Some machine learning strategies and Geostatistics

Mathieu Ribatet—Full Professor of Statistics

0. Reminder

Statistical learning workflow

1. Descriptive analysis:
   - ☐ have glimpse at the data
   - ☐ check for possible outliers
   - ☐ detect any interesting pattern
2. Inferential statistics:
   - ☐ Hypothesis testing (χ^2–tests, anova. . .)
   - ☐ Model's definition that answer the question (linear models, survival models, random forests, . . .)
3. Model fitting (maximum likelihood / robust / sparse estimator, loss minimization, . . .)
4. Model selection / checking (bias, homoscedasticity, leverage, influence. . .)
5. Evaluate models' predictive performances

We will focus on supervised tasks only.

Model based measures

- Model based measures are used for model selection and (often) rely on probabilistic arguments, e.g., convergence to a χ^2 distribution.
- Popular measures are:
  - Information Criterion, typically of the form

$$-\text{goodness of fit} + \text{model complexity},$$

    such as Akaïke or Bayesian Information Criterion (AIC / BIC).
  - Hypothesis testing, typically of the form

$$H_0 : \text{Nested model is true} \qquad H_1 : \text{More complex model is true},$$

    such as the likelihood ratio test or the t–test.

This lecture will not cover model based performance measures. . . (discussed in another lecture)

Loss functions

Definition 1. A function ℓ defined on $\mathcal{Y} \times \mathcal{Y}$, where $\mathcal{Y}$ is the outcome set, is said to be a loss function if for all $y, y' \in \mathcal{Y}$ we have

$$\ell(y, y) = 0, \qquad \ell(y, y') > 0, \quad y \neq y'.$$

- Loss functions are used for model fitting or evaluating models' performances.
- Common loss functions:

$$\ell(y, y') = \begin{cases} \|y - y'\|_2^2, & \text{quadratic}/\ell_2 \text{ loss} \quad \text{(regression)} \\ \|y - y'\|_1, & \text{absolute}/\ell_1 \text{ loss} \quad \text{(robust regression)} \\ 1_{\{y \neq y'\}} & \text{0–1 loss} \quad \text{(hard classification)} \\ -y^\top \log y', & \text{cross entropy} \quad \text{(soft classification)} \\ -\sum_i \log f(y_i), & \text{Neg. log-lik.} \quad \text{(model based)} \end{cases}$$

Technically speaking, the negative log–likelihood is not a loss function but is central in statistics so I had to mention it here!

Generalization error

- Evaluate model's performance is useful to:
  - Estimate the Generalization Error

$$\mathbb{E}_{(Y,\mathbf{X})}\left[\ell(Y,\hat{Y})\right],$$

    where ℓ is a given loss function, $(Y,\mathbf{X})$ is the distribution of a future observation and $\hat{Y}=\hat{f}(\mathbf{X})$ is the prediction from a known fitted model given $\mathbf{X}$.
  - Pick the best model among several fitted models $\hat{Y}_j=\hat{f}_j(\mathbf{X})$, $j=1,\ldots,J$, (checked!) candidates

$$\underset{j=1,\ldots,J}{\arg\min}\ \mathbb{E}_{(Y,\mathbf{X})}\left[\ell\left(Y,\hat{Y}_j\right)\right]$$

Generalization error estimation

$$R_{\text{theo}}(\hat{\theta}) = \mathbb{E}_{(Y,\mathbf{X})}\left[\ell(Y, \hat{Y})\right]$$

- The generalization error is unknown since the distribution of $(Y, \mathbf{X})$ is unkown
- A consistent estimator of the generalization error is

$$R_{\text{emp}}(\hat{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell(Y_i, \hat{Y}_i) \longrightarrow \mathbb{E}_{(Y,\mathbf{X})}\left[\ell(Y, \hat{Y})\right], \qquad n \to \infty, \qquad (1)$$

  where $(Y_i, \mathbf{X}_i)$ are independent copies of $(Y, \mathbf{X})$.
- In the above expression the estimator $\mathbf{x} \mapsto \hat{Y} = \hat{f}(\mathbf{x})$ is considered as known.

Never (ever!) use (1) on the data set used to fit $\hat{f}$, i.e., training set. You must compute (1) using either hold out or cross-validation strategies—see later.

Overfitting and underfitting

- As said previously $R_{\text{emp}}(\hat{\theta})$ is different from $R_{\text{theo}}(\hat{\theta})$
- Actually $R_{\text{emp}}(\hat{\theta})$ computed on the training set underestimates the generalization error and often leads to overfitting.
- Need another estimate of the generalization error.

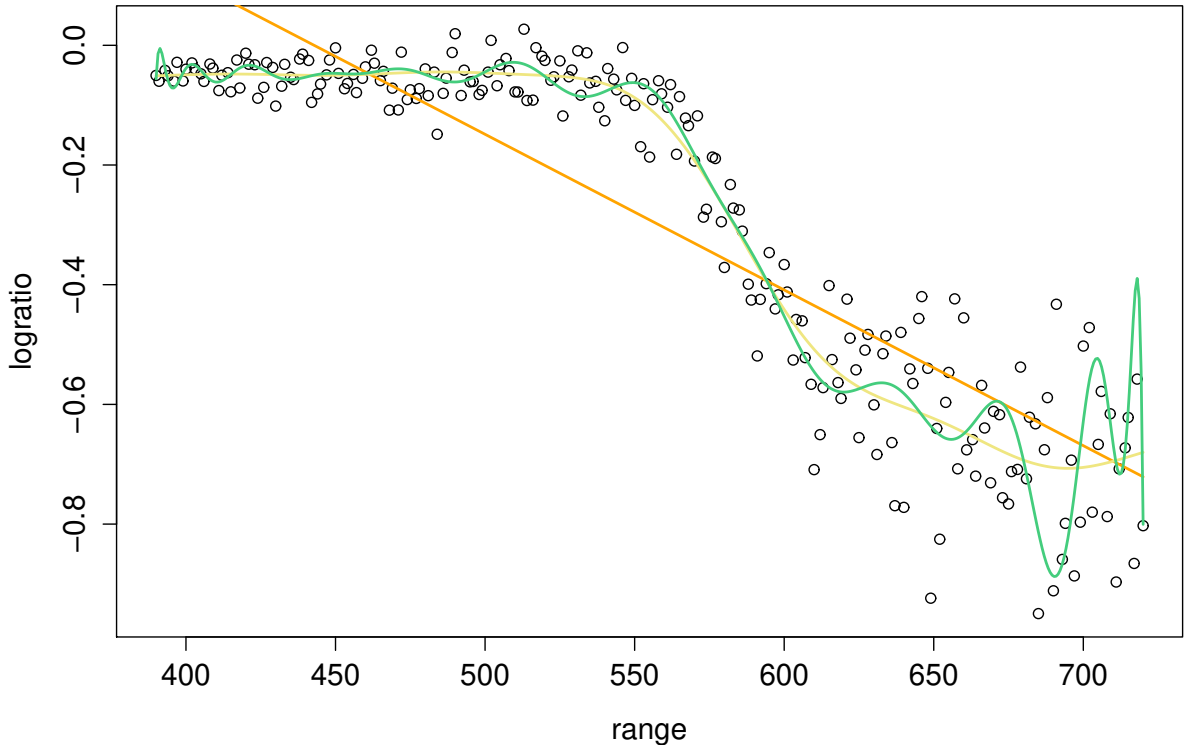


Figure 1: *Illustration of underfitting and overfitting on the* `lidar` *dataset. col2: Underfitting; Kakhi: Right fit; Green: Overfitting.*

Test/Validation risk

- Suppose we have a second, independent of $\mathscr{D}_n$, dataset, say $\tilde{\mathscr{D}}_{n_2}$[1].
- We can then compute the test/validation risk

$$R_{\text{test}}(\hat{\theta}) = \frac{1}{n_2} \sum_{i=1}^{n_2} \ell(Y_i; f(X_i; \hat{\theta})).$$

- We then have

$$\mathbb{E}\left\{ R_{\text{test}}(\hat{\theta}) \mid \mathscr{D}_n \right\} = R_{\text{theo}}(\hat{\theta}).$$

[1] Or that we have divided into two pieces the original dataset $\mathscr{D}_n$...

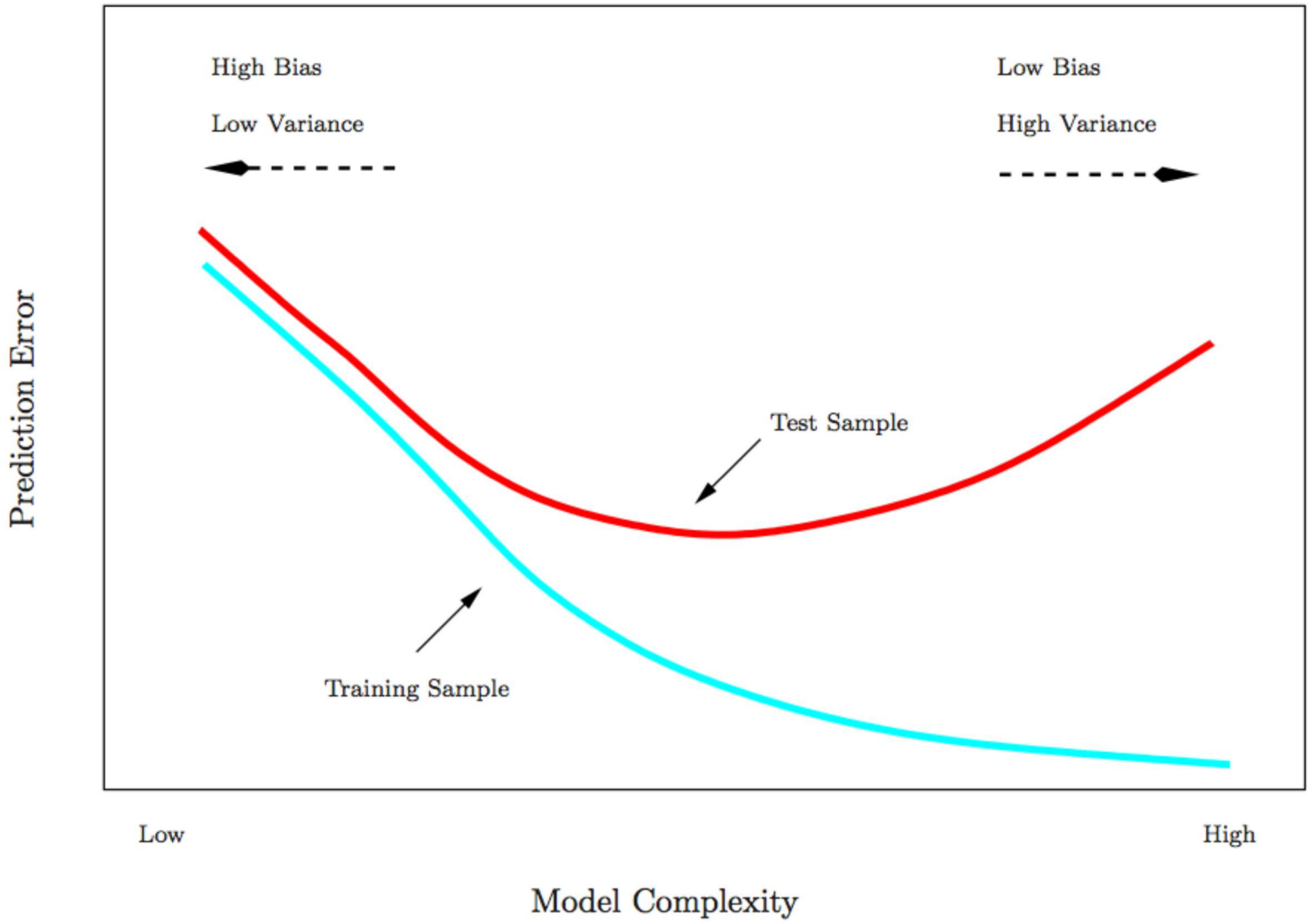


Figure 2: *Typical evolution of the training error and test error as the model complexity increases.*

Validation set

Algorithm 1: "Hold-out validation" algorithm.

input : A dataset $\mathscr{D}_n$, a training set $\mathcal{T} \subset \{1, \ldots, n\}$ and a validation set $\mathcal{V} \subset \{1, \ldots, n\}$ such that $\mathcal{T} \cap \mathcal{V} = \emptyset$ and $\mathcal{T} \cup \mathcal{V} = \{1, \ldots, n\}$.

output: An estimate of the generalization error $R_{\text{theo}}(\hat{\theta})$.

1 Compute the decision rule

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \ell(Y_i; f(X_i; \theta))$$

2 Return an estimate of the generalization error

$$\hat{R}_{\text{theo}}(\hat{\theta}) = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \ell(Y_i; f(X_i; \hat{\theta})).$$

Such an approach has a main drawback: the generalization error estimate heavily relies on the validation set. It would be better to have several validation sets to mitigate this effect.

Cross–validation "leave p out"

Algorithm 2: "Leave p out cross validation" algorithm.

input : A dataset $\mathscr{D}_n$ and an integer $p < n$.
output: An estimate of the generalization error $R_{\text{theo}}(\hat{\theta})$.

1 Compute the $n_p = \binom{n}{p}$ subsets, say $\mathcal{V}_1, \ldots, \mathcal{V}_{n_p}$, of $\{1, \ldots, n\}$ having p elements.
2 **for** $k \leftarrow 1$ **to** s_{n_p} **do**
3 Build the training set $\mathcal{T}_k = \mathscr{D}_n \setminus \mathcal{V}_k$;
4 Compute the decision rule $\hat{\theta}_k = \arg\min_{\theta \in \Theta} \frac{1}{|\mathcal{T}_k|} \sum_{i \in \mathcal{T}} \ell(Y_i; f(X_i; \theta))$;
5 Compute an estimate of the generalization error for $\mathcal{V}_k$

$$\hat{R}_{\text{theo},k}(\hat{\theta}_k) = \frac{1}{|\mathcal{V}_k|} \sum_{i \in \mathcal{V}_k} \ell(Y_i; f(X_i; \hat{\theta}_k)).$$

6 Return an estimate of the generalization error $\tilde{R}_{\text{theo}}(\hat{\theta}) = \frac{1}{n_p} \sum_{k=1}^{n_p} \hat{R}_{\text{theo},k}(\hat{\theta}_k)$;

Cross–validation "leave p out"

Algorithm 2: "Leave p out cross validation" algorithm.

input : A dataset $\mathscr{D}_n$ and an integer $p < n$.
output: An estimate of the generalization error $R_{\text{theo}}(\hat{\theta})$.

1 Compute the $n_p = \binom{n}{p}$ subsets, say $\mathcal{V}_1, \ldots, \mathcal{V}_{n_p}$, of $\{1, \ldots, n\}$ having p elements.
2 **for** $k \leftarrow 1$ **to** s_{n_p} **do**
3 Build the training set $\mathcal{T}_k = \mathscr{D}_n \setminus \mathcal{V}_k$;
4 Compute the decision rule $\hat{\theta}_k = \arg\min_{\theta \in \Theta} \frac{1}{|\mathcal{T}_k|} \sum_{i \in \mathcal{T}} \ell(Y_i; f(X_i; \theta))$;
5 Compute an estimate of the generalization error for $\mathcal{V}_k$

$$\hat{R}_{\text{theo},k}(\hat{\theta}_k) = \frac{1}{|\mathcal{V}_k|} \sum_{i \in \mathcal{V}_k} \ell(Y_i; f(X_i; \hat{\theta}_k)).$$

6 Return an estimate of the generalization error $\tilde{R}_{\text{theo}}(\hat{\theta}) = \frac{1}{n_p} \sum_{k=1}^{n_p} \hat{R}_{\text{theo},k}(\hat{\theta}_k)$;

- The drawback of such an approach is that it may be very time consuming (unless $p \in \{1, 2\}$).
- The widely used case $p = 1$ is often called leave one out for obvious reasons.

Cross–validation "K fold"

Algorithm 3: "K fold cross validation" algorithm.

input : A dataset $\mathscr{D}_n$ and $K \geq 2$ divisor of n.
output: An estimate of the generalization error $R_{\text{theo}}(\hat{\theta})$.

1 Compute K disjoint subsets, say $\mathcal{V}_1, \ldots, \mathcal{V}_K$, of $\{1, \ldots, n\}$.
2 **for** $k \leftarrow 1$ **to** s_{n_p} **do**
3 Build the training set $\mathcal{T}_k = \mathscr{D}_n \setminus \mathcal{V}_k$;
4 Compute the decision rule $\hat{\theta}_k = \arg\min_{\theta \in \Theta} \frac{1}{|\mathcal{T}_k|} \sum_{i \in \mathcal{T}} \ell(Y_i; f(X_i; \theta))$;
5 Compute an estimate of the generalization error for $\mathcal{V}_k$

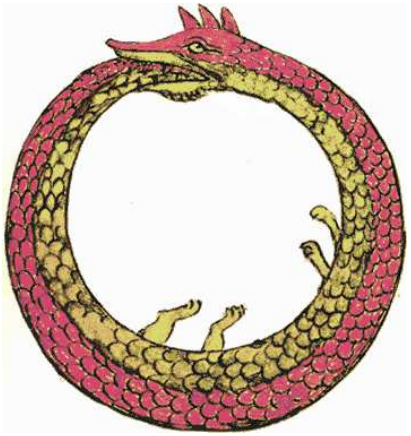
$$\hat{R}_{\text{theo},k}(\hat{\theta}_k) = \frac{K}{n} \sum_{i \in \mathcal{V}_k} \ell(Y_i; f(X_i; \hat{\theta}_k))$$

6 Return an estimate of the generalization error $\tilde{R}_{\text{theo}}(\hat{\theta}) = \frac{1}{K} \sum_{k=1}^{K} \hat{R}_{\text{theo},k}(\hat{\theta}_k)$;

Typical choices: $K \in \{5, 10, n\}$. When $K = n$ it is the leave one out cross validation.

Train + Validation + Test

- Watch out if when building your decision rule you tuned some hyper-parameters[2], then you must use the train + validation + test.
- Essentially the validation set will be used to tune these hyperparameters while the test set will be used to estimate the generalization error.
- Indeed if you tune and estimate the generalization error on the same dataset, i.e., the test dataset, then you will underestimate the generalization error since you



did set those hyperparameters to minimize the generalization error!!!

[2] Don't worry we will define what it is later. Right now you can just assume that these are paramters that you do not estimate but rather held fix to some value of your choice.

Small / Moderate sample size

- Splitting the orginal data set into training / testing or training / validation / testing sets requires a large amount of observations, e.g., $n > 1000$.
- For smaller sample sizes it is good practice to use K–fold validation strategies.
- Without to much details, we simply:

  1. Partition the original data set into K non overlapping parts (of equal size)
  2. Fit the model using $K - 1$ parts
  3. Estimate the generalization error using the remaining block
  4. Iterate with a different split and average all estimations

- Some guidelines for choosing K
  - For moderate sample sizes, e.g., $250 \leq n \leq 1000$, $K = 5, 10$
  - For small sample size, e.g., $n < 250$, $K = n$ known as leave one out.

> **i** Successive estimations will be dependent and, under mild regularity conditions, the K–fold estimator is still consistent (convergence may be slower though).

Classification

Classifier

Definition 2. A classifier is just a mapping

$$f : E \longrightarrow \{1, \ldots, K\}$$
$$x \longmapsto f(x)$$

where E is the input space and here the output space has K possible outcomes.

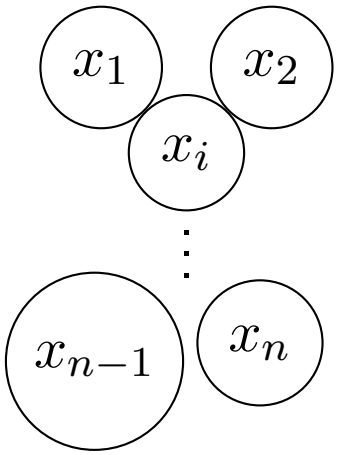
- There are different situations for the input space E:
  - E is only a "feature space", i.e., no label, we talk about unsupervised classification
  - $E =$ feature space $\times \{1, \ldots, K\}$, i.e., with labels, we talk about supervised classification.

What is, simply put, classification?

Definition 3. Classification, clustering or segmentation refers to a statistical framework that puts a label to each (potentially new) observation.

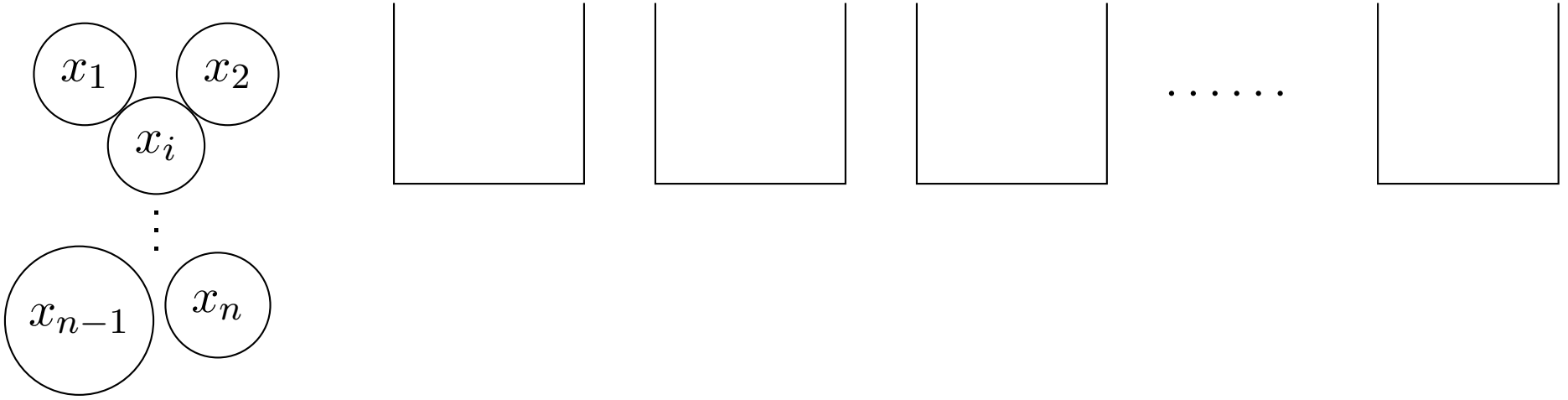
What is, simply put, classification?

Definition 3. Classification, clustering or segmentation refers to a statistical framework that puts a label to each (potentially new) observation.



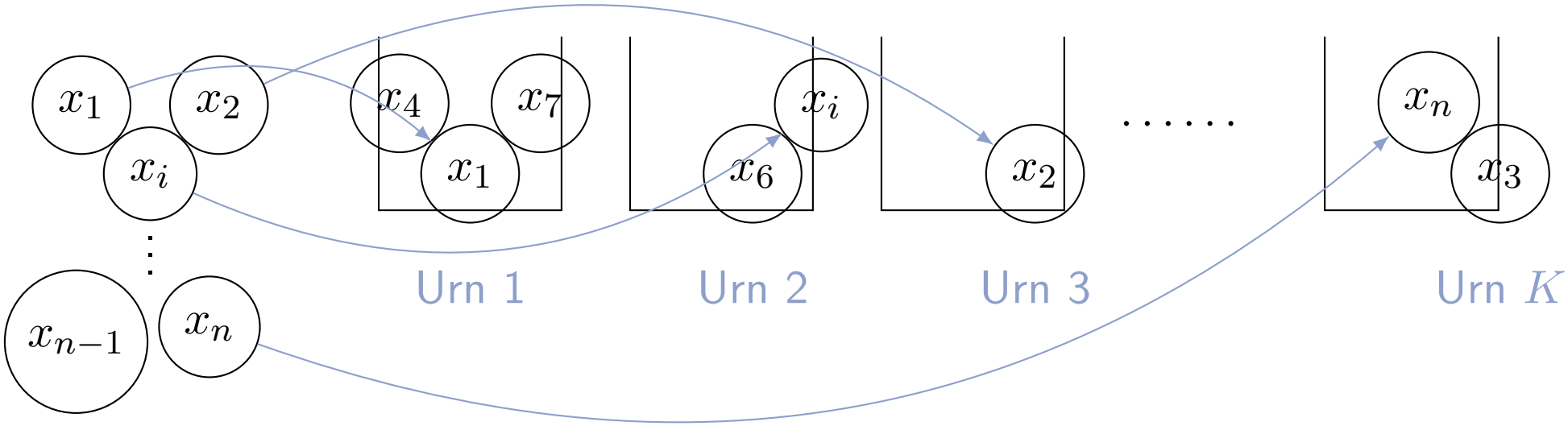
What is, simply put, classification?

Definition 3. Classification, clustering or segmentation refers to a statistical framework that puts a label to each (potentially new) observation.



What is, simply put, classification?

Definition 3. Classification, clustering or segmentation refers to a statistical framework that puts a label to each (potentially new) observation.



Remark. I will indifferently talk about urn, cluster, label or classes to talk about the group associated to a given observation.

Hard and soft/fuzzy classification

- In my previous example, each ball was placed in a single urn. It is called hard classification or simple classification.
- Sometimes one may wish to put a ball in several urns with some probabilities. This is known as soft classification or fuzzy classification.
- Soft classification often outputs a vector of probabilities that belongs to the unit K–simplex, i.e.,

$$p \in \mathbb{S}_K = \{u \in (0, \infty)^K : \sum_{j=1}^{K} u_j = 1\}$$

and where the conditional probability of membership are given by

$$p_j(x) = \Pr(\text{Belongs to label } j \mid \text{given characteristics } x)$$

How to fill in urns?

Remark. Consider the case where we have n balls and K urns. The number of possible partitions using those K (non empty) urns corresponds to the Stirling numbers of the second kind $S(n, K)$.

How to fill in urns?

Remark. Consider the case where we have n balls and K urns. The number of possible partitions using those K (non empty) urns corresponds to the Stirling numbers of the second kind $S(n, K)$.

- We thus need a way to "order" all these possible configurations.
- The main idea is to have as much as possible:
  - "homogeneous element within urns"
  - "non homogeneous set of clusters"

Different ways to "measure this homogeneity" will lead to different classifiers.

Examples of classifiers

Supervised

- Linear discriminant analysis
- Quadratic discriminant analysis
- Logistic regression
- Random forests

Unsupervised

- k–means
- k–medoids
- Gaussian mixtures
- Hierarchical clustering
- Spectral clustering

Examples of classifiers

Supervised

- Linear discriminant analysis
- Quadratic discriminant analysis
- Logistic regression
- Random forests

Unsupervised

- k–means
- k–medoids
- Gaussian mixtures
- Hierarchical clustering
- Spectral clustering

i We talk about classification for the supervised case and clustering for the unsupervised case. In French there is no such a distinction although some talk about "classification automatique" for the unsupervised case.

- Recall that a K–class (supervised) classifier is just a mapping

$$f : \mathcal{X} \longrightarrow \{1, \ldots, K\}$$
$$\mathbf{x} \longmapsto y = f(\mathbf{x}),$$

  where $\mathcal{X}$ is the covariates / features space.
- In concrete situations, the mapping f is estimated from a (supervised) dataset $\mathcal{D}_n = \{(Y_i, \mathbf{X}_i) : i = 1, \ldots, n\}$
- To ease notations we will write f for $\hat{f}$ but still use $\hat{Y}$

It leads to spurious notations since we have $\hat{Y} = f(\mathbf{X})$.

Confusion matrix

The confusion matrix is a contingency table between true labels and predicted labels, i.e.,

		Predictions		
		Positive	Negative	Total
Truth	Positive	True Positive (TP)	False Negative (FN)	P
	Negative	False Positive (FP)	True Negative (TN)	N
	Total	PP	PN	n

Extension to more than 2 classes is straightforward using a one vs. all approach.

Summary statistics of a confusion matrix

- Comparing confusion matrix can be difficult
- It is convenient to summarize the confusion matrix into some numerical values to get an ordering.
- Common choices are
  - accuracy
  - sensitivity, recall, true positive rate
  - specificity, true negative rate
  - precision
  - F1 score
  - prevalence

Accuracy

- The Accuracy is given by

$$\frac{\text{number of true positives and true negatives}}{\text{sample size}} = \frac{TP + TN}{n}$$

- Evaluates the overall performance. labels.
- It is the sample version of

$$\Pr(Y = \hat{Y} \mid \mathbf{X}) \qquad \texttt{(how likely predictions are correct)}$$

		Predictions		
		Positive	Negative	Total
Truth	Positive	52	18	70
	Negative	21	9	30
	Total	73	27	100

$$\text{Accuracy} = \frac{52 + 9}{100} = 61\%$$

Recall / Sensitivity / True Positive Rate

- The recall is given by

$$\frac{\text{number of true positives}}{\text{number of positive cases}} = \frac{TP}{P}$$

- Puts emphasis on positive cases.
- It is the sample version of

$$\Pr(\hat{Y} = 1 \mid Y = 1, \mathbf{X}) \qquad \texttt{(given it is } \oplus\texttt{, how likely prediction is correct)}$$

		Predictions		
		Positive	Negative	Total
Truth	Positive	52	18	70
	Negative	21	9	30
	Total	73	27	100

$$\text{Recall} = \frac{52}{70} \approx 74\%$$

Specificity / True Negative Rate

- The specificity is given by

$$\frac{\text{number of true negatives}}{\text{number of negative cases}} = \frac{TN}{N}.$$

- Puts emphasis on negative cases.
- It is the sample version of

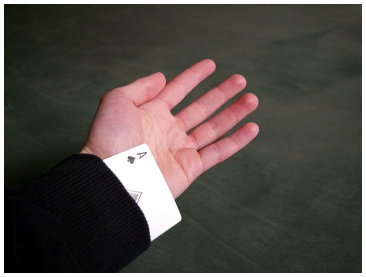
$$\Pr(\hat{Y} = 0 \mid Y = 0, \mathbf{X})$$ `(given it is ⊖, how likely prediction is correct)`

		Predictions		
		Positive	Negative	Total
Truth	Positive	52	18	70
	Negative	21	9	30
	Total	73	27	100

$$\text{Specificity} = \frac{9}{30} = 30\%$$

Cheating with recall and specificity

`Disclaimer: I am not responsible for any damage or harm caused by the use of the following classifiers ;-)`



To get a perfect recall, use the classifier $\mathbf{X} \mapsto \hat{Y} = 1$ to get

		Predictions		
		Positive	Negative	Total
Truth	Positive	P	0	P
	Negative	N	0	N
	Total	n	0	n

$$\text{Recall} = \frac{\text{P}}{\text{P}} = 100\%$$

Similarly to get a perfect specificity, use the classifier $\mathbf{X} \mapsto \hat{Y} = 0$ to get

		Predictions		
		Positive	Negative	Total
Truth	Positive	0	P	P
	Negative	0	N	N
	Total	0	n	n

$$\text{Specificity} = \frac{\text{N}}{\text{N}} = 100\%$$

☞ We need additional metrics to assess the performance of a classifier.

Precision

- The precision is given by

$$\frac{\text{number of true predicted positive case}}{\text{number of predicted positive cases}} = \frac{TP}{PP}.$$

- Puts emphasis on false alarms, i.e., $\hat{Y} = 1$ while $Y = 0$.
- It is the sample version of

$$\Pr\left(Y = 1 \mid \hat{Y} = 1\right). \qquad \texttt{(given you predict } \oplus\texttt{, how likely it is correct?)}$$

		Predictions		
		Positive	Negative	Total
Truth	Positive	52	18	70
	Negative	21	9	30
	Total	73	27	100

$$\text{Precision} = \frac{52}{73} \approx 71\%$$

Classical interview question

- You go from A to B at speed $50km/h$ and $40km/h$ on your way back.
- What is your average speed?

Classical interview question

- You go from A to B at speed $50km/h$ and $40km/h$ on your way back.
- What is your average speed?

Not hired Well, hmmm, $45km/h$

Classical interview question

- You go from A to B at speed $50km/h$ and $40km/h$ on your way back.
- What is your average speed?

Not hired Well, hmmm, $45km/h$

Still in line Well, hmmm, total distance is $2d$, total time is $d/50 + d/40$. So average speed is $2d/(d/50 + d/40)$

Classical interview question

- You go from A to B at speed $50km/h$ and $40km/h$ on your way back.
- What is your average speed?

Not hired Well, hmmm, $45km/h$

Still in line Well, hmmm, total distance is $2d$, total time is $d/50 + d/40$. So average speed is $2d/(d/50 + d/40)$

Hired Well I can do the math but quickly since we're talking about averaging speeds, therefore rates, it is the harmonic mean.

Classical interview question

- You go from A to B at speed $50km/h$ and $40km/h$ on your way back.
- What is your average speed?

Not hired Well, hmmm, $45km/h$

Still in line Well, hmmm, total distance is $2d$, total time is $d/50 + d/40$. So average speed is $2d/(d/50 + d/40)$

Hired Well I can do the math but quickly since we're talking about averaging speeds, therefore rates, it is the harmonic mean.

Definition 4. The harmonic mean of (positive) real numbers $x_1, \ldots, x_n$ is

$$\bar{x}_{\text{harm}} = \left(\frac{\sum_{i=1}^{n} x_i^{-1}}{n} \right)^{-1}.$$

The harmonic mean is the right one when dealing with rates.

