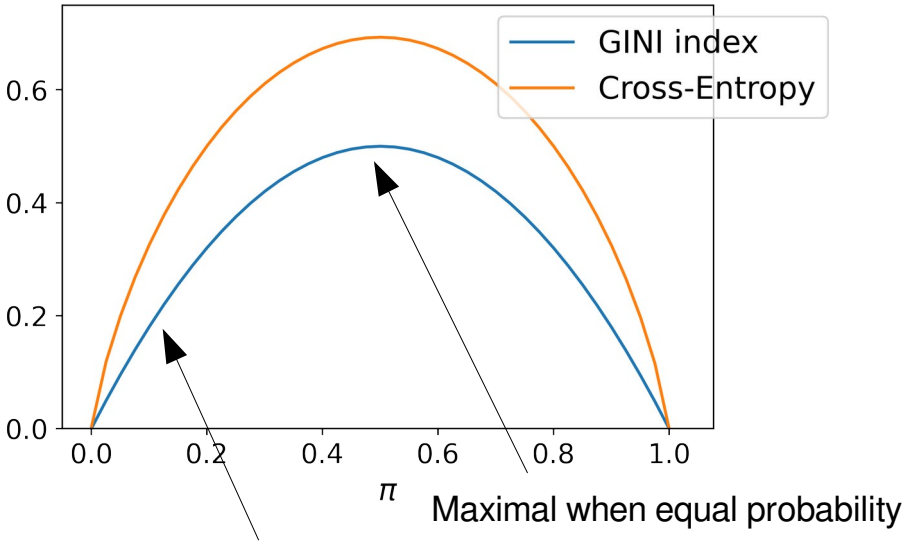
Decreases probabilities are dissimilar

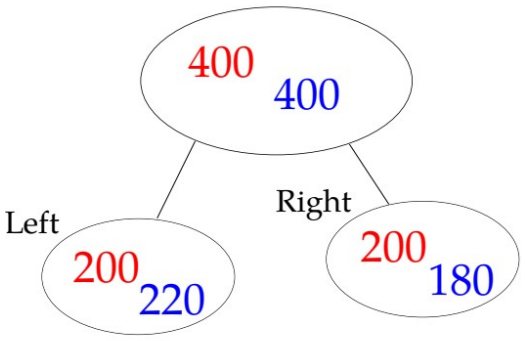
Decision Trees

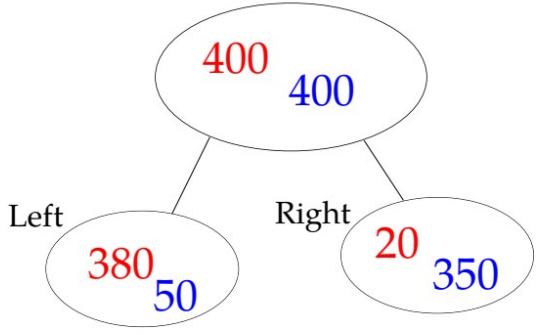
Simple example: 1 splitting, 2 classes

$$\mathrm{GI}[\pi] = 2\pi(1-\pi)$$

$$\mathrm{CE}[\pi] = -\pi \log \pi - (1-\pi) \log (1-\pi)$$





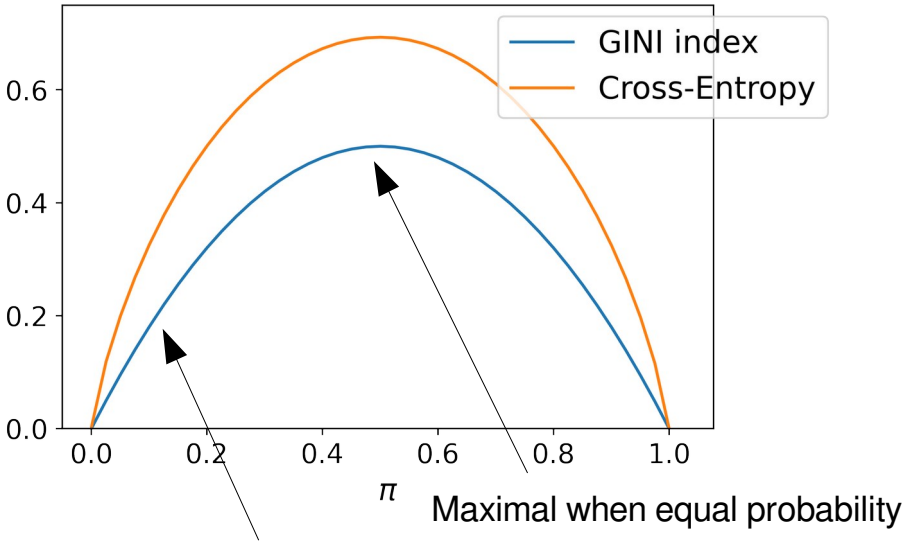


Decision Trees

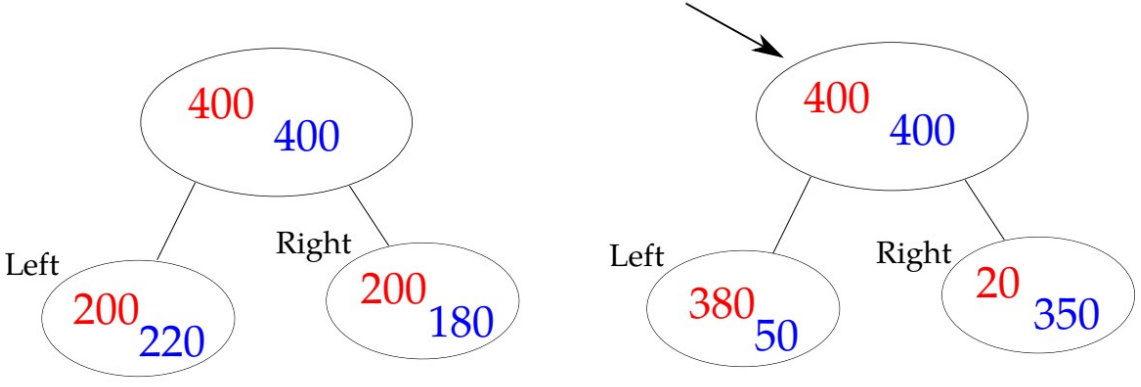
Simple example: 1 splitting, 2 classes

$$\mathrm{GI}[\pi] = 2\pi(1-\pi)$$

$$\mathrm{CE}[\pi] = -\pi \log \pi - (1-\pi) \log (1-\pi)$$



This split provides more 'information' about the data separation into classes



$GI = 0.525\, GI^L + 0.475\, G^R = 0.5$

$GI = 0.5375\, GI^L + 0.4625\, G^R = 0.16$

The splitting that minimizes the Gini index or the cross-entropy leads to regions dominated by 1 class

Decision Trees

GI and CE are therefore functions to minimise to achieve a classification task.

*

Decision Trees

GI and CE are therefore functions to minimise to achieve a classification task.

Recall that a tree = set of splits

$$\text{split } (j,s) = \left\{ \begin{array}{l} \{x_j : x_j < s\} =: R_1(j,s)\} \\ \{x_j : x_j \geq s\} =: R_2(j,s)\} \end{array} \right\}$$

A given split is chosen by:

$$\min_{j,s} \left[\mathbb{p}_{R_1} \mathrm{GI}\big(\boldsymbol{\pi}(R_1), j, s\big) + \mathbb{p}_{R_2} \mathrm{GI}\big(\boldsymbol{\pi}(R_2), j, s\big) \right]^*$$

Fraction of samples ending up in that region

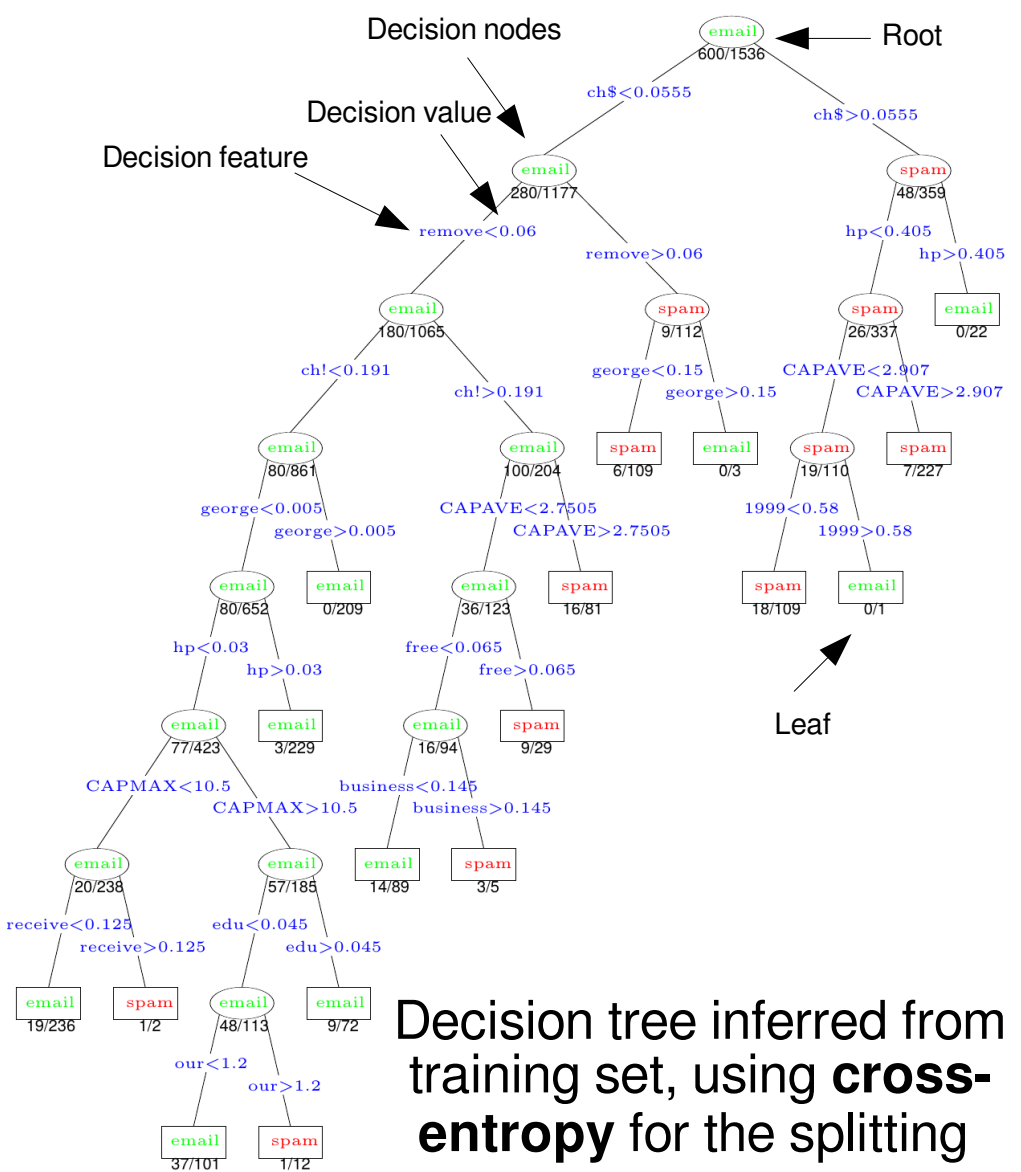
To repeat until a stopping criterion is met.

