Machine Learning for econometrics

Flexible models for tabular data

Matthieu Doutreligne

February 18th, 2025

A lot of today's content is taken from the excellent sklearn mooc (Estève et al., 2022)

Reminder from previous session

- Statistical learning 101: bias-variance trade-off
- Regularization for linear models: Lasso, Ridge, Elastic Net
- Transformation of variables: polynomial regression

Reminder from previous session

- Statistical learning 101: bias-variance trade-off
- Regularization for linear models: Lasso, Ridge, Elastic Net
- Transformation of variables: polynomial regression

🤔 But… How to select the best model? the best hyper-parameters?

Table of contents

Model evaluation and selection with cross-validation

Example with the Wage dataset

- Raw dataset: (N=534, p=11)

EDUCATION	SOUTH	SEX	EXPERIENCE	UNION	WAGE	AGE	RACE	OCCUPATION	SECTOR	MARR
8	no	female	21	not_member	5.10	35	Hispanic	Other	Manufacturing	Married
9	no	female	42	not_member	4.95	57	White	Other	Manufacturing	Married
12	no	male	1	not_member	6.67	19	White	Other	Manufacturing	Unmarried
12	no	male	4	not_member	4.00	22	White	Other	Other	Unmarried
12	no	male	17	not_member	7.50	35	White	Other	Other	Married

-

-

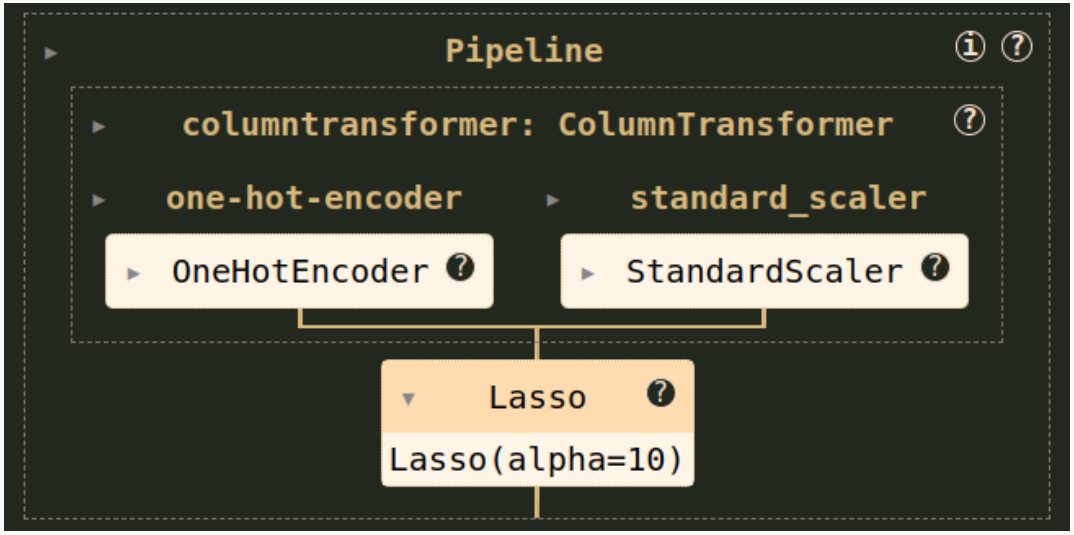
Example with the Wage dataset

- Raw dataset: (N=534, p=11)
- Transformation: encoding categorical data, scaling numerical data: (N=534, p=23)

one-hot-encoder__SOUTH_no	one-hot-encoder__SOUTH_yes	one-hot-encoder__SEX_female	one-hot-encoder__SEX_male	one-hot-encoder__UNION_member	encoder__UNION_not
1.0	0.0	1.0	0.0	0.0	
1.0	0.0	1.0	0.0	0.0	
1.0	0.0	0.0	1.0	0.0	
1.0	0.0	0.0	1.0	0.0	
1.0	0.0	0.0	1.0	0.0	

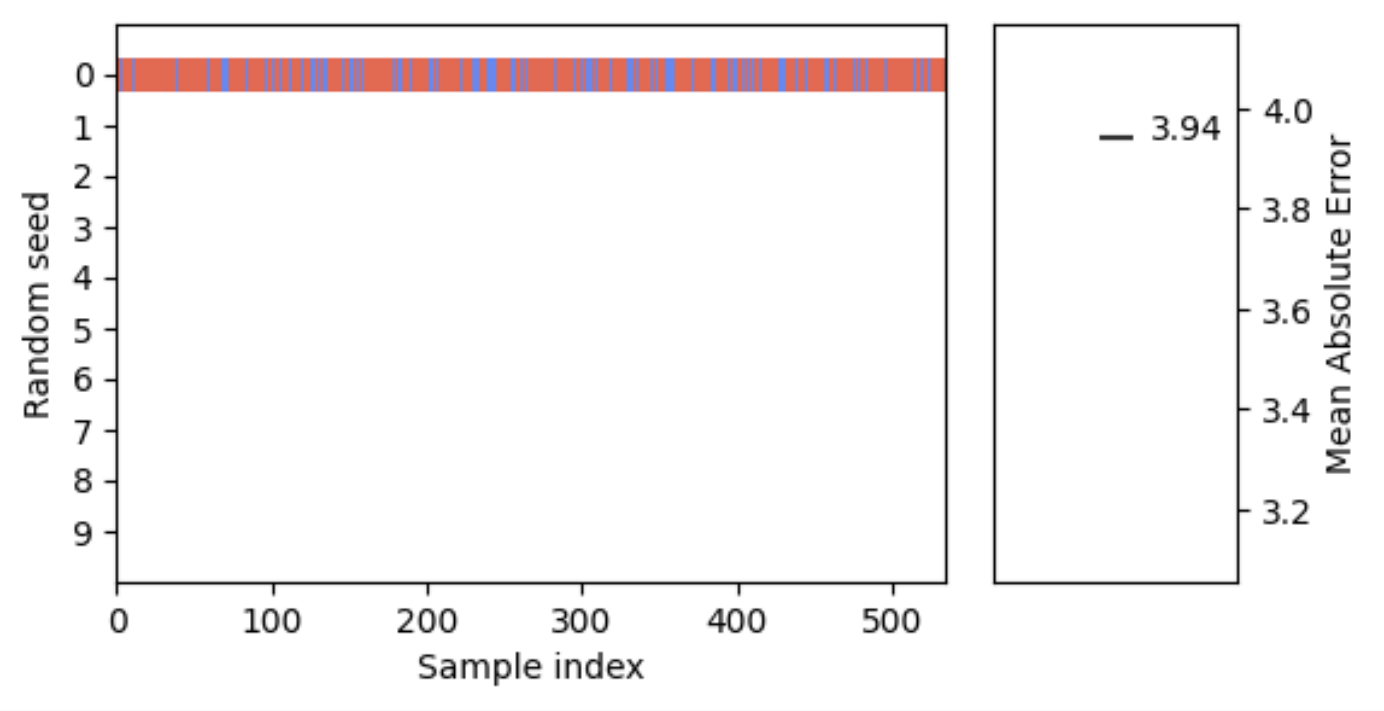
Example with the Wage dataset

- Raw dataset: (N=534, p=11)
- Transformation: encoding categorical data, scaling numerical data: (N=534, p=23)
- Regressor: Lasso with regularization parameter ($\alpha = 10$)