$F1$–score

- $F1$–score is the harmonic mean of precision and recall
- The $F1$–score is given by

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}}$$ `(Performance on positive cases (predicted and true))`

		Predictions		
		Positive	Negative	Total
Truth	Positive	52	18	70
	Negative	21	9	30
	Total	73	27	100

$$F_1 = \frac{2}{73/52 + 70/52} \approx 73\%$$

Prevalence

- The prevalence is given by

$$\frac{\text{Number of positive cases}}{\text{sample size}} = \frac{P}{n}.$$

- It is the sample version of

$$\Pr(Y = 1 \mid \mathbf{X}) \qquad \texttt{(how likely it is to be } \oplus\texttt{)}$$

- It is not a performance metrics but rather a description of the population.

		Predictions		
		Positive	Negative	Total
Truth	Positive	52	18	70
	Negative	21	9	30
	Total	73	27	100

$$\text{Prevalence} = \frac{70}{100} = 70\%$$

To sum up

> ‘‘Our classifier is not very accurate overall (22% better than random guessing). It is pretty good at correctly identifying positive cases but performs poorly for negative cases. However when a positive case is detected, it is likely that it is true. Overall, performance for positive cases is pretty good. However note that the population is unbalanced and we may take into account that feature.’’

- ☐ Accuracy of 61%
- ☐ Recall of 74%
- ☐ Specificity of 30%
- ☐ Precision of 71%
- ☐ $F1$–score of 73%
- ☐ Prevalence of 70%

Soft classifier

- A K–class soft classifier is a mapping

$$\begin{aligned} p\colon \mathcal{X} &\longrightarrow \mathbb{S}_K \\ \mathbf{x} &\longmapsto p(\mathbf{x}) = \{p_1(\mathbf{x}), \ldots, p_K(\mathbf{x})\}^\top, \end{aligned}$$

  where $\mathbb{S}_K = \left\{\mathbf{u} \in (0,1)\colon \sum_{k=1}^K u_k = 1\right\}$, i.e., unit simplex.
- Note that the p_k's are class conditional probabilities estimators, i.e.,

$$p_k(\mathbf{x}) = \widehat{\Pr}\left(Y = k \mid \mathbf{X} = \mathbf{x}\right)$$

- From the above soft classifier we can get class prediction using the Maximum A Posteriori (MAP) estimator, i.e.,

$$\underset{k \in \{1,\ldots,K\}}{\arg\max}\ p_k(\mathbf{x}).$$

Soft classifier: binary case

- The MAP estimator for a binary soft classifier is, given $\mathbf{X} = \mathbf{x}$

$$\hat{Y} = \begin{cases} 1, & \text{if } p_1(\mathbf{x}) > u \\ 0, & \text{otherwise,} \end{cases} \qquad \text{with } u = 0.5.$$

- In general one can pick any cutoff value $u \in (0, 1)$ and we have
  - As u increases, recall decreases while precision increases.
  - As u decreases, recall increases while precision decreases.
- Depending on the situation, it may be a desirable behaviour, e.g.,

  $u \approx 1$ not too many false alarm, e.g., spam as you'll miss some emails.
  $u \approx 0$ not too many false negative, e.g., fraud detection as you don't want to miss any fraud.

ROC Curve (for binary classification only!)

- Receiver Operating Characteristic (ROC) curves assess the impact of the cutoff.
- Plots the sensitivity as 1 - specificity varies.

- It passes through the points

  $(0, 0)$ Always predicts negative, i.e., $\hat{Y} \equiv 0$
  $(1, 1)$ Always predicts positive, i.e., $\hat{Y} \equiv 1$

- The "higher" the curve is, the better is the classifier.

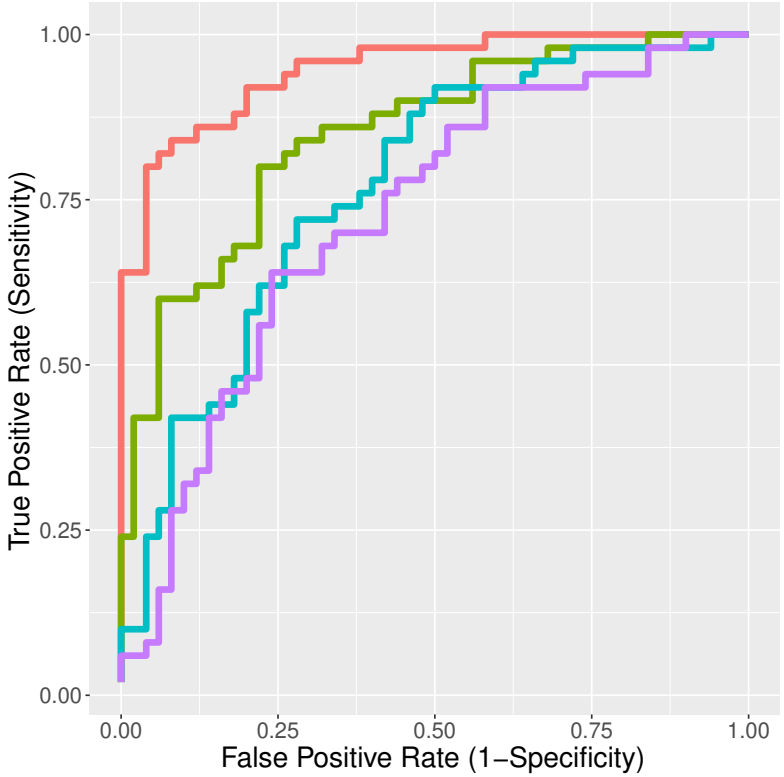


Figure 3: *ROC curves for 4 classifiers.*

Exercise 1. What are the ROC curves for:

- ☐ the "p–coin classifier" C_1, i.e., independently from the covariates value X, $\hat{Y}_p \sim \text{Ber}(p)$?
- ☐ the "we known the truth" classifier C_2, i.e., $\hat{Y} = Y$?

Solution 1.

$$1 - \text{Specificity} = 1 - \Pr(\hat{Y} = 0 \mid Y = 0, X) = 1 - \Pr(\hat{Y} = 0 \mid Y = 0) = p$$

$$\text{Sensitivity} = \Pr(\hat{Y} = 1 \mid Y = 1, X) = \Pr(\hat{Y} = 1 \mid Y = 0) = p$$

$$1 - \text{Specificity} = 1 - \Pr(\hat{Y} = 0 \mid Y = 0, X) = 0$$

$$\text{Sensitivity} = \Pr(\hat{Y} = 1 \mid Y = 1, X) = 1$$

Area Under the Curve (AUC)

- Because of crossings, it is always complicated to compare curves by eyes.
- A widely used choice for summarizing a ROC curve is to compute the Area Under the ROC Curve (AUC).
- From the AUC summary statistics we can easily compare different classifiers:
  - the largest, the better
  - $AUC = 1$ corresponds to the perfect classifier
  - If $AUC < 0.5$, the classifier is doing worse than tossing a coin![3]

[3] If you ever face this situation it is a red flag about your statistical training ;-) More seriously just "reverse the binary classifier" $\tilde{Y} = 1 - \hat{Y}$.

Confusion matrix

If the data set is supervised, we can thus compute the confusion matrix

Table 1: *Confusion matrix for the k–means clustering on the iris dataset.*

	1	2	3
setosa	33	0	17
versicolor	0	46	4
virginica	0	50	0

Confusion matrix

If the data set is supervised, we can thus compute the confusion matrix

Table 1: *Confusion matrix for the k–means clustering on the iris dataset.*

	1	2	3
setosa	33	0	17
versicolor	0	46	4
virginica	0	50	0

Table 2: *Confusion matrix on the same data set—with label switching.*

	1	2	3
setosa	33	17	0
versicolor	0	4	46
virginica	0	0	50

Confusion matrix

If the data set is supervised, we can thus compute the confusion matrix

Table 1: *Confusion matrix for the k–means clustering on the iris dataset.*

	1	2	3
setosa	33	0	17
versicolor	0	46	4
virginica	0	50	0

Table 2: *Confusion matrix on the same data set—with label switching.*

	1	2	3
setosa	33	17	0
versicolor	0	4	46
virginica	0	0	50

☞ Clustering is not able to distinguish the `versicolor` and `virginica` species.

Table 3: *Confusion matrix on the same data set—with label switching.*

		Predictions			
		1	2	3	Total
Truth	setosa	50	0	0	50
	versicolor	0	48	2	50
	virginica	0	14	36	50
	Total	50	62	38	150

$$\text{Accuracy} = \frac{134}{150} \approx 0.89$$

$$\text{Recall}_{\text{setosa}} = \frac{50}{50} = 1 \qquad \text{Recall}_{\text{versicolor}} = \frac{48}{50} = 0.96 \qquad \text{Recall}_{\text{virginica}} = \frac{36}{50} = 0.72$$

$$\text{Spec.}_{\text{setosa}} = \frac{50}{100} = 1 \qquad \text{Spec.}_{\text{versicolor}} = \frac{50+36}{50} = 0.86 \qquad \text{Spec.}_{\text{virginica}} = \frac{50+48}{100} = 0.98$$

$$\text{Prec.}_{\text{setosa}} = \frac{50}{50} = 1 \qquad \text{Prec.}_{\text{versicolor}} = \frac{48}{62} \approx 0.77 \qquad \text{Prec.}_{\text{virginica}} = \frac{36}{38} \approx 0.95$$

$$\text{Accuracy} = \frac{134}{150} \approx 0.89$$

$$\text{Recall}_{\text{setosa}} = \frac{50}{50} = 1 \qquad \text{Recall}_{\text{versicolor}} = \frac{48}{50} = 0.96 \qquad \text{Recall}_{\text{virginica}} = \frac{36}{50} = 0.72$$

$$\text{Spec.}_{\text{setosa}} = \frac{50}{100} = 1 \qquad \text{Spec.}_{\text{versicolor}} = \frac{50+36}{50} = 0.86 \qquad \text{Spec.}_{\text{virginica}} = \frac{50+48}{100} = 0.98$$

$$\text{Prec.}_{\text{setosa}} = \frac{50}{50} = 1 \qquad \text{Prec.}_{\text{versicolor}} = \frac{48}{62} \approx 0.77 \qquad \text{Prec.}_{\text{virginica}} = \frac{36}{38} \approx 0.95$$

"Overall our classifier is (very) good. If a flower is either setosa or virinica, we are able to identify them correctly and we are even more accurate in predicting which species it is not. Note that we get slightly worse performance for versicolor. Overall we can have high confidence in the predictions with slightly worse performance for versicolor."

ROC curve and AUC

- The k–means is not a soft classifier and, as so, we cannot compute ROC curve and AUC.
- Actually it is not quite true. . .
- K–means is a specific case of Gaussian mixture models
- Gaussian mixture models gives class conditional probabilities estimates, i.e.,

$$\Pr(\hat{Y} = k \mid \mathbf{X} = \mathbf{x}), \qquad k = 1, \ldots, K.$$

- Using this framework one can thus get ROC and AUC

Regression

- Evaluating model's performance for regression problem is rather easy.
- Shortly you simply estimate the generalization error estimate for a given loss function ℓ

$$\frac{1}{n}\sum_{i=1}^{n}\ell(Y_i,\hat{Y}_i)$$

- The choice of ℓ is driven by your application and the desirable properties of your predictions.

Loss functions for regression (some)

- Quadratic based losses, a.k.a., ℓ_2–losses,

$$\ell_2(y, y') = \|\varphi(y) - \varphi(y')\|_2, \qquad \text{for some mapping } \varphi.$$

- Absolute based losses, a.k.a., ℓ_1–losses,

$$\ell_1(y, y') = \|\varphi(y) - \varphi(y')\|_1, \qquad \text{for some mapping } \varphi.$$

- Absolute–Quadratic losses

$$\ell(y, y') = \begin{cases} c_1 \ell_1(y, y'), & \|y - y'\|_1 < \delta \\ c_2 \ell_2(y, y'), & \text{otherwise,} \end{cases} \qquad c_1, c_2 \text{ normalizing constants.}$$

- Quantile losses

$$\ell(y, y') = \begin{cases} \gamma |y - y'|, & y < y' \\ (1 - \gamma)|y - y'|, & y \geq y', \end{cases} \qquad 0 \leq \gamma \leq 1.$$

Quadratic losses

$$\ell(y, y') = \|\varphi(y) - \varphi(y')\|_1$$

- Two common choices for φ are
  - $\varphi : y \mapsto y$ gives the mean squared error
  - $\varphi : y \mapsto \log(1+y)$ gives the mean squared log error and is valid for positive outcomes.
- Aim at accurate predictions overall
- Loss can be driven by a single large error—the $\log$ mitigates a bit.
- Kind of risk adversarial losses

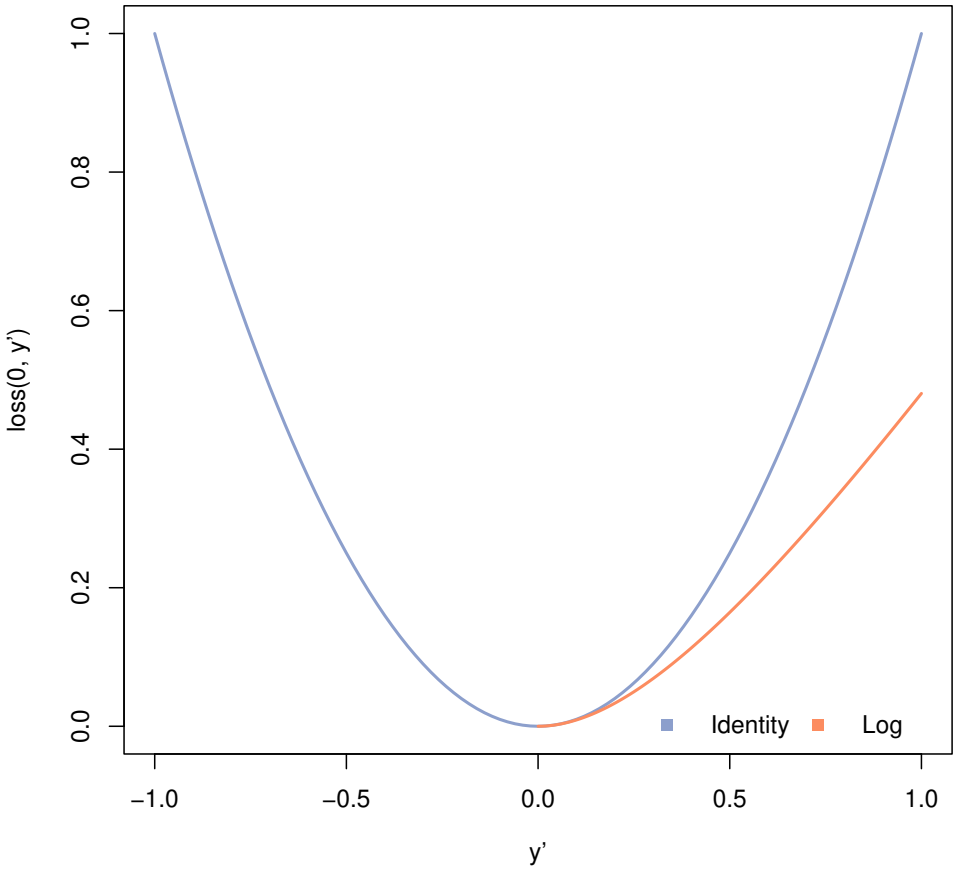


Figure 4: *Evolution of the loss as the prediction y' deviates from the true value $y = 0$.*

Absolute losses

$$\ell(y, y') = \|\varphi(y) - \varphi(y')\|_1$$

- Two common choices for φ are
  - $\varphi : y \mapsto y$ gives the mean absolute error
  - $\varphi : y \mapsto \log(1+y)$ gives the mean absolute log error and is valid for positive outcomes.
- More robust to a single large error than quadratic losses
- If no outliers, not as accurate as quadratic loss.

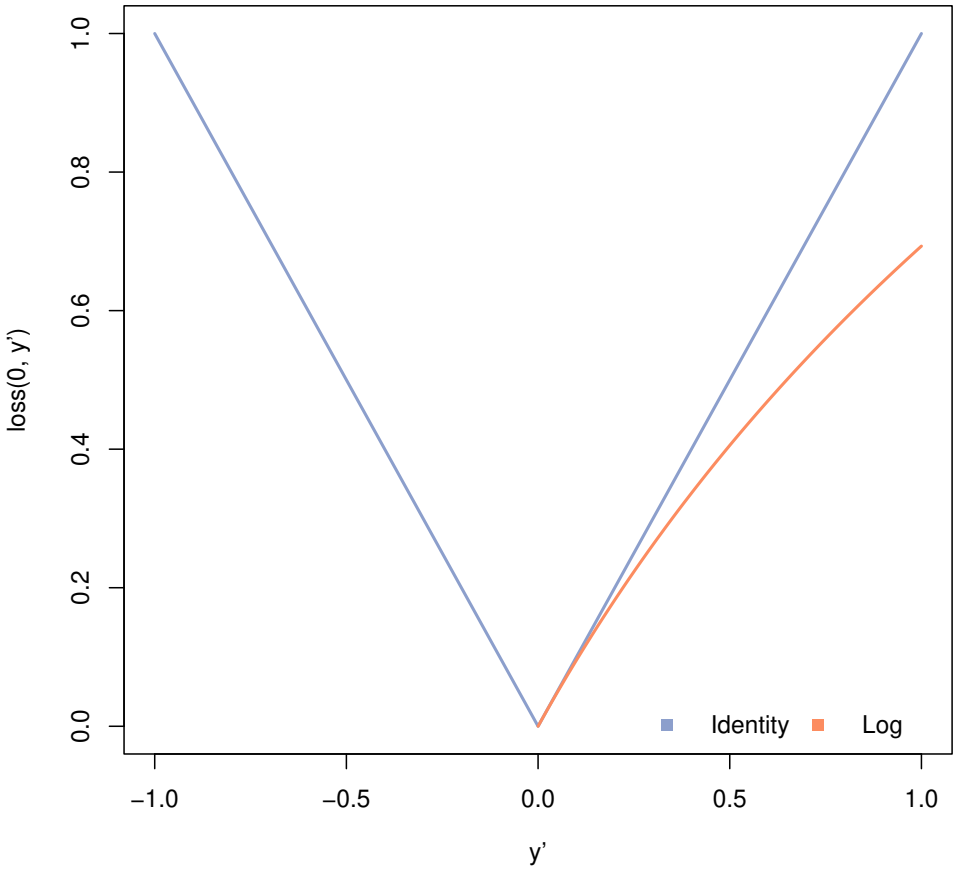


Figure 5: *Evolution of the loss as the prediction y' deviates from the true value $y = 0$.*

Huber loss

$$\ell(y, y') = \begin{cases} c_1 \|y - y'\|_2^2, & \|y - y'\|_1 < \delta \\ c_2 \|y - y'\|_1, & \text{otherwise} \end{cases}$$

- Huber loss: $c_1 = 1$ and $c_2 = \delta$
- When δ is

  large similar to quadratic loss
  small similar to absolute loss

- It is a tradeoff between absolute and quadratic losses
- The hyperparameter δ should be specified.

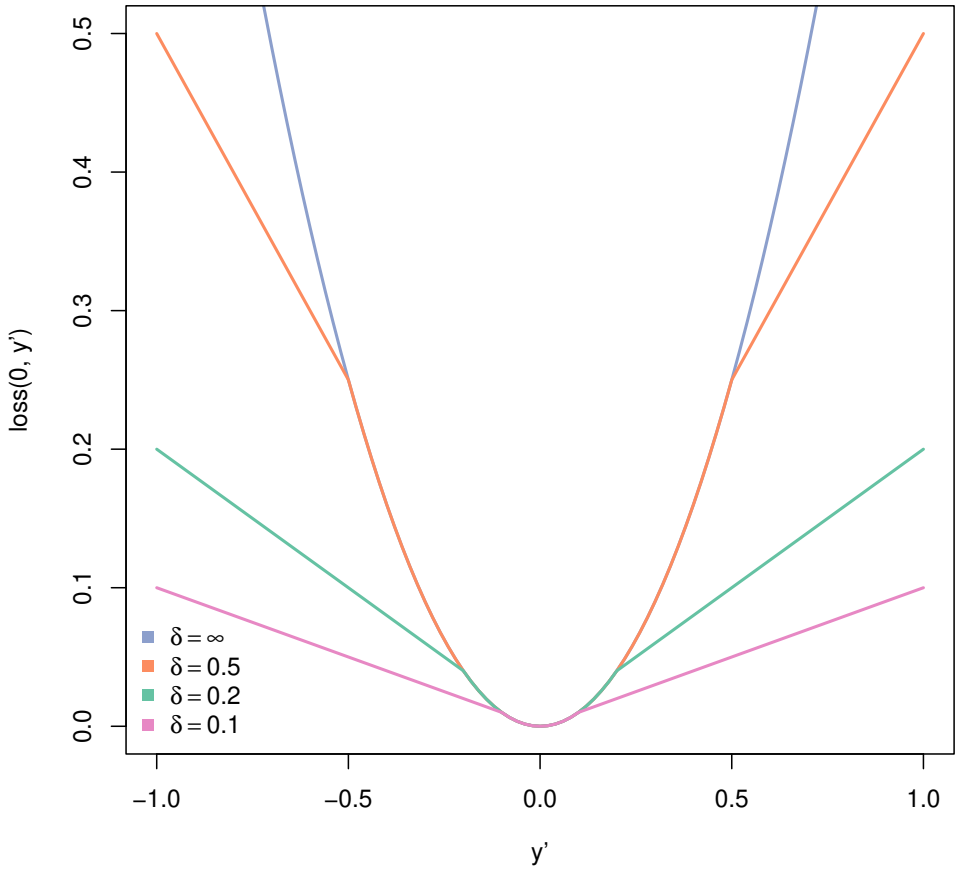


Figure 6: *Plot of Huber losses as δ varies. When $\delta \to \infty$, Huber loss converges to the quadratic loss.*

Quantile losses

$$\ell(y, y') = \begin{cases} \gamma|y - y'|, & y < y' \\ (1-\gamma)|y - y'|, & y \geq y' \end{cases}$$

- Hyperparameter $\gamma \in [0, 1]$
- It controls how to penalize under/over estimation:

  $\gamma = 0.5$ no difference
  $\gamma < 0.5$ penalize underestimations
  $\gamma > 0.5$ penalize overestimations

- γ should be specified

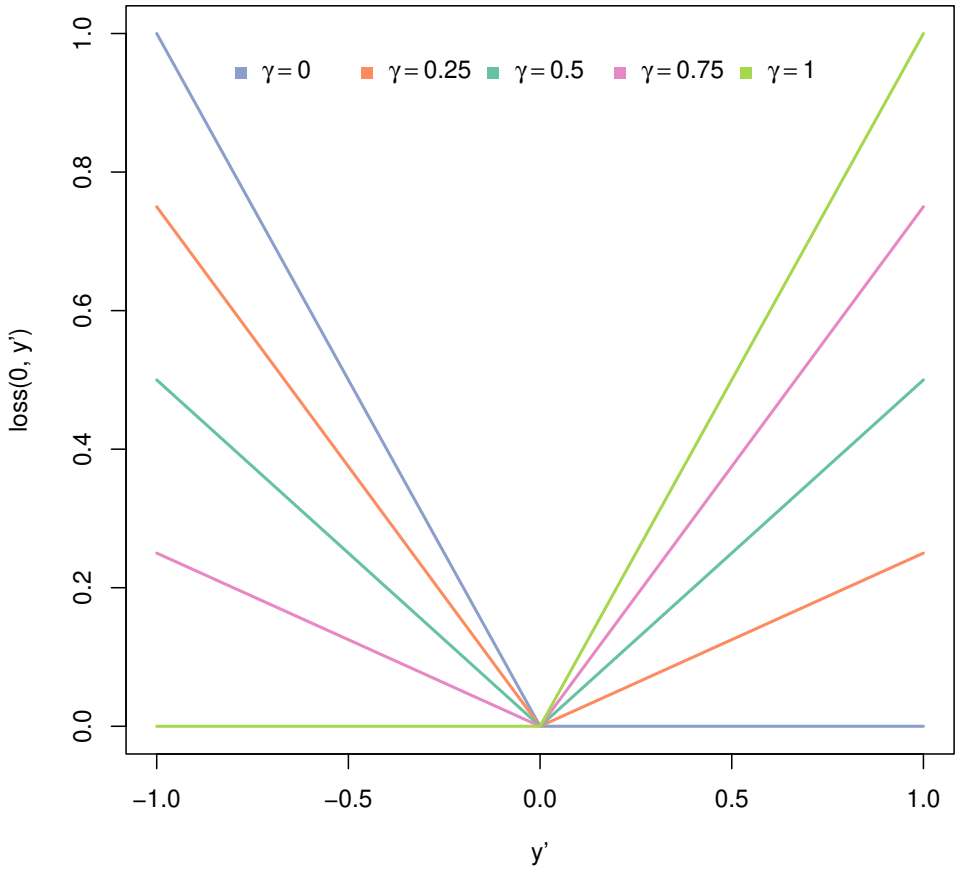


Figure 7: *Plot of quantile losses as γ varies. When $\gamma = 1, 0$ it behaves like a ReLu or $1-$Relu respectively and as the absolute loss when $\gamma = 0.5$.*

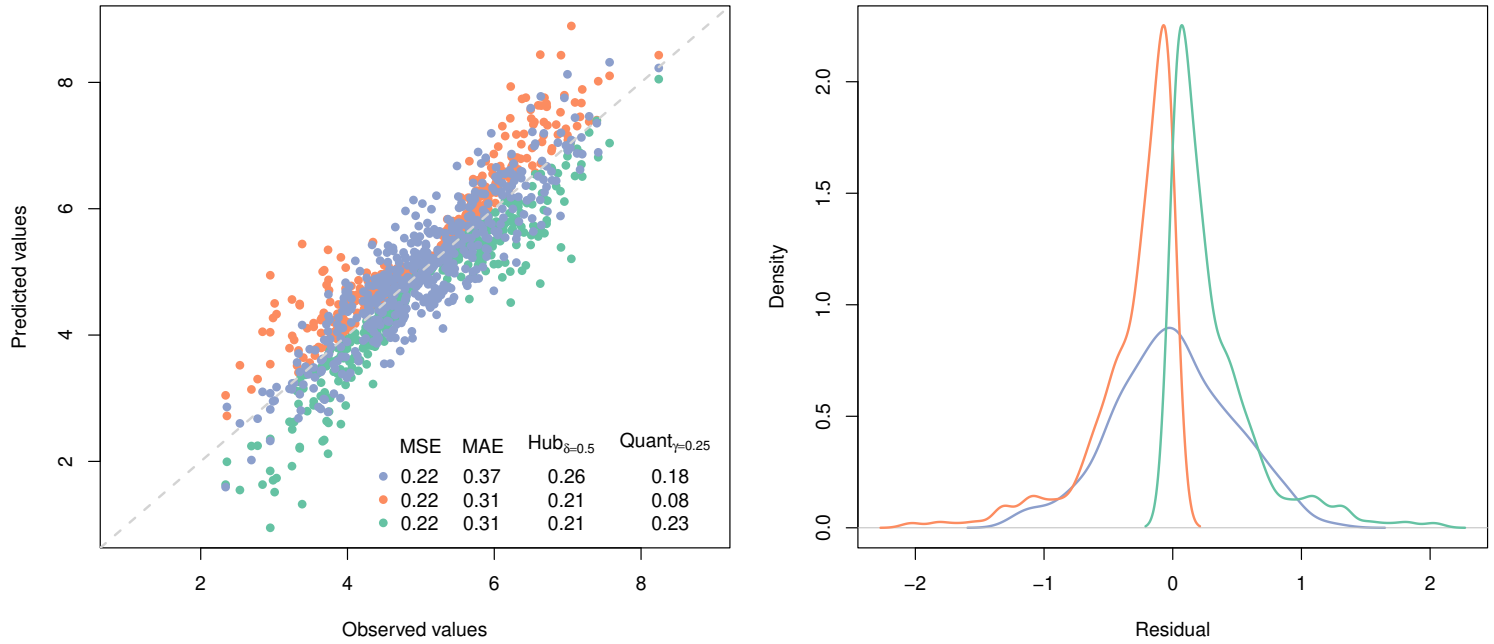


Figure 8: *Left: Comparisons between observations and predicted values for 3 differents estimators with respective losses. Right: Kernel density estimation of the residuals.*

- Same MSE for all estimators but absolute error larger for blue model. This model is overall less accurate but without (a few) large errors.
- Huber loss confirms this statement.
- Orange model has the smallest quantile error ($\gamma = 0.25$) has it (almost) never underestimates.

I. Trees

You'll learn what's behind...

```
> library(tidymodels)
> rf_model <-
   rand_forest(mtry = 8, min_n = 7, trees = 1000) %>%
   set_engine("ranger", num.threads = cores, importance = "impurity") %>%
   set_mode("regression")
> fit <- rf_model %>% fit(log10(Sale_Price) ~ ., data = train)
```

```
from sklearn.ensemble import RandomForestRegressor
my_rf = RandomForestRegressor(n_estimators=10, max_features=4)
fit = my_rf.fit(X_train, y_train)
```

Quick overview

- Random forests are a fairly recent learning strategy (00's)
- It is based on classification and regression trees or CART for short.
- It is a modification of bagging to mitigate dependence between trees.

> Bagging and random forests heavily rely on bootstrap and, as so, both are ensemble methods.

Why averaging?

Proposition 1. *Let $T_1, \ldots, T_B$ be independent copies of T with $Var(T) = \sigma^2$. We have*

$$Var\left(\bar{T}_B\right) = \frac{\sigma^2}{B}, \qquad \bar{T}_B = \frac{1}{B}\sum_{b=1}^{B} T_b.$$

Proposition 2. *Let $T_1, \ldots, T_B$ be dependent copies of T with $Var(T) = \sigma^2$ and pairwise correlation $\rho > 0$. We have*

$$Var\left(\bar{T}_B\right) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2, \qquad \bar{T} = \frac{1}{B}\sum_{b=1}^{B} T_b.$$

Since

$$\text{Var}\left(\bar{T}_B\right) \longrightarrow \rho\sigma^2, \qquad B \to \infty,$$

the pairwise correlation ρ mainly controls the variance of $\bar{T}_B$ as long as B is large enough. Random forests aims at reducing ρ without increasing (too much) σ^2.

1. CART

What is a binary tree?

Definition 5. A tree is a collection of connected nodes. It is often used to display a hierarchical structure in a graphical way.
Nodes without any children are called leaves or terminal nodes.

Definition 6. A binary tree is a tree whose nodes have at most two children.