Example: spam data

- Classifier of ‘spam’/’e-mail’ messages
- 57 continuous predictors (words, special characters, sequences of capital letters)
- Training set (3065) and Test set (1536)

Class	Features				
	CAPTOT	ch$	remove	business	
email	0.11	0.053	0.21	0.003	...
spam	0.10	0.015	0.25	0.081	...
email	0.21	0.012	0.32	0.001	...
		...			

Example discussed in Hastie, Tibshirani, Friedman, Elements of statistical learning, chap. 9, 15



Decision tree inferred from training set, using **cross-entropy** for the splitting

Which stopping criterion to use?

How much should we grow the tree?

1. **Setting constraints on tree size**: minimum node size, minimum number of samples for a leaf, maximum number of leaves, maximum depth.

Example: spam data

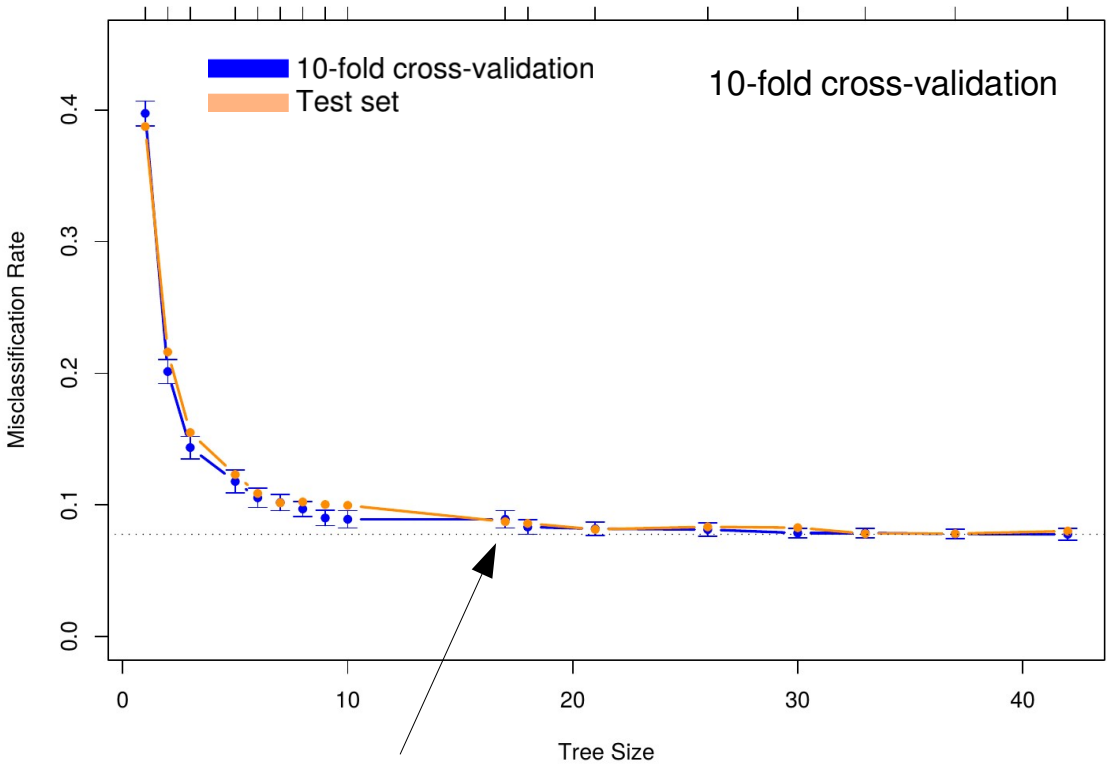
Which stopping criterion to use?

How much should we grow the tree?

1. **Setting constraints on tree size**: minimum node size, minimum number of samples for a leaf, maximum number of leaves, maximum depth.

2. **Cross-validation**: the tree size is a hyper-parameter, to tune based on out-of-sample performance (like here).

They stop when there is a plateau in performance on the validation set.



17-nodes tree chosen by cross-validation

Example discussed in Hastie, Tibshirani, Friedman, Elements of statistical learning, chap. 9, 15

Example: spam data

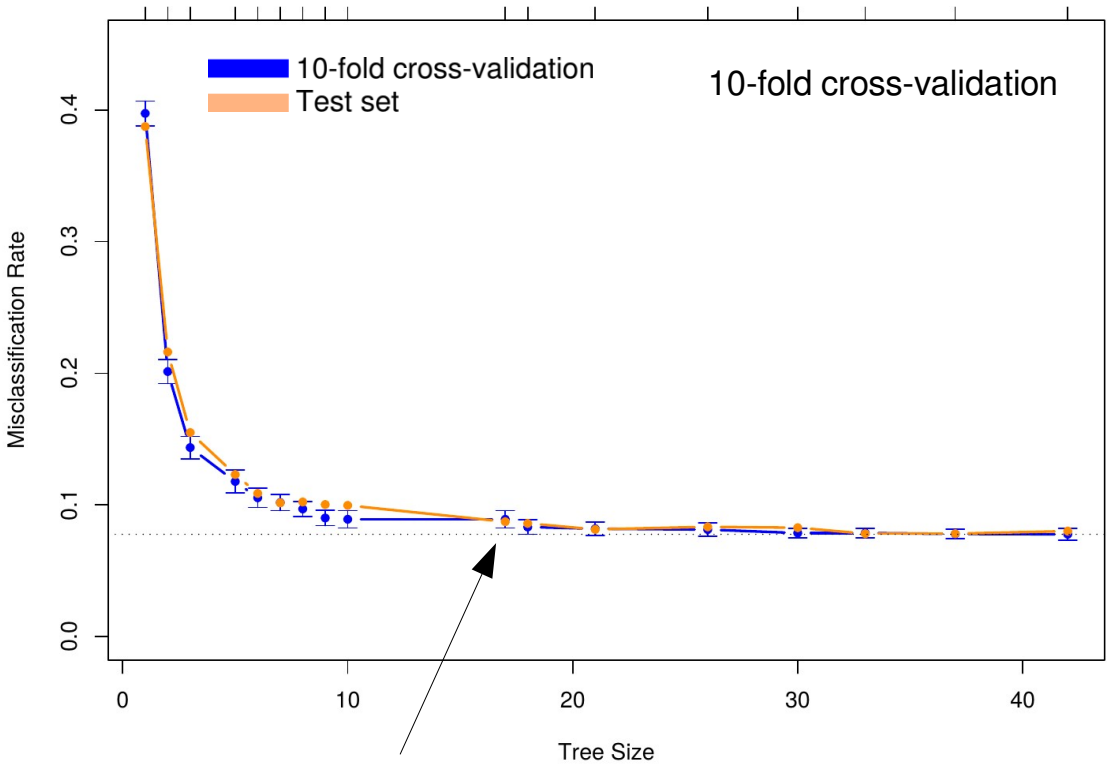
Which stopping criterion to use?

How much should we grow the tree?

1. **Setting constraints on tree size**: minimum node size, minimum number of samples for a leaf, maximum number of leaves, maximum depth.

2. **Cross-validation**: the tree size is a hyper-parameter, to tune based on out-of-sample performance (like here).

They stop when there is a plateau in performance on the validation set.