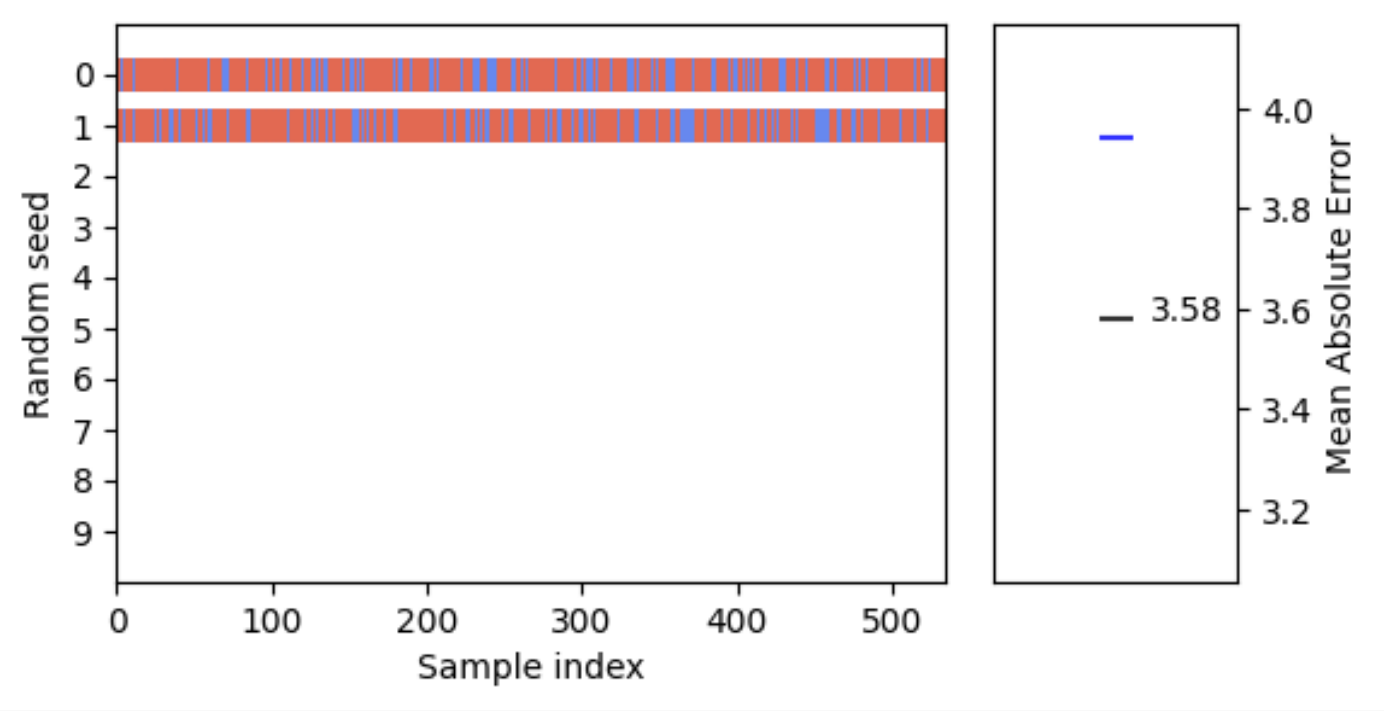
Repeated train/test splits

Splitting once: In red, the training set, in blue, the test set



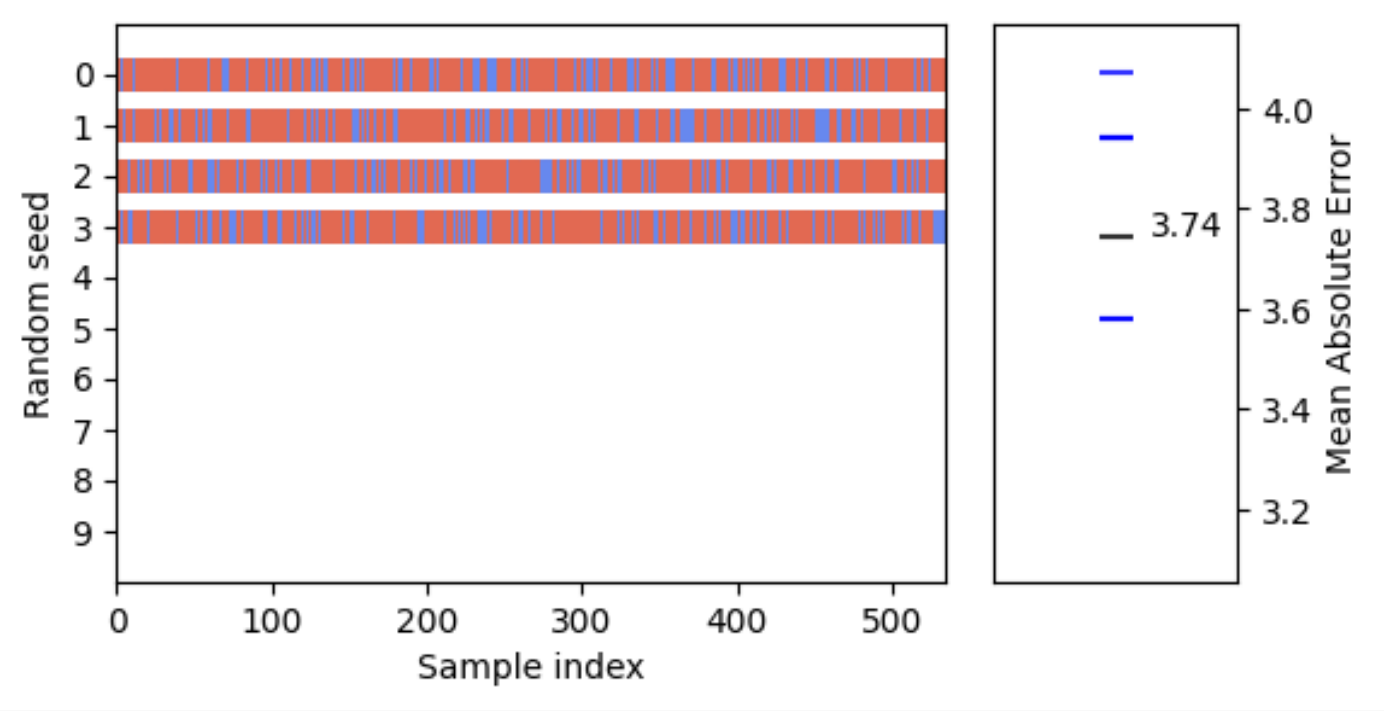
Repeated train/test splits

But we could have chosen another split ! Yielding a different MAE

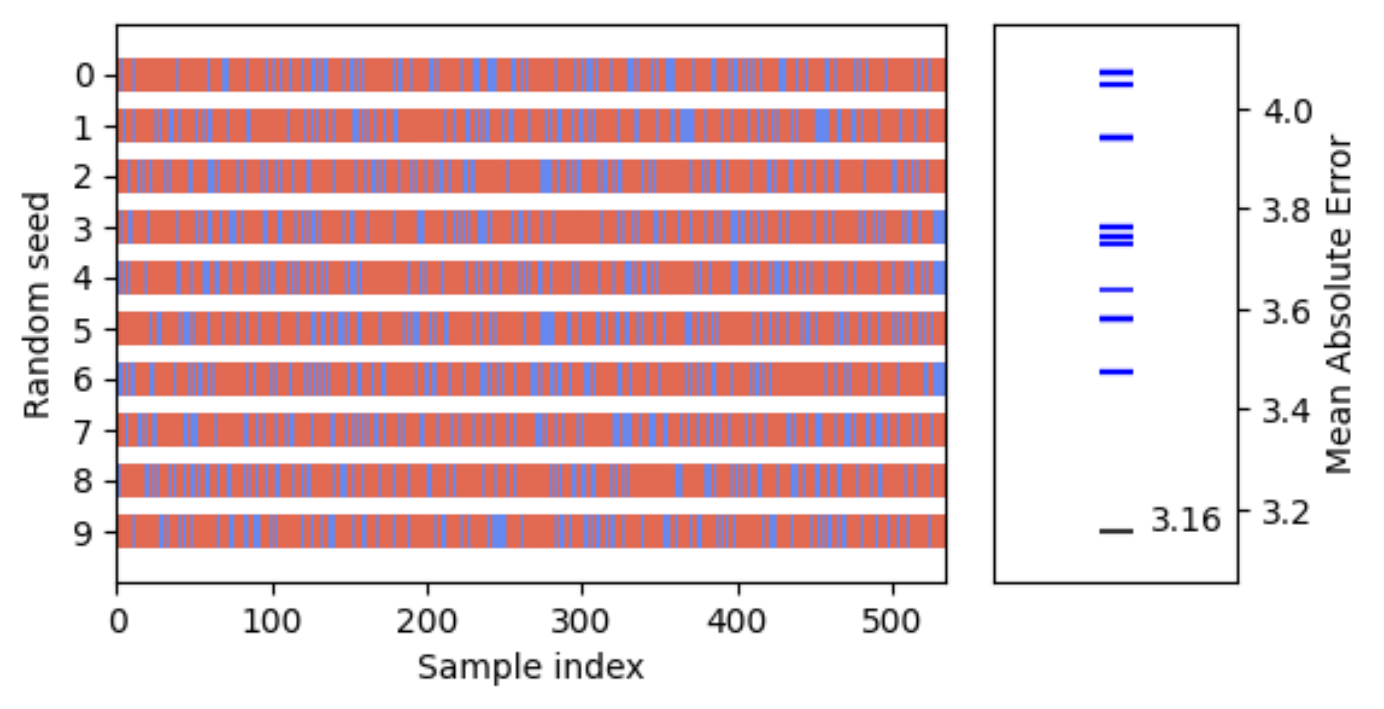


Repeated train/test splits

And another split...



Splitting ten times



🎉 **Distribution of MAE:** 3.71 ± 0.26

Repeated exclusive train/test splits = Cross-validation