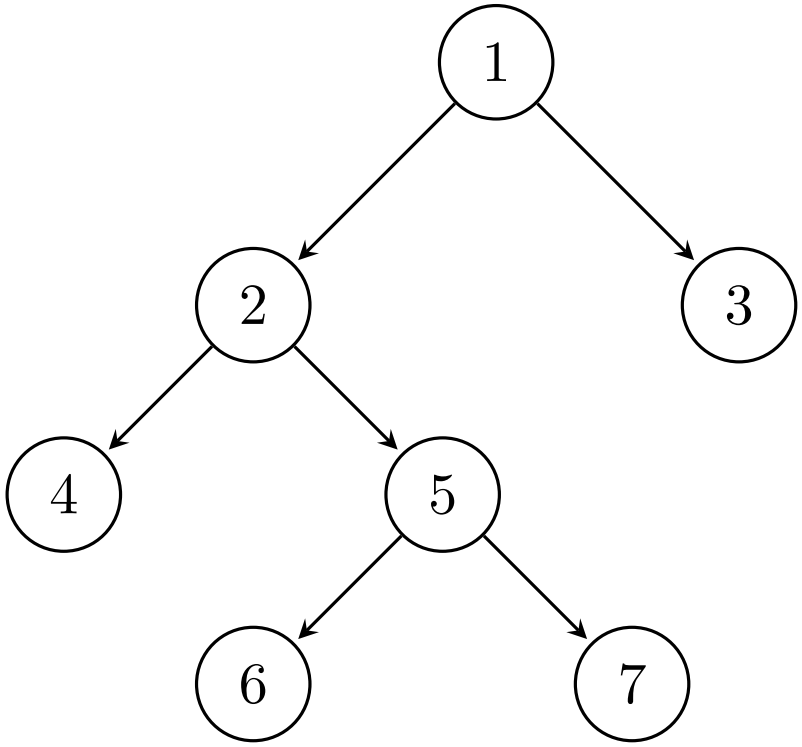


Figure 9: *A (rooted) binary tree.*

Classification And Regression Trees (CART)

- A tree may define an estimator where each leaf is an estimator.
- Can be used both for both regression and classification hence the phrasing

Classification And Regression Trees (CART)

The CART estimator is

$$\hat{Y} = T(\mathbf{X}) = \sum_{\ell=1}^{\#\text{ leaves}} c_\ell 1_{\{\mathbf{X} \text{ ends in the } \ell\text{-th leaf}\}}, \qquad \mathbf{X} \in \mathcal{X},$$

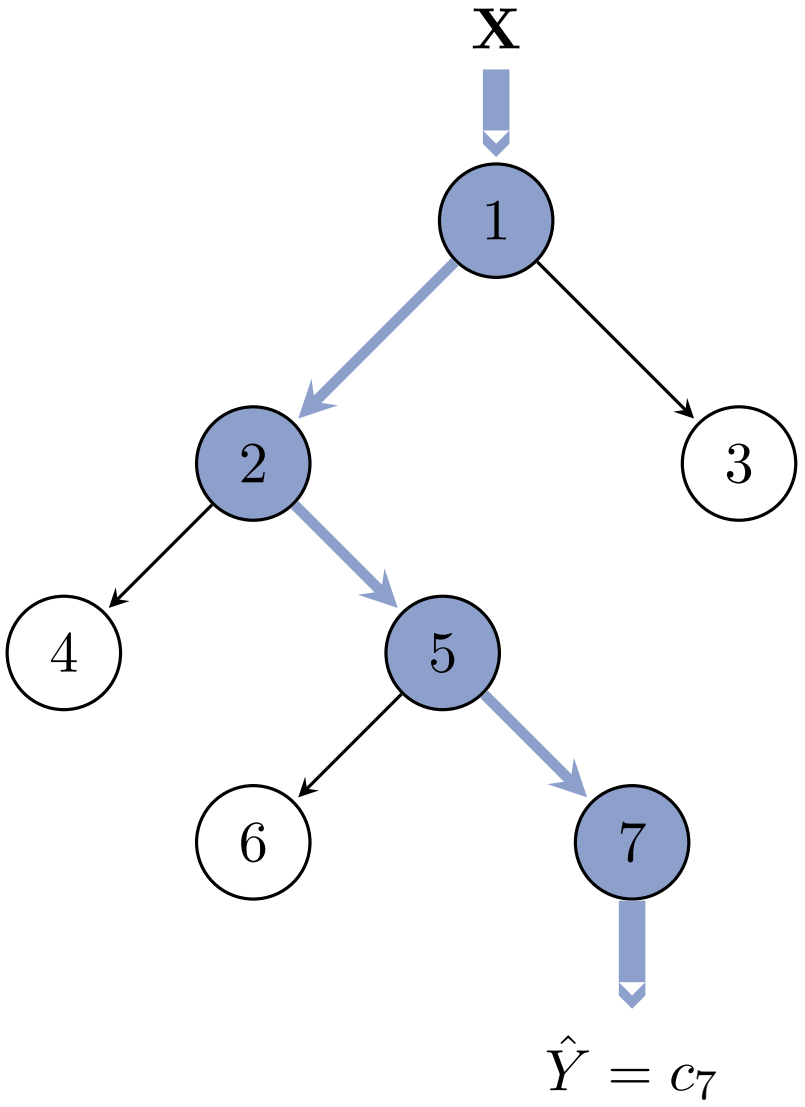


Figure 10: *Prediction of an invidual* $\mathbf{X}$ *from a CART.*

Region R_ℓ

- For any new individuals $\mathbf{X}_* \in \mathcal{X}$, $\mathbf{X}_*$ should end in a leaf, i.e., get a prediction.
- Each leaf thus corresponds to a subset $R_\ell \subset \mathcal{X}$ such that

$$\cup_{\ell=1}^{\#\text{ leaves}} R_\ell = \mathcal{X} \qquad \texttt{(covering of } \mathcal{X}\texttt{)}$$

- To get a unique prediction, we further impose

$$R_{\ell_1} \cap R_{\ell_2} = \emptyset, \quad \ell_1 \neq \ell_2. \qquad \texttt{(partition of } \mathcal{X}\texttt{)}$$

Due to successive binary splits, not all partitions of $\mathcal{X}$ are valid!

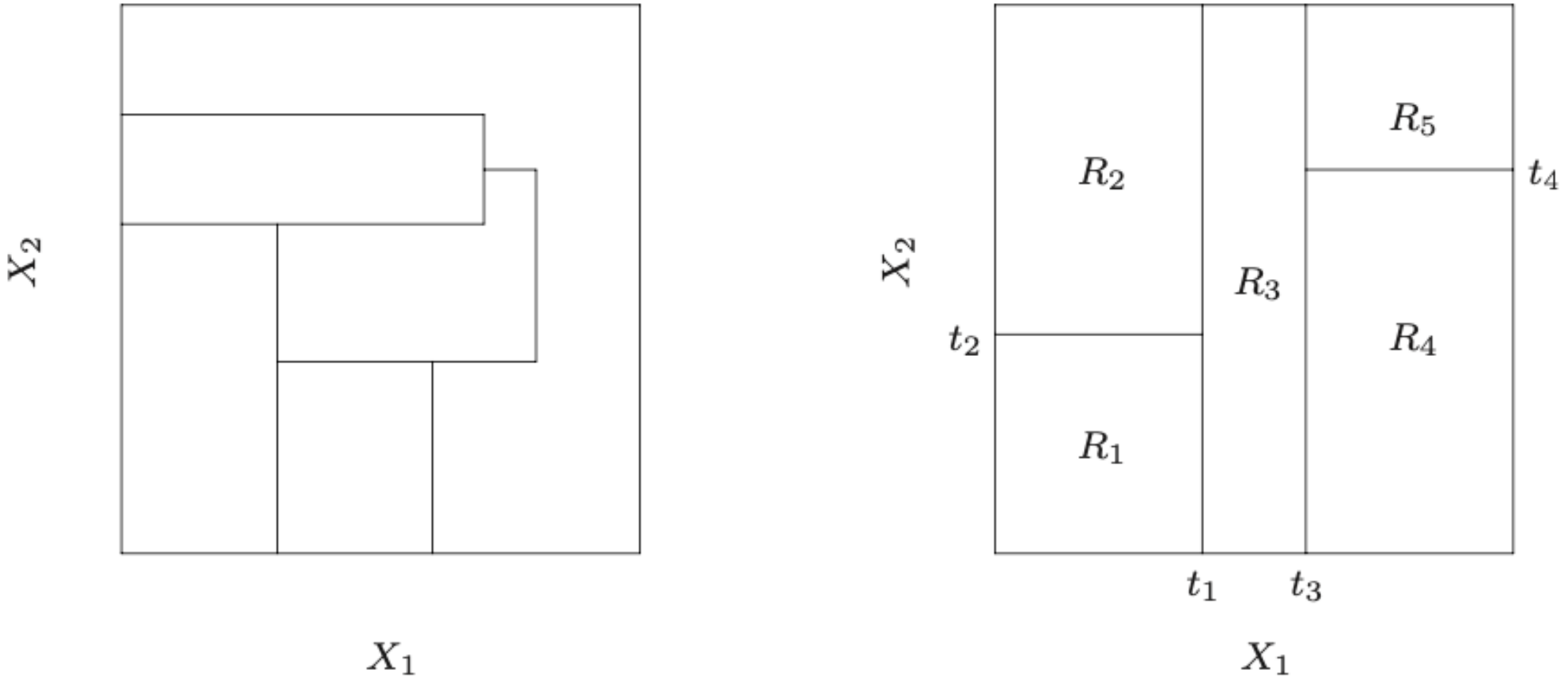


Figure 11: *Two partitions of $\mathcal{X}$. Only one is admissible! Taken from Elements of Statistical Learning (Second edition).*

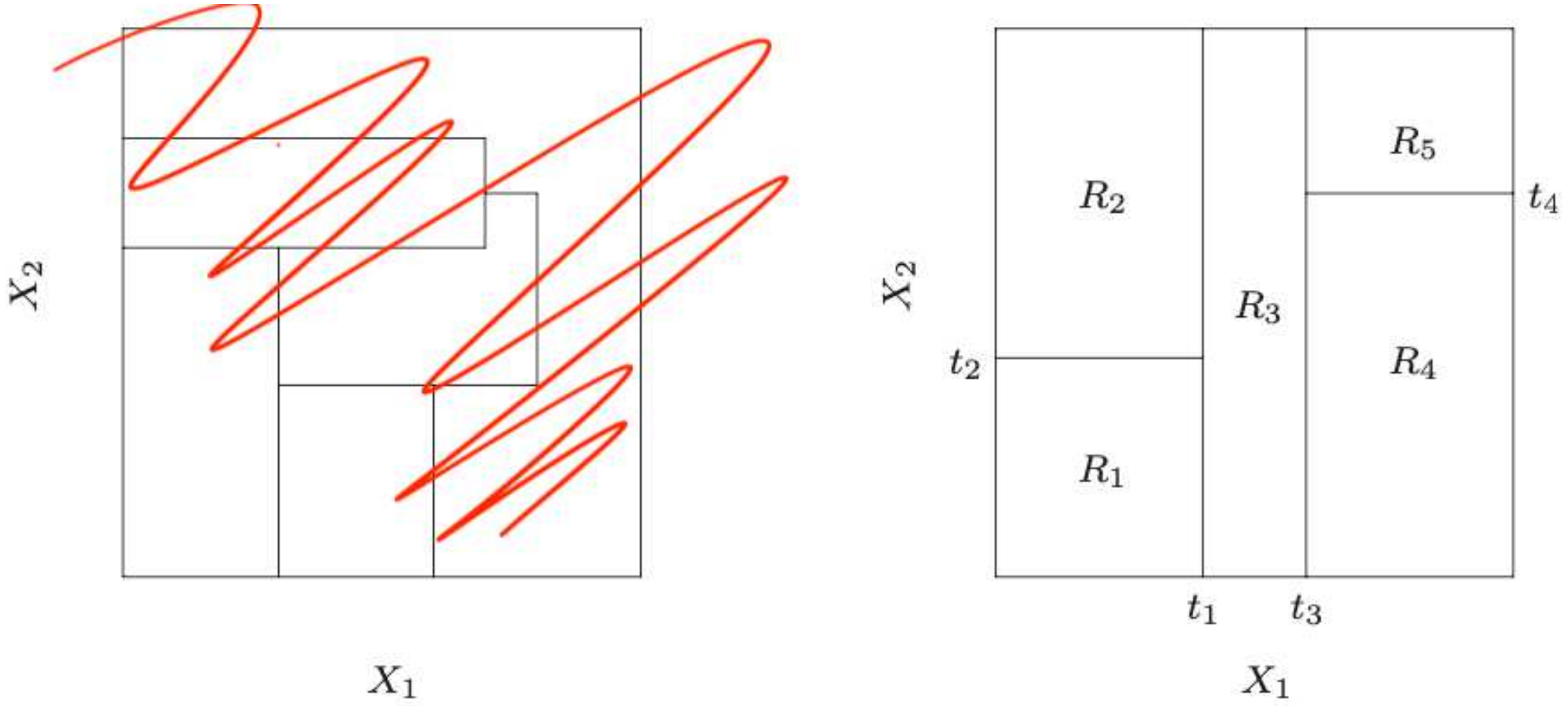


Figure 11: *Two partitions of $\mathcal{X}$. Only one is admissible! Taken from Elements of Statistical Learning (Second edition).*

Growing a CART (splitting nodes)

- Suppose we have p features, i.e., $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_p$.
- Splitting a node into child nodes N_ℓ, N_r is done from a binary decision rule

$$\text{individual } \mathbf{X} \text{ moves to: } \begin{cases} N_\ell, & \text{if } X \in R \\ N_r, & \text{otherwise,} \end{cases}$$

  for some splitting set $R \subset \mathcal{X}$.
- For CART, the splitting set is defined using a single variable, i.e.,

$$R = \{\mathbf{x} \in \mathcal{X} : x_j \in S_j \subset \mathcal{X}_j\}, \qquad \text{for some } j \in \{1, \ldots, p\}.$$

Growing a CART consists in iteratively finding the optimal split.

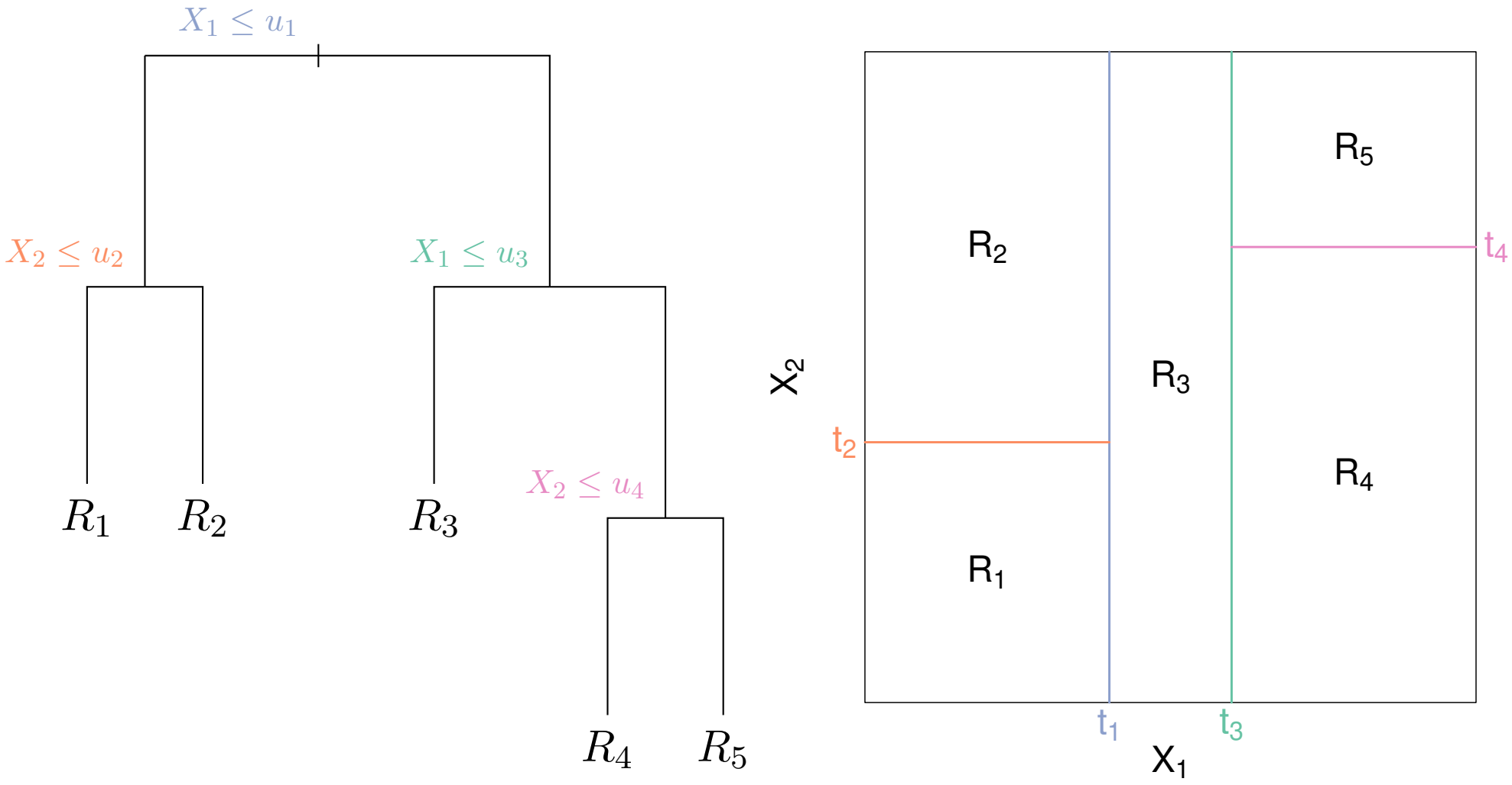


Figure 12: *A CART with $\mathcal{X} \subset \mathbb{R}^2$ (left) and the corresponding partition of $\mathcal{X}$.*

Optimal split

- Optimal split of a node consists amounts to get
  - the best variable X_j from which split relies on;
  - the best splitting set S_j.
- Optimal split is the one that minimizes a given loss function ℓ, i.e.,

$$\underbrace{\underset{j=1,\ldots,p}{\operatorname{argmin}} \left[\underbrace{\underset{S_j \in \mathcal{X}_j}{\operatorname{argmin}} \left\{ \underbrace{\min_{c_L} \sum_{i=1}^{n} \ell(Y_i, c_L) 1_{\{X_{i,j} \in S_j\}}}_{\text{minimize loss in the left child}} + \underbrace{\min_{c_R} \sum_{i=1}^{n} \ell(Y_i, c_R) 1_{\{X_{i,j} \notin S_j\}}}_{\text{minimize loss in the right child}} \right\}}_{\text{best splitting set } S_j \text{ for feature } X_j} \right]}_{\text{best splitting variable } X_j}$$

Splitting set

- The splitting set definition depends on the type of splitting variable X_j
- If X_j is quantitative the binary decision rule is

$$\mathbf{X} \text{ moves to: } \begin{cases} N_\ell, & \text{if } \mathbf{X} \in R(u) \\ N_r, & \text{otherwise,} \end{cases} \qquad R(u) = \{\mathbf{x} \in \mathcal{X} : x_j \leq u\},$$

  for some suitable cutoff value $u \in \mathbb{R}$.
- If X_j is qualitative with K levels, the binary decision rule is

$$\mathbf{X} \text{ moves to: } \begin{cases} N_L, & \text{if } X_j \in R \\ N_R, & \text{otherwise,} \end{cases}, \qquad R \subseteq \{1, \ldots, K\} \setminus \{E \cup \emptyset\}$$

As an aside

For any quantitative variable X_j, $j = 1, \ldots, p$, and cutoff value $u \in \mathbb{R}$, the optimal values for c_L and c_R are

$$\hat{c}_L = \underbrace{\frac{1}{|R_N(u)|} \sum_{i=1}^{n} Y_i 1_{\{\mathbf{X}_i \in R_N(u)\}}}_{\text{mean response in the left child}},$$

$$\hat{c}_R = \underbrace{\frac{1}{n - |R_N(u)|} \sum_{i=1}^{n} Y_i 1_{\{\mathbf{X}_i \notin R_N(u)\}}}_{\text{mean response in the right child}},$$

where

$$|R_N(u)| = \sum_{i=1}^{n} 1_{\{\mathbf{X}_i \in R_N(u)\}} = \#\text{ observations in the left child of } N$$

Split with a qualitative variable

Suppose X_j is categorical with levels $E = \{1, \ldots, K\}$. The binary decision rule is now

$$\mathbf{X} \text{ moves to: } \begin{cases} N_L, & \text{if } \mathbf{X} \in R_N \\ N_R, & \text{otherwise,} \end{cases}, \qquad R_N \subseteq \{1, \ldots, K\} \setminus \{E \cup \emptyset\}$$

There are

$$\frac{\overbrace{2^K}^{\#\text{ partitions}} - \overbrace{2}^{\text{omit } \emptyset \text{ and } E}}{\underbrace{2}_{\substack{\text{symmetry} \\ N_L \leftrightarrows N_R}}} = 2^{K-1} - 1$$

ways to partition E with 2 non overlapping sets. It induces a computational burden. However optimal splitting region R_N can be found from only $K-1$ evaluations (not trivial at all!)

Classification trees

- For classification problems, i.e., $Y \in \{1, \ldots, K\}$, we cannot use the quadratic loss anymore

$$\mathbb{E}_{(Y,\mathbf{X})}\left[(Y - \hat{Y})^2\right].$$

- We must use a measure of non homogeneity, i.e., node impurity,

Classification error

$$1 - \Pr_{(Y,\mathbf{X})}(\hat{Y} = Y)$$

Gini's index

$$\mathbb{E}_{(Y,\mathbf{X})}\left[1 - \Pr(\hat{Y} = Y)\right]$$

Cross–entropy

$$-\mathbb{E}_{(Y,\mathbf{X})}\left[\log \Pr(\hat{Y} = Y)\right]$$

Remember that $\hat{Y}$ obviously is a function of $\mathbf{X}$.

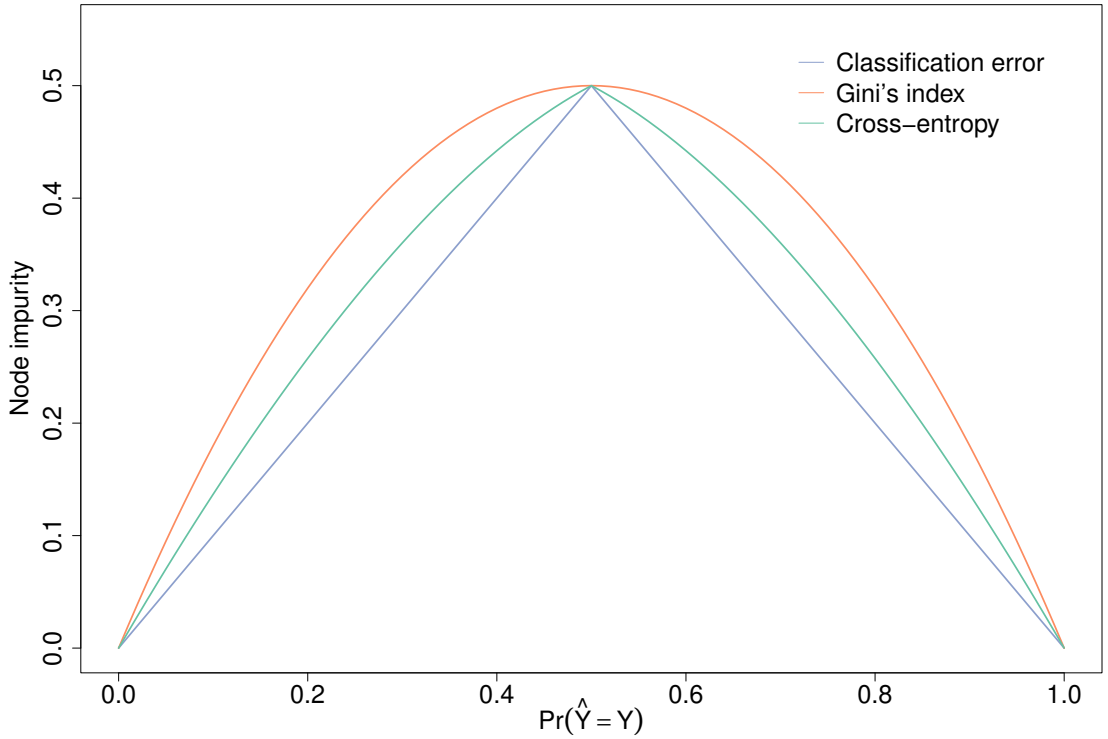


Figure 13: *Different node impurity measures for a binary classification problem. The cross–entropy impurity has been scaled by* $1/\log 2$ *so that it is equal to 0.5 at* $x = 0.5$.

- Similar overall pattern
- Miss-classification is not differentiable
- Gini and cross–entropy are:
  - differentiable: helps optimization
  - favour pure nodes

Gini and cross–entropy should be used to grow and one can use any impurity measure while pruning—miss-classification rate is often use though.

$$\underset{j,s}{\operatorname{argmin}} \left\{ \sum_{i:\, \mathbf{X}_i \in R_1(j,s)} (Y_i - \hat{c}_1)^2 + \sum_{i:\, \mathbf{X}_i \in R_2(j,s)} (Y_i - \hat{c}_2)^2 \right\}$$

- Finding the optimal cutoff value s and feature j are relatively easy.
- For feature X_j, possible cutoff values s are the observed outcome of X_j
- Hence the above optimization problem is solved using a brute–force search.

All possible cutoff values s can be computed once for all at the beginning of the learning stage.

Pruning a tree

- Pruning a tree is about simplifying a tree
- It consists in "collapsing the internal nodes of a tree" based on some criterion.
- The criterion is often cost–complexity pruning

$$\mathcal{P}_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|, \qquad \alpha \geq 0,$$

where T is a binary tree with $|T|$ terminal nodes $R_1, \ldots, R_{|T|}$ and

$$N_m = \sum_{i=1}^{n} 1_{\{\mathbf{X}_i \in R_m\}}, \qquad Q_m(T) = \frac{1}{N_m} \sum_{i=1}^{n} (Y_i - \hat{c}_m)^2 1_{\{\mathbf{X}_i \in R_m\}}.$$

$$\mathcal{P}_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|, \qquad \alpha \geq 0.$$

- Tuning parameter α drives the tradeoff between goodness of fit and complexity:
  - large values of α yields to small trees `(simple models)`
  - small values of α gives large trees. `(complex models)`
- Pruning is done in an iterative way by
  1. collapsing the internal node that gives the smallest quadratic loss increase to get a sub–tree $\tilde{T}$
  2. iterate on the new tree until we get the single–node tree, i.e., new tree is the root.

Pros and cons

- 👍 Almost no data pre-processing, e.g., data scaling
- 👍 Robust to outliers
- 👍 Allows for missing values
- 👍 Intuitive and easy to explain to non specialist
- 👍 Kind of interpretable
- 👎 Highly instable: small change in $\mathbf{X}$ may give a completly different answer
- 👎 CPU demanding
- 👎 Prone to overfitting—pruning mitigates this drawback
- 👎 Non continuous predictor in regression
- 👎 High bias for unbalanced designs—must "re-balanced" it

❗ It is highly recommended to not use CART but rather random forests.

2. Random forest

Towards random forest

$$\text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} T_b\right) \approx \rho\sigma^2, \qquad \text{Cov}(T_b, T_{b'}) = \rho\sigma^2, \qquad B \gg 1,\ \rho > 0.$$

- If we have a low (positive) correlation, the variance is reduced.
- We need to find a way to get almost uncorrelated trees.
- We use the same rationale as bagging, i.e.,

  1. Generate a synthetic data set by boostraping both individuals and covariates;
  2. Fit a CART
  3. Repeat

  where
- Having "different dataset" reduces correlations.

Algorithm 4: Random forest for regression and classification.

input : Supervised data set $\mathcal{D}_n = \{(\mathbf{X}_i, Y_i) \colon i = 1, \ldots, n\}$, number of trees B

1 **for** $b \leftarrow 1$ **to** B **do**

2 Draw a boostrap sample $\mathcal{D}_b$ of size n from $\mathcal{D}_n$;

3 Grow a tree T_b from $\mathcal{D}_b$ using the following steps:

1. Select m variables at random from the p variables
2. Pick the best variable / split–point among the m
3. Split the node into two children nodes

4 Output the ensemble of trees $\{T_b \colon b = 1, \ldots, B\}$ and predictors

Regression (`averaging`)

$$\hat{f}_B \colon \mathbf{x} \longmapsto \frac{1}{B} \sum_{b=1}^{B} T_b(\mathbf{x}),$$

Classification (`majority vote`)

$$\hat{C}_B \colon \mathbf{x} \longmapsto \underset{k}{\operatorname{argmax}} \sum_{b=1}^{B} 1_{\{T_b(\mathbf{x})=k\}}$$

Recommendations, for regression use $m = \lfloor p/3 \rfloor$ with minimum node size is 5; for classification use $m = \lfloor \sqrt{p} \rfloor$ with minimum node size 1. But these are just guidelines and in practice you should consider fine tuning.

Out of bag samples

Algorithm 5: Bootstraping the observations give OOB sample.

input : Supervised data set $\mathcal{D}_n = \{(\mathbf{X}_i, Y_i) : i = 1, \ldots, n\}$, number of trees B

for $b \leftarrow 1$ **to** B **do**
- Draw a boostrap sample $\mathcal{D}_b$ of size n from $\mathcal{D}_n$;
- Grow a tree T_b from $\mathcal{D}_b$ using the following steps:
  1. Select $\tilde{p}$ variables at random from the p variables
  2. Pick the best variable / split–point among the m
  3. Split the node into two children nodes

- By construction, some observations will be discarded while fitting tree T_b.
- These observations are called Out Of Bag (OOB)
- As a consequence we can estimate the generalization error based on OOB samples

$$\frac{1}{n}\sum_{i=1}^{n}\frac{1}{N_i}\sum_{b=1}^{B}\text{loss}\{Y_i, T_b(\mathbf{X}_i)\}1_{\{(\mathbf{X}_i,Y_i)\notin\mathcal{D}_b\}}, \qquad N_i = \sum_{b=1}^{B}1_{\{(\mathbf{X}_i,Y_i)\notin\mathcal{D}_b\}}$$

Algorithm 6: Bootstraping the observations gives OOB sample.

input : Supervised data set $\mathcal{D}_n = \{(\mathbf{X}_i, Y_i) : i = 1, \ldots, n\}$, number of trees B

for $b \leftarrow 1$ **to** B **do**
- Draw a boostrap sample $\mathcal{D}_b$ of size n from $\mathcal{D}_n$;
- Grow a tree T_b from $\mathcal{D}_b$ using the following steps:
  1. Select $\tilde{p}$ variables at random from the p variables
  2. Pick the best variable / split–point among the m
  3. Split the node into two children nodes

ℹ Recall that bagging trees consists in bootsrapping the individuals to get an ensemble of trees. It is actually a specific random forest with $m = p$!

3. Feature importance

Motivation

- A CART is (quite) interpretable since it is based on binary splits
- It is more challenging for random forests since we are "averaging" over multiple binary trees.
- How to know which feature has a large impact on predictions?
- This stage is known as variable importance
- Not all statistical models enable variable importance measures but random forests do!
- Let's see how

Mean decrease impurity

- Binary trees split node N into (N_L, N_R) maximizing the decrease in impurity

$$\Delta i(N) = i(N) - \underbrace{\{p(L)i(N_L) + p(R)i(N_R)\}}_{\text{impurity decrease after splitting } N}, \qquad p(L) = \frac{|N_L|}{|N|},\ p(R) = 1 - p(L),$$

  where $i(T)$ and $|T|$ are the impurity and cardinal of node T.
- Mean Decrease Impurity (MDI) aggregates over nodes of T

$$\text{MDI}_T(X_j) = \sum_{N \in T} p(N) \{i(N) - \Delta i(N)\} 1_{\{\text{split of } N \text{ uses } X_j\}}.$$

- For a random forest $\mathcal{F} = \{T_1, \ldots, T_B\}$, we average over trees, i.e.,

$$\text{MDI}_{\mathcal{F}}(X_j) = \frac{1}{B} \sum_{b=1}^{B} \text{MDI}_{T_b}(X_j).$$

Intuitively, if X_j is not influential, splits based on X_j should not decrease impurity and MDA should be small.

Boston data set

```
> head(Boston)
     crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat medv
1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98 24.0
2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14 21.6
3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03 34.7
4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94 33.4
5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33 36.2
6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21 28.7
```

- Aim is to predict the price of house (regression problem)
- Sample size is $n = 506$
- We have $p = 13$ covariates (only a few categorical)

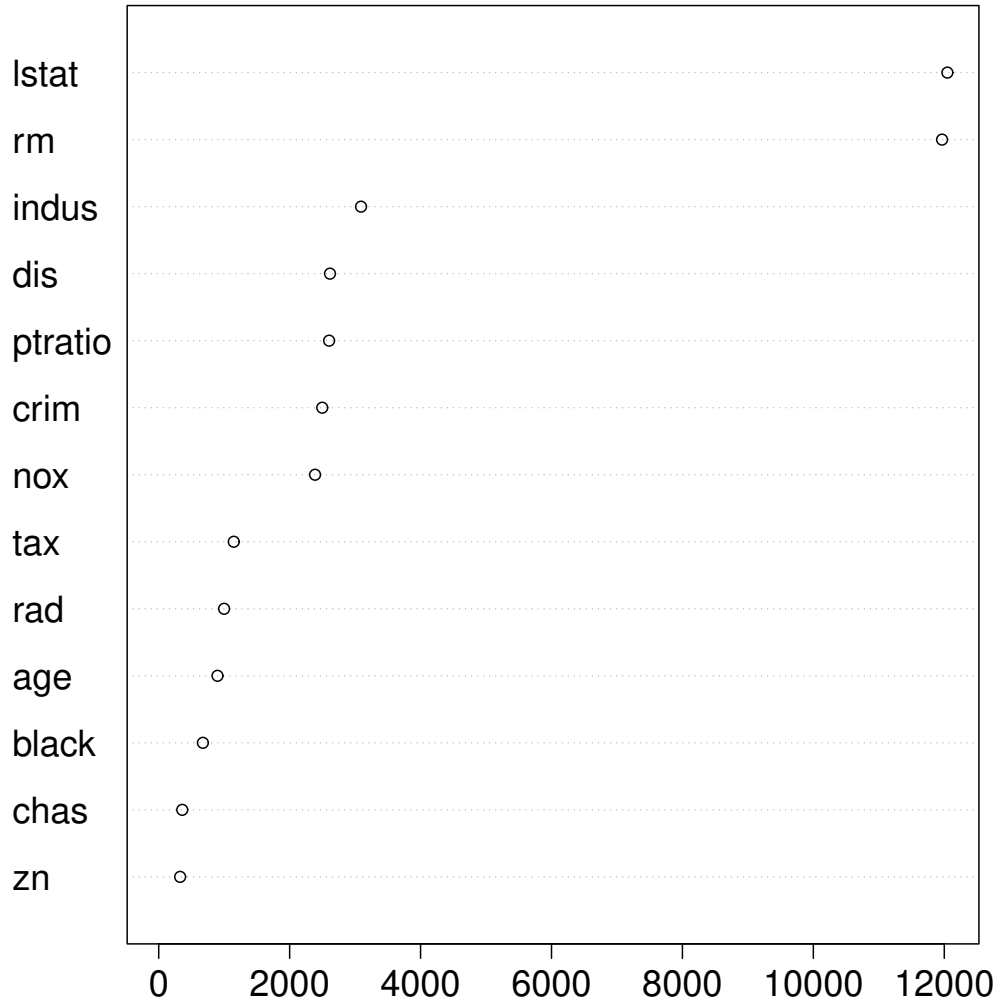


Figure 14: *Mean decrease impurity for the Boston housing regression problem.*

Mean decrease accuracy (a.k.a. permutation importance)

- The Mean Decrease Accuracy (MDA) for a tree (fitted) T is

$$\text{MDA}_T(X_j; \mathcal{D}_n) = \frac{1}{n}\sum_{i=1}^{n} \text{loss}\left\{Y_i, T(\mathcal{D}_{j,n})\right\} - \frac{1}{n}\sum_{i=1}^{n} \text{loss}\left\{Y_i, T(\mathcal{D}_n)\right\},$$

  where $\mathcal{D}_{j,n}$ is similar to the original data $\mathcal{D}_n$ except that feature X_j has been randomly shuffled.
- For a random forest $\mathcal{F} = (T_1, \ldots, T_B)$, we average over all trees, i.e.,

$$\text{MDA}(\mathcal{F}, \mathcal{D}_n) = \frac{1}{B}\sum_{b=1}^{B} \text{MDA}_{T_b}(X_j, \tilde{\mathcal{D}}_{n,b})$$

  where $\tilde{\mathcal{D}}_{n,b}$ is the out-of-bag sample of tree T_b.