17-nodes tree chosen by cross-validation

Confusion matrix

	Predicted	
True	email	spam
email	57.3%	4.0%
spam	5.3%	33.4%

Misclassification rate = 9.3%

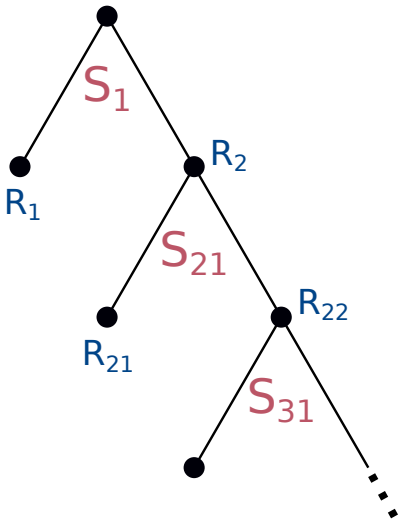
Example discussed in Hastie, Tibshirani, Friedman, Elements of statistical learning, chap. 9, 15

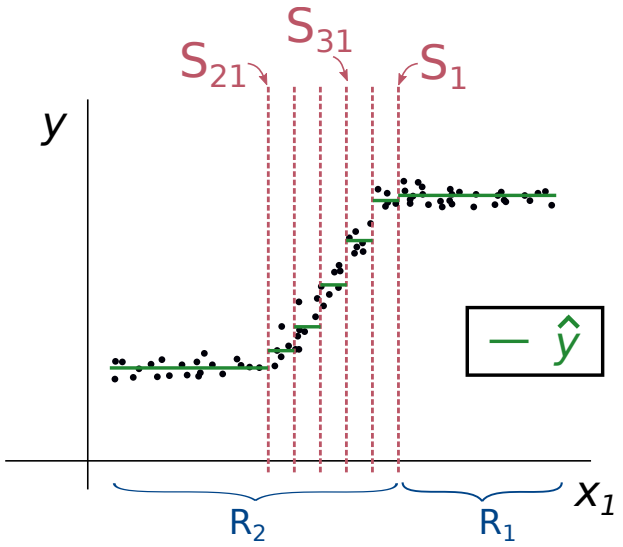
Decision Trees

Decision trees are also applied to regression. The only change is in the cost function:

$$\min_{j,s} RSS(\mathbf{y}; j, s), \qquad RSS(\mathbf{y}; j, s) = \left[\sum_{\mathbf{x}^{(i)} \in R_1(j,s)} (y^{(i)} - \overline{y}_{R_1})^2 + \sum_{\mathbf{x}^{(i)} \in R_2(j,s)} (y^{(i)} - \overline{y}_{R_2})^2 \right]$$

Mean of output variable in that region



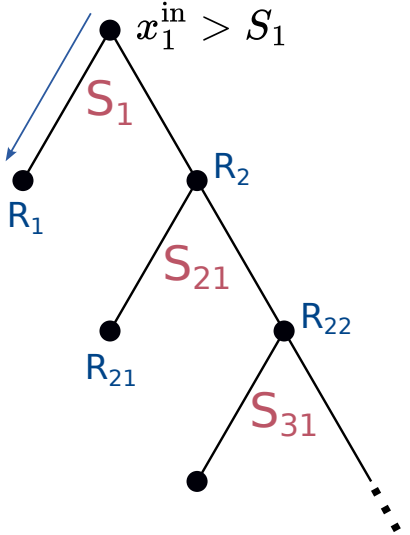


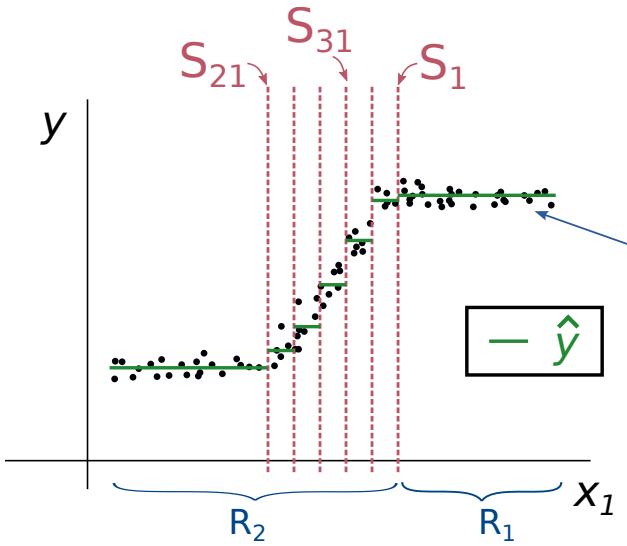
Decision Trees

Decision trees are also applied to regression. The only change is in the cost function:

$$\min_{j,s} RSS(\mathbf{y}; j, s), \qquad RSS(\mathbf{y}; j, s) = \left[\sum_{\mathbf{x}^{(i)} \in R_1(j,s)} (y^{(i)} - \overline{y}_{R_1})^2 + \sum_{\mathbf{x}^{(i)} \in R_2(j,s)} (y^{(i)} - \overline{y}_{R_2})^2 \right]$$

Mean of output variable in that region





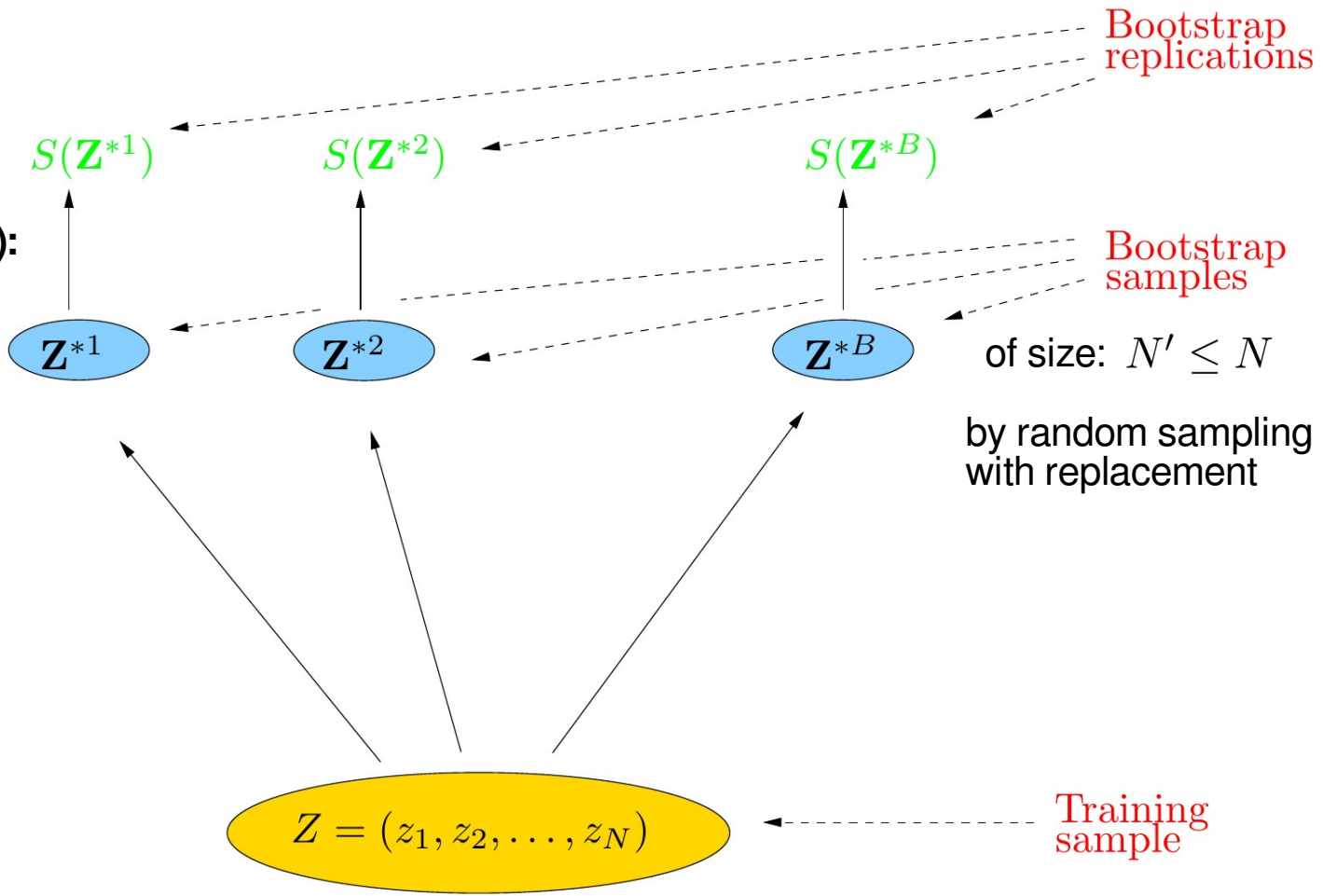
Prediction = average output in the region where the input lies

$$\hat{y}^{\text{DT}}(\mathbf{x}^{\text{in}}) = \overline{y}_{R_\alpha} \text{ with } \mathbf{x}^{\text{in}} \in R_\alpha$$

Pros and Cons

- **Flexible:** They can easily handle both continuous and categorical data multi-class classification, regression. ✓
- **Powerful:** perform well with nonlinear relationships among features ✓
- **Prone to overfitting** - need to reduce the variance ✗

Bootstrap Aggregation (Bagging):
average over many trees to reduce the variance of the DT estimator.

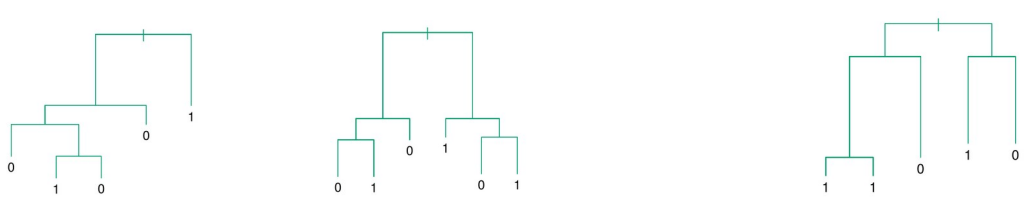


From Hastie, Tibshirani, Friedman,

Elements of statistical learning, chap. 7

For each bootstrap sample grow a tree, giving the ensemble

$$\left\{ \hat{f}_b^{\mathrm{DT}} \right\}_{b=1}^{B}$$



Bootstrap replications

$S(\mathbf{Z}^{*1})$ $S(\mathbf{Z}^{*2})$ $S(\mathbf{Z}^{*B})$

Bootstrap Aggregation (Bagging): average over many trees to reduce the variance of the DT estimator.