Practical usage with sklearn: `cross_validate`.

```python
from sklearn.model_selection import cross_validate
cv_results = cross_validate(
    regressor, data, target, cv=5, scoring="neg_mean_absolute_error"
)
```

Repeated exclusive train/test splits = Cross-validation

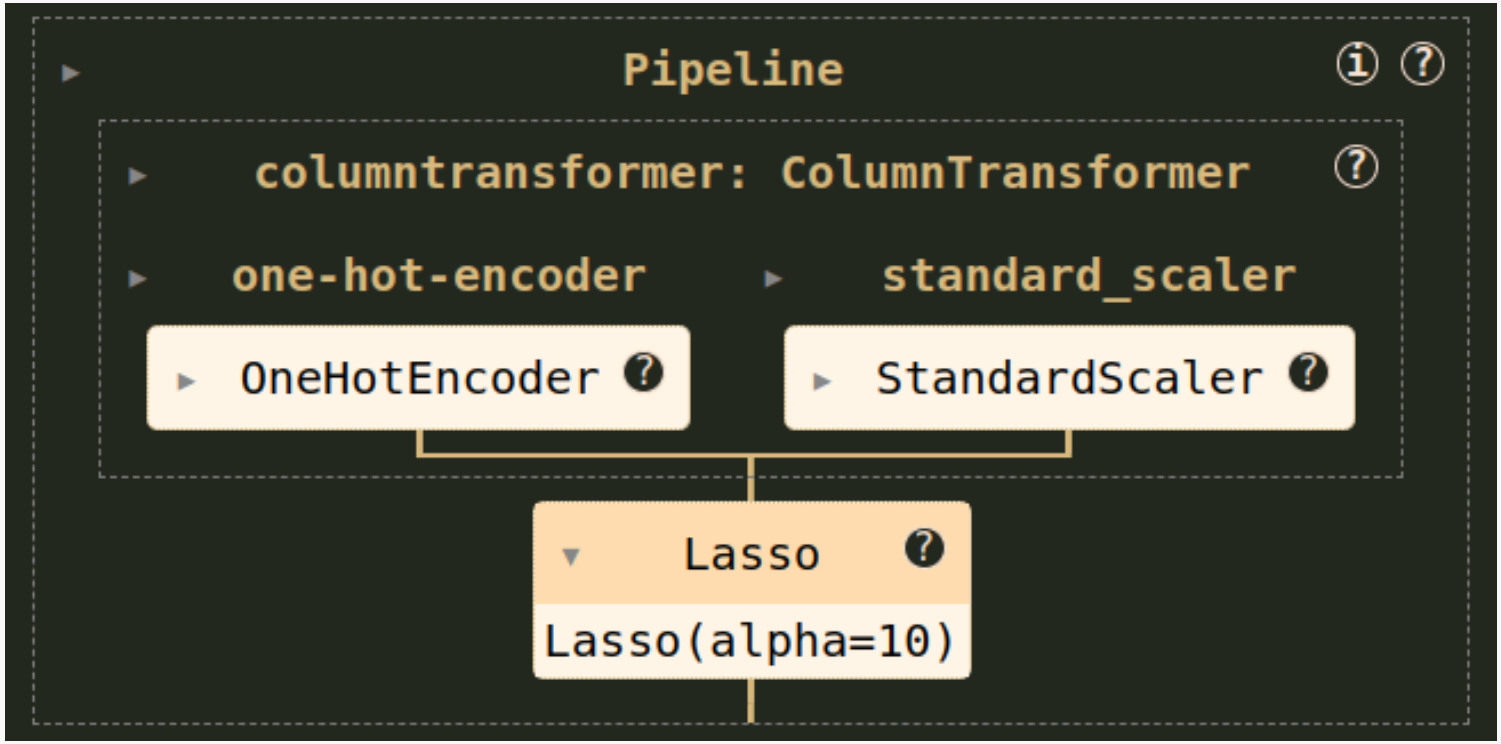
Practical usage with sklearn: `cross_validate`.

```python
from sklearn.model_selection import cross_validate
cv_results = cross_validate(
    regressor, data, target, cv=5, scoring="neg_mean_absolute_error"
)
```