> Intuitively, if X_j is not influential, prediction performance where X_j is shuffled should not be degraded too much hence MDA should be small.

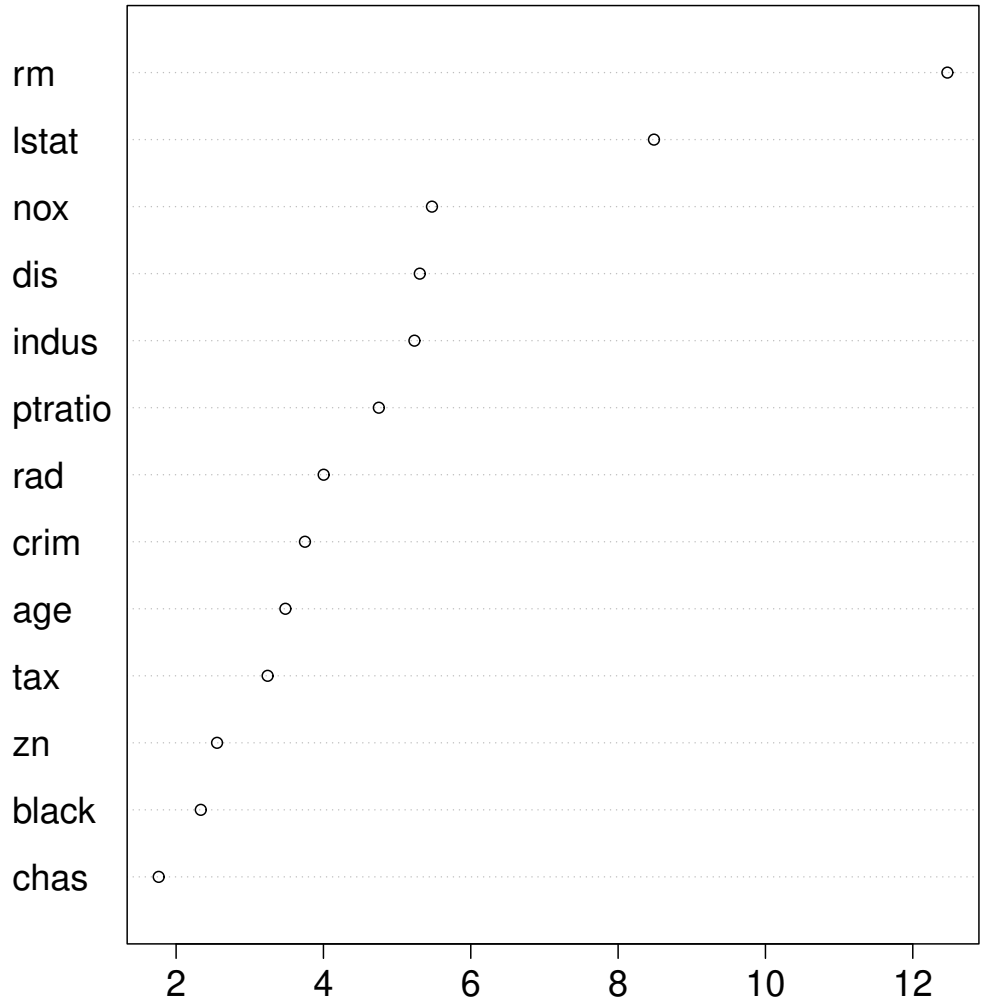


Figure 15: *Mean decrease accuracy for the Boston housing regression problem.*

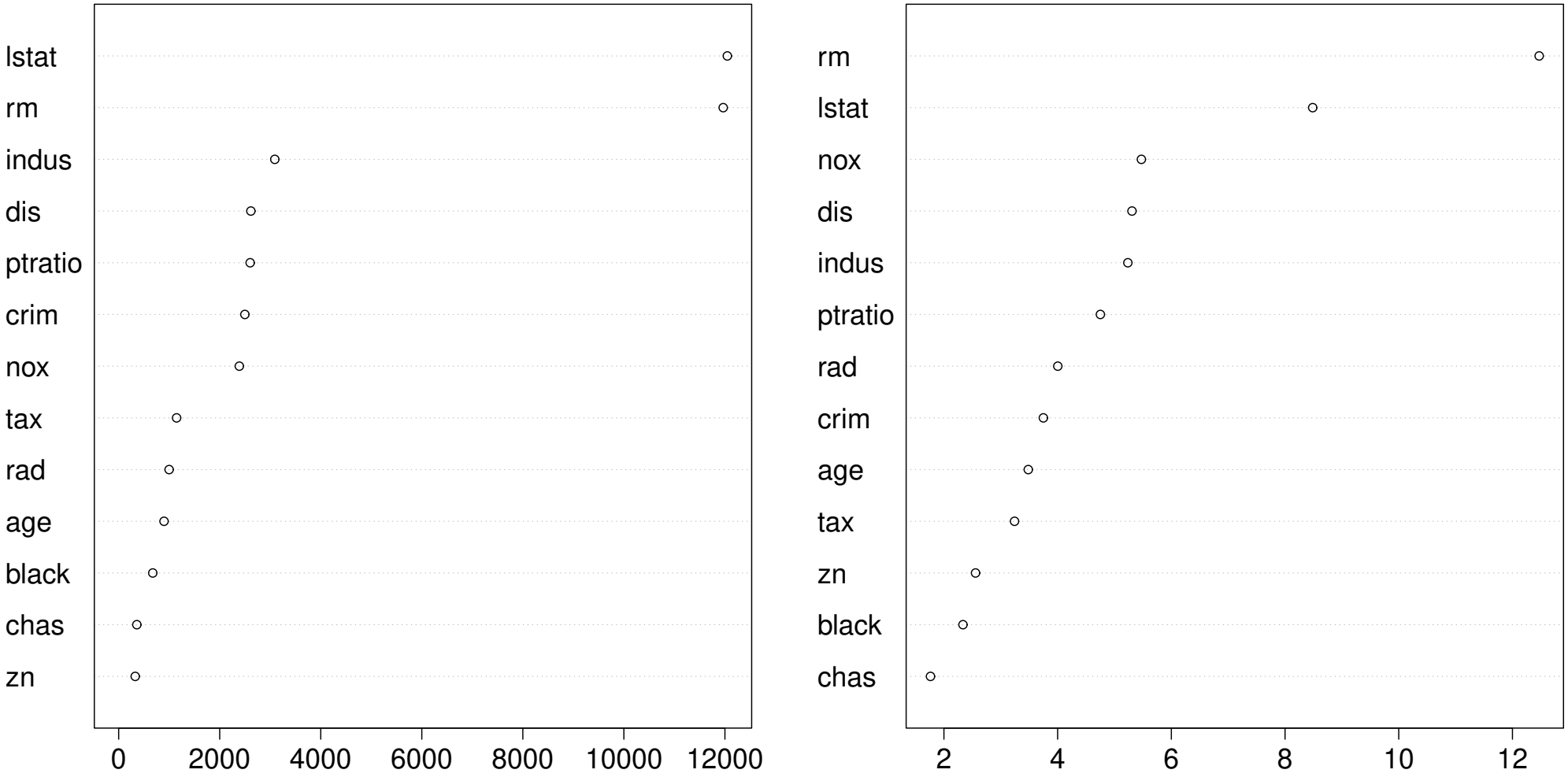


Figure 16: *Comparison of feature importance measures.*

Shapley values

- Shapley values came from game theory and hence are agnostic.
- Within a (coalitional) game with n players, they give the "fair" distribution of the (maximal) profit
- For our concern, we have

  Game predict Y given $\mathbf{X}$;
  Players features;
  Profit model's prediction for an observation $(Y_i, \mathbf{X}_i)$.

- Shapley value of the i–th observation and j–th feature (p features in total) is

$$\text{Shapley}(X_{i,j}) = \frac{1}{p} \sum_{S \subseteq \{1,\ldots,p\} \setminus \{j\}} \underbrace{\binom{n-1}{|S|}^{-1}}_{\substack{\text{number of partitions} \\ \text{of size } |S| \text{ without } j}} \underbrace{\{\nu(X_{i,S} \cup X_{i,j}) - \nu(X_{i,S})\}}_{\substack{\text{marginal contribution of} \\ X_{i,j} \text{ to } X_{i,S} \cup X_{i,j}}}$$

Intuitively, if $X_{i,j}$ is not influential, as before prediction performance should not be degraded too much hence Shapley values should be small.

Shapley value estimation

- Recall that Shapley values uses terms of the form $\nu(X_S)$, $S \subseteq \{1, \ldots, p\}$.
- For our concern, it implies to fit the model to all subset S, i.e., $2^p - 1$ models.
- To reduce the computational burden, one can use an estimation based on a marginalization approach to get $\nu(X_{S_1})$ from $\nu(X_{S_1} \cup X_{S_2})$, $S_1 \cap S_2 = \emptyset$, i.e.,

$$\hat{\nu}(X_{i,S_1}) = \frac{1}{n} \sum_{\ell=1}^{n} \nu(X_{i,S_1} \cup X_{\ell,S_2})$$

The above estimator is called Shapley sampling values.

Shapley additive explanation values

- Shapley Addidtive exPlanation (SHAP) values are Shapley values with

$$\nu(X_S) = \mathbb{E}\left[\hat{f}(\mathbf{X}) \mid X_S\right]$$

- From basic probability theory, we easily get

$$\mathbb{E}\left[\hat{f}(\mathbf{X}) \mid X_S = x_S\right] = \int \hat{f}(\mathbf{x}) p(\mathbf{x} \mid x_S) \mathrm{d}x_{-S}.$$

- SHAP values can be estimated in two ways:
  - using the same marginalization strategy, i.e., SHAP sampling values;
  - using a Kernel approach, i.e., Kernel SHAP.
- We will now focus on Kernel SHAP.

Kernel SHAP

- Kernel SHAP assumes independence between covariates, i.e., $X_{S_1} \mid X_{S_2} \sim X_{S_1}$ when $S_1 \cap S_2 = \emptyset$, so that

$$\mathbb{E}\left[\hat{f}(\mathbf{X}) \mid X_S\right] = \int \hat{f}(\mathbf{x}) p(\mathbf{x} \mid x_S) \mathrm{d}x_{-S} = \int \hat{f}(\mathbf{x}) p(x_{-S}) \mathrm{d}x_{-S}.$$

- Using (as often), we can define the Kernel SHAP estimator

$$\hat{\nu}(X_{i,j}) = \frac{1}{L} \sum_{\ell=1}^{J} \hat{f}(X_{i,j}, \tilde{X}_{-j}),$$

where the $X_{\ell,-j}$'s are sampled independently from $X_{i,j}$.

If covariates are highly dependent, estimates will be completely off. Some variations exists to enable dependent features, e.g., using a multivariate Gaussian distribution.

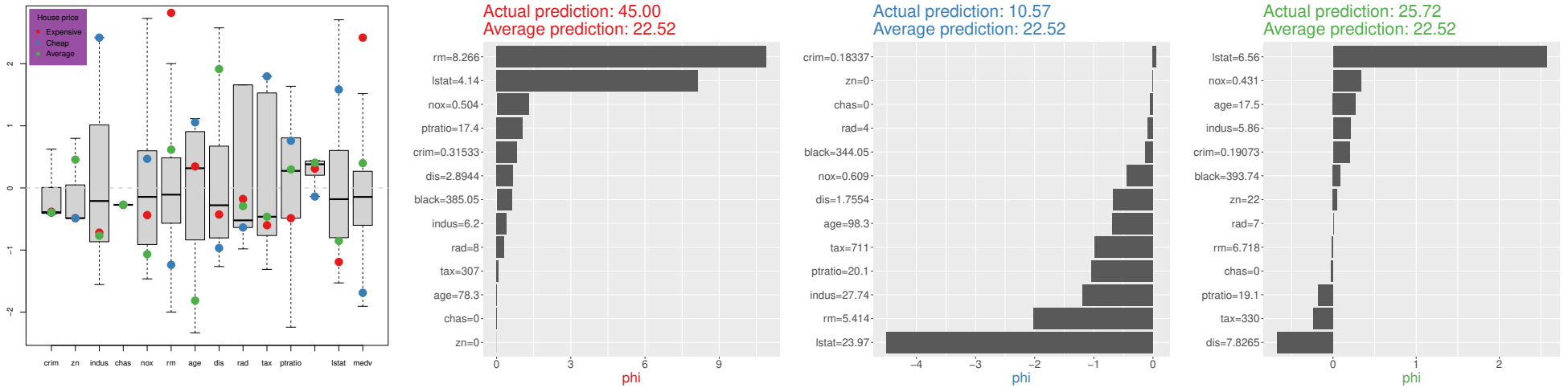


Figure 17: *Boxplot (scaled Boston) and Shapley values for the Boston housing regression problem.*

4. Workflow

R workflow

```
> library(tidymodels)
> data <- tidymodels::ames
> data
# A tibble: 2,930 × 74
   MS_SubClass    MS_Zoning Lot_Frontage Lot_Area Street Alley Lot_Shape
 * <fct>          <fct>            <dbl>    <int> <fct>  <fct> <fct>
 1 One_Story_194... Resident...          141    31770 Pave   No_A... Slightly...
 2 One_Story_194... Resident...           80    11622 Pave   No_A... Regular
 3 One_Story_194... Resident...           81    14267 Pave   No_A... Slightly...
 4 One_Story_194... Resident...           93    11160 Pave   No_A... Regular
 5 Two_Story_194... Resident...           74    13830 Pave   No_A... Slightly...
 6 Two_Story_194... Resident...           78     9978 Pave   No_A... Slightly...
 7 One_Story_PUD... Resident...           41     4920 Pave   No_A... Regular
 8 One_Story_PUD... Resident...           43     5005 Pave   No_A... Slightly...
 9 One_Story_PUD... Resident...           39     5389 Pave   No_A... Slightly...
10 Two_Story_194... Resident...           60     7500 Pave   No_A... Regular
# i 2,920 more rows
# i 67 more variables: Land_Contour <fct>, Utilities <fct>, ...
```

- Prediction of the price (`Sale_Price`) of a house (regression problem)
- We have $n = 2930$ individuals and $p = 74$ covariates (some are categorical)

Data visualization: Outcome

```
> ggplot(data, aes(Sale_Price)) + geom_hist()
> ggplot(data, aes(log(Sale_Price))) + geom_hist()
```

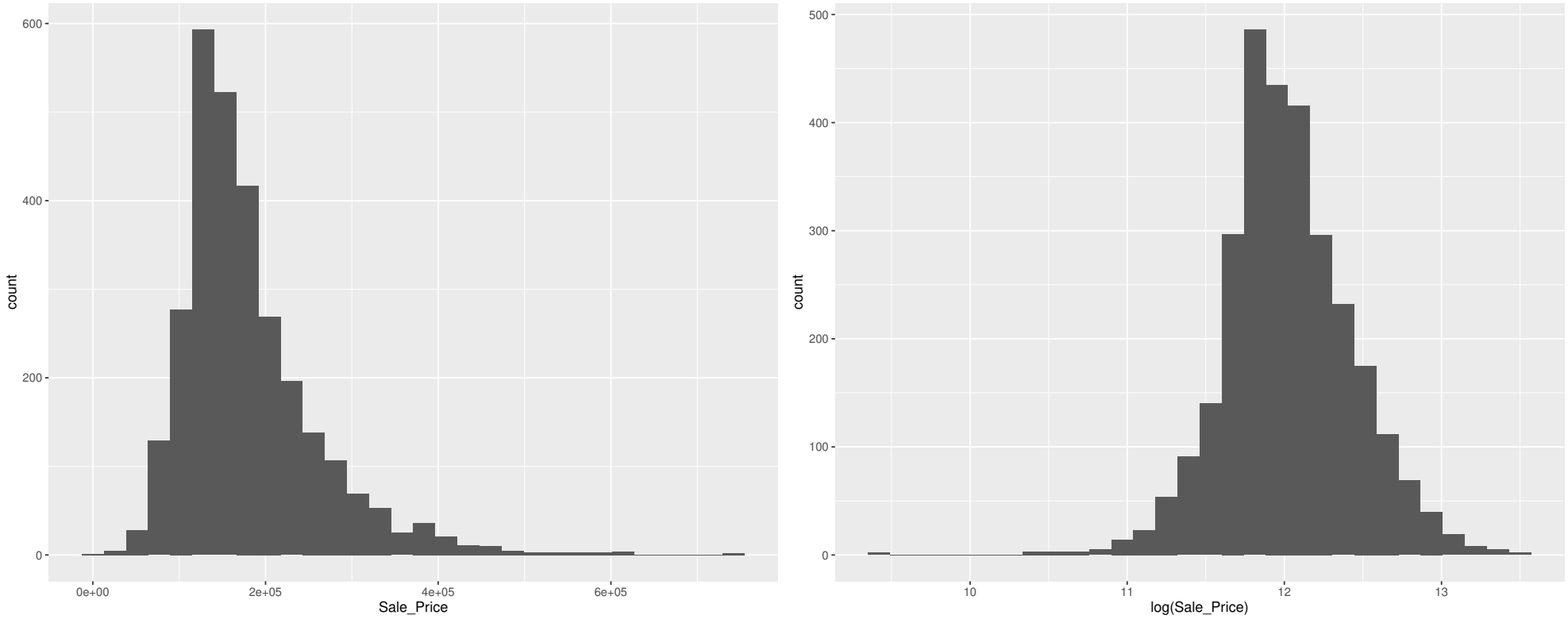


Figure 18: *Left: Histrogram of the sale price. The outcome is (severly) right-skewed and variance stabilization may be suitable. Right: Histogram on the log-transformed price.*

Feature visualization

```
> library(GGally)
> subdata <- data %>% select_if(is.numeric)
> ggpairs(subdata %>% select(sample(ncol(subdata), 6))
```

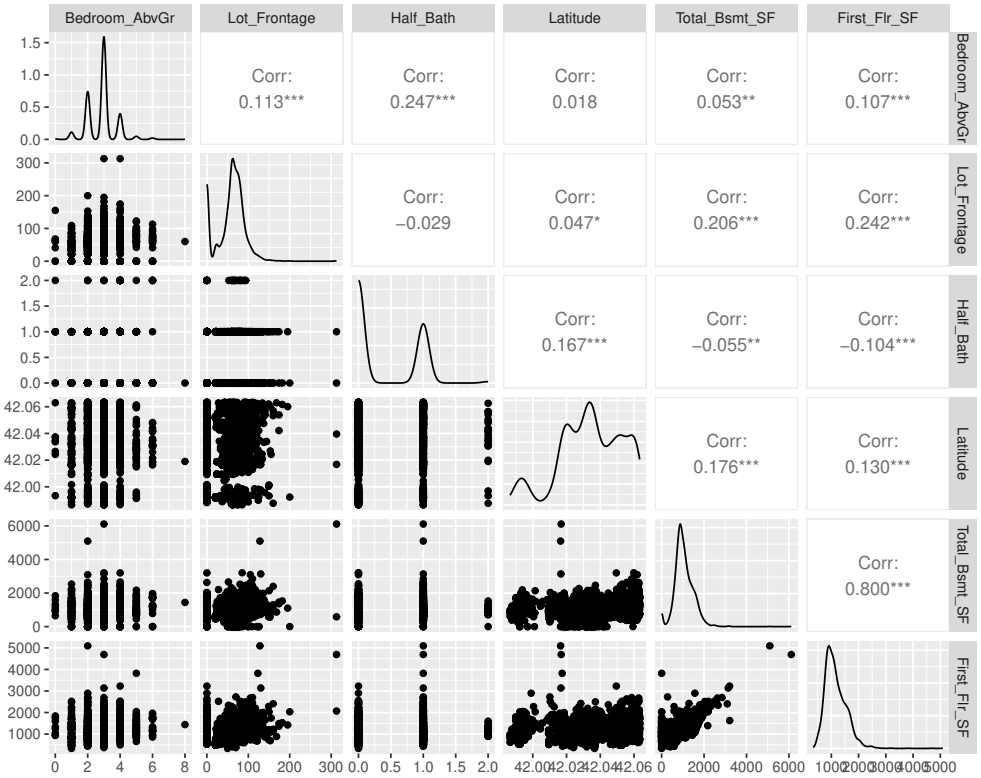


Figure 19: *Pair plot of some quantitative features. Plot show that some features are correlated. (automatic plot is nasty—integer valued variables!*

Splitting the dataset

```
## Split the dataset in training (3/4) and test set (1/4)
> splits <- initial_split(data, prop = 0.75)
> train <- training(splits)
> test  <- testing(splits)
```

ℹ Here there is no validation set since we won't do fine tuning. Most often you will so you may rather use the `initial validation split` function.

A fitted random forest model

```
> rf_model <-
   rand_forest(mtry = 8, min_n = 7, trees = 1000) %>%
   set_engine("ranger", num.threads = cores, importance = "impurity") %>%
   set_mode("regression")
> fit <- rf_model %>% fit(log10(Sale_Price) ~ ., data = train)
> fit
parsnip model object

Ranger result

Call:
 ranger::ranger(x = maybe_data_frame(x), ...

Type:                             Regression
Number of trees:                  1000
Sample size:                      2197
Number of independent variables:  73
Mtry:                             8
Target node size:                 7
Variable importance mode:         impurity
Splitrule:                        variance
OOB prediction error (MSE):       0.003977174
R squared (OOB):                  0.8727639
```

Model's performance

```
## Get prediction
> pred <- fit %>% predict(new_data = test)

## Compute performance metrics
> df <- data.frame(test, pred)
> df <- df %>% mutate(LogSale_Price = log(Sale_Price))
> df %>% metrics(truth = LogSale_Price, estimate = .pred)
# A tibble: 3 × 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard      0.135
2 rsq     standard      0.900
3 mae     standard      0.0922
> ## Compare observation from prediction
df %>% ggplot(aes(LogSale_Price, .pred)) + geom_point() +
  geom_abline(intercept = 0, slope = 1) +
  xlab("Log Sale Price") + ylab("Prediction")
```

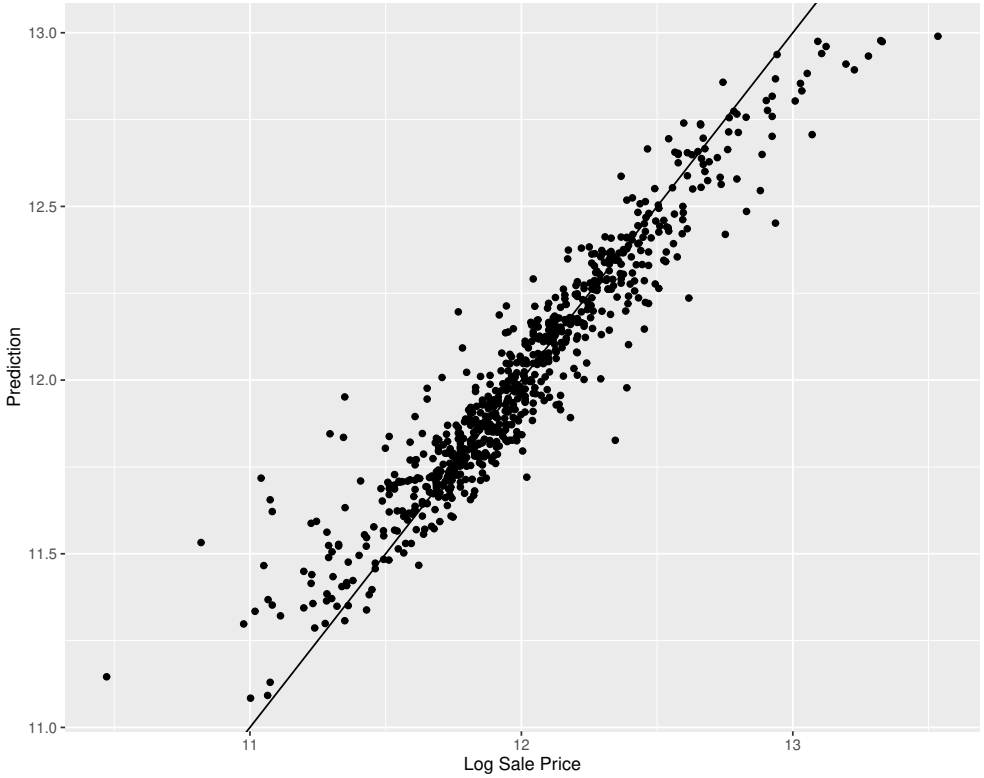


Figure 20: *Comparison between observed log sale prices and predictions on the test set.*

Feature importance

```
> library(vip)
> fit %>% vip(num_features = 20)
```

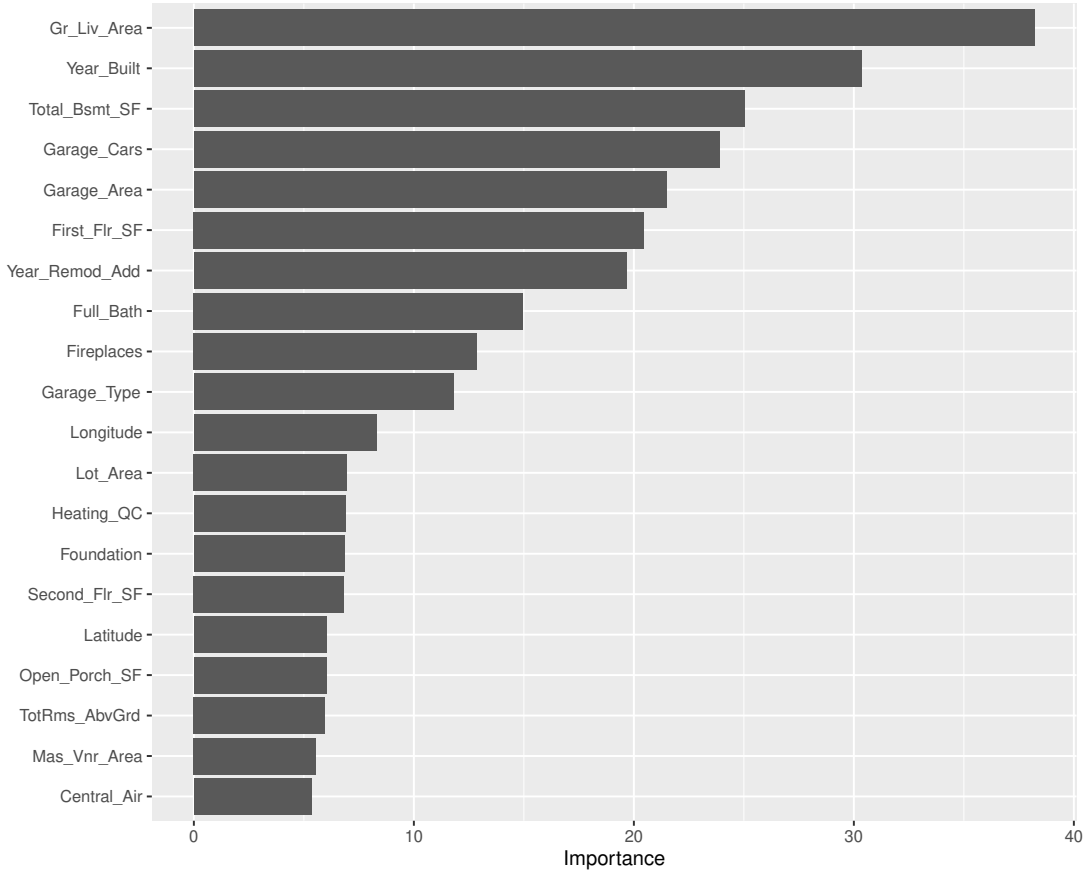


Figure 21: *Feature importance for our fitted random forest model.*

Python workflow

```
import pandas as pd
data = pd.read_csv('https://mribatet.perso.math.cnrs.fr/CentraleNantes/Data/ames.csv')
data
```

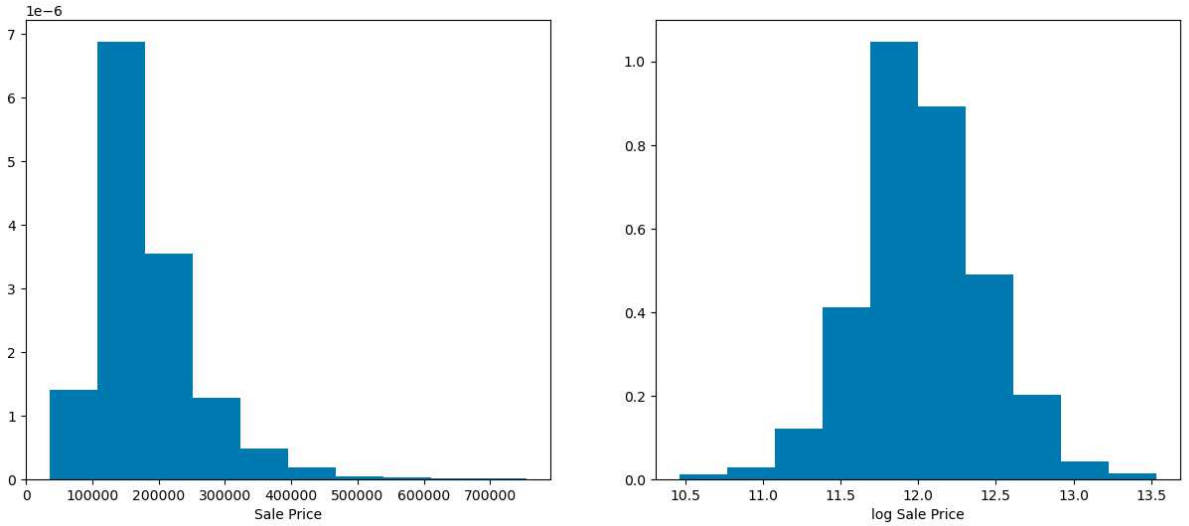
	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	P
0	1	60	RL	65.0	8450	Pave	?	Reg	Lvl	AllPub	...	0	
1	2	20	RL	80.0	9600	Pave	?	Reg	Lvl	AllPub	...	0	
2	3	60	RL	68.0	11250	Pave	?	IR1	Lvl	AllPub	...	0	
3	4	70	RL	60.0	9550	Pave	?	IR1	Lvl	AllPub	...	0	
4	5	60	RL	84.0	14260	Pave	?	IR1	Lvl	AllPub	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	?	Reg	Lvl	AllPub	...	0	
1456	1457	20	RL	85.0	13175	Pave	?	Reg	Lvl	AllPub	...	0	
1457	1458	70	RL	66.0	9042	Pave	?	Reg	Lvl	AllPub	...	0	
1458	1459	20	RL	68.0	9717	Pave	?	Reg	Lvl	AllPub	...	0	
1459	1460	20	RL	75.0	9937	Pave	?	Reg	Lvl	AllPub	...	0	

1460 rows × 81 columns

Data visualization: Outcome

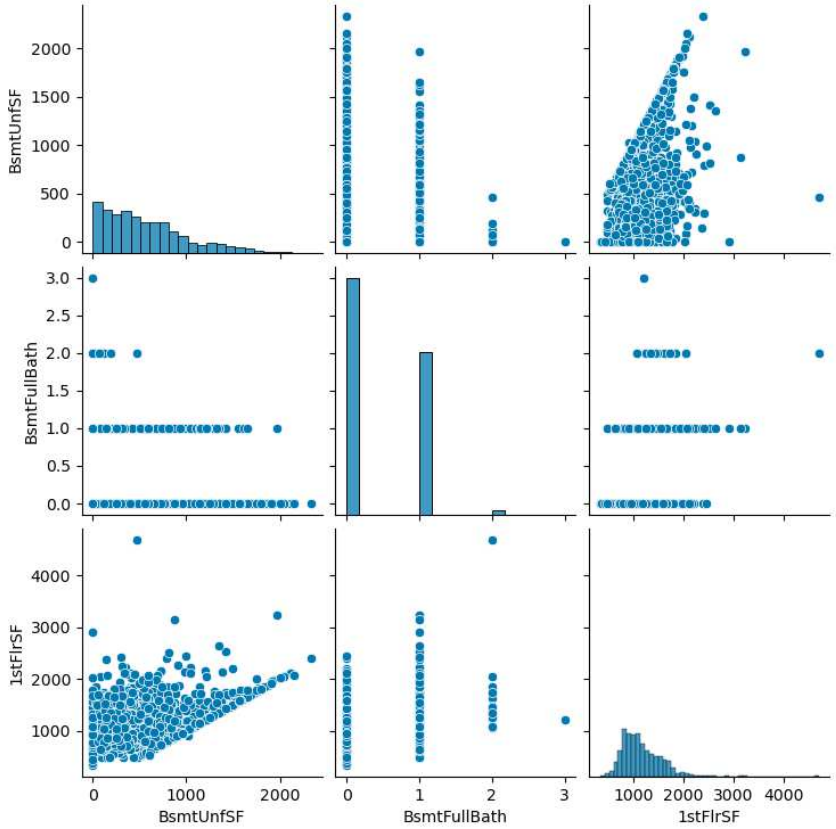
```
import numpy as np
import matplotlib.pyplot as plt

data = data.drop(columns="Id")
X, Y = (data.drop(columns="SalePrice"), data["SalePrice"])
plt.figure(figsize=(15,6))
plt.subplot(1, 2, 1)
plt.hist(Y, density=True)
plt.xlabel("Sale Price")
plt.subplot(1, 2, 2)
plt.hist(np.log(Y), density = True)
plt.xlabel("log Sale Price")
```



Data visualization: feature visualization

```
import seaborn as sns
idx = np.random.choice(X.shape[1], 6)
print(X.shape)
print(idx)
sns.pairplot(X.iloc[:,idx])
```



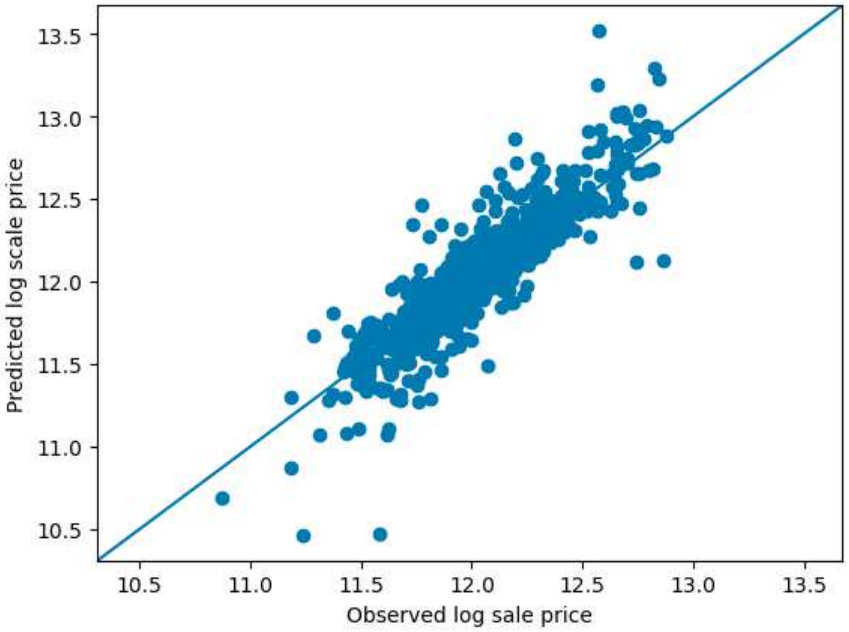
A fitted random forest

```
## Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, np.log(Y))

## Fit
from sklearn.ensemble import RandomForestRegressor
my_rf = RandomForestRegressor(n_estimators=10, max_features=4)
fit = my_rf.fit(X_train, y_train)
```

Model's performance

```
plt.scatter(my_rf.predict(X_test), y_test)
plt.axline((y_test.min(), y_test.min()), (y_test.max(), y_test.max()))
plt.xlabel("Observed log sale price")
plt.ylabel("Predicted log scale price")
```



II. Boosting

You will know what's behind...

```
> library(gbm)##other libraries exist (must be installed first)
> fit <- gbm(y ~ ., data = train, distribution = "adaboost", n.trees = 500)
> pred <- predict(fit, test)

> library(xgboost)
> model <- xgboost(data = Xtrain, label = Ytrain, max.depth = 2,
                   nrounds = 500)
> predict(model, Xtest)
```

```
from sklearn.ensemble import AdaBoostClassifier
boost = AdaBoostClassifier(n_estimator = 500, max_depth = 1,
                           learning_rate = 1)##decision stumps
fit = boost.fit(X_train, Y_train)
y_pred = model.predict(X_test)

from xgboost import XGBRegressor##for regression of course
model = XGBRegressor()
model.fit(Xtrain, Ytrain)
pred = model.predict(Xtest)
```

1. Introduction

Brief history of Boosting

1994 Open question about boosting
1990 First mathematical algorithm (non adaptive)
1991 Almost optimal algorithm (non adpative)
1995 AdaBoost
1996– Use of Adaboost for applications
1999–2001 Casting AdaBoost as (functional) gradient descent
2016 XGBoost (Newton–Raphson)

Quick overview

- Boosting is a fairly recent learning strategy (90's / 00's)
- Still some theoretical developments and variants
- The idea is to mix poor learners to get a better one.
- Originally introduced for classification, regression is possible though.
- It belongs to the class of ensemble methods, i.e., combination of several models.
- The first boosting algorithm AdaBoost.