Bootstrap samples

$\mathbf{Z}^{*1}$ $\mathbf{Z}^{*2}$ $\mathbf{Z}^{*B}$

of size: $N' \leq N$

by random sampling with replacement

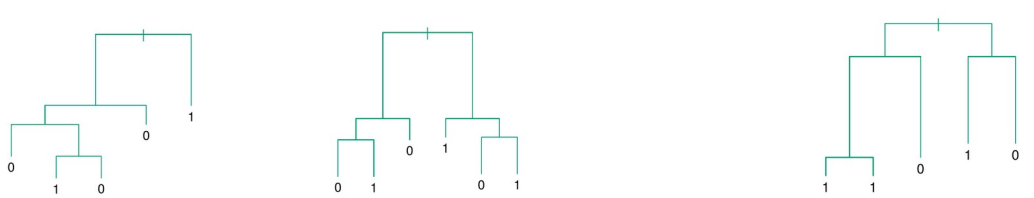
$Z = (z_1, z_2, \ldots, z_N)$

Training sample

From Hastie, Tibshirani, Friedman, Elements of statistical learning, chap. 7

For each bootstrap sample grow a tree, giving the ensemble

$$\left\{ \hat{f}_b^{\mathrm{DT}} \right\}_{b=1}^{B}$$



Bootstrap replications

$S(\mathbf{Z}^{*1})$ $S(\mathbf{Z}^{*2})$ $S(\mathbf{Z}^{*B})$

Bootstrap Aggregation (Bagging): average over many trees to reduce the variance of the DT estimator.

Bootstrap samples

$\mathbf{Z}^{*1}$ $\mathbf{Z}^{*2}$ $\mathbf{Z}^{*B}$

of size: $N' \leq N$

by random sampling with replacement

*Note: it's a **general procedure** for variance reduction by averaging over ensembles of models*

$Z = (z_1, z_2, \ldots, z_N)$

Training sample

From Hastie, Tibshirani, Friedman, Elements of statistical learning, chap. 7

If there are correlations between variables, one can't reduce the variance beyond a certain limit...

B random variables (in this case, the trees), that are simply i.d. (identically distributed, but not necessarily independent), with positive pairwise correlation ρ and each of them with variance σ^2, the variance of their average as an estimator is:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

Random forests (Breiman, 2001) attempt further variance reduction by decreasing the correlation

Random Forests:

Key idea: before each split, select at random $m \leq p$ features as candidates for splitting

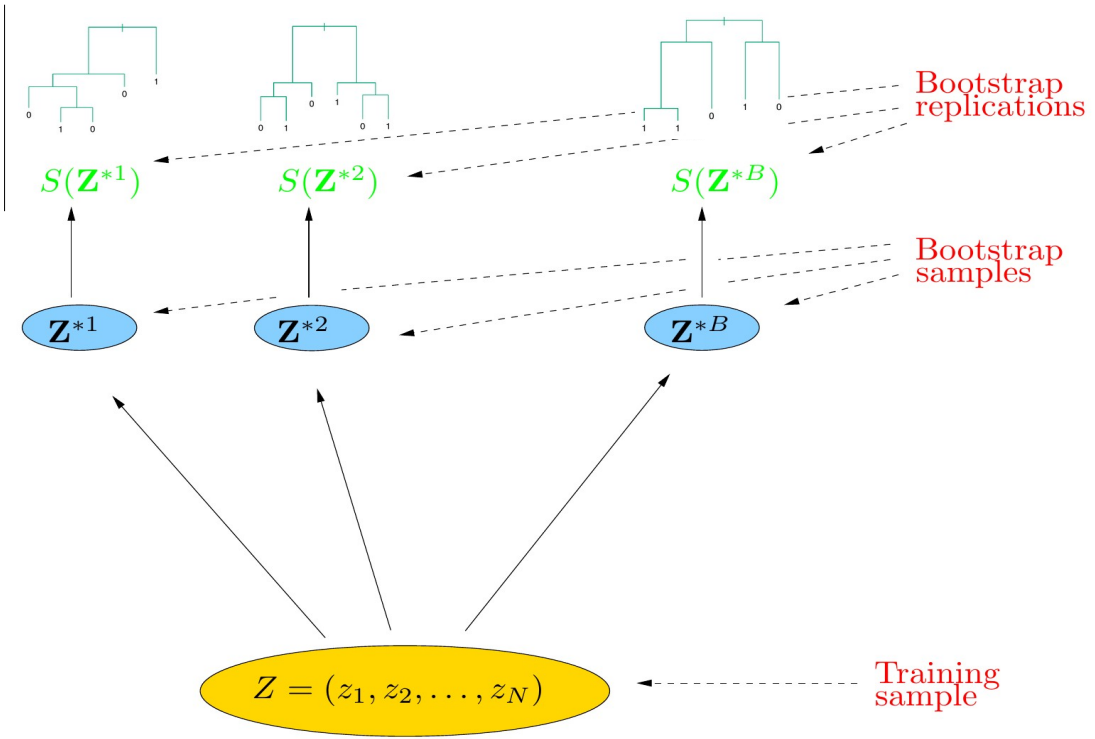
Random Forests:

Key idea: before each split, select at random $m \leq p$ features as candidates for splitting

For each bootstrap sample grow a tree (bagging)

+

random selection of predictors at each splitting



Random Forests:

Algorithm: Random Forests for Regression and Classification

Training.
For $b = 1$ to B:

(1) Draw a bootstrap sample $\mathbf{Z}^{*b}$ from the training data.

(2) Train a tree $\hat{f}_b^{\mathrm{DT}}$ on the bootstrapped sample as follows.
For each terminal node at the current level, repeat:
(i): Draw at random a subset of m features from the p features.
(ii): Find the optimal split (j, s) (according to the relevant cost function), choosing j only among the m features randomly sampled in *(i)*.
(iii) Perform the split into two daughter nodes.
Until the stopping criterion is reached.

Making predictions (aggregating). Given a new data point $\mathbf{x}^{\mathrm{in}}$:

(1) *Regression:*

$$\hat{y} = \hat{f}(\mathbf{x}^{\mathrm{in}}) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b^{\mathrm{DT}}(\mathbf{x}^{\mathrm{in}})$$

(2) *Classification:*

$$\hat{y} = \underset{q}{\operatorname{argmax}}\, \boldsymbol{\pi}, \text{ with } \boldsymbol{\pi} = \frac{1}{B} \sum_{b=1}^{B} \boldsymbol{\pi}_b^{\mathrm{DT}}$$

Random Forests:

Algorithm: Random Forests for Regression and Classification

Training.
For $b = 1$ to B:

(1) Draw a bootstrap sample $\mathbf{Z}^{*b}$ from the training data.

(2) Train a tree $\hat{f}_b^{\mathrm{DT}}$ on the bootstrapped sample as follows.
For each terminal node at the current level, repeat:
(i): Draw at random a subset of m features from the p features.
(ii): Find the optimal split (j, s) (according to the relevant cost function), choosing j only among the m features randomly sampled in *(i)*.
(iii) Perform the split into two daughter nodes.
Until the stopping criterion is reached.

Making predictions (aggregating). Given a new data point $\mathbf{x}^{\mathrm{in}}$:

(1) *Regression:*

$$\hat{y} = \hat{f}(\mathbf{x}^{\mathrm{in}}) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b^{\mathrm{DT}}(\mathbf{x}^{\mathrm{in}})$$

(2) *Classification:*

Impurity of the region (node) where the sample ends in tree *b*

$$\hat{y} = \underset{q}{\operatorname{argmax}}\, \boldsymbol{\pi}, \text{ with } \boldsymbol{\pi} = \frac{1}{B}\sum_{b=1}^{B} \boldsymbol{\pi}_b^{\mathrm{DT}}$$

Random Forests:

Algorithm: Random Forests for Regression and Classification

Training.
For $b = 1$ to B:

(1) Draw a bootstrap sample $\mathbf{Z}^{*b}$ from the training data.

(2) Train a tree $\hat{f}_b^{\mathrm{DT}}$ on the bootstrapped sample as follows.
For each terminal node at the current level, repeat:
(i): Draw at random a subset of m features from the p features.
(ii): Find the optimal split (j, s) (according to the relevant cost function), choosing j only among the m features randomly sampled in *(i)*.
(iii) Perform the split into two daughter nodes.
Until the stopping criterion is reached.

Additional hyper-parameters to choose!