- 🙂 Robustly estimate generalization performance.
- 🤩 Estimate data variability of the performance : bigger source of variation (Bouthillier et al., 2021).
- 🚀 Let's use it to select the best models among several candidates!

Cross-validation for model selection: choose best α for lasso

Wage pipeline



Cross-validation for model selection: choose best α for lasso

Wage pipeline

Random search over a distribution of α values

Python

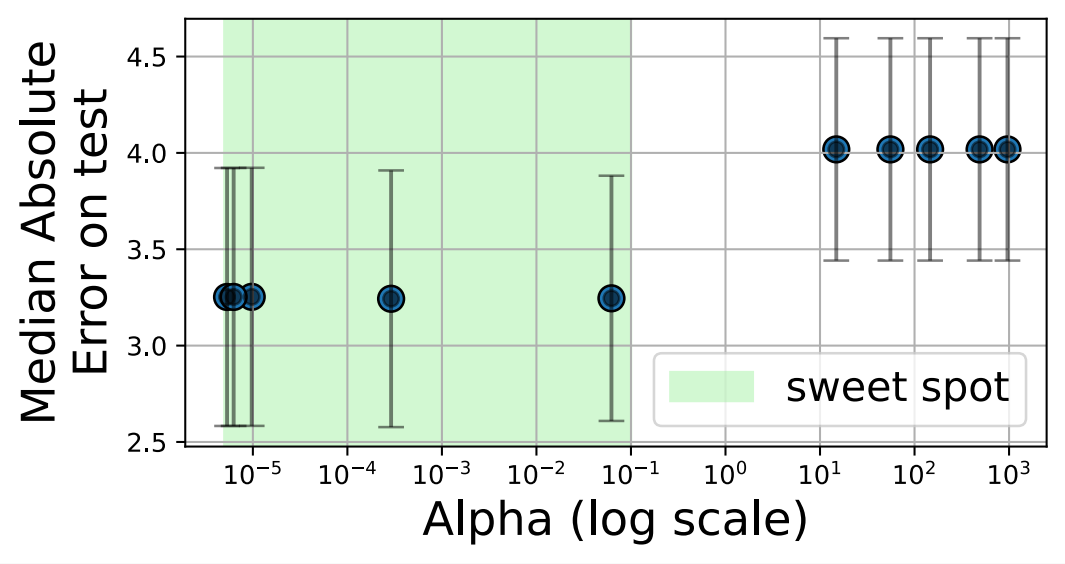
```python
param_distributions = {"lasso__alpha": loguniform(1e-6, 1e3)}
model_random_search = RandomizedSearchCV(
    pipeline,
    param_distributions=param_distributions,
    n_iter=10, # number of hyper-parameters sampled
    cv=5, # number of folds for the cross-validation
    scoring="neg_mean_absolute_error", # score to optimize
)
model_random_search.fit(X, y)
```

Cross-validation for model selection: choose best α for lasso

Wage pipeline

Random search over a distribution of α values

Goal: Identify the best α value(s)



What final model to use for new prediction?

- Often used in practice: refit on the full data the model with the best hyper-parameters.
-

What final model to use for new prediction?

- Often used in practice: refit on the full data the model with the best hyper-parameters.
- Theorically motivated: Aggregate the outputs from the cross-validate estimators of the best model:

$$\hat{y} = \frac{1}{K} \sum_{k=1}^{K} \hat{y}_k$$

where $\hat{y}_k$ is the prediction of the model trained on the k-th fold.

What final model to use for new prediction?

- Often used in practice: refit on the full data the model with the best hyper-parameters.
- Theorically motivated: Aggregate the outputs from the cross-validate estimators of the best model:

$$\hat{y} = \frac{1}{K} \sum_{k=1}^{K} \hat{y}_k$$

  where $\hat{y}_k$ is the prediction of the model trained on the k-th fold.

- Averaging cross-validate estimators selects the best model asymptotically among a family of models (Lecué & Mitchell, 2012)

Naive cross-validation to select and estimate the best performances

Hyper-parameters selection is a kind of model fitting

Using a single loop of cross-validation, the full dataset is used to:

- Select the best hyper-parameters
- Estimate the generalization performance of the selected model

Naive cross-validation to select and estimate the best performances

Hyper-parameters selection is a kind of model fitting

Using a single loop of cross-validation, the full dataset is used to:

- Select the best hyper-parameters
- Estimate the generalization performance of the selected model

⚠️ Naive cross-validation can lead to overfitting and over-optimistic performance estimation

Naive cross-validation to select and estimate the best performances

Hyper-parameters selection is a kind of model fitting

Using a single loop of cross-validation, the full dataset is used to:

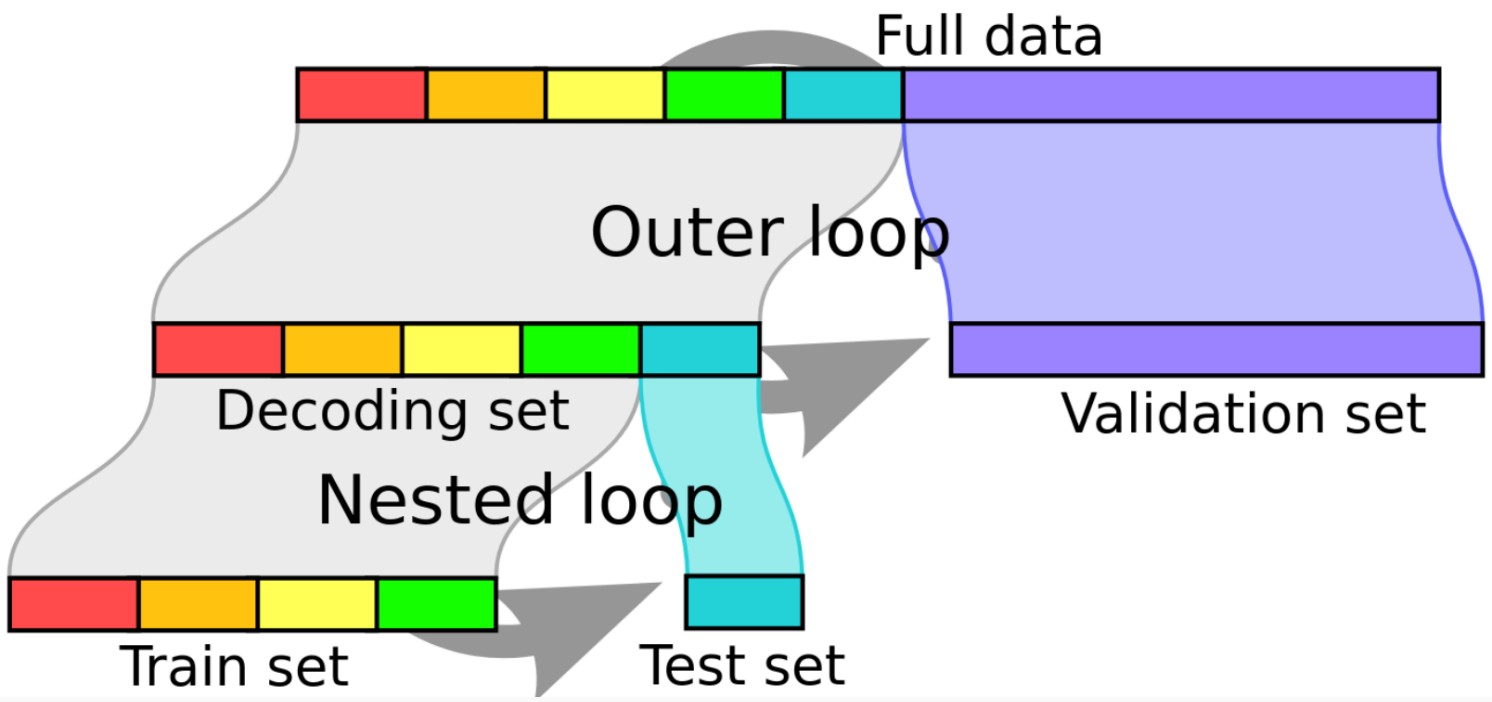
- Select the best hyper-parameters
- Estimate the generalization performance of the selected model

⚠️ Naive cross-validation can lead to overfitting and over-optimistic performance estimation

🚀 Solution: Nested cross-validation (Varoquaux et al., 2017)

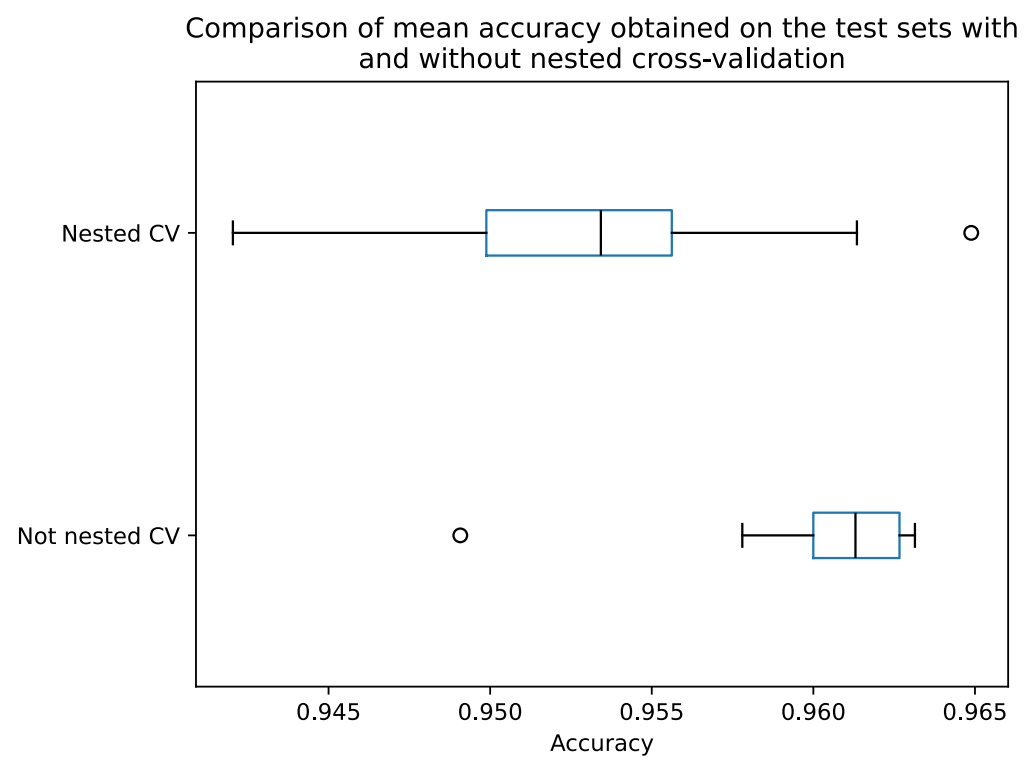
Nested cross-validation to select and estimate the best performances

- Inner CV loop to select the best hyper-parameters
- Outer loop to estimate the generalization performance of the selected model



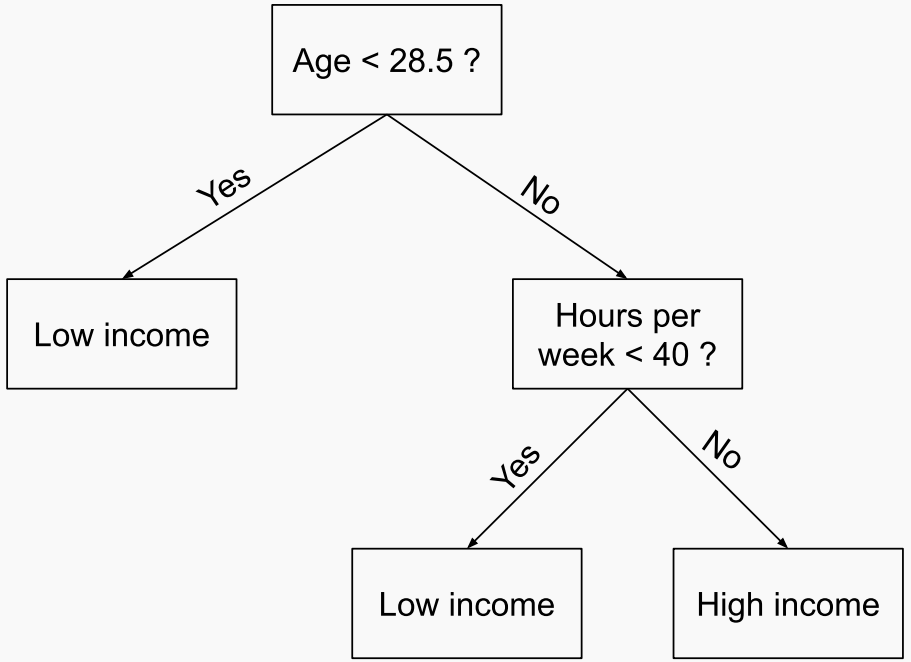
Over-optimistic performance estimation: example