To introduce the theory we will (first) focus on classification.

Classifier (reminder)

Definition 7. A K–class classifier is a mapping

$$\begin{aligned} H \colon \mathcal{X} &\longrightarrow \{1, \ldots, K\} \\ \mathbf{x} &\longmapsto H(\mathbf{x}), \end{aligned}$$

where $\mathcal{X}$ is the covariable / feature space.
When $K = 2$, it is a binary classification problem.
The accuracy of a classifier can be assessed from its (theoretical) error rate

$$\mathbb{E}_{(\mathbf{X},Y)}\left[1_{\{Y \neq H(\mathbf{X})\}}\right] = \Pr\{Y \neq H(\mathbf{X})\}.$$

However note that other relevant metrics that the error rate exists, e.g., sensitivity, specificity, F–score, ROC, AUC, ...

Definition 8. A regression estimator (regressor beurk) is a mapping

$$\begin{aligned} f : \mathcal{X} &\longrightarrow \mathbb{R}^d \\ \mathbf{x} &\longmapsto f(\mathbf{x}), \end{aligned}$$

where $\mathcal{X}$ is the covariable / feature space.
Most often $d = 1$ and corresponds to a univariate regression problem.

The accuracy of a regression estimator can be assessed from many (theoretical) error/loss such as

$$\frac{1}{2}\mathbb{E}_{(\mathbf{X},Y)}\left[\|f(\mathbf{X}) - Y\|_2^2\right], \quad \texttt{(Mean squared error)}$$

$$\mathbb{E}_{(\mathbf{X},Y)}\left[\|f(\mathbf{X}) - Y\|_1\right], \quad \texttt{(Mean absolute error)}$$

$$\mathbb{E}_{(\mathbf{X},Y)}\left[\frac{1}{2}\|f(\mathbf{X}) - Y\|_2^2 1_{\{\|Y - f(\mathbf{X})\|_1 \leq \delta\}} + \left\{\delta\|Y - f(\mathbf{X})\|_1 - \frac{\delta^2}{2}\right\} 1_{\{\|Y - f(\mathbf{X})\| > \delta\}}\right], \quad \texttt{(Huber)}$$

Definition 9. A K–class classifier H is said to be a weak learner if its error rate is at least a little better than that of a random guess, i.e.,

$$\text{error rate(weak learner)} \leq \text{error rate(random guess)} = 1 - \frac{1}{K} = \frac{K-1}{K}.$$

Definition 10. A K–class classifier H is said to be a strong learner if it converges in probability to the true classifier H_*, i.e., for all $\varepsilon > 0$ we have

$$\Pr\left\{|H_*(X) - H(X)| > \varepsilon\right\} \longrightarrow 0, \qquad n \to \infty.$$

2. Adaboost

Open questions?

> Is it possible to get a strong learner from a set of weak learners?

- Clearly no if we just fit weak classifiers on the same data
- However the answer is YES!
- But we need to change the data set in a sensible way.

AdaBoost (binary classification $Y \in \{-1, 1\}$)

Algorithm 7: AdaBoost for the binary classification task.

input : Supervised data set $\mathcal{D}_n = \{(\mathbf{X}_i, Y_i) : i = 1, \ldots, n\}$, $T > 1$
output: A (hopefully) strong classifier

1 Initialize weights $\omega_1(i) = 1/n$, $i = 1, \ldots, n$;
2 **for** $t \leftarrow 1$ **to** T **do**
3 Fit a weak classifier h_t from $\mathcal{D}_n$ with weights $\omega_t(i)$;
4 Compute the (weighted) empirical error rate

$$\varepsilon_t = \text{error}_t(\mathcal{D}_n) = \sum_{i=1}^{n} \omega_t(i) 1_{\{Y_i \neq h_t(\mathbf{X}_i)\}}$$

5 Compute learner weights

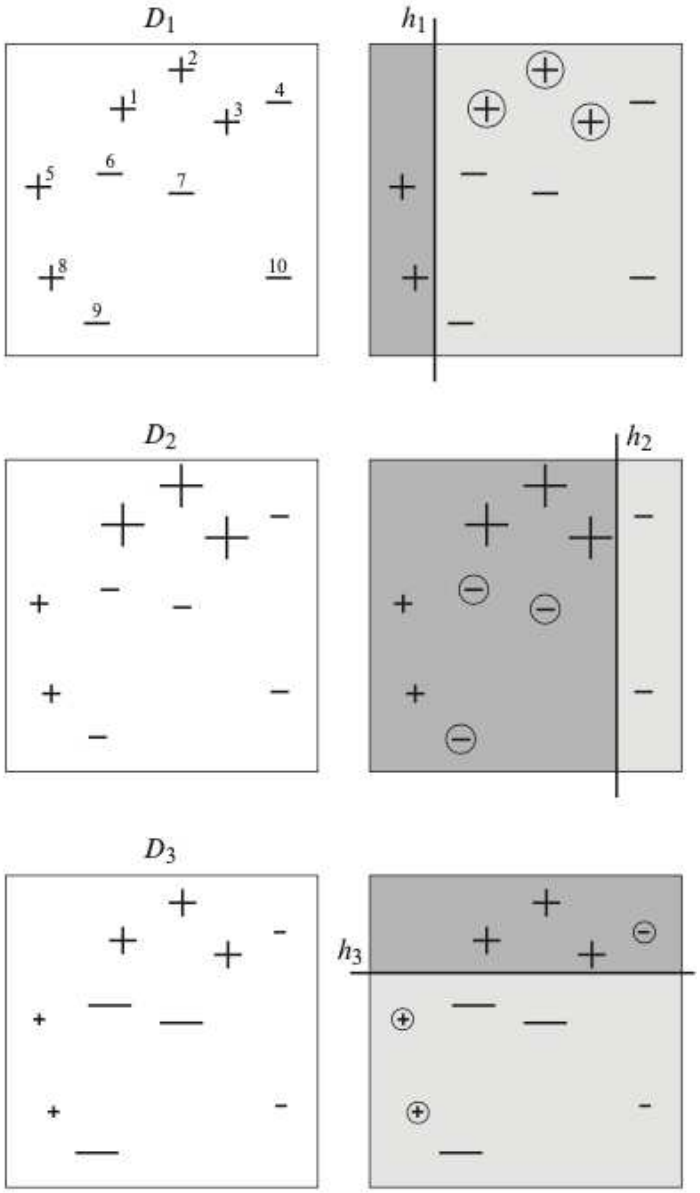
$$\alpha_t = \frac{1}{2} \text{logit}(1 - \varepsilon_t)$$

6 Update the (un-normalized) observation weights

$$\omega_{t+1}(i) \propto \omega_t(i) \exp\left\{-\alpha_t Y_i h_t(X_i)\right\}, \qquad i = 1, \ldots, n$$

7 Return the final binary classifier

$$H : \mathbf{x} \longmapsto \text{sign}\left\{\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right\}$$



- Taken from *Boosting: Foundations and Algorithms. R. Schapire and Y. Freund. (2012)*.
- Each row depicts one iteration.
- Left column: Illustration of distributions $\omega_t(i)$ (size proportional to weights).
- Right column: Prediction from weak classifiers. Darker regions indicates positive predictions.
- Missclassified observations are circled.

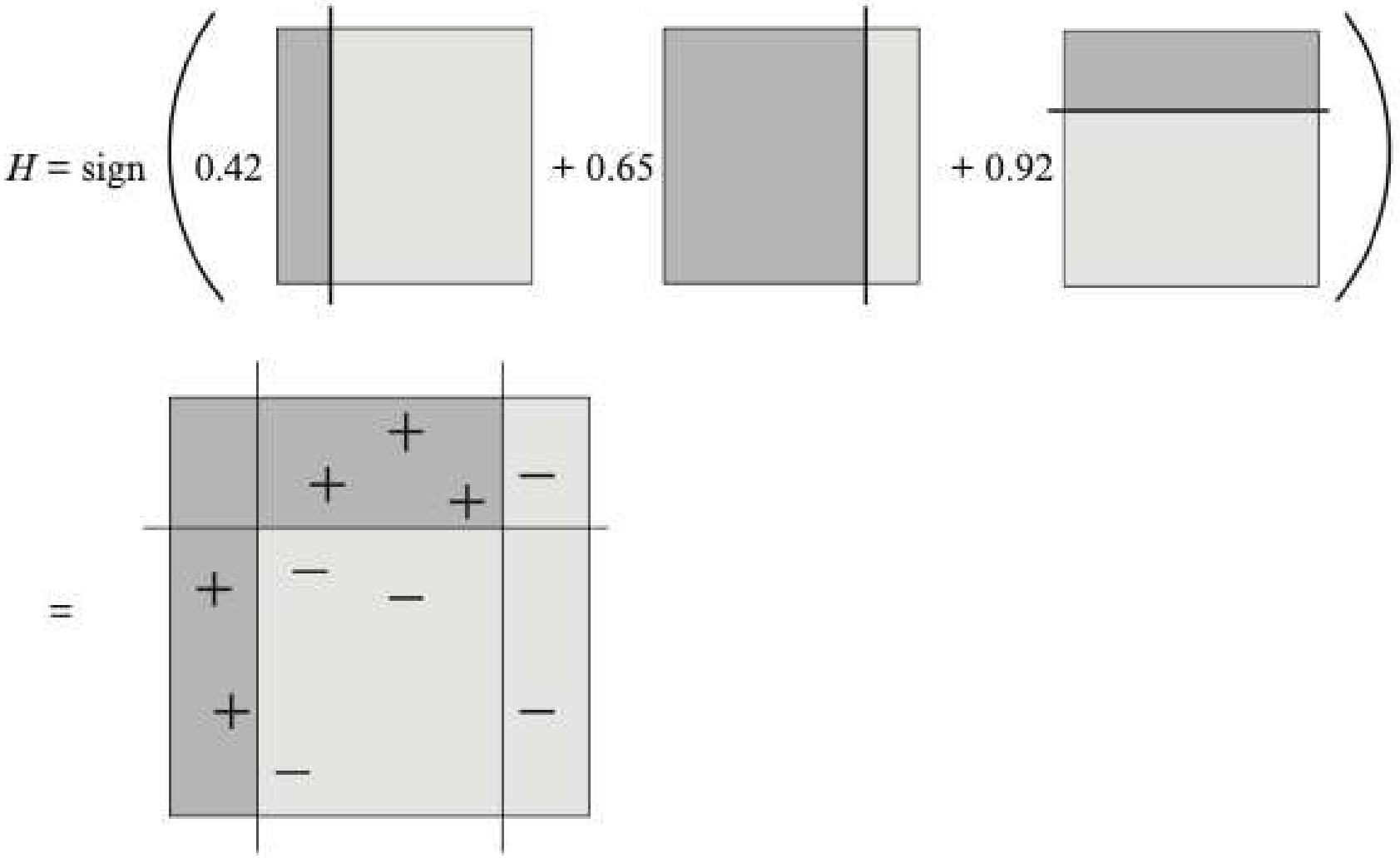


Figure 22: *Taken from* Boosting: Foundations and Algorithms. R. Schapire and Y. Freund. (2012).

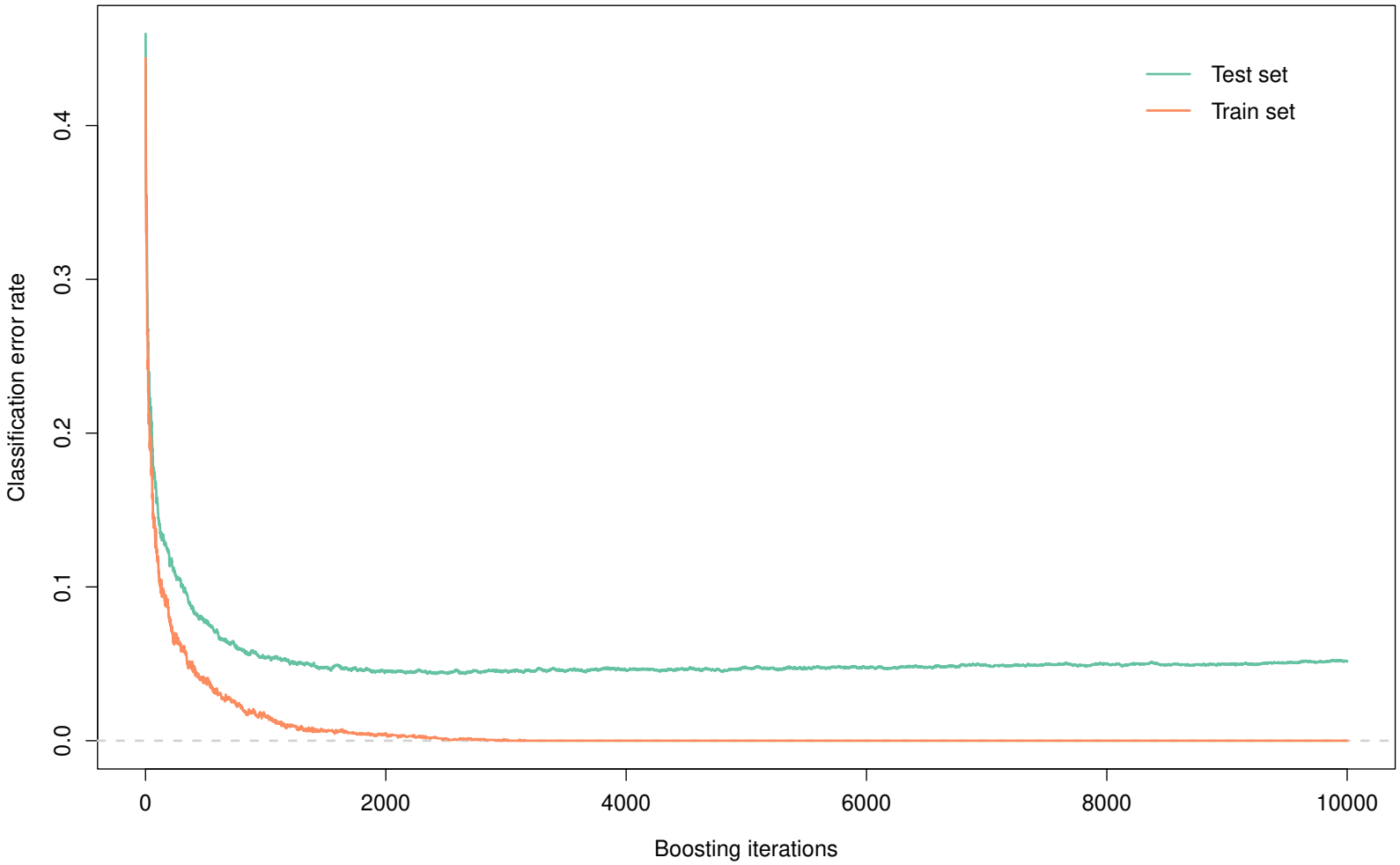


Figure 23: *Evolution of the boosting (test) error as the number of iterations increases.*

Comments on AdaBoost

- Pronounce "ADD–uh–boost" (short for "adaptive boosting").
- Agnostic as weak learners are not specified.
- Uses a sequence of distributions over $\mathcal{D}_n$, i.e., the observation weights $\omega_t(i)$:
  - The weight $\omega_t(i)$ "measures" how badly h_t predicts Y_i.
  - Poorly predicted observations will get increased weights next round.
- Popular choices for weak learners are
  - naive Bayes
  - logistic regression
  - support vector machine
  - shallow decision trees, in particular decision stumps

Random guessing?

- Why should we have a classification error rate smaller than $1/2$?
- The reason is in line 5 of AdaBoost, i.e.,

$$\alpha_t = \frac{1}{2}\text{logit}(1-\varepsilon_t) \qquad \texttt{(learner weights)}$$

- If $\varepsilon = 1/2$, $\alpha_t = 0$ and AdaBoost get stuck to its current state.
- If h_t and $-h_t$ belong to the same family of weak learner, we cannot have $\varepsilon_t > 1/2$:
  - If h_t has error rate $\varepsilon_t > 1/2$, then $-h_t$ has error rate $1 - \varepsilon_t < 1/2$.
  - Thus $-h_t$ has better performance than h_t and contradicts line 3.
- It doesn't make sense to have

$$\underbrace{\alpha_t < 0}_{\text{negative weight!}} \iff \text{logit}(1-\varepsilon_t) < 0 \iff \varepsilon_t > 0.5$$

A surrogate optimization problem

- Our goal is to minimize the classification loss for our "final estimator H", i.e., minimizing

$$\text{loss}(H) = \frac{1}{n}\sum_{i=1}^{n} 1_{\{H(\mathbf{X}_i)\neq Y_i\}}$$

- It is easily seen that

$$\text{loss}(H) \leq \frac{1}{n}\sum_{i=1}^{n} e^{-Y_i \sum_{t=1}^{T} \alpha_t h_t(\mathbf{X}_i)} = U_T$$

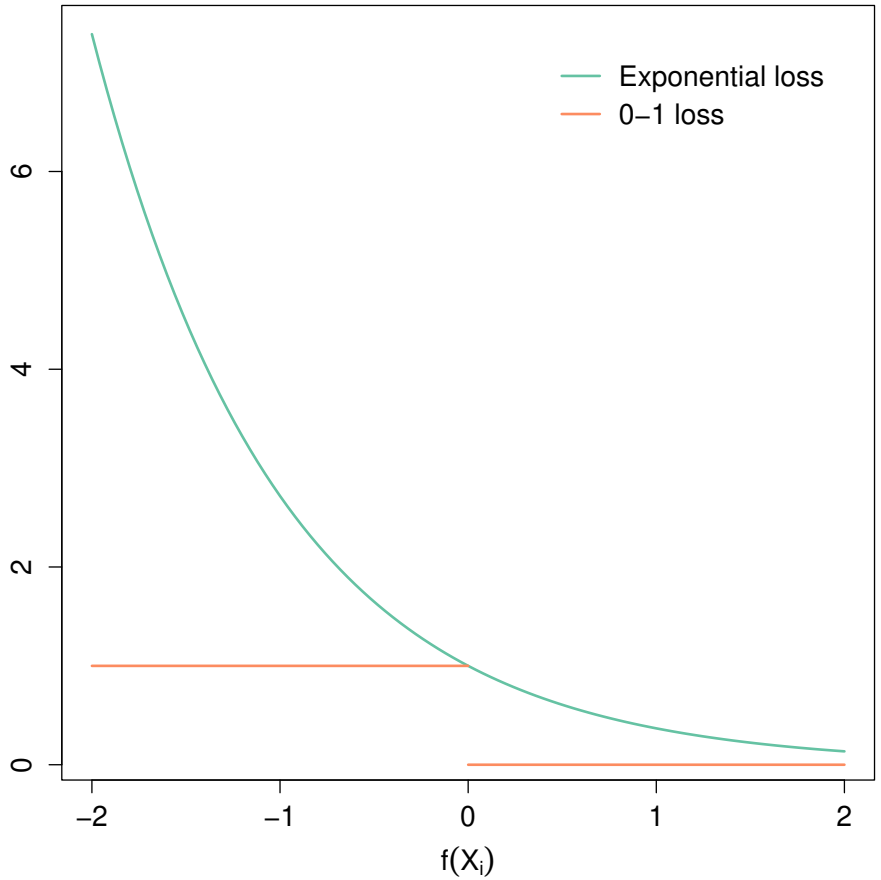


Figure 24: *The 0–1 loss is upper bounded by the exponential loss. Here we assume $Y = 1$.*

☞ Minimizing the loss by minimizing the upper bound U_T (which is convex).

Investigating the upper bound U_T

$$N_t = \sum_{i=1}^{n} \omega_t(i) e^{-\alpha_t Y_i h_t(\mathbf{X}_i)} \qquad \texttt{(normalizing constant of } \omega_{t+1}\texttt{)}$$

We thus have

$$\begin{aligned}
1 &= N_T^{-1} \sum_{i=1}^{n} \omega_T(i) e^{-\alpha_T Y_i h_T(\mathbf{X}_i)} \\
&= N_T^{-1} N_{T-1}^{-1} \sum_{i=1}^{n} \omega_{T-1}(i) e^{-\alpha_{T-1} Y_i h_{T-1}(\mathbf{X}_i)} e^{-\alpha_T Y_i h_T(\mathbf{X}_i)} \\
&= \texttt{...recursion...} = \left(\prod_{t=1}^{T} N_t^{-1} \right) \underbrace{\sum_{i=1}^{n} n^{-1} e^{\sum_{t=1}^{T} \alpha_t Y_i h_t(\mathbf{X}_i)}}_{U_T}
\end{aligned}$$

☞ We have shown that $U_T = \prod_{t=1}^{T} N_t$

Minimizing the upper bound U_T

$$U_T = \prod_{t=1}^{T} N_t, \qquad N_t = \sum_{i=1}^{n} \omega_t(i) e^{-\alpha_t Y_i h_t(\mathbf{X}_i)}$$

- To minimize U_T, one can minimize each N_t w.r.t. α_t and h_t.
- Minimization is done using a greedy approach,

Algorithm 8: Greedy minimization of N_t w.r.t. α_t and ω_t.

1 **for** $t = 1, \ldots, T$ **do**
2 Minimize N_t w.r.t. to h_t;
3 Minimize N_t w.r.t. α_t;
4 Return the final binary classifier H;

☞ Line 2 amounts to fit the (weak) classifier h_t while line 3 is a bit more involved.

Minimization of N_t w.r.t. α_t

$$
\begin{aligned}
N_t &= \sum_{i=1}^{n} \omega_t(i) e^{-\alpha_t Y_i h_t(\mathbf{X}_i)} \\
&= e^{\alpha_t} \underbrace{\sum_{i=1}^{n} \omega_t(i) 1_{\{Y_i \neq h_t(\mathbf{X}_i)\}}}_{\text{weighted error rate}} + e^{-\alpha_t} \sum_{i=1}^{n} \omega_t(i)_{\{Y_i = h_t(\mathbf{X}_i)\}} \\
&= \varepsilon_t e^{\alpha_t} + (1 - \varepsilon_t) e^{-\alpha_t},
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial N_t}{\partial \alpha_t} = 0 &\Longleftrightarrow \varepsilon_t e^{\alpha^t} - (1 - \varepsilon_t) e^{-\alpha_t} = 0 \\
&\Longleftrightarrow \exp(2\alpha_t) = \frac{1 - \varepsilon_t}{\varepsilon_t} \\
&\Longleftrightarrow \alpha_t = \frac{1}{2} \mathsf{logit}(1 - \varepsilon_t).
\end{aligned}
$$

Algorithm 9: AdaBoost for the binary classification task.

input : Supervised data set $\mathcal{D}_n = \{(\mathbf{X}_i, Y_i) : i = 1, \ldots, n\}$, $T > 1$
output: A (hopefully) strong classifier

1 Initialize weights $\omega_1(i) = 1/n$, $i = 1, \ldots, n$;
2 **for** $t \leftarrow 1$ **to** T **do**
3 Fit a weak classifier h_t from $\mathcal{D}_n$ with weights $\omega_t(i)$;
4 Compute the (weighted) empirical error rate

$$\varepsilon_t = \text{error}_t(\mathcal{D}_n) = \sum_{i=1}^{n} \omega_t(i) 1_{\{Y_i \neq h_t(\mathbf{X}_i)\}}$$

5 Compute learner weights

$$\alpha_t = \frac{1}{2}\text{logit}(1 - \varepsilon_t)$$

6 Update the (unnormalized) observation weights

$$\omega_{t+1}(i) \propto \omega_t(i) \exp\left\{-\alpha_t Y_i h_t(X_i)\right\}, \qquad i = 1, \ldots, n$$

7 Return the final binary classifier

$$H : \mathbf{x} \longmapsto \text{sign}\left\{\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right\}$$

Investigating N

$$N_t = \varepsilon_t e^{\alpha_t} + (1-\varepsilon_t)e^{-\alpha_t}, \qquad \alpha_t = \frac{1}{2}\log\frac{1-\varepsilon_t}{\varepsilon_t}$$
$$= \varepsilon_t\sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}} + (1-\varepsilon_t)\sqrt{\frac{\varepsilon_t}{1-\varepsilon_t}}$$

☞ We thus have

$$N_t = 2\sqrt{\varepsilon_t(1-\varepsilon_t)} = \sqrt{1-(1-2\varepsilon_t)^2}.$$

Convergence rate

$$\begin{aligned}\text{loss}(C_T) \leq U_T &= \prod_{t=1}^{T} N_t = \prod_{t=1}^{T} \sqrt{1-(1-2\varepsilon_t)^2} \\ &= \exp\left[\frac{1}{2}\sum_{t=1}^{T}\log\left\{1-(1-2\varepsilon_t)^2\right\}\right] \\ &\leq \exp\left\{-\frac{1}{2}\sum_{t=1}^{T}(1-2\varepsilon_t)^2\right\} \\ &= \exp\left\{-2\sum_{t=1}^{T}\left(\frac{1}{2}-\varepsilon_t\right)^2\right\}.\end{aligned}$$

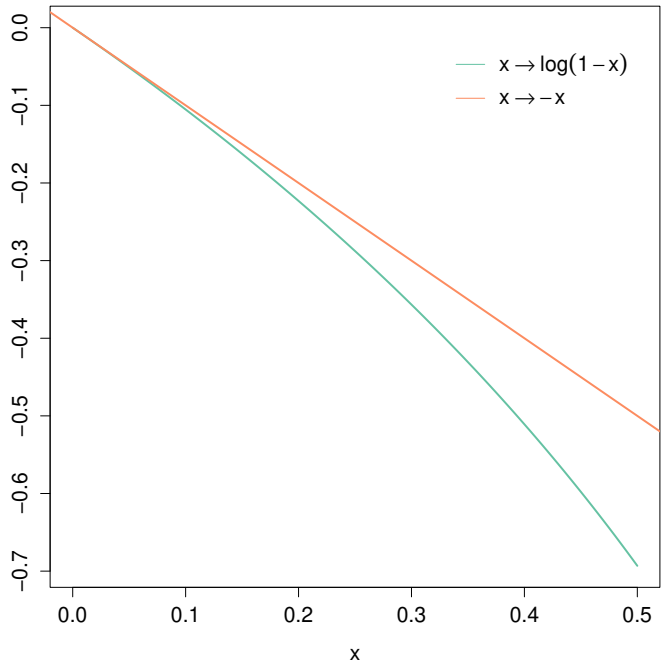


Figure 25: *Illustration that the mapping* $x \mapsto \log(1-x)$ *is upper bounded by* $x \mapsto -x$.

☞ A $T \to \infty$, the training set error rate converges to 0 at an exponential rate.

Boosting behaviour

- Overfitting might occur when weak learners are
  - too complex as it induce large variance
  - too close to random guessing
- As usual we face a bias variance trade off:
  - T large: small bias and large variance
  - T small: large bias and small variance
- About outliers:
  - Boosting can cope with outlier by construction
  - Too many outliers may degrade performance or dramatically slows down convergence

Towards multi-class classification

- We introduced AdaBoost for binary classification
- We show that we need to have weak learner weights $\alpha_t > 0$, i.e., $\varepsilon_t > 1/2$.
- When we have K–classes, the classification error rate is

$$\varepsilon_t = 1 - \frac{1}{K} = \frac{K-1}{K}.$$

- When $K > 2$, might be difficult to find a classifier h_t satisfying this constraint.
- How to bypass this problem:
  - use a one-vs-all strategy: not a good idea
  - use SAMME
  - use something different (teasing)

Algorithm 10: SAMME for a K–class classification task.

input : Supervised data set $\mathcal{D}_n = \{(\mathbf{X}_i, Y_i) : i = 1, \ldots, n\}$, $T > 1$
output: A (hopefully) strong classifier

1 Initialize weights $\omega_1(i) = 1/n$, $i = 1, \ldots, n$;
2 **for** $t \leftarrow 1$ **to** T **do**
3 Fit a weak classifier h_t from $\mathcal{D}_n$ with weights $\omega_t(i)$;
4 Compute the (weighted) empirical error rate

$$\varepsilon_t = \text{error}_t(\mathcal{D}_n) = \sum_{i=1}^{n} \omega_t(i) 1_{\{Y_i \neq h_t(\mathbf{X}_i)\}}$$

5 Compute learner weights

$$\alpha_t = \frac{1}{2}\text{logit}(1 - \varepsilon_t) + \frac{1}{2}\log(K - 1)$$

6 Update the (un-normalized) observation weights

$$\omega_{t+1}(i) \propto \omega_t(i) \exp\left\{-\alpha_t Y_i h_t(X_i)\right\}, \qquad i = 1, \ldots, n$$

7 Return the final K–class classifier

$$H : \mathbf{x} \longmapsto k \in \underset{\text{argmax}}{\{1, \ldots, K\}} \sum_{t=1}^{T} \alpha_t 1_{\{h_t(\mathbf{x}) = k\}};$$

Comments on SAMME

- Stagewise Additive Modeling using a Multi–class Exponential loss function.
- Clearly when $K = 2$ we get the original AdaBoost
- As the name suggests, the classification error is now bounded by the multi-class exponential loss

$$\mathbf{x} \longmapsto \exp\left\{-\frac{1}{K}\sum_{k=1}^{K} Y^{(k)} h^{(k)}(\mathbf{x})\right\},$$

  with a constraint to ensure that we predict at least one class.
- With this modification we now have

$$\alpha_t > 0 \Longleftrightarrow \frac{1-\varepsilon_t}{\varepsilon_t} > 1 - K \Longleftrightarrow 1 - \varepsilon_t > \frac{1}{K}$$

☞ The classifier h_t should be (as before) a weak classifier for K–classes and is much more easier to get.