Making predictions (aggregating). Given a new data point $\mathbf{x}^{\mathrm{in}}$:

(1) *Regression:*

$$\hat{y} = \hat{f}(\mathbf{x}^{\mathrm{in}}) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b^{\mathrm{DT}}(\mathbf{x}^{\mathrm{in}})$$

(2) *Classification:*

Impurity of the region (node) where the sample ends in tree *b*

$$\hat{y} = \underset{q}{\mathrm{argmax}}\, \boldsymbol{\pi}, \text{ with } \boldsymbol{\pi} = \frac{1}{B}\sum_{b=1}^{B} \boldsymbol{\pi}_b^{\mathrm{DT}}$$

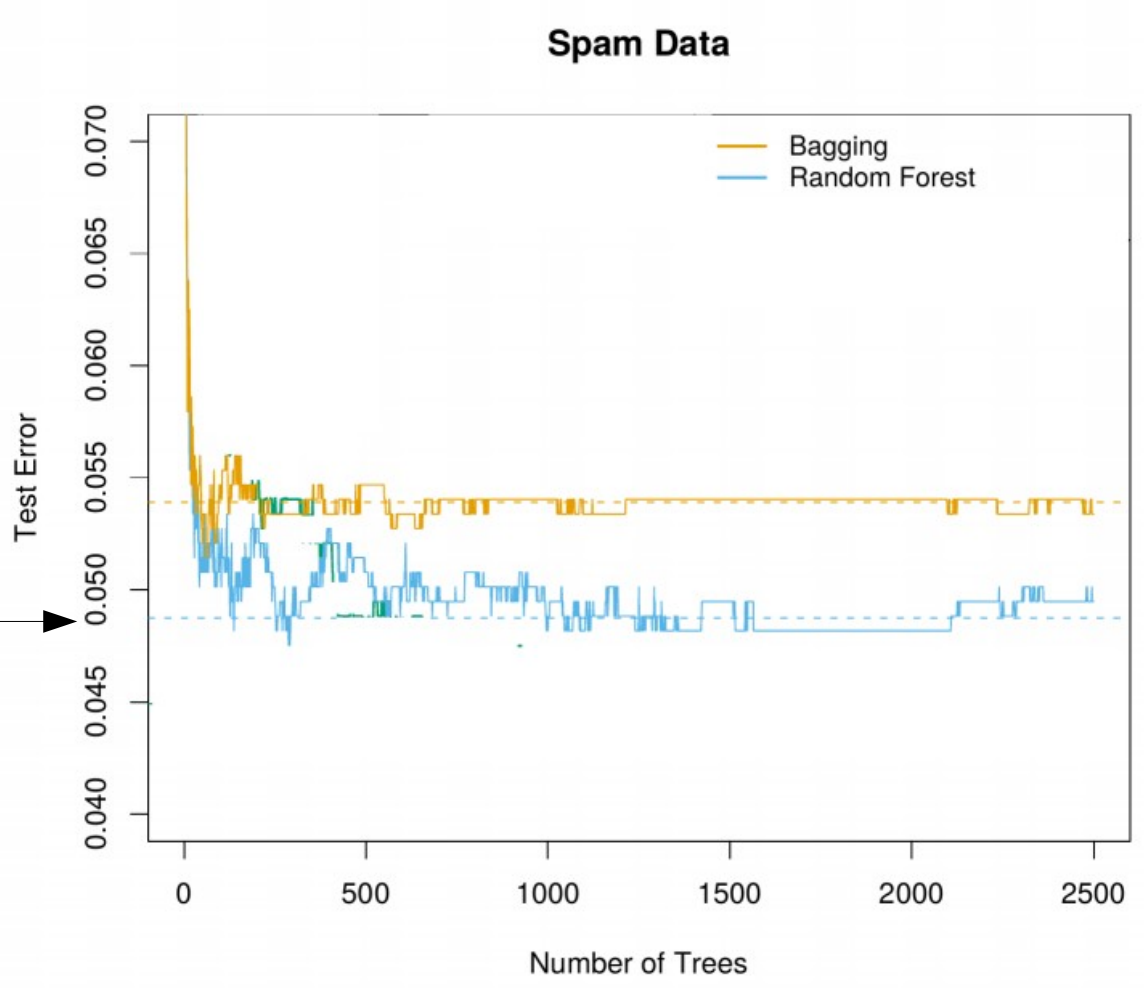
The spam example data illustrates:
1. Bagging improves over decision tree: reducing variance by averaging
2. Random forests improve over bagging alone by de-correlating the bagging trees

Bagging: error rate 5.4%

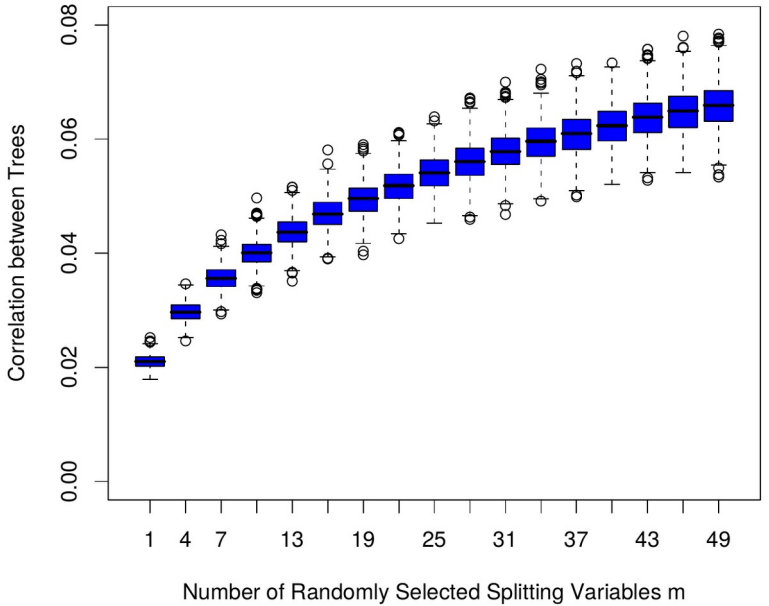
Random Forest: error rate 4.9%

As a comparison:

Simple Decision tree: 9.3%
Logistic regression: 7.6%



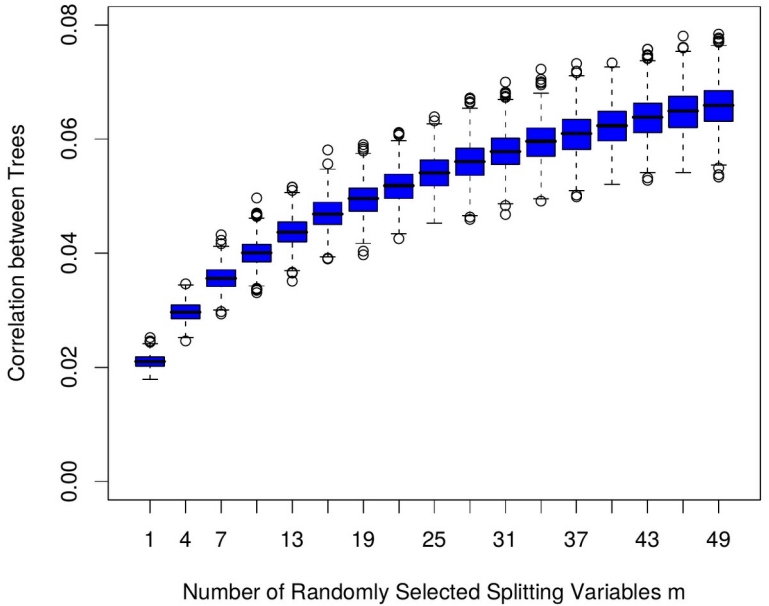
Number of splitting variables *m*



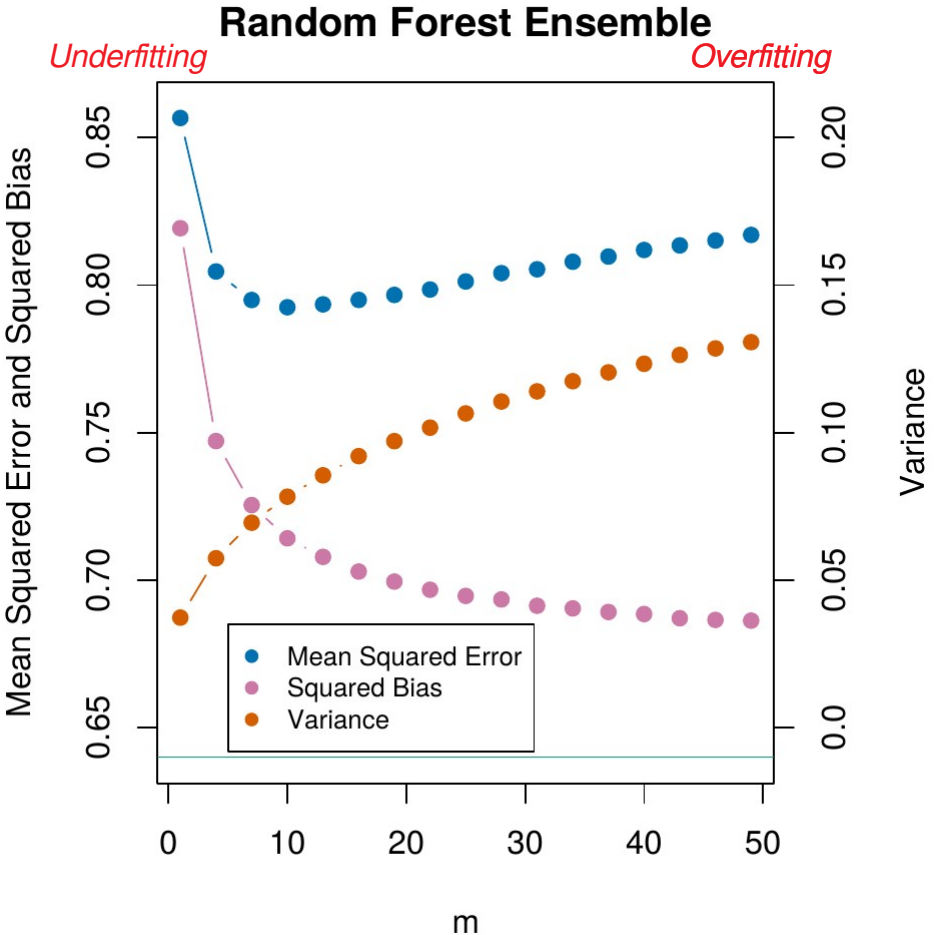
Less splitting variables lead to more dissimilar trees