- Dataset: Breast cancer (N, p) = (569, 30)
- Classifier: RandomForestClassifier with multiple choices of hyper-parameter

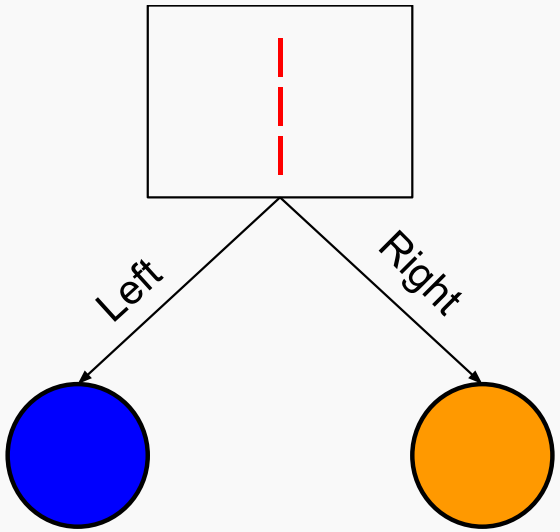


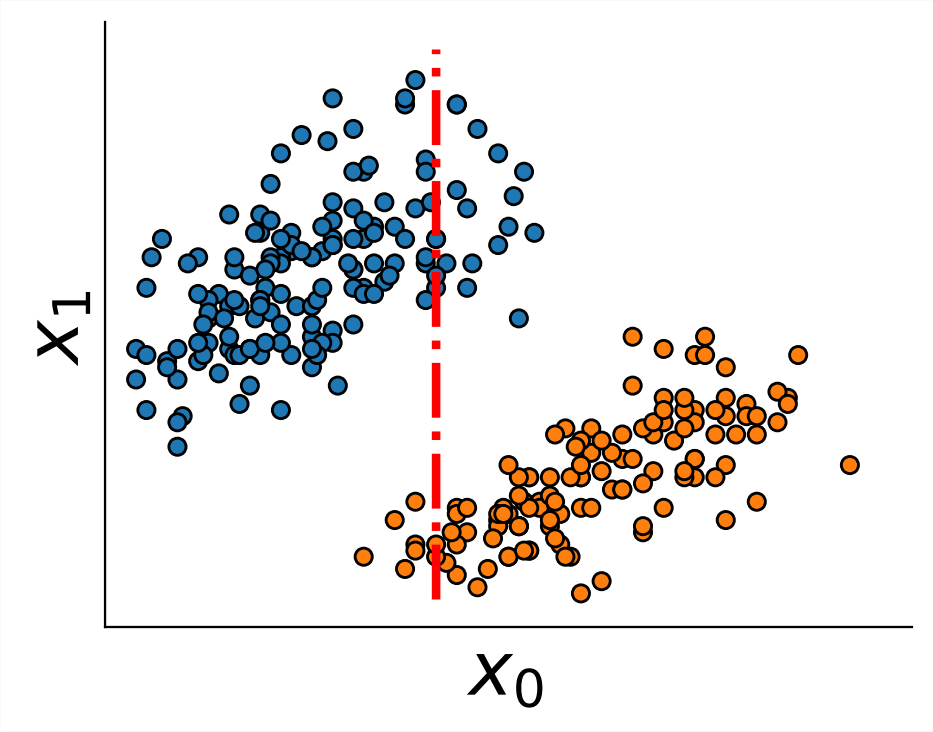
Flexible models: Tree, random forests and boosting

What is a decision tree? An example.