SAMME.R

- The SAMME algorithm uses weak classifiers $h_t(\mathbf{x}) \in \{1, \ldots, K\}$, i.e.,

$$h_t(\mathbf{x}) = \underset{k}{\operatorname{argmax}} \hat{\Pr}(Y = k \mid \mathbf{X} = \mathbf{x})$$

- Another approach is to treat it as a regression problem

$$h_t(\mathbf{x}) \propto \hat{\Pr}(Y = k \mid \mathbf{X} = \mathbf{x})$$

- This is idea of `SAMME.R` where `R` stands for real.

AdaBoost with R

```
> library(gbm)##other libraries exist (must be installed first)
> fit <- gbm(y ~ ., data = train, distribution = "adaboost", n.trees = 500)
> pred <- predict(fit, test)
```

AdaBoost with Python

```
from sklearn.ensemble import AdaBoostClassifier
boost = AdaBoostClassifier(n_estimator = 500, max_depth = 1,
                           learning_rate = 1)##decision stumps
fit = boost.fit(X_train, Y_train)
y_pred = model.predict(X_test)
```

There is an additional learning rate (`lr`) argument while the original doesn't. It is a generalization where

$$\alpha_t = \frac{\texttt{lr}}{2} \times \text{logit}(1 - \varepsilon_t) + \frac{1}{2}\log(K-1).$$

3. Gradient Boosting

Towards Gradient Boosting

- AdaBoost can be thought as a gradient descent.
- Introducing the AdaBoost, we show that

$$\text{loss}_{\text{0-1}}(H) \leq \text{loss}_{\text{exp}}(H), \qquad H(\mathbf{x}) = \text{sign}\left\{\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right\}.$$

- AdaBoost minimizes the exponential loss rather than the 0–1 loss.
- Minimization is done using a greedy gradient descent.
- Could we use the same approach for a generic loss function ℓ?

This is the idea beyond Gradient Boosting (regression trees).

- Boosting is a forward stage-wise estimator, i.e.,

$$f_{t+1}(\mathbf{x}) = f_t(\mathbf{x}) + \alpha h_{t+1}(\mathbf{x}), \qquad h_{t+1} = \arg\min_{h \in \mathcal{H}} \ell(f_t + \alpha h). \tag{2}$$

- Suppose that the loss ℓ is (sub–)differentiable and convex
- For α small enough and fixed, a (functional) Taylor expansion around any $f \in \mathcal{H}$ gives

$$\ell(f + \alpha h) \approx \ell(f) + \alpha < \nabla \ell(f), h > .$$

- Hence treating the above approximation as exact, we get

$$\arg\min_{h \in \mathcal{H}} \ell(f + \alpha h) = \arg\min_{h \in \mathcal{H}} \ < \nabla \ell(f), h >$$

- Since $\ell(H) = \sum_{i=1}^{n} \ell(\delta_{X_i} H)$, one thus have

$$\arg\min_{h \in \mathcal{H}} < \nabla \ell(f), h >= \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} \frac{\partial \ell}{\partial f(\mathbf{x}_i)} h(\mathbf{x}_i)$$

- And having current estimator f_t we get a new one from

$$f_{t+1} = f_t + \alpha h_{t+1}, \qquad h_{t+1} = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} \frac{\partial \ell}{\partial f_t(\mathbf{x}_i)} h(\mathbf{x}_i)$$

i The quantities

$$r_t(\mathbf{x}_i) = \frac{\partial \ell}{\partial f_t(\mathbf{x}_i)}, \qquad i = 1, \ldots, n,$$

are often referred to as the residuals.

Residuals for common losses

- Quadratic loss:

$$\ell(f) = \frac{1}{2} \sum_{i=1}^{n} \{f(\mathbf{x}_i) - y_i\}^2 \Longrightarrow \frac{\partial \ell}{\partial f(\mathbf{x}_i)} = f(\mathbf{x}_i) - y_i$$

- ℓ_1 loss:

$$\ell(f) = \sum_{i=1}^{n} |f(\mathbf{x}_i) - y_i| \Longrightarrow \frac{\partial \ell}{\partial f(\mathbf{x}_i)} = \text{sign}\{f(\mathbf{x}_i) - y_i\}, \qquad \texttt{(sub-gradient)}$$

with the convention that $\text{sign}(0) = [-1, 1]$.

- Exponential loss:

$$\ell(f) = \sum_{i=1}^{n} \exp\{-f(\mathbf{x}_i) y_i\} \Longrightarrow \frac{\partial \ell}{\partial f(\mathbf{x}_i)} = -y_i \exp\{-f(\mathbf{x}_i) y_i\}.$$

Assume for now that $\sum_i h^2(\mathbf{x}_i) = c$, c a constant. We have

$$\begin{aligned}
\underset{h\in\mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} r_t(\mathbf{x}_i)h(\mathbf{x}_i) &= \underset{h\in\mathcal{H}}{\operatorname{argmin}} -2\sum_{i=1}^{n} t_t(\mathbf{x}_i)h(\mathbf{x}_i), \qquad t_t(\mathbf{x}_i) = -r_t(\mathbf{x}_i) \\
&= \underset{h\in\mathcal{X}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \underbrace{t_t(\mathbf{x}_i)^2}_{\text{ind. of } h} - 2\sum_{i=1}^{n} t_t(\mathbf{x}_i)h(\mathbf{x}_i) + \underbrace{h^2(\mathbf{x}_i)}_{\sum_i h^2(\mathbf{x}_i) = c} \right\} \\
&= \underset{h\in\mathcal{X}}{\operatorname{argmin}} \left\{ t_t(\mathbf{x}_i) - h(\mathbf{x}_i) \right\}^2
\end{aligned}$$

i At stage $t+1$, the optimal h_{t+1} corresponds to a regression tree to fit the negative residuals $t_t(\mathbf{x}_i)$ minimizing the squared loss whatever ℓ is (sub–differentiable and convex though).

Algorithm 11: Gradient boosting.

input : Supervised data set $\mathcal{D}_n = \{(\mathbf{X}_i, Y_i) : i = 1, \ldots, n\}$, loss function ℓ, step size α
output: A (hopefully) strong classifier H

Initialization of strong learner $H \equiv 0$;
for $t \leftarrow 1$ **to** T **do**
    Compute negative residuals

$$t_t(\mathbf{X}_i) = Y_i - H(\mathbf{X}_i)$$

    Find optimal weak learner

$$h_{t+1} = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \{h(\mathbf{X}_i) - t_t(\mathbf{X}_i)\}^2$$

    Update strong learner (gradient descent)

$$H \leftarrow H + \alpha h_{t+1}$$

Return the final learner H;

Why should we have $\sum_i h^2(\mathbf{x}_i) = c$?

- For binary classification problems, we have

$$\sum_{i=1}^{n} h^2(\mathbf{x}_i) = n, \qquad \text{since } h : \mathbb{R}^p \to \{-1, 1\}.$$

- For regression problems, it is typically not the case and tweak the update scheme, i.e., use

$$H \leftarrow H + \tilde{\alpha} \frac{h_{t+1}}{\|h_{t+1}\|}, \qquad \tilde{\alpha} = \|h_{t+1}\| \alpha.$$

> **i** For regression, the update uses the "direction" of the gradient h_{t+1} not its magnitude and that completely makes sense.

Regularization

- Regularization is often used to avoid overfitting
- For gradient boosting several options are possible:
  - reduce the number of weak learners
  - reduce the depth of trees, e.g., use decision stumps
  - limit the minimum number of observations in tree's terminal nodes
  - use stochastic gradient, i.e., use a random sub-sample at each gradient descent step
  - penalize the loss function, i.e.,

$$\ell(H) + \nu P(H), \qquad \nu > 0,$$

where is a penalization term of H, e.g., $P(h)$ is a ℓ_2–penalization and ν controls the amount of penalization.

XGBoost

- XGBoost stands for Extreme gradient boosting
- Provides a scalable, portable and distributed gradient boosting library
- Open–source software
- Python (`scikit-learn`) and R (`caret`) packages are available
- Widely used in `Kaggle` competitions

XGBoost with R

```
> install.packages("xgboost")
> library(xgboost)
> model <- xgboost(data = Xtrain, label = Ytrain, max.depth = 2,
                   nrounds = 500)
> predict(model, Xtest)
```

```
pip install xgboost
from xgboost import XGBRegressor##for regression of course
model = XGBRegressor()
model.fit(Xtrain, Ytrain)
pred = model.predict(Xtest)
```

III. Geostatistics

1. Introduction

Motivation

- Many variables are spatial in extent, e.g., rainfall, petroleum, elevation
- The use of univariate or even multivariate statistical models may be too restrictive.
- An example would be to try to estimate the expected surface of a pollutant exceeding some critical level u_{Craft} in a study region $\mathcal{X} \subset \mathbb{R}^d$, i.e.,

$$\text{Area}(u_{\text{crit}}) = \mathbb{E}\left[\int_{\mathcal{X}} 1_{\{Y(s)>u_{\text{crit}}\}} \text{d}s\right],$$

where $Y(s)$ is the amount of pollutant at location s.

The use of univariate models may still be useful provided the focus is on pointwise quantities, e.g., quantiles at $s_* \in \mathcal{X}$.

Different type of spatial data

- geostatistical data: data are defined continuously on $\mathcal{X}$, e.g., rainfall;
- punctual data: the data are points falling randomly over some space $\mathcal{X}$, e.g., tree locations.
- lattice data: data are aggregated over sub-regions, e.g., number of citizen in counties.

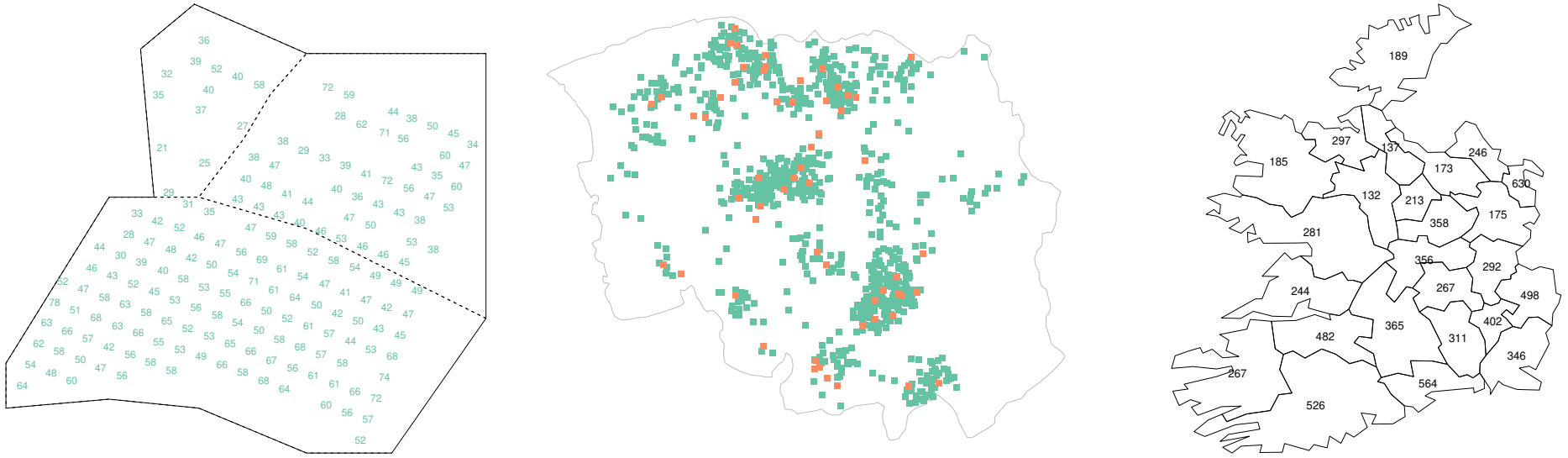
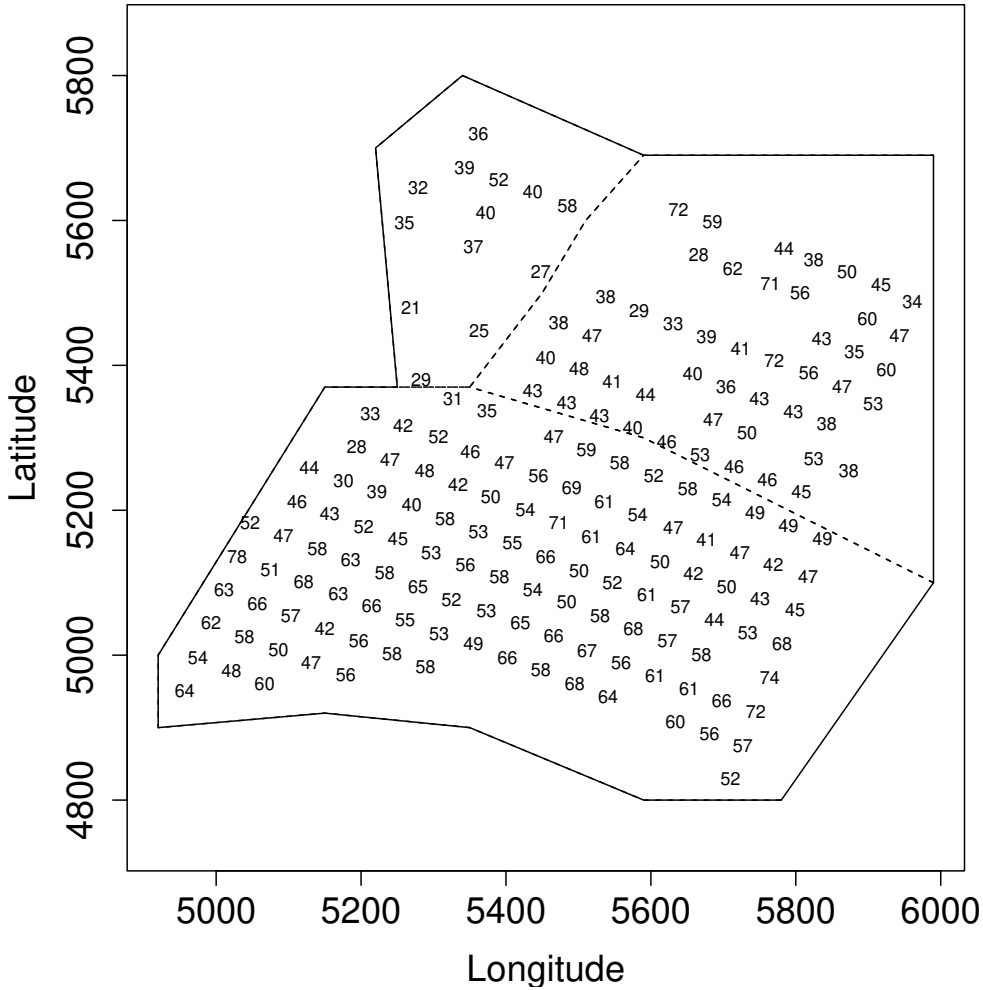


Figure 26: *The three different type of spatial data. From left to right: geostatistical (Calcium concentration), punctual (location of lung and larynx) and lattice data.*

The Ca_{20} data set



Focus is on geostatiscal data only!

2. Framework

Definition 11. A stochastic process defined on $\mathcal{X}$ is a collection of random variables indexed by $\mathcal{X}$ on the same probability space $(\Omega, \mathcal{F}, \Pr)$.

Proposition 3. *A stochastic process $\{Y(s)\colon s \in \mathcal{X}\}$ is completely characterised from its finite dimensional distribution functions, i.e., for any $k \geq 1$ and $s_1, \ldots, s_k \in \mathcal{X}$*

$$\Pr\{Y(s_1) \leq A_1, \ldots, Y(s_k) \leq A_k\}, \qquad A_1, \ldots, A_k \text{ Borel sets},$$

(provided they satisfy the hypothesis of the Kolmogorov extension theorem, i.e., invariance to permutation and consistent marginalisation)

Strictly stationary processes

Definition 12. A stochastic process $\{Y(s)\colon s \in \mathcal{X}\}$ is said (strictly) stationary if its finite dimensional distribution functions are invariant by translation, i.e., for any $k \geq 1$, $s_1, \ldots, s_k \in \mathcal{X}$ and $h \in \mathcal{X}$ we have

$$\Pr\{Y(s_1+h) \leq A_1, \ldots, Y(s_k+h) \leq A_k\} = \Pr\{Y(s_1) \leq A_1, \ldots, Y(s_k) \leq A_k\},$$

where A_j are Borel sets.

☞ In practice, strict stationarity is too strong and cannot be checked. Need a weaker hypothesis.

Definition 13. A second order stochastic process is a stochastic process whose second order moment exists, i.e., $\text{Var}[Y(s)] < \infty$ for all $s \in \mathcal{X}$.

- Working with second order processes allows to define
  - the mean function / trend / drift

$$\begin{aligned} \mu : \mathcal{X} &\longrightarrow \mathbb{R} \\ s &\longmapsto \mathbb{E}[Y(s)], \end{aligned}$$

  - the covariance function

$$\begin{aligned} K : \mathcal{X} \times \mathcal{X} &\longrightarrow \mathbb{R} \\ (s, s') &\longmapsto \text{Cov}\{Y(s), Y(s')\}. \end{aligned}$$

Definition 14. A second order process is said weakly stationary, or just stationary, if for any $s, s' \in \mathcal{X}$ and $h \in \mathcal{X}$ we have

$$\mu(s+h) = \mu(s), \qquad K(s+h, s'+h) = K(s, s'). \qquad \texttt{(translation invariance)}$$

Definition 15. A stochastic process $\{Y(s) \colon s \in \mathcal{X}\}$ is said isotropic if for any rotation matrix R, i.e., $|R| = 1$ and $R^{-1} = R^T$, we have

$$\{Y(Rs) \colon s \in \mathcal{X}\} \overset{\mathsf{d}}{=} \{Y(s) \colon s \in \mathcal{X}\}. \qquad \texttt{(rotation invariance)}$$

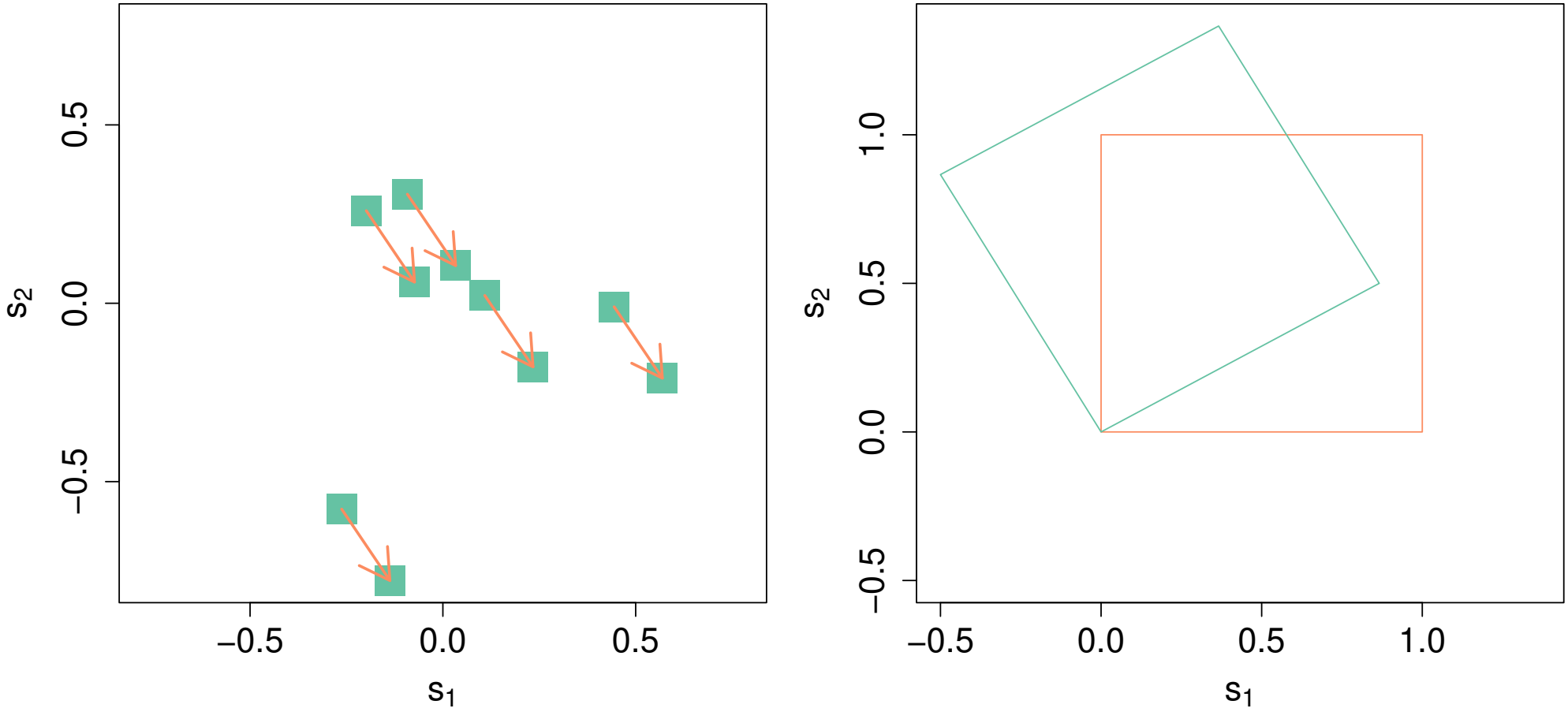


Figure 27: *Illustration of stationarity and isotropy.*

Consequences

- If a process is stationary we have

$$K(s, s') = K(o, s' - s) = K(h),$$

where $h = s - s'$ and is even since

$$\text{Cov}\{Y(s), Y(s')\} = \text{Cov}\{Y(s'), Y(s)\}.$$

- If we further assume isotropy, the covariance function now satisfies

$$\begin{aligned} K(Rh) &= K(h) \\ &= K(\|h\|) \\ &= K(-\|h\|). \end{aligned}$$

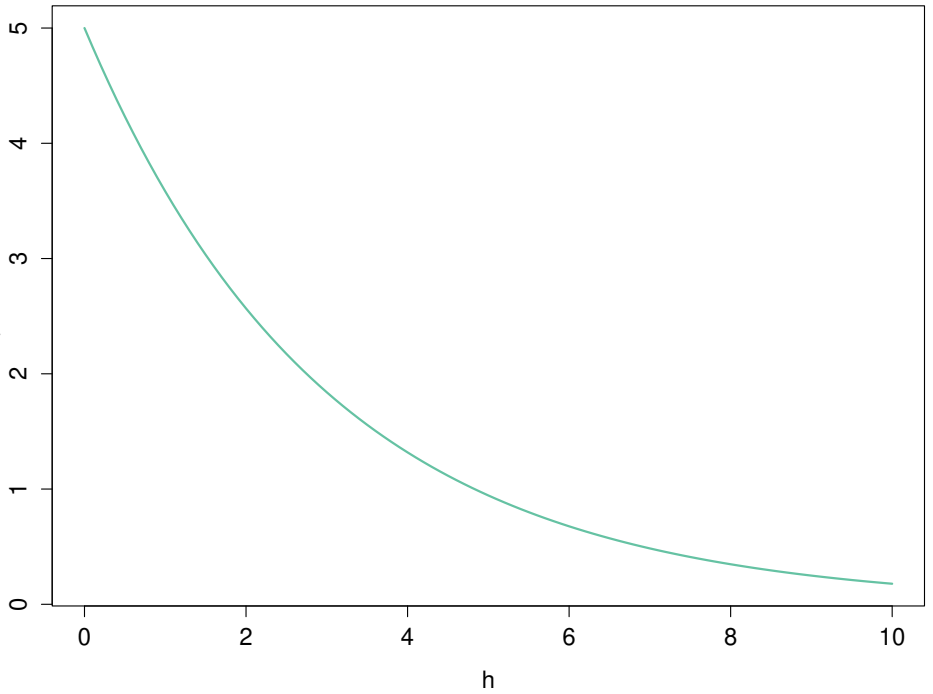


Figure 28: *Plot of a stationary isotropic covariance function. What is $K(0)$?*

Definition 16. A stochastic process $\{Y(s) : s \in \mathcal{X}\}$ is said to have stationary increments if for all $s \in \mathcal{X}$ and $h \in \mathcal{X}$, the distribution of

$$Y(s+h) - Y(s) \overset{\text{fin}}{=} Y(h) - Y(o),$$

i.e., depends only on the lag h and where $o \in \mathcal{X}$ is an arbitrary origin.

- The motivation for using stationary increments processes is that we are no longer restricted to stationary processes.
- We can even work with non second order processes and simply assume

$$\text{Var}[Y(h) - Y(o)] < \infty.$$

Example 1. Consider the following random walk defined on $\mathcal{X} = \mathbb{Z}$

$$Y(s+1) = Y(s) + \varepsilon_{s+1}, \qquad \varepsilon_j \overset{\text{iid}}{\sim} N(0, \sigma^2).$$

It has indeed stationary increments since

$$Y(s+h) - Y(s) = \sum_{j=0}^{h-1} \underbrace{\{Y(s+h-j) - Y(s+h-j-1)\}}_{=\varepsilon(s+h-j)} = \sum_{j=0}^{h-1} \varepsilon_{s+h-j} \sim N(0, h\sigma^2).$$

but is not stationary. Even worse we have $\text{Var}\{Y(s)\} \to \infty$ as $s \to \infty$.

> **i** Extension of the above random walk to $\mathcal{X} = \mathbb{R}^d$ leads to the so-called Brownian random fields. If we further assume dependence across increments we get fractional Brownian processes.

Semi–variogram

- The covariance function is a summary statistic of the spatial dependence function for at most second order processes.
- To get an analogue for stationary increment processes we rather consider the semi–variogram

$$\gamma(h) = \frac{1}{2}\mathsf{Var}[Y(h) - Y(o)] = \frac{1}{2}\mathbb{E}\left[\{Y(h) - Y(o)\}^2\right].$$

If the process is indeed second order we have

$$\gamma(h) = \frac{1}{2}\{2K(o,o) - 2K(o,h)\} = K(o,o)\{1 - \rho(h)\},$$

where $h \mapsto \rho(h)$ is the correlation function and

$$\gamma(h) \longrightarrow K(o,o), \qquad \|h\| \to \infty, \quad \texttt{(as long as } \rho(h) \to 0\texttt{)}$$

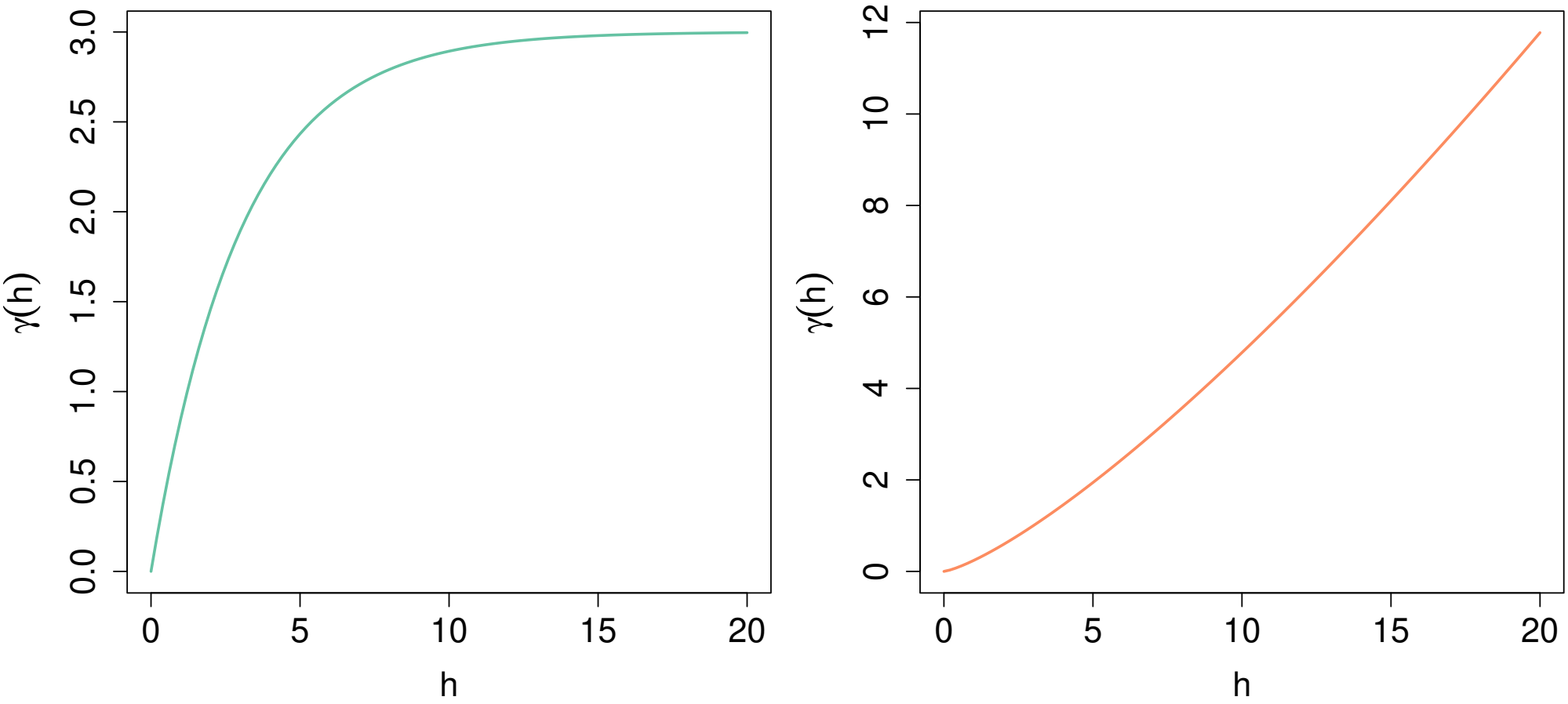


Figure 29: *Bounded (left) and unbounded (right) semi–variograms. If it exists, what is $\gamma(\infty)$?*

Some isotropic stationary correlation functions and variograms

Family	$\rho(h)$	$\gamma(h)$	Support
Exponential	$\exp(-h/\lambda)$	$1-\exp(-h/\lambda)$	$\lambda>0$
Gaussian	$\exp\left\{-(h/\lambda)^2\right\}$	$1-\exp\left\{-(h/\lambda)^2\right\}$	$\lambda>0$
Stable / Powered exponential	$\exp\left\{-(h/\lambda)^\kappa\right\}$	$1-\exp\left\{-(h/\lambda)^\kappa\right\}$	$\lambda>0, 0\leq\kappa\leq 2$
Whittle–Matérn	$\frac{2^{1-\kappa}}{\Gamma(\kappa)}\left(\frac{u}{\lambda}\right)^\kappa K_\kappa\left(\frac{u}{\lambda}\right)$	$1-\frac{2^{1-\kappa}}{\Gamma(\kappa)}\left(\frac{u}{\lambda}\right)^\kappa K_\kappa\left(\frac{u}{\lambda}\right)$	$\lambda>0, \kappa>0$
Fractional	—	$(h/\lambda)^\kappa$	$0\leq\kappa\leq 2$

- The parameters λ and κ are known as the range and smooth parameters.
- Associated covariance functions are derived using a sill parameter τ, i.e.,

$$K(h)=\tau\rho(h), \qquad \tau>0. \qquad (\tau=\mathtt{K(o)})$$

- The smooth and range parameters drives respectively the smoothness of the random process and the range of spatial dependence.

The practical range h_p is the distance such that $\rho(h_p)=0.05$.