*Simulated data for **regression task**, from Hastie, Tibshirani, Friedman, Elements of statistical learning, chap.15*

Number of splitting variables *m*



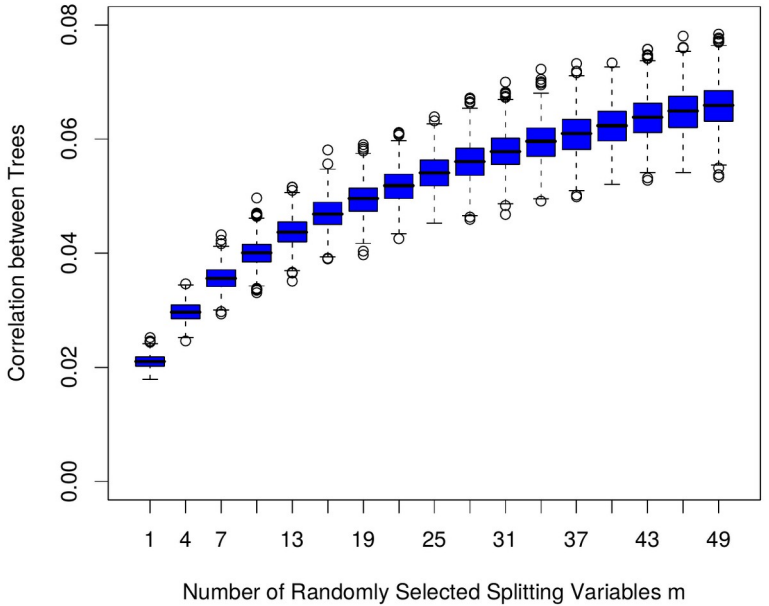
Less splitting variables lead to more dissimilar trees



m controls the bias-variance trade-off

*Simulated data for **regression task**, from Hastie, Tibshirani, Friedman, Elements of statistical learning, chap.15*

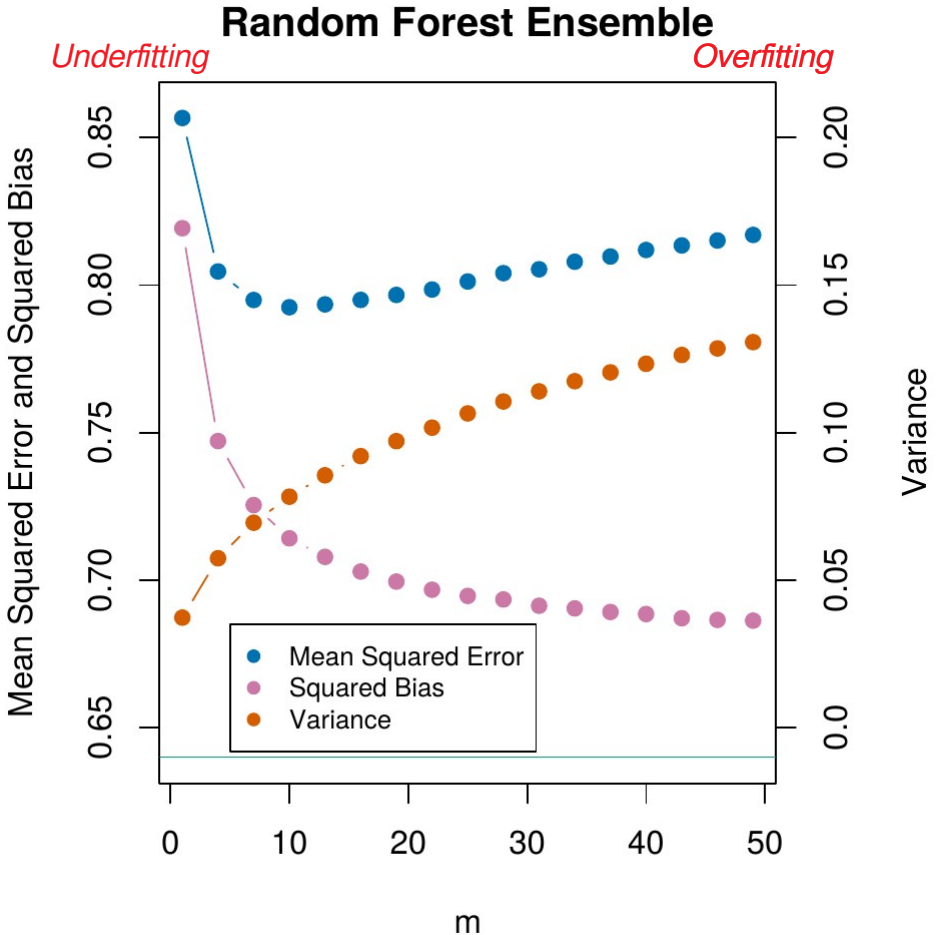
Number of splitting variables *m*



Less splitting variables lead to more dissimilar trees

Default choice (authors' recommendation):

- Regression $m \sim p/3$
- Classification $m \sim \sqrt{p}$



m controls the bias-variance trade-off

*Simulated data for **regression task**, from Hastie, Tibshirani, Friedman, Elements of statistical learning, chap.15*

Out-Of-Bag (OOB) samples

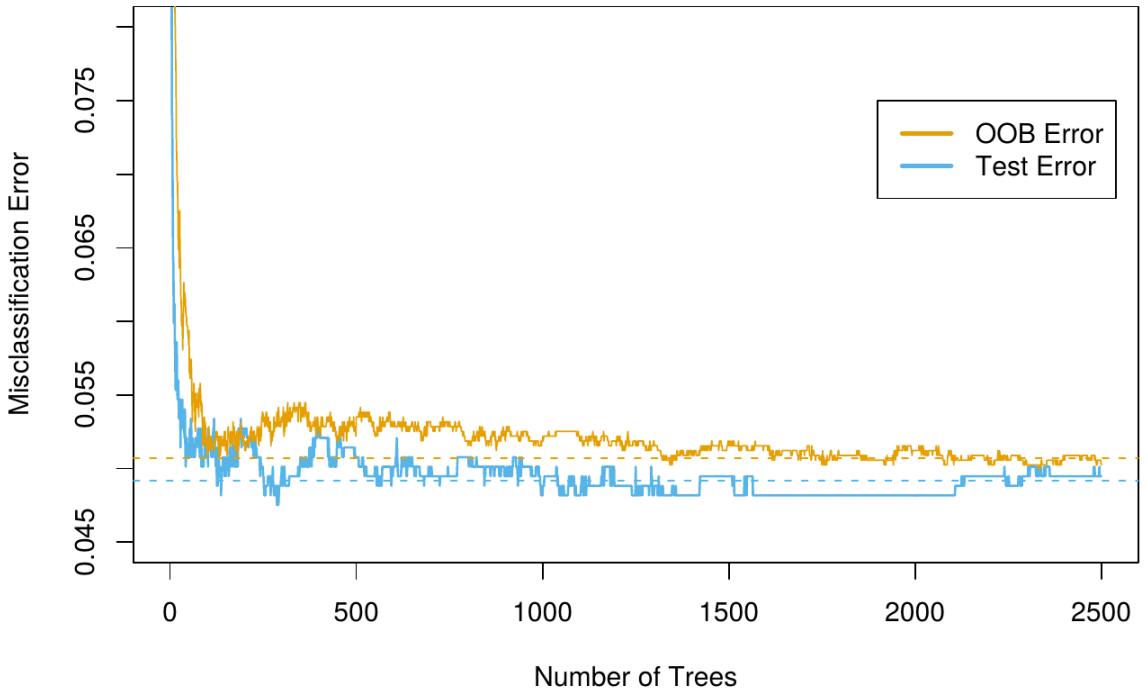
Samples that were *not* picked in a certain bootstrap sample. For the corresponding tree, OOB samples are *unseen* samples, thus can be used the OOB validation (hyper-parametric search)

Out-Of-Bag (OOB) samples

Samples that were *not* picked in a certain bootstrap sample. For the corresponding tree, OOB samples are *unseen* samples, thus can be used the OOB validation (hyper-parametric search)

Spam example

OOB error: calculated by taking as predictor, for a data point, the average of predictions by trees corresponding to bootstrap samples in which the data point did not appear.