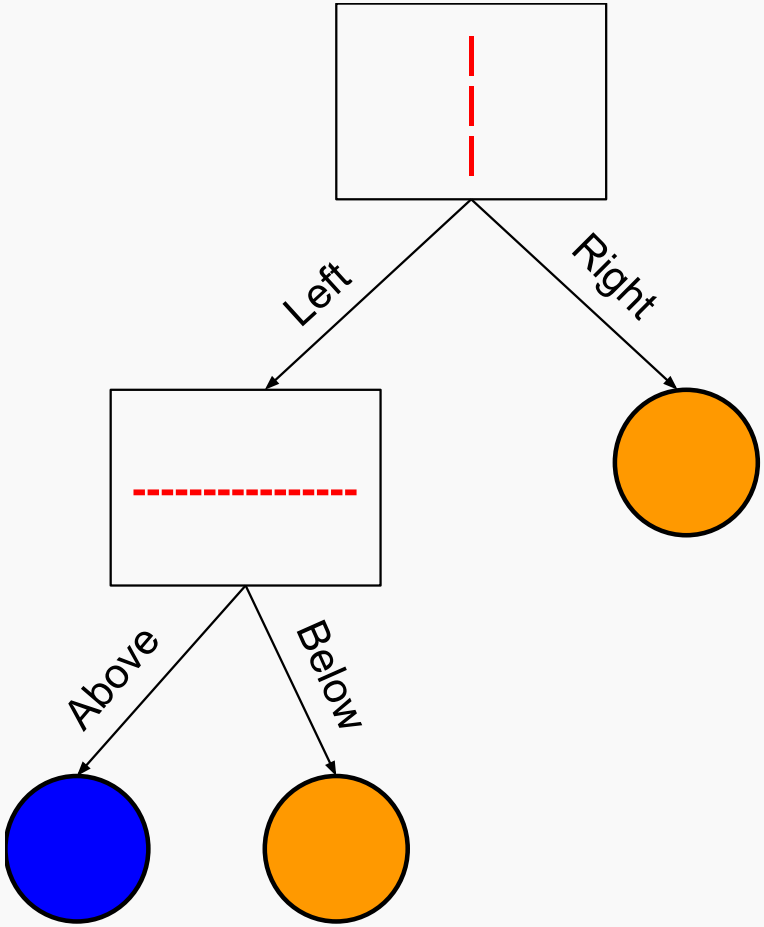


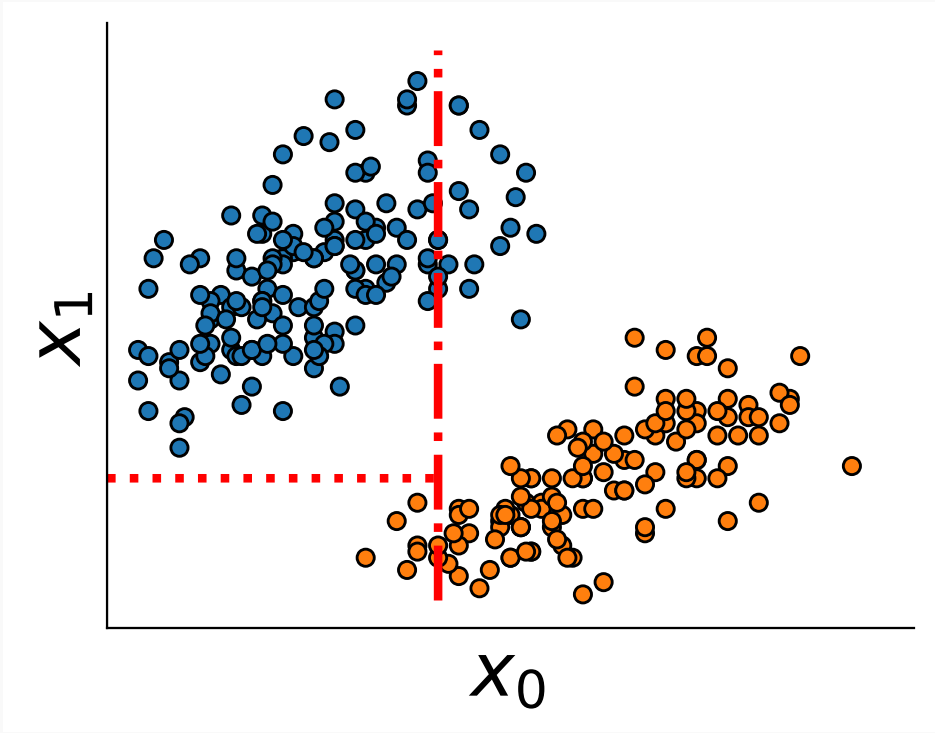
Growing a classification tree



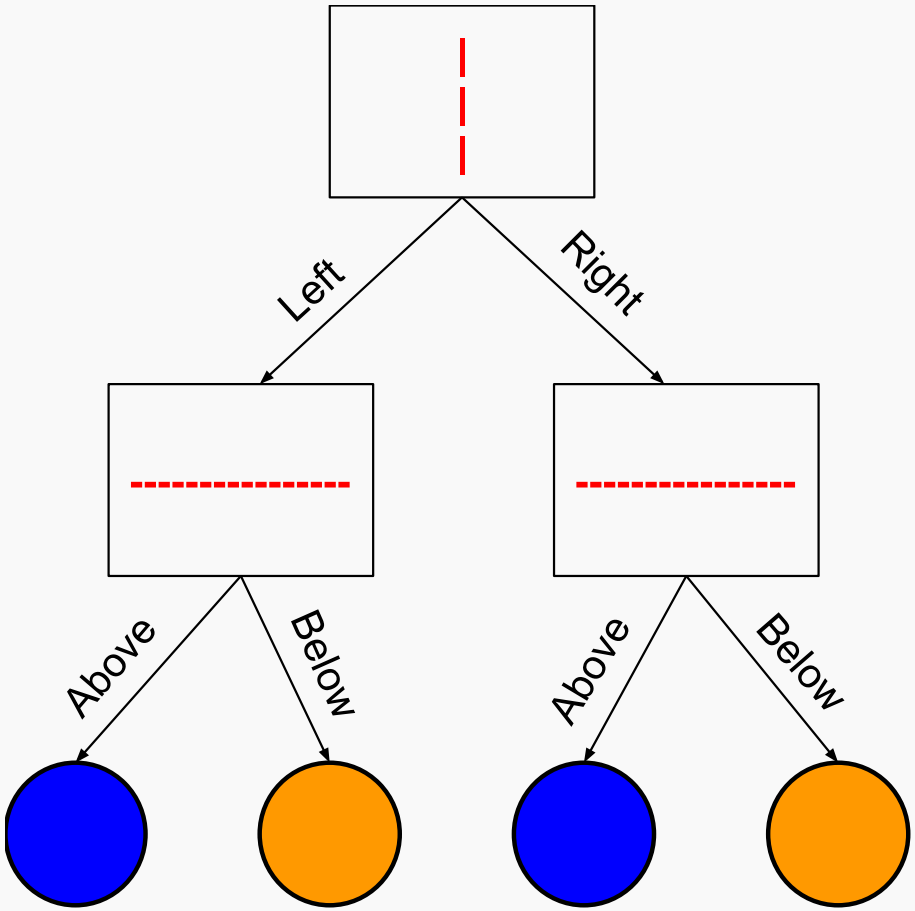


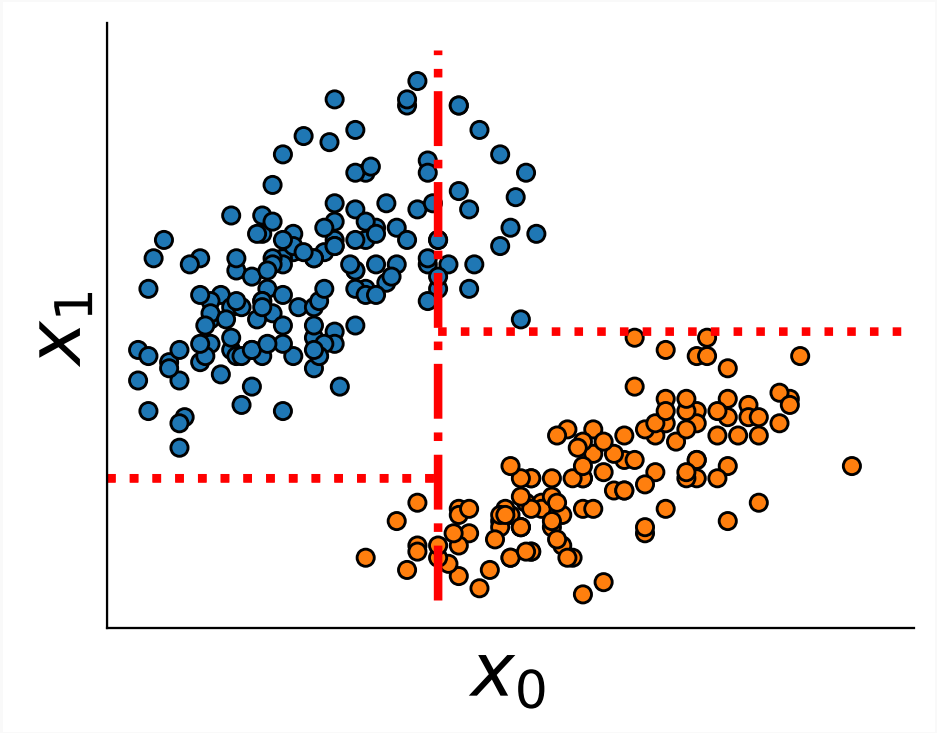
Growing a classification tree



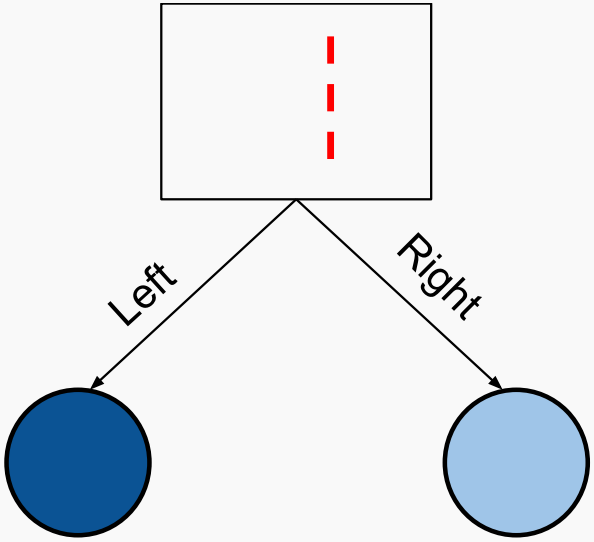


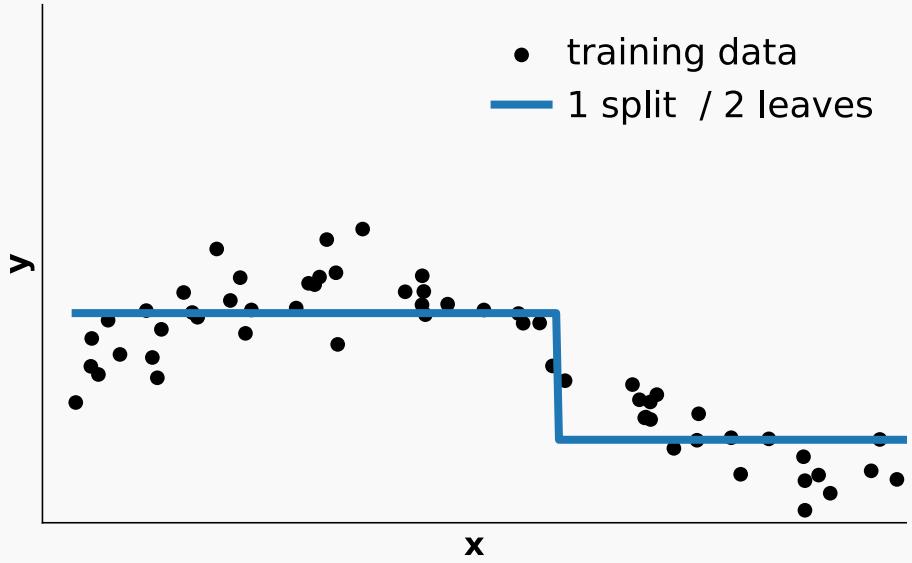
Growing a classification tree



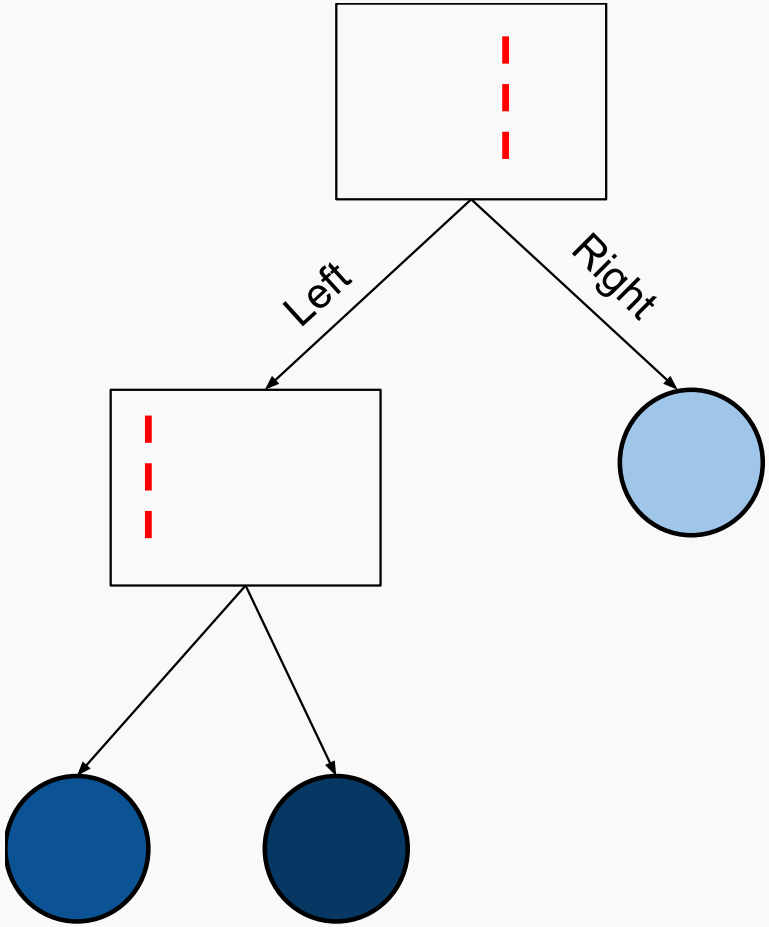


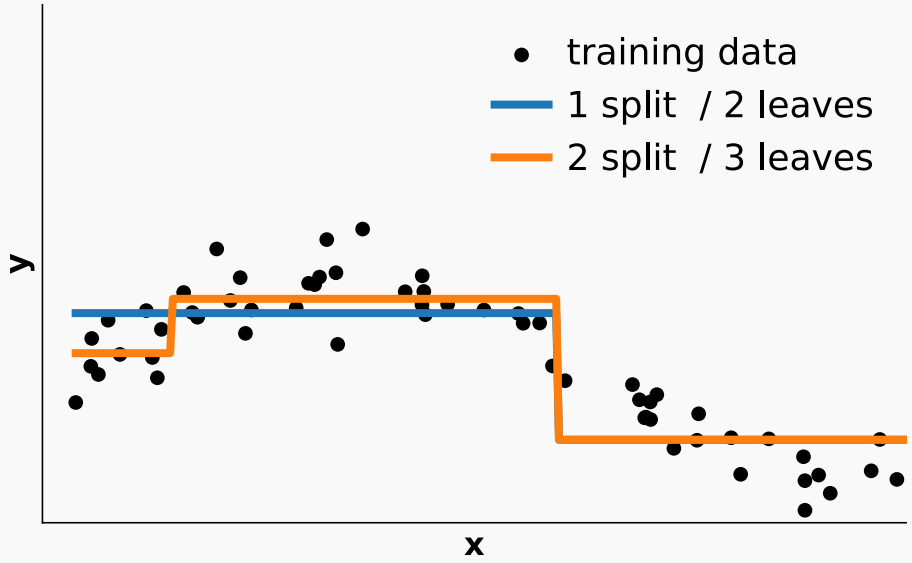
Growing a regression tree