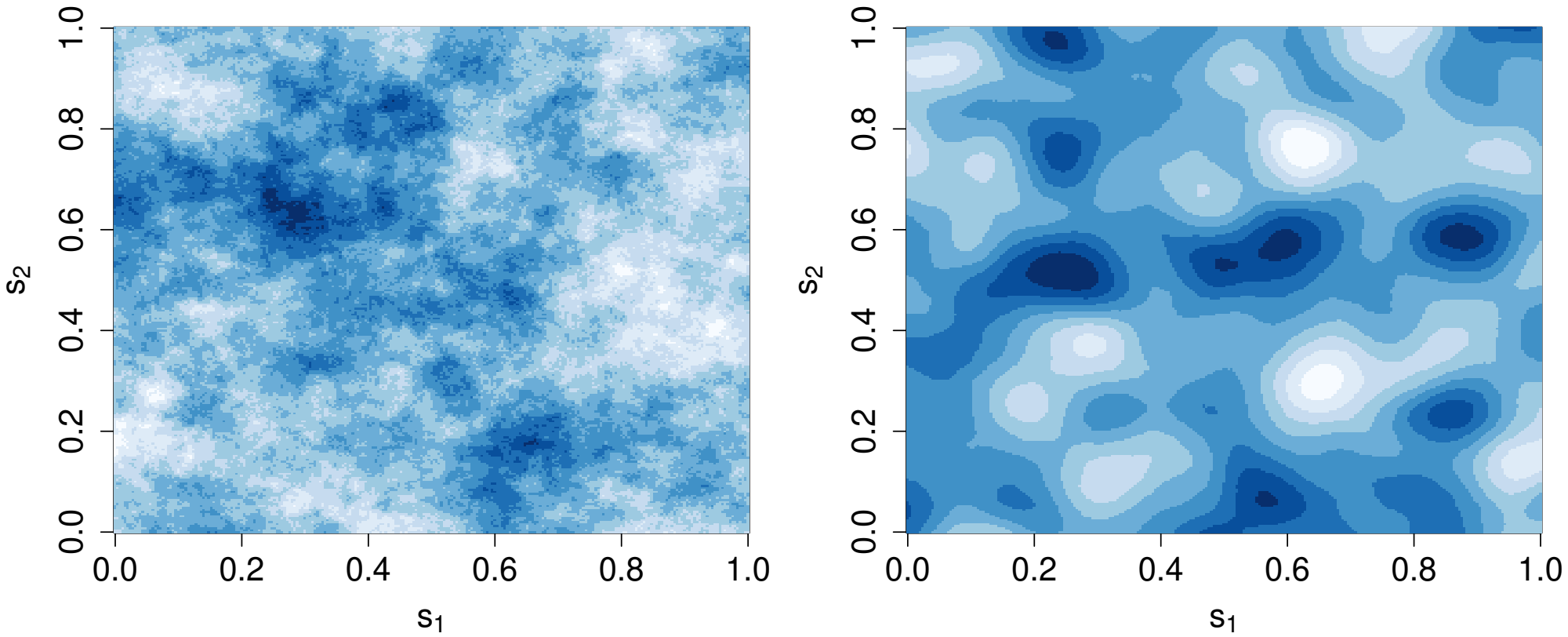
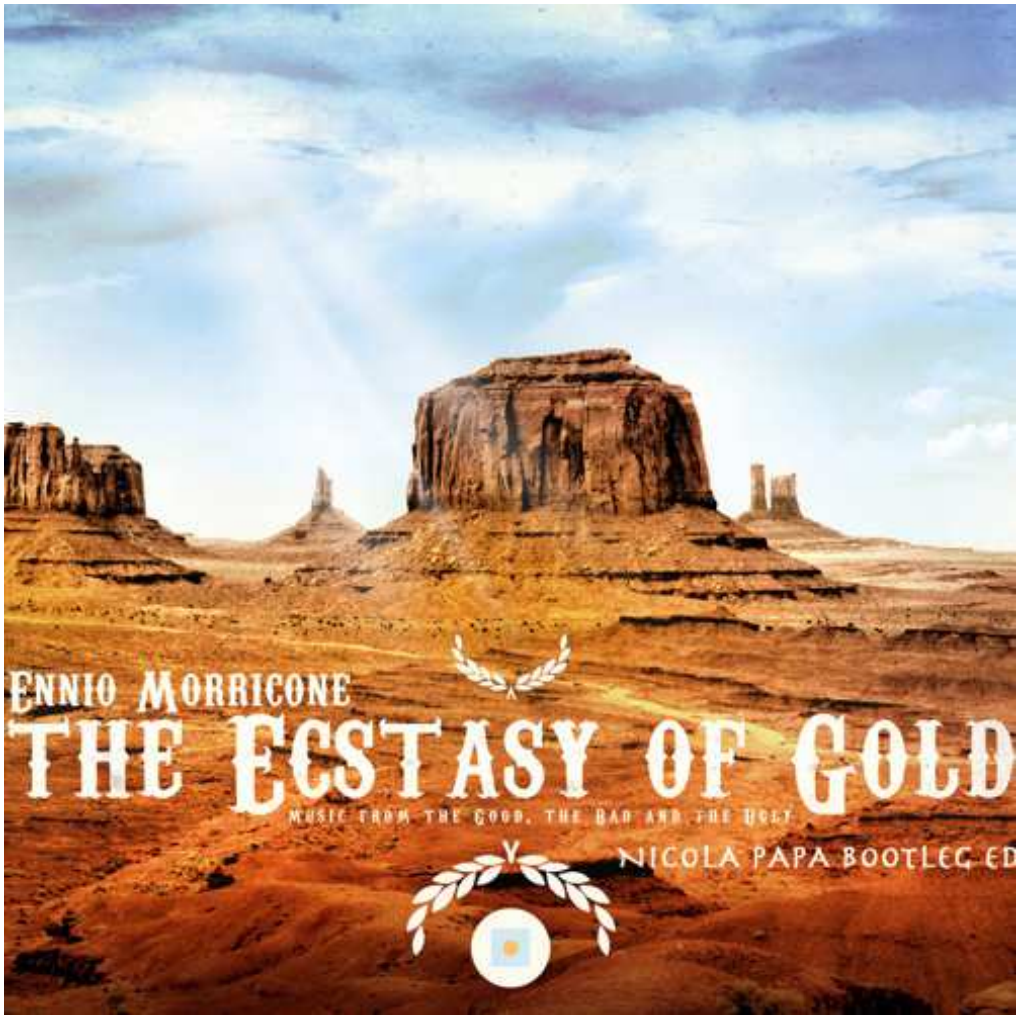


Figure 30: *Two realisations of a random fields with a powered exponential correlation function. Left:* $\kappa = 1$. *Right:* $\kappa = 2$.


ENNIO MORRICONE
THE ECSTASY OF GOLD
MUSIC FROM THE GOOD, THE BAD AND THE UGLY
NICOLA PAPA BOOTLEG ED

- The covariance function may have a discontinuity at the origin, called nugget effect, i.e.,

$$K(h) = \begin{cases} \eta + \tau, & h = 0, \\ \tau\rho(h), & h > 0. \end{cases}$$

- The nugget effect may have two possible interpretations:
  - error in measurements, i.e., $Y(s) = S(s) + \varepsilon(s)$
  - spatial variation on a scale smaller than the minimum distance between measurements (if no replicate)

Proposition 4. *If a correlation of a stationary process is discontinuous, then discontinuity has to be at the origin.*
If a stationary process has a correlation function which is continuous (at the origin) then it is continuous and if twice differentiable, the process is differentiable (both from a L^2 sense).

Extension to higher orders are possible!

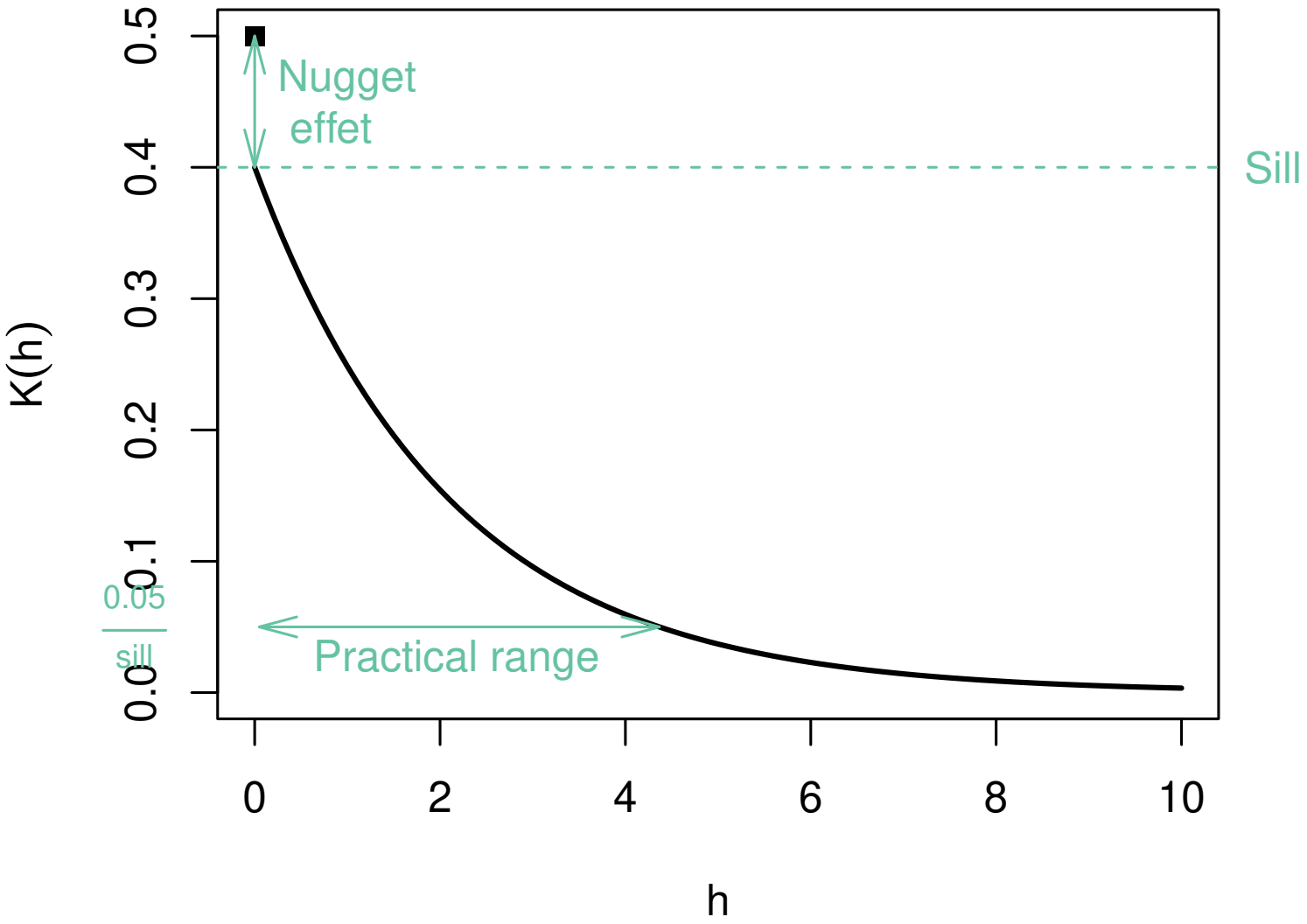


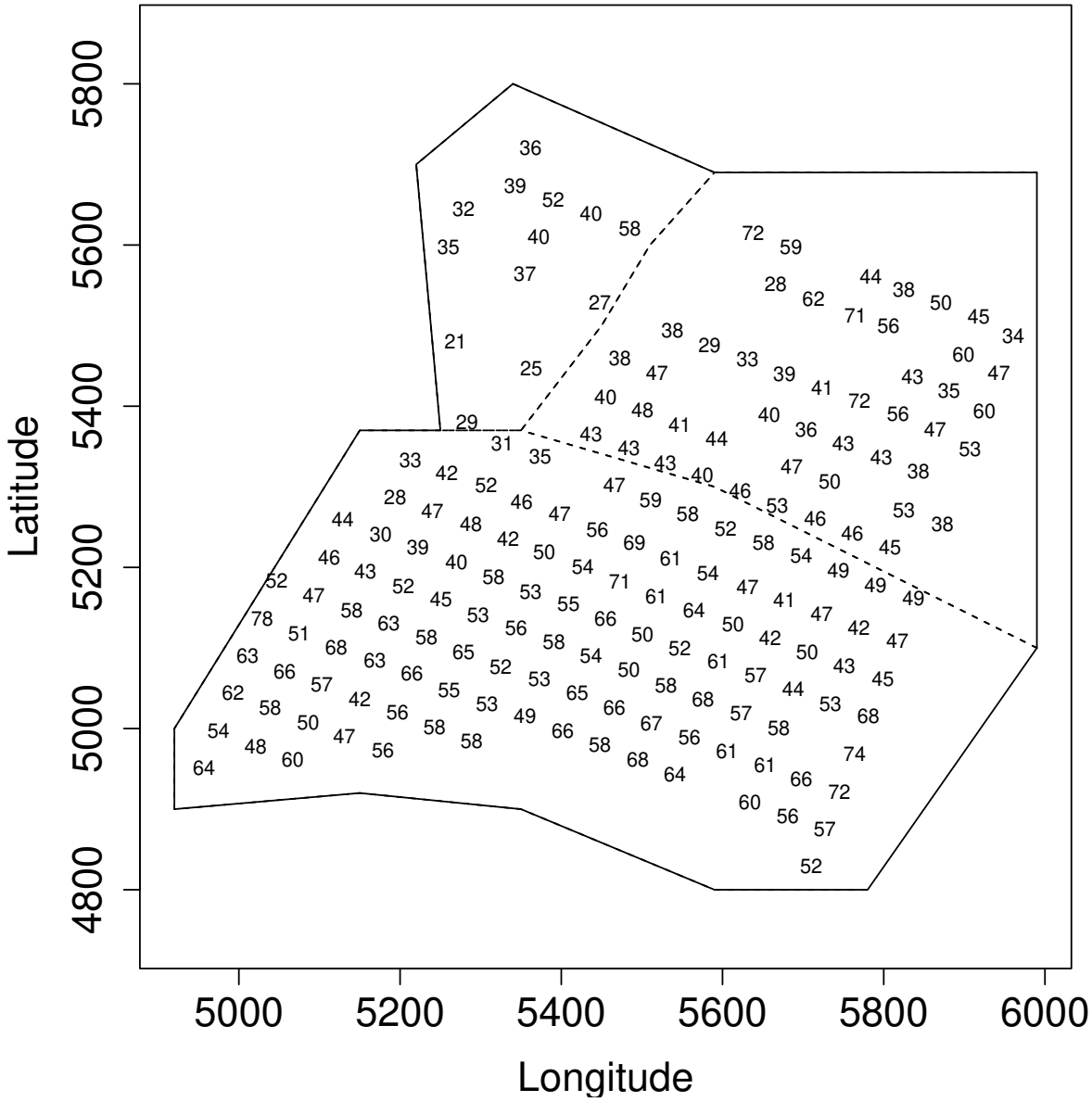
Figure 31: *Illustration of the nugget effect, the sill parameter and the practical range.*

3. Inference

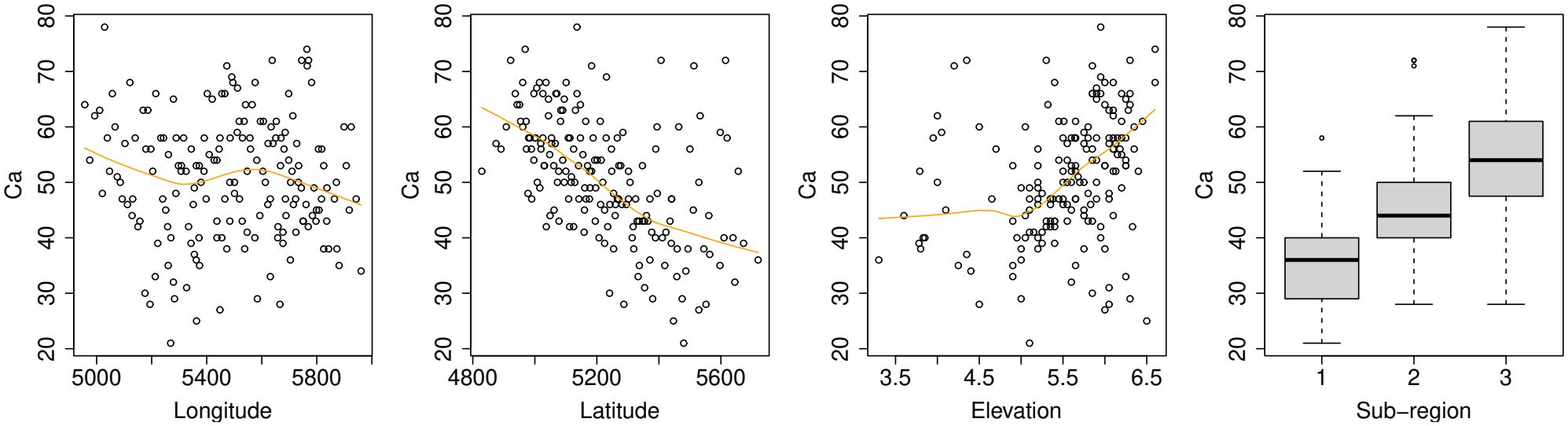
Descriptive analysis

- Before trying to model the data we need to check whether the data can safely be assumed stationary / isotropic / ...
- Essentially we start with a descriptive analysis which, four our context, consists in
  - checking for any trend in the mean function $s \mapsto \mu(s)$
  - inspecting the semi–variogram.
- The first stage is very simple. Just plot data w.r.t. some covariates, e.g., longitude, latitude, ...

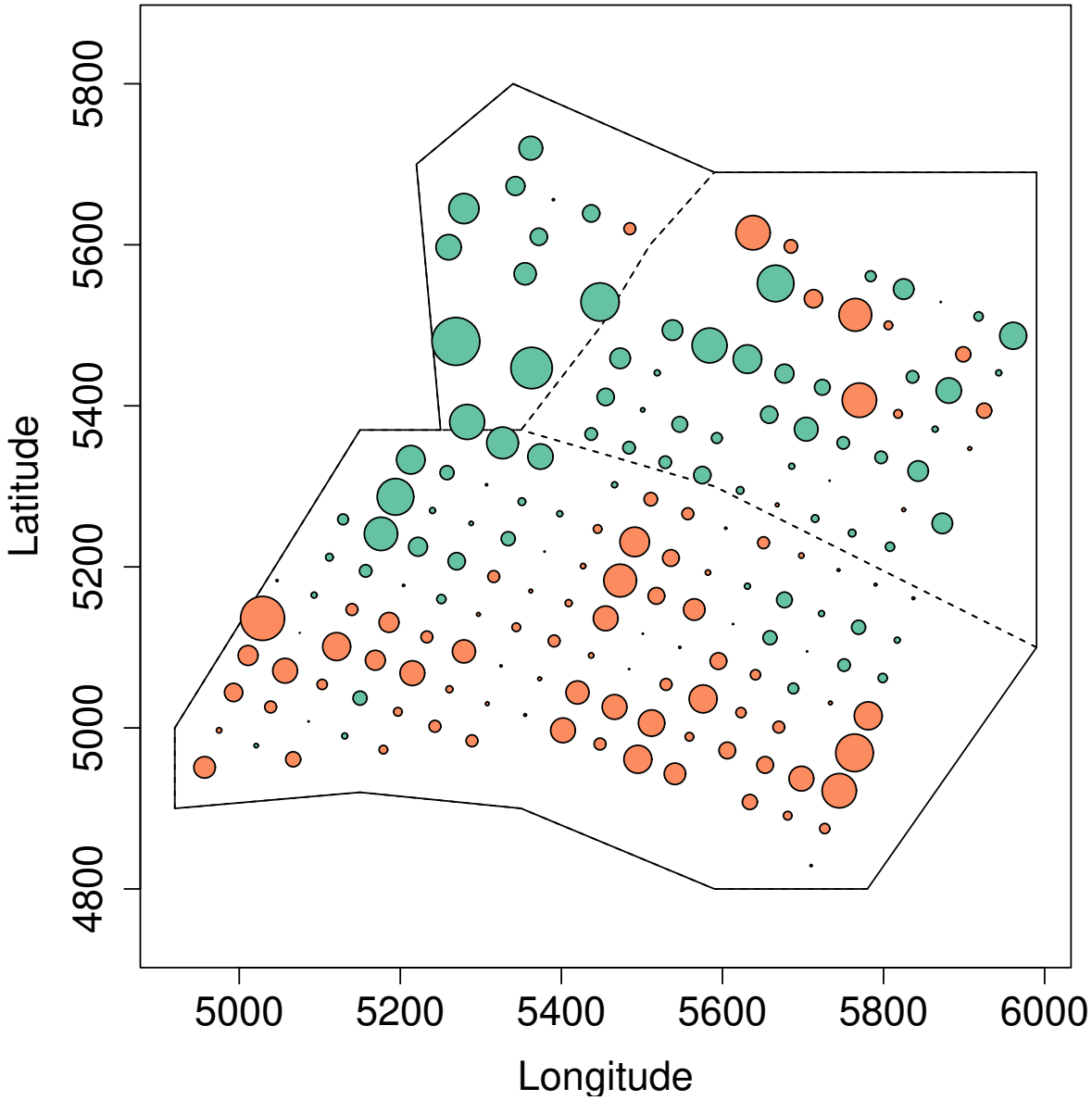
Ca20 data set.



Ca20 data set.



Ca20 data set.



Empirical variograms

- Given some data $\mathcal{D}_n = \{Y_i(s_j) \colon i = 1, \ldots, n,\ j = 1, \ldots, k\}$, we easily estimate the semi–variogram

$$\hat{\gamma}(h_{j,\ell}) = \frac{1}{2n} \sum_{i=1}^{n} \{Y_i(s_j) - Y_i(s_\ell)\}^2, \qquad h_{j,\ell} = \|s_j - s_\ell\|.$$

- We may have $n = 1$ so that the above estimator has huge variance and we rather use a binned version, i.e.,

$$\tilde{\gamma}(h_b) = \frac{1}{2|B_b|} \sum_{i=1}^{n} \sum_{j,\ell=1}^{k} \{Y_i(s_j) - Y_i(s_\ell)\}^2 1_{\{\|s_j - s_\ell\| \in B_b\}},$$

where $\{B_b \colon b = 1, \ldots, B\}$ is a partition of $(0, \max h_{j,\ell})$ and h_b is the centroid of B_b.

The binned estimator is however biased but has a (much) lower variance.

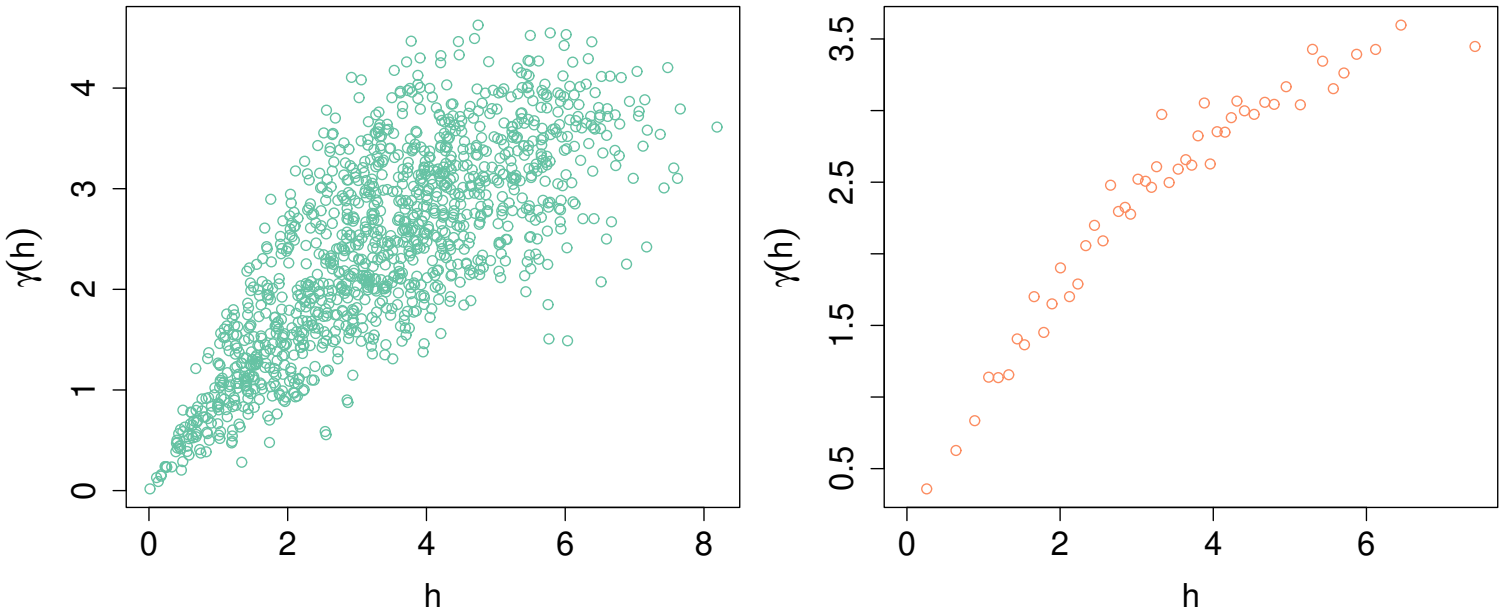


Figure 32: *Empirical variograms. Left: raw. Right: binned.*

The above estimator makes sense only if your data can be considered as stationary or at least with stationary increments.

You may want to remove any possible trends (using a linear model for instance) and estimate the variogram on the residuals.

Least square fitting of a variogram

- Suppose we have fitted a mean function, e.g., from linear models.
- We can fit any parametric variogram $\gamma(\cdot;\psi)$ minimizing using the (weighted) least square estimator on the empirical variogram $\hat{\gamma}$, i.e.,

$$\hat{\psi} = \arg\min_{\psi\in\Psi} \sum_{j,\ell} \omega_{j,\ell} \left\{\hat{\gamma}(h_{j,\ell}) - \gamma(h_{j,\ell};\psi)\right\}^2 .$$

- The two fitted quantities are all we need to enable predictions!

Nasty optimization problem: use several initial values!
Often fix the smooth parameter to some values, e.g., $\kappa = 0.25, 0.5, \ldots, 2$.
Always question yourself if a nugget effect makes sense.

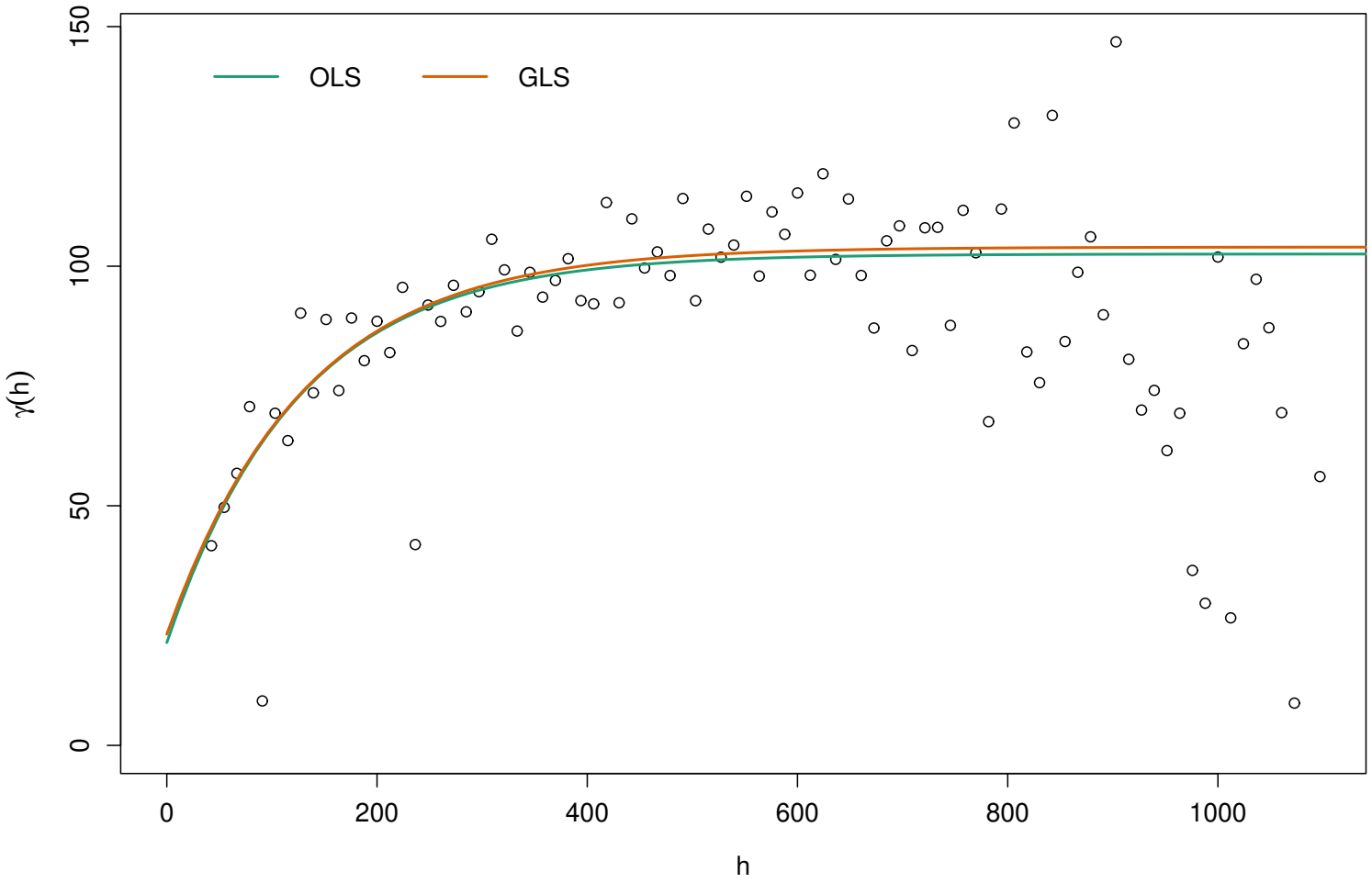
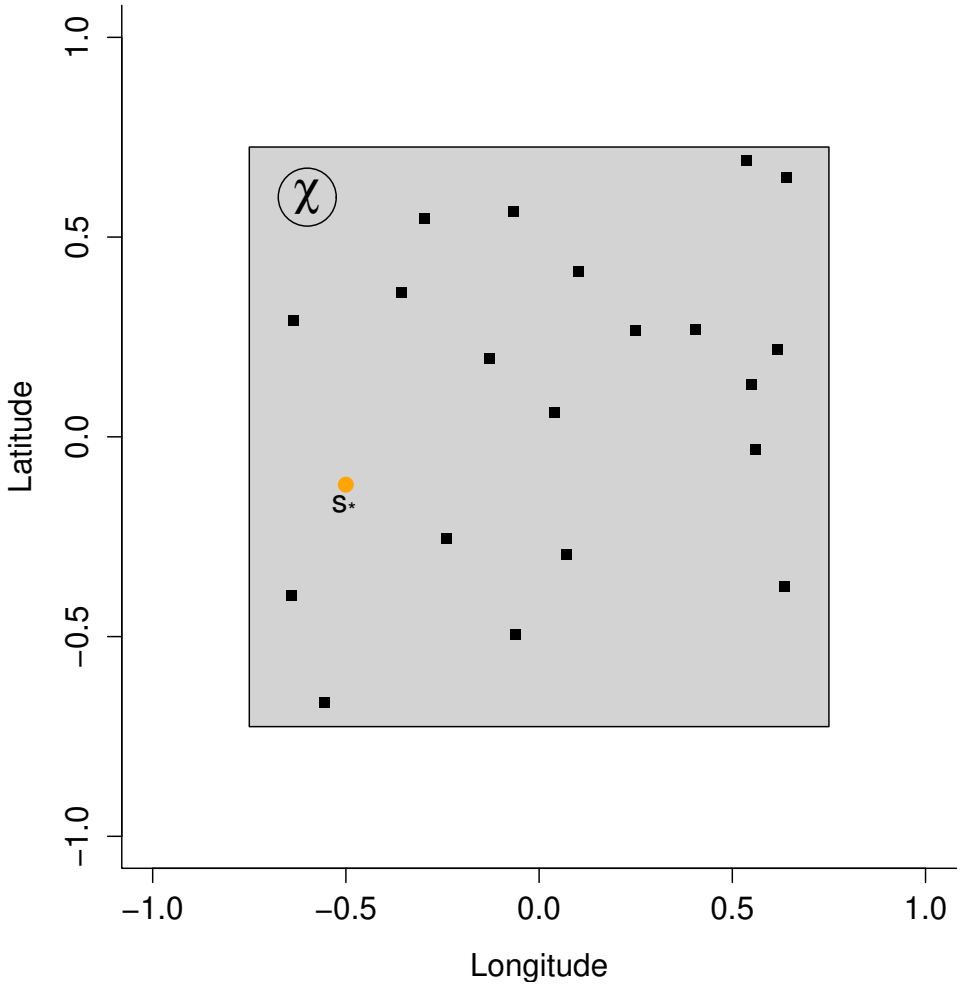


Figure 33: *Least square fitting of a parametric variogram on the Calcium data set.*



- Prediction of $Y(s_*)$ based on observations $Y(s_1), \ldots, Y(s_k)$.
- Restriction to unbiased linear estimators, i.e.,

$$\hat{Y}(s_*) = \sum_{j=1}^{k} \lambda_j Y(s_j),$$

with $\mathbb{E}[\hat{Y}(s_*)] = \mu(s_*)$.