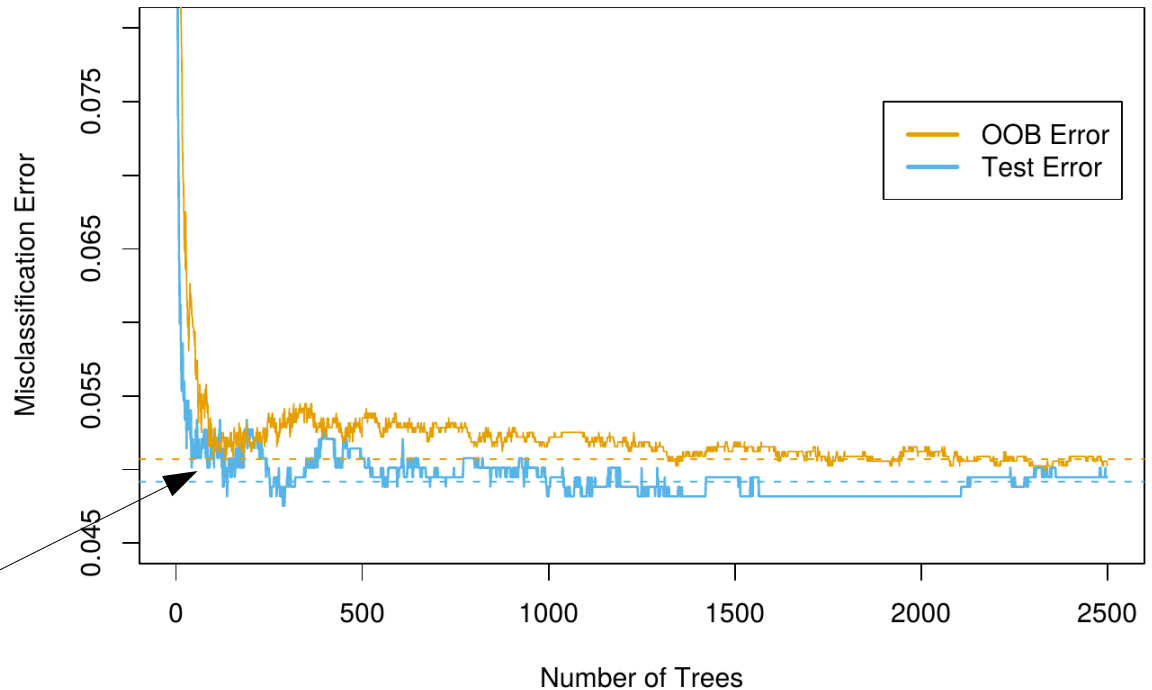


Out-Of-Bag (OOB) samples

Samples that were *not* picked in a certain bootstrap sample. For the corresponding tree, OOB samples are *unseen* samples, thus can be used the OOB validation (hyper-parametric search)

Spam example

OOB error: calculated by taking as predictor, for a data point, the average of predictions by trees corresponding to bootstrap samples in which the data point did not appear.



About 200 trees are already sufficient

OOB error is equivalent to the one obtained by cross-validation, hence when the OOB error stabilizes one can stop increasing B

A way to gain interpretability: Variable Importance

For each tree:

OOB samples

Class	Features				
	CAPTOT	ch$	remove	business	
email	0.11	0.053	0.21	0.003	...
spam	0.10	0.015	0.25	0.081	...
email	0.21	0.012	0.32	0.001	...
		...			

Class	Features				
	CAPTOT	ch$	remove	business	
email	0.11	0.053	0.25	0.003	...
spam	0.10	0.015	0.32	0.081	...
email	0.21	0.012	0.21	0.001	...
		...	*k*		

Importance of variable *k* = |OOB accuracy - OOB accuracy with variable *k* permuted|

(To average over the trees)

Spam example from Hastie, Tibshirani, Friedman, Elements of statistical learning, chap. 15

A way to gain interpretability: Variable Importance

For each tree:

OOB samples

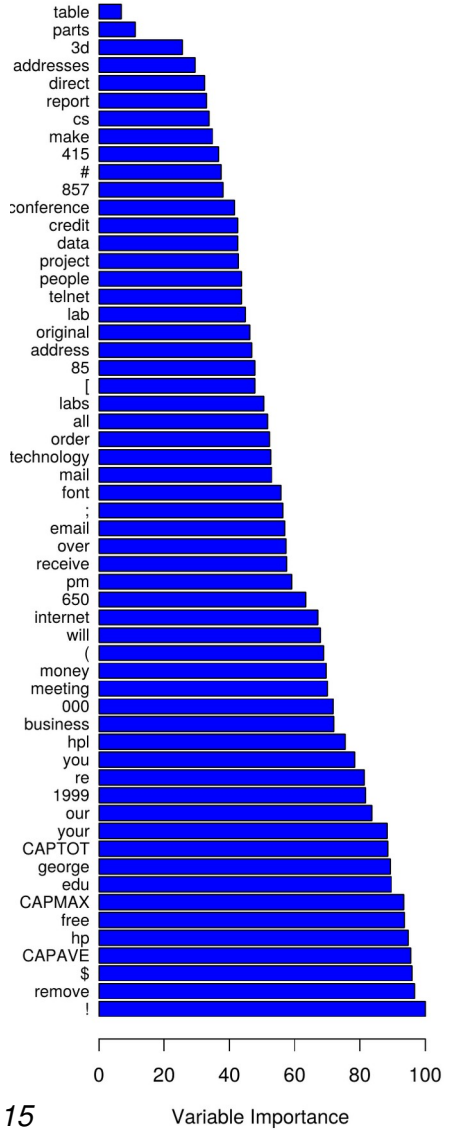
Class	CAPTOT	ch$	remove	business	...
email	0.11	0.053	0.21	0.003	...
spam	0.10	0.015	0.25	0.081	...
email	0.21	0.012	0.32	0.001	...
		...			

Class	CAPTOT	ch$	remove	business	...
email	0.11	0.053	0.25	0.003	...
spam	0.10	0.015	0.32	0.081	...
email	0.21	0.012	0.21	0.001	...
		...	*k*		

Importance of variable *k* = |OOB accuracy - OOB accuracy with variable *k* permuted|

(To average over the trees)

1. Some predictors are more important than others in separating spam from e-mail



Spam example from Hastie, Tibshirani, Friedman, Elements of statistical learning, chap. 15

A way to gain interpretability: Variable Importance

For each tree:

OOB samples

Class	Features				
	CAPTOT	ch$	remove	business	
email	0.11	0.053	0.21	0.003	...
spam	0.10	0.015	0.25	0.081	...
email	0.21	0.012	0.32	0.001	...
	...				

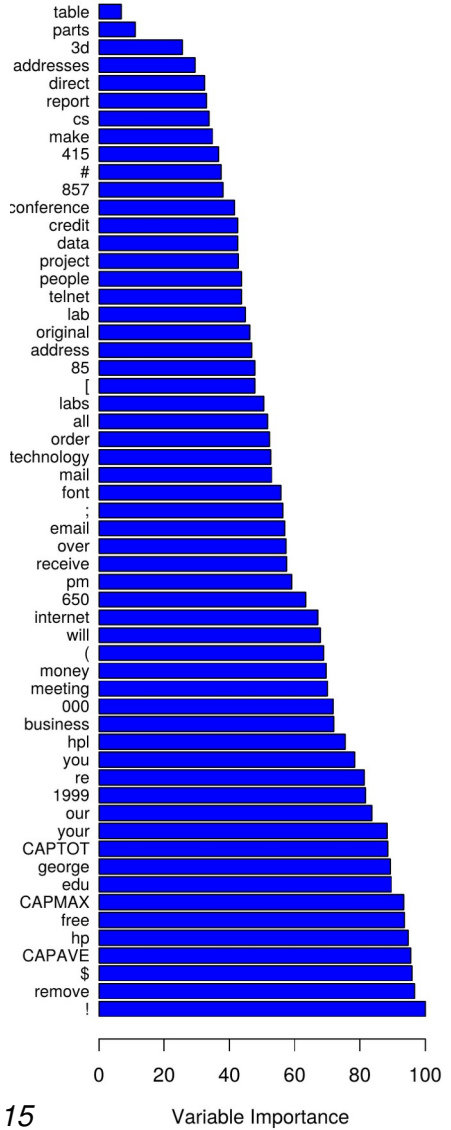
Class	Features				
	CAPTOT	ch$	remove	business	
email	0.11	0.053	0.25	0.003	...
spam	0.10	0.015	0.32	0.081	...
email	0.21	0.012	0.21	0.001	...
		...	*k*		

Importance of variable *k* = |OOB accuracy - OOB accuracy with variable *k* permuted|

(To average over the trees)

1. Some predictors are more important than others in separating spam from e-mail

2. In general, it's important to identify what is the predictive power of each feature (e.g. get insights into the main determinants of a given outcome).



Spam example from Hastie, Tibshirani, Friedman, Elements of statistical learning, chap. 15

Table 6. Mean-squared test set error.

Data set	Bagging	Adapt. bag	Forest
Boston Housing	11.4	9.7	10.2
Ozone	17.8	17.8	16.3
Servo $\times 10 - 2$	24.5	25.1	24.6
Abalone	4.9	4.9	4.6
Robot Arm $\times 10 - 2$	4.7	2.8	4.2
Friedman #1	6.3	4.1	5.7
Friedman #2 $\times 10 + 3$	21.5	21.5	19.6
Friedman #3 $\times 10 - 3$	24.8	24.8	21.6

See additional reading for other examples and comparisons

Table 8. Mean-squared test set error.

Random feature selection + noise on the output works better than bagging alone →

Data set	With bagging	With Noise
Boston Housing	10.2	9.1
Ozone	17.8	16.3
Servo $\times 10 - 2$	24.6	23.2
Abalone	4.6	4.7
Robot Arm $\times 10 - 2$	4.2	3.9
Friedman #1	5.7	5.1
Friedman #2 $\times 10 + 3$	19.6	20.4
Friedman #3 $\times 10 - 3$	21.6	19.8

Examples of RF for regression, from Breiman 2001

Gradient Boosting:

Given a *strong* model (like DT): it's the iterative addition of *weak* models to reduce the error of the strong one.

Gradient Boosting:

Given a *strong* model (like DT): it's the iterative addition of *weak* models to reduce the error of the strong one.

Take a regressor with loss=squared error

Shrinkage: controls the rate of update

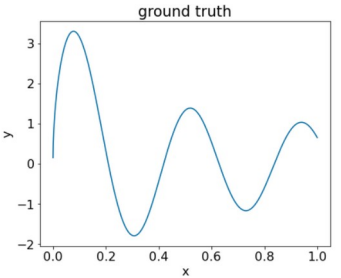
$$S_{t+1} = S_t - \nu w_t$$

Strong model prediction at iteration t+1

Weak model prediction at t

*Weak model is trained to predict the **error** of the strong model at iteration t*

Gradient Boosting:



Given a *strong* model (like DT): it's the iterative addition of *weak* models to reduce the error of the strong one.

Take a regressor with loss=squared error

Shrinkage: controls the rate of update

$$S_{t+1} = S_t - \nu w_t$$

Strong model prediction at iteration t+1

Weak model prediction at t

*Weak model is trained to predict the **error** of the strong model at iteration t*

Example from https://developers.google.com/machine-learning/decision-forests/intro-to-gbdt

Gradient Boosting:

Given a *strong* model (like DT): it's the iterative addition of *weak* models to reduce the error of the strong one.

Take a regressor with loss=squared error

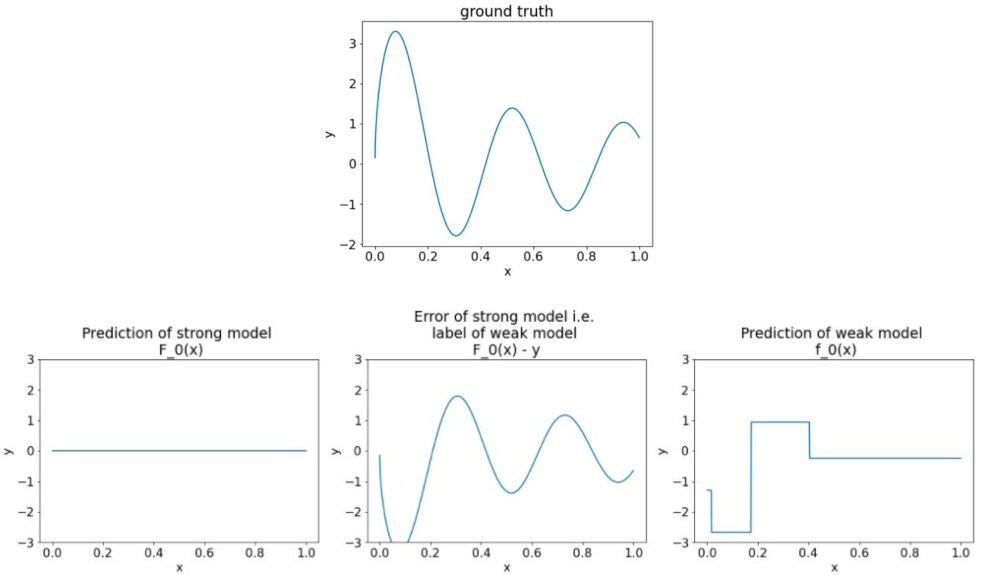
Shrinkage: controls the rate of update

$$S_{t+1} = S_t - \nu w_t$$

Strong model prediction at iteration t+1

Weak model prediction at t

*Weak model is trained to predict the **error** of the strong model at iteration t*



Example from https://developers.google.com/machine-learning/decision-forests/intro-to-gbdt

Gradient Boosting:

Given a *strong* model (like DT): it's the iterative addition of *weak* models to reduce the error of the strong one.

Take a regressor with loss=squared error

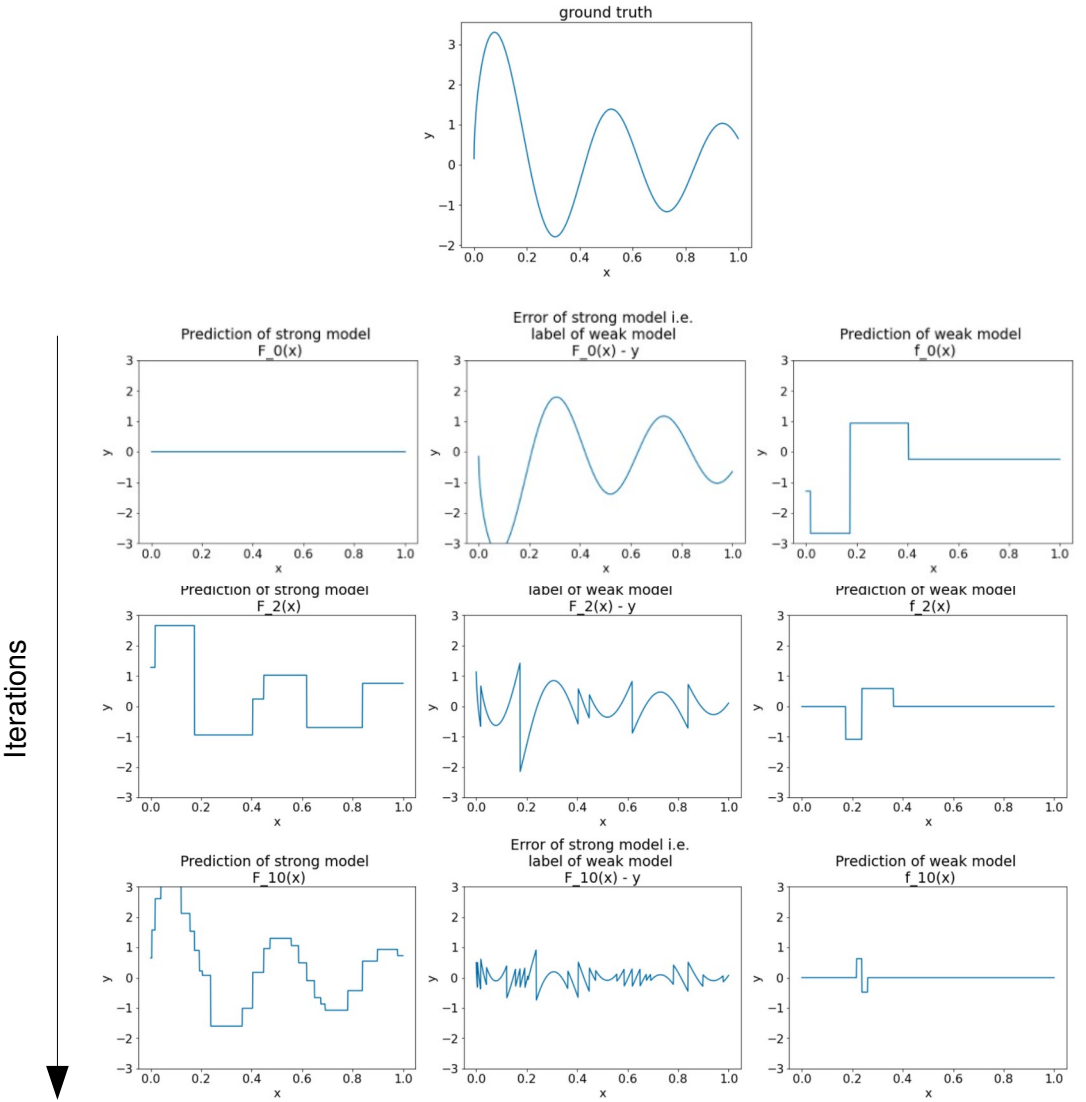
Shrinkage: controls the rate of update

$$S_{t+1} = S_t - \nu w_t$$

Strong model prediction at iteration t+1

Weak model prediction at t

*Weak model is trained to predict the **error** of the strong model at iteration t*



Example from https://developers.google.com/machine-learning/decision-forests/intro-to-gbdt

Gradient Boosting:

Given a *strong* model (like DT): it's the iterative addition of *weak* models to reduce the error of the strong one.

More generally:

Shrinkage: controls the rate of update

$$S_{t+1} = S_t - \nu w_t$$

Strong model prediction at iteration t+1

Weak model prediction at t

Weak models are trained iteratively to predict the ***gradient of the loss*** *wrt the strong predictions:*

$$w_t = \partial L(y, S_t)/\partial S_t$$

Gradient Boosting:

Given a *strong* model (like DT): it's the iterative addition of *weak* models to reduce the error of the strong one.

More generally:

Shrinkage: controls the rate of update

$$S_{t+1} = S_t - \nu w_t$$

Strong model prediction at iteration t+1

Weak model prediction at t

Weak models are trained iteratively to predict the ***gradient of the loss*** *wrt the strong predictions:*

$$w_t = \partial L(y, S_t)/\partial S_t$$

With a squared error loss, one recovers the strong ***signed error*** *as learning target for the weak model*

$$L(y, S_t) = (y - S_t)^2$$

$$\frac{\partial L(y, S_t)}{\partial S_t} = -2(y - S_t)$$

Summary:

1. Decision trees:

- Minimising CE and GINI index produces regions dominated by 1 class
- Tree size: one can set constraints or use cross-validation (see spam example)
- Gradient boosting: iterative procedure fitting the error to improving performance

Summary:

1. Decision trees:

- Minimising CE and GINI index produces regions dominated by 1 class
- Tree size: one can set constraints or use cross-validation (see spam example)
- Gradient boosting: iterative procedure fitting the error to improving performance

2. Need to reduce the variance (cause of overfitting):

Random Forests = bagging + random selection of features

Summary:

1. Decision trees:

- Minimising CE and GINI index produces regions dominated by 1 class
- Tree size: one can set constraints or use cross-validation (see spam example)
- Gradient boosting: iterative procedure fitting the error to improving performance

2. Need to reduce the variance (cause of overfitting):

Random Forests = bagging + random selection of features

- Number of decision trees and splitting features needs to be selected
- OOB estimates can be used for: hyper-parametric search, variable importance