- Estimator is the one minimizing the mean squared error, i.e.,

$$\hat{Y}(s_*) = \arg\min_T \mathbb{E}\left[\{T - Y(s_*)\}^2\right].$$

(Universal) Kriging

- There are several of Kriging:

  Simple $\mu(s) \equiv 0$
  Ordinary $\mu(s) = m$, m unknown parameter
  Universal $\mu(s) = \mathbf{x}(s)^\top \boldsymbol{\beta}$,$\boldsymbol{\beta}$ unknown parameter, $\mathbf{x}(s)$ vector of covariates,
  Co-kriging Y is multivariate

  and their intrinsic counterpart.
- Explicit expressions for $\hat{Y}(s_*)$ are available but not given here (nasty).
- We can also get expression for the kriging variance, i.e.,

$$\text{Var}\left\{\hat{Y}(s_*) - Y(s_*)\right\}.$$

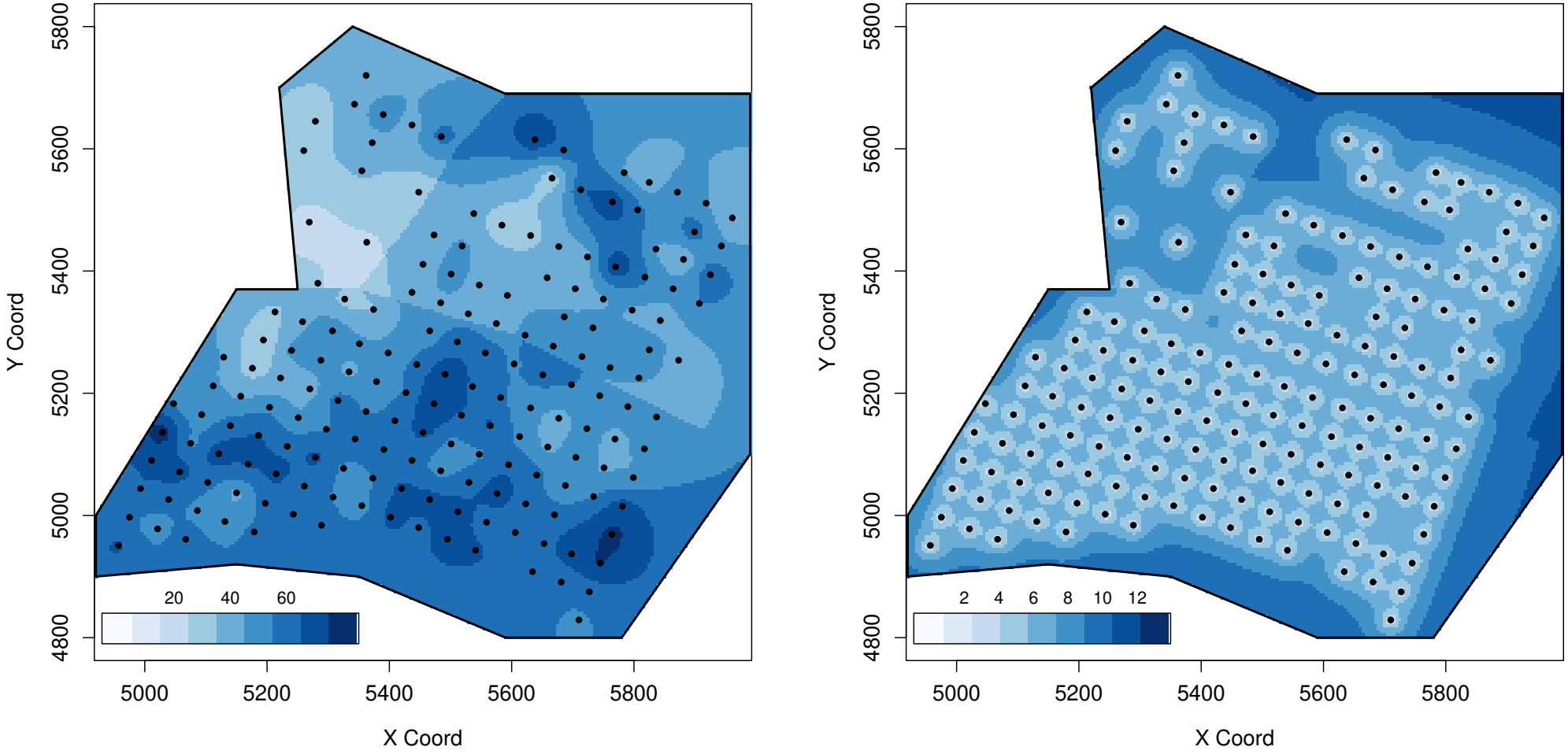


Figure 34: *Kriging estimator (left) and kriging standard error (right) for the Ca_{20} data set.*

4. Model–based geostatistics

Definition 17. The multivariate Gaussian distribution defined on $\mathbb{R}^d$, $d \geq 1$, has probability density function

$$f(\mathbf{y}) = (2\pi)^{-d/2}|\Sigma|^{-1/2}\exp\left\{-\frac{1}{2}(\mathbf{y}-\mu)^\top\Sigma^{-1}(\mathbf{y}-\mu)\right\}, \qquad \mathbf{y}\in\mathbb{R}^d, \tag{3}$$

where $\mu \in \mathbb{R}^d$ is the mean vector and $\Sigma \in M_d(\mathbb{R})$ is the covariance matrix.

The Mahalanobis distance is given by

$$a^2(\mathbf{y}) = (\mathbf{y}-\mu)^\top\Sigma^{-1}(\mathbf{y}-\mu)$$

Definition 18. A Gaussian process $\{Y(s) : s \in \mathcal{X}\}$ is a stochastic process whose finite dimensional distribution functions are multivariate Gaussian.

Proposition 5. *A Gaussian process is completely characterized through its mean function and covariance function.*

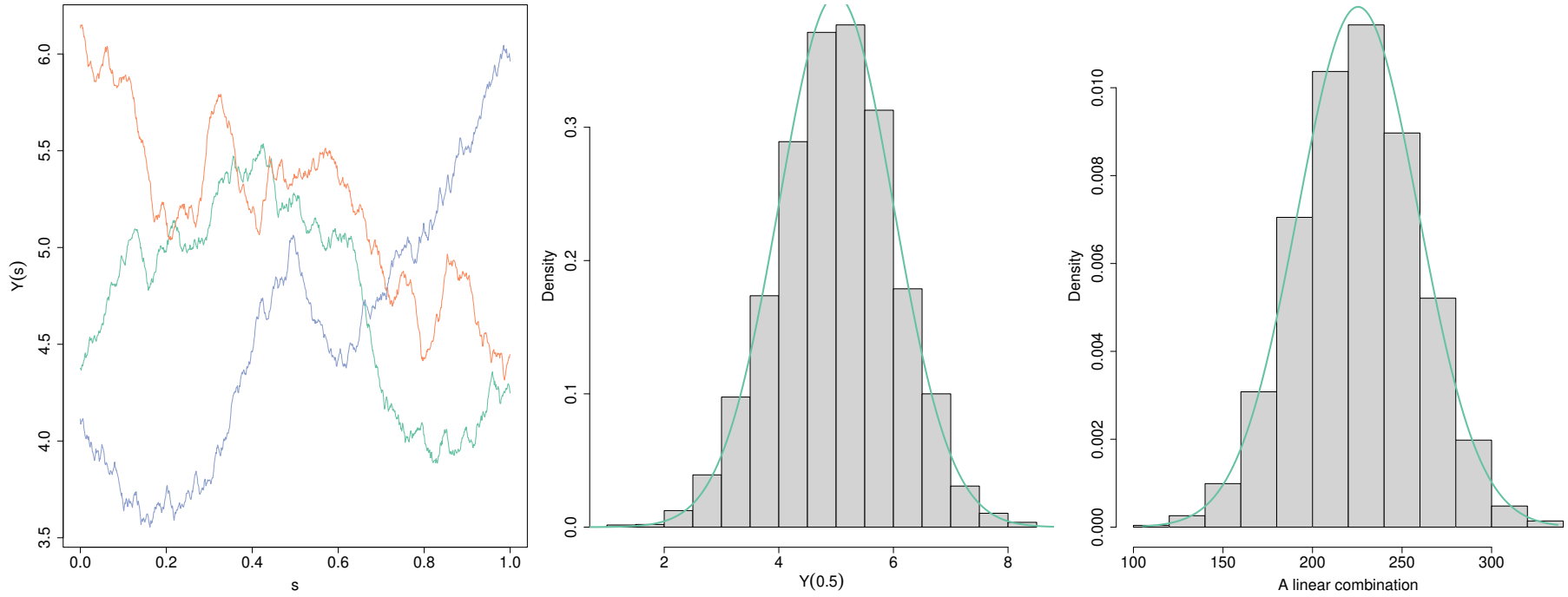


Figure 35: *Numerical illustration of a Gaussian process.*

(semi) Definite positive functions

Definition 19. A function $f : \mathbf{s} \in \mathbb{R}^d \mapsto f(\mathbf{s})$ is said to be (semi) definite positive if it is symmetric and

$$\boldsymbol{\lambda}^\top A \boldsymbol{\lambda} > 0, \qquad A = (a_{i,j} = f(s_i - s_j) : i, j = 1, \ldots, d), \quad x_1, \ldots, x_p \in \mathbb{R}^d,$$

for any non–zero vector $\boldsymbol{\lambda} \in \mathbb{R}^p$. It is semi definite positive if the above inequality is not strict.

The covariance function γ is (semi) definite positive to ensure that the Mahalanobis distance

$$a^2(\mathbf{s}, \mathbf{y}) = (\mathbf{y} - \mu)^\top \Sigma^{-1}(\mathbf{s})(\mathbf{y} - \mu), \qquad \Sigma(\mathbf{s}) = \{\sigma_{i,j} = \gamma(s_i, s_j)\}$$

is always positive and the multivariate Gaussian density is properly defined.

Fitting a Gaussian process

- Having observed n independent observations at k spatial locations, i.e., $\mathcal{D}_n = \{y_i(s_j) \colon i = 1, \ldots, n, j = 1, \ldots, k\}$ $s_1, \ldots, s_k$, we define the log–likelihood as

$$\ell(\mu, \gamma; \mathcal{D}_n) = -\frac{nd}{2}\log(2\pi) - \frac{n}{2}\log|\Sigma(\mathbf{s})| - \frac{1}{2}\sum_{i=1}^{n} a^2(\mathbf{s}, \mathbf{y}_i).$$

💔 no likelihood theory here, but Gaussian processes can (easily) be estimated by maximizing the log–likelihood.

Parametric assumptions

- The above likelihood has some flaws:
  - it has $d + d(d+1)/2$ parameters to estimate which is typically too large;
  - cannot enable prediction at a new location s_* since both mean and covariance function cannot be computed at s_*.
- Hence we further place some parametric structures on
  - the mean function $\mu(s)$, e.g.,

$$\mu(s; \boldsymbol{\beta}) = \mathbf{x}(s)^\top \boldsymbol{\beta},$$

    where $\mathbf{x}(s)$ is a vector of additional covariates at s and $\boldsymbol{\beta}$ a parameter vector to be estimated.
  - the covariance function $\gamma(s, s') = \gamma(s, s'; \psi)$ using some prescribed parametric expressions as the ones presented earlier.

Non isotropic/stationary covariance functions

- Defining non isotropic / stationary covariance functions is a current research field and is far from being trivial.
- A quick and dirty way to get non isotropic covariance functions is to use any isotropic correlation function on a transformed space $\mathcal{X}'$ given by

$$\begin{aligned} \phi\colon \mathcal{X} &\longrightarrow \mathcal{X}' \\ s &\longmapsto \phi(s;\kappa), \end{aligned}$$

  for some prescribed parametric one–one mapping $\phi(\cdot;\kappa)$.
- A specific case, known as geometric anisotropy, is to set

$$\phi(s;\kappa) = C(\kappa)s, \qquad C(\kappa) = \begin{bmatrix} \cos\kappa_1 & -\sin\kappa_1 \\ \sin\kappa_1 & \cos\kappa_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \kappa_2^{-1} \end{bmatrix},$$

  κ_1, κ_2 are respectively the anisotropy angle and ratio.

Interpolation

- As usual the best estimator we can reached (in a L^2 sense) is the conditional expectation, i.e.,

$$\hat{Y}(s_*) = \mathbb{E}\left\{Y(s_*) \mid Y(s_1), \ldots, Y(s_k)\right\}.$$

- For the Gaussian case, the conditional expectation is linear in the $Y(s_j)$.
- Hence the above estimator is actually the kriging estimator!

🔊 You will sometimes hear: "kriging is the optimal estimator" in a L^2 sense. It is wrong unless if we assume Gaussian. However it is indeed optimal if we restrict to linear unbiased estimators.

By the way...

- What if my data are not Gaussian, e.g., rainfall amount.
- A quick and dirty way is to work on a transformation of your data, e.g., $\log Y(s)$, so that Gaussian is a sensible choice.
- One widely used choice for positive variable is the Box–Cox transformation

$$y \longmapsto \begin{cases} \frac{y^\lambda - 1}{\lambda}, & \lambda \neq 0 \\ \log y, & \lambda = 0. \end{cases}$$

- However it implicitly assumes that the data are stationary so you need to remove any trend first to estimate the shape parameter λ in the Box–Cox transformation.


Tu bluffes Martoni, il bluffe

5. Simulation

(Unconditional) Simulations

- It is rather straightforward to simulated Gaussian process at a moderate number of locations, e.g., $k \leq 3000$, from the Cholesky decomposition of the covariance matrix.
- More precisely for any $\mathbf{s} = (s_1, \ldots, s_k) \in \mathcal{X}$, we have

$$Y(\mathbf{s}) \overset{\mathsf{d}}{=} \mu(\mathbf{s}) + C(\mathbf{s})^\top \boldsymbol{\varepsilon}, \qquad \Sigma(\mathbf{s}) = C(\mathbf{s})^\top C(\mathbf{s}),$$

where $\boldsymbol{\varepsilon}$ is a vector of k independent standard normal random variables.

More sophisticated techniques, e.g., turning bands, circulant embedding methods, exist to get faster simulations on large (gridded) number of locations.

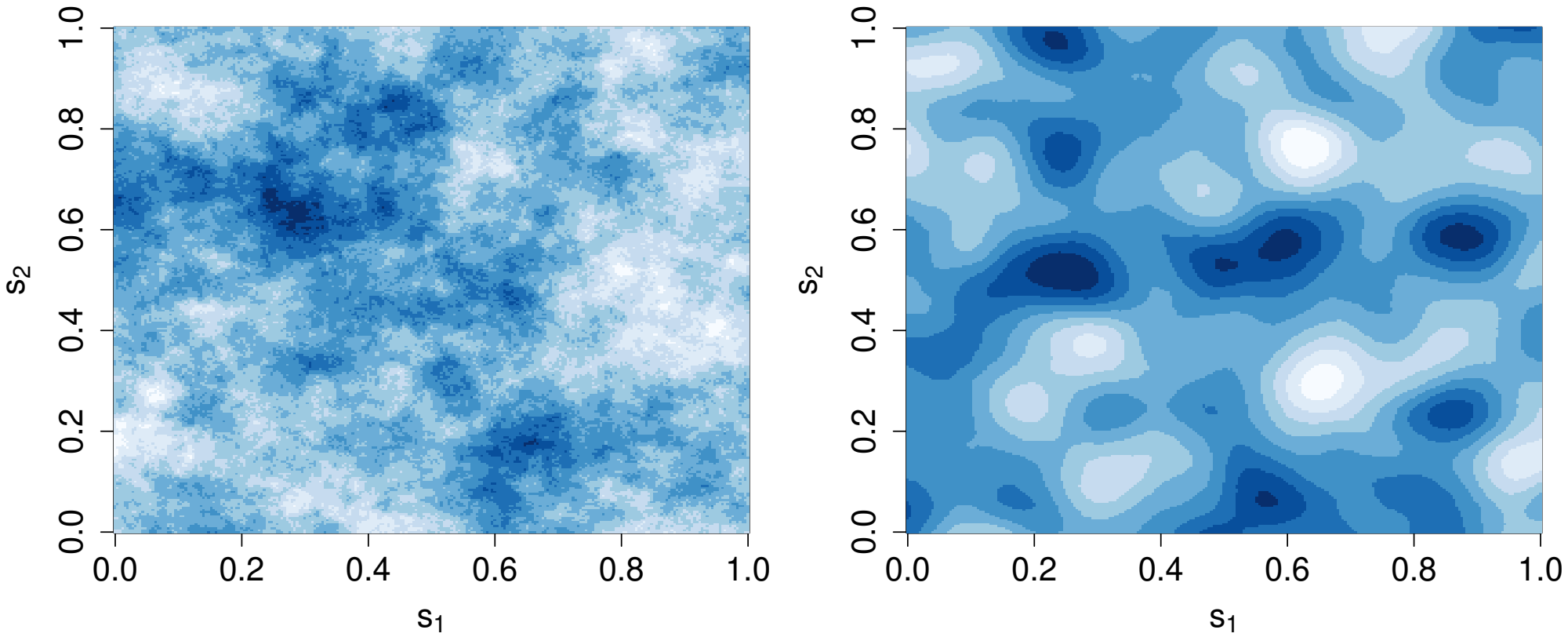


Figure 36: *Two realizations of a random fields with a powered exponential correlation function. Left:* $\kappa = 1$. *Right:* $\kappa = 2$.

Conditional simulations

- Estimating areal quantities from kriging may be too smooth.
- Conditional simulations can be used to get Monte Carlo estimate (and thus the entire distribution) of it.
- Conditional simulations are random simulations that honors some constraints, e.g., simulating from

$$Y(s_*) \mid Y(\mathbf{s}) = \mathbf{y},$$

  where $\mathbf{y}$ is the vector of held fixed values at prescribed location $\mathbf{s}$.
- Under the Gaussian setting, one can use the decomposition

$$Y(s_*) \mid \{Y(\mathbf{s}) = \mathbf{y}\} \overset{\mathsf{d}}{=} \underbrace{Y_{\mathsf{krig}}(s_*)}_{\text{kriging of } Y} + \tilde{Y}(s_*) - \underbrace{\tilde{Y}_{\mathsf{krig}}(s_*)}_{\text{kriging of } \tilde{Y}},$$

  where $\tilde{Y}$ is an independent copy of Y.

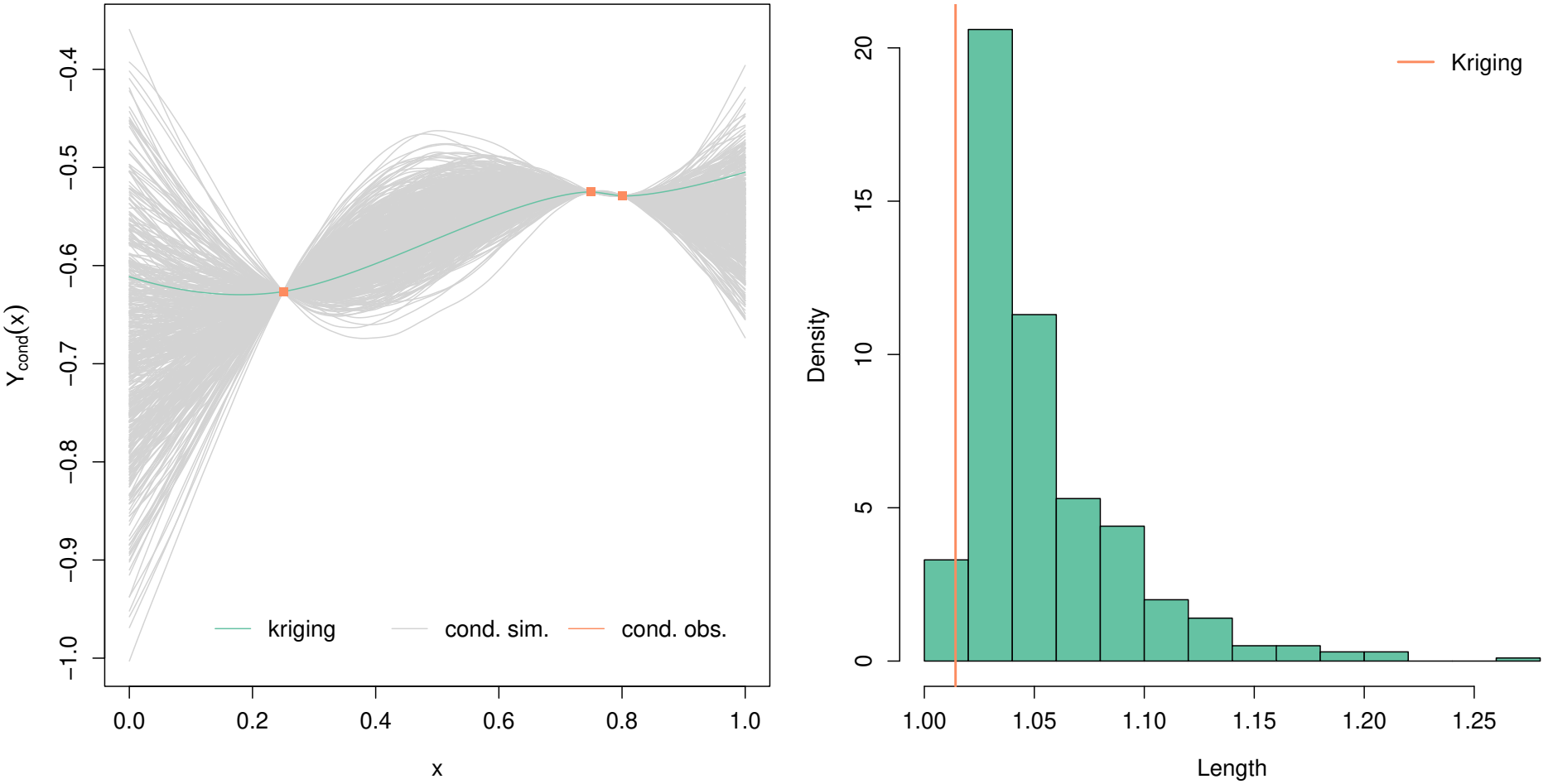


Figure 37: *Comparison between conditional simulations and kriging. Right: length of the curve estimated from kriging and conditional simulations.*

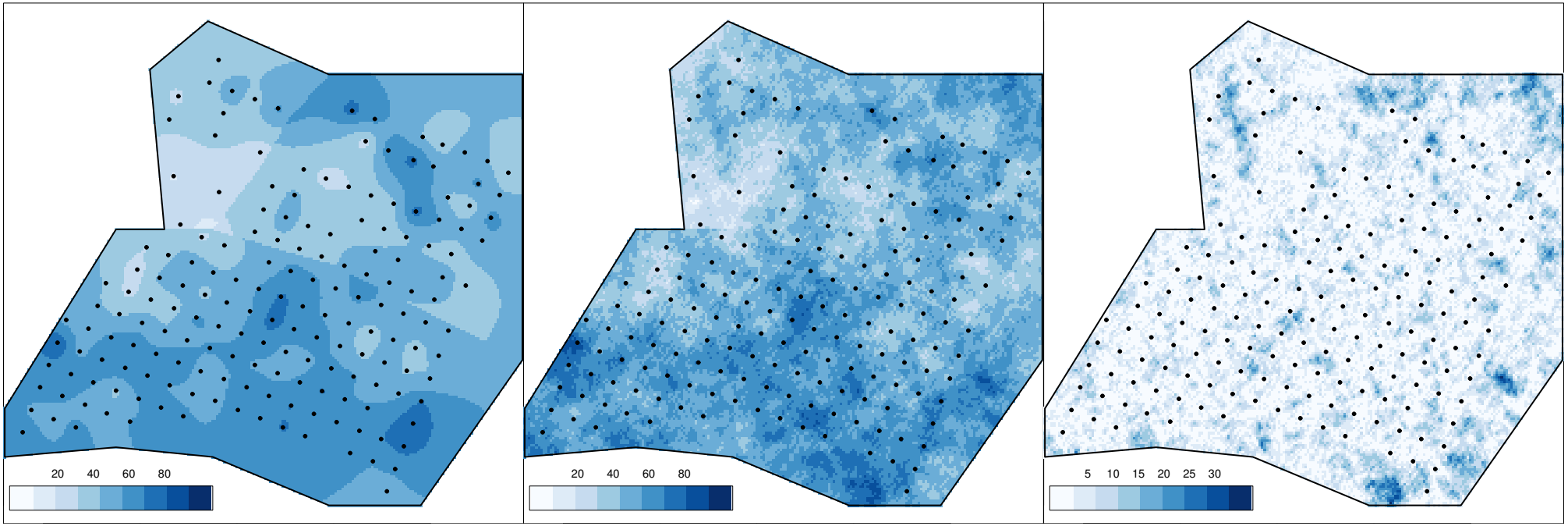


Figure 38: *Comparison between kriging (left) and a conditional simulation (middle). Right: absolute difference of the two.*

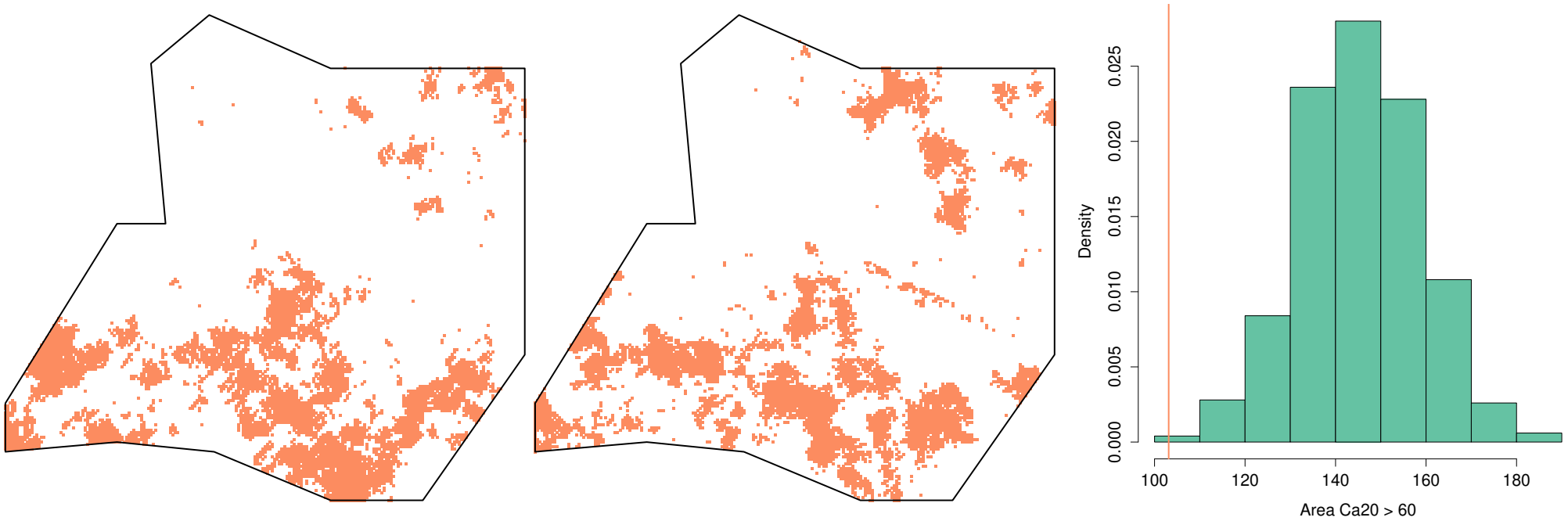


Figure 39: *Left and middle: Two sampled level sets with $u_{crit} = 60$. Right: Distribution of the expected level set area from conditional simulations (histogram) and kriging (vertical line).*

☞ As expected, the kriging–based estimator underestimates.

6. Big data

- Broadly speaking, there are two different type of "big data":

Type I when the number of covariates p is large
Type II when the sample size n is large

- From a statistical standpoint, Type I is the most challenging as parameter estimation is tricky or even impossible.
- Type II induces computational burden and we need numerical/optimization tricks.

You can have the two of us!

High–dimensional setting, a.k.a., big data I

- Fitting a Gaussian process when the number of location is large, i.e., $k \gg 1$, is challenging.
- As stated previously, the most CPU demanding parts of the likelihood is the evaluation of $|\Sigma(\mathbf{s})|$ and the Mahalanobis distance $a^2(\mathbf{s})$.
- To bypass this hurdle one can (at least) use one of the following options:
  - composite likelihoods
  - covariance tapering

Definition 20. A composite log–likelihood is a linear combination of log-likelihoods of "smaller dimensions".

Example 2. The independent composite likelihood uses only univariate densities, i.e.,

$$\ell_{\text{ind}}(\psi; \mathcal{D}_n) = \sum_{j=1}^{k} \omega_j \underbrace{\sum_{i=1}^{n} \log f\{y_i(s_j); \psi\}}_{\text{univariate log-likelihood}},$$

and the pairwise composite likelihood makes use of bivariate densities, i.e.,

$$\ell_{\text{pair}}(\psi; \mathcal{D}_n) = \sum_{j=1}^{k-1} \sum_{\ell=j+1}^{k} \omega_{j,\ell} \underbrace{\sum_{i=1}^{n} \log f\{y_i(s_j), y_i(s_\ell); \psi\}}_{\text{bivariate log-likelihood}},$$

where ω_j and $\omega_{j,\ell}$ are (positive) weights.

Covariance Tapering

- Computational burden heavily relies on the inversion of the covariance matrix $\Sigma(\mathbf{s})$
- Tapering consists in modify $\Sigma(\mathbf{s})$ to get a sparse structure, i.e., many zeros.

pros efficient computation using sparse matrix algebra

cons approximate inference

Proposition 6. *Let f_1 and f_2 be two definite positive functions. Then the function*

$$f : s \mapsto f_1(s) f_2(s)$$

is definite positive.

- We can get a sparse version of $\Sigma(\mathbf{s})$ from the above property. More precisely

$$\Sigma(\mathbf{s})_{\mathsf{tap}} = \Sigma(\mathbf{s}) \odot \Sigma_c(\mathbf{s}),$$

  where $\odot$ stands for the direct product, i.e., componentwise, and $\Sigma_c(\mathbf{s})$ is a covariance matrix obtained from a covariance function with compact support.
- The associated Cholesky decomposition will be sparse as well (up to a sensible permutation)

Two taper approximation

- The tapering introduced above induce a bias in the parameter estimation.
- The bias can be severe if the tapering range is small compared to the practical range—prediction are slightly affected though.
- One may rather use a two–taper version where the Mahalanobis distance is now substituted with

$$\tilde{a}^2(\mathbf{s}, \mathbf{y}) = y^\top \left[\{\Sigma(\mathbf{s}) \odot \Sigma_c(\mathbf{s})\}^{-1} \Sigma_c(\mathbf{s}) \right] y.$$

- The two–taper strategy yields unbiased parameters estimation
- The price to pay is that the computational cost is larger than the one–taper version

Another approach consists in using a truncated SVD, as for PCA.

High–dimensional setting, a.k.a., big data II

- Fitting a Gaussian process when the number of replicates is large, i.e., $n \gg 1$, is challenging.
- In such situations evaluation of the likelihood is demanding due to the sum in n, i.e.,

$$\ell(\psi; \mathcal{D}_n) = \sum_{i=1}^{n} \log \varphi(\mathbf{y}_i; \boldsymbol{\mu}, \Sigma).$$

- Two (related) possible approaches are:
  - mini–batch gradient ascent
  - stochastic gradient ascent

Gradient ascent (reminder)

Proposition 7. *Let ψ_0 be an initial state. The sequence*

$$\psi_{n+1} = \psi_n + \eta \nabla J(\psi_n), \qquad n \geq 0,$$

will converge to a local maxima (if it does), where η is known as the step size (learning rate if you're a noob!).

- The step size can be adaptive, i.e., η now depends on t and / or ψ_n.
- Current popular choices are Nesterov adaptive schemes, i.e., so called momentum, where

$$\psi_{n+1} = \psi_n + \mu_n v_n + \eta_n \nabla J(\psi_n), \qquad v_n \text{ some "measure of velocity".}$$

If minimizing, use gradient descent $\psi_{n+1} = \psi_n - \eta \nabla J(\psi_n)$.

Mini–batch gradient ascent

- Consider the following optimization problem

$$\arg\max_{\psi\in\Psi} J(\psi), \qquad J(\psi) = \sum_{i=1}^{n} J_i(\psi).$$

- If $n \gg 1$, evaluation of J is prohibitive and prevent the use of gradient ascent.
- One can minimize the CPU cost using mini–batch gradient ascent

$$\psi_{n+1,b+1} = \psi_{n,b} + \eta \sum_{i\in B_b} \nabla J_i(\psi_{n,b}), \qquad b = 1, \ldots, B,$$

and where by convention $\psi_{n+1,1} = \psi_{n,B+1}$ and

$$\cup_{b=1,\ldots,B} B_b = \{1, \ldots, n\}, \qquad B_b \cap B_{b'} = \emptyset,$$

i.e., a partition of $\{1, \ldots, n\}$.

If you want to show off (a bit)

- The gradient update step is done after browsing each batch B_b
- The computational cost is thus reduced.
- One loop over the entire data set is called an epoch.

Stochastic gradient ascent

- Stochastic gradient ascent is somehow similar to mini–batch gradient ascent as it compute the gradient on subset of the data set $\mathcal{D}_n$.
- The main difference is that these subsets are now random.
- The basic stochastic gradient ascent scheme is

$$\psi_{n+1} = \psi_n + \eta \nabla J_I(\psi_n), \qquad I \sim \text{Unif}\{1, \ldots, n\}.$$

- Some generalization are possible:
  - random mini-batches where we drawn random batches
  - use a other distribution than the discrete uniform.
- Stochastic gradient ascent will converge to a local maxima as long as the learning rate goes to 0.
- Its randomness may helps escaping from local maxima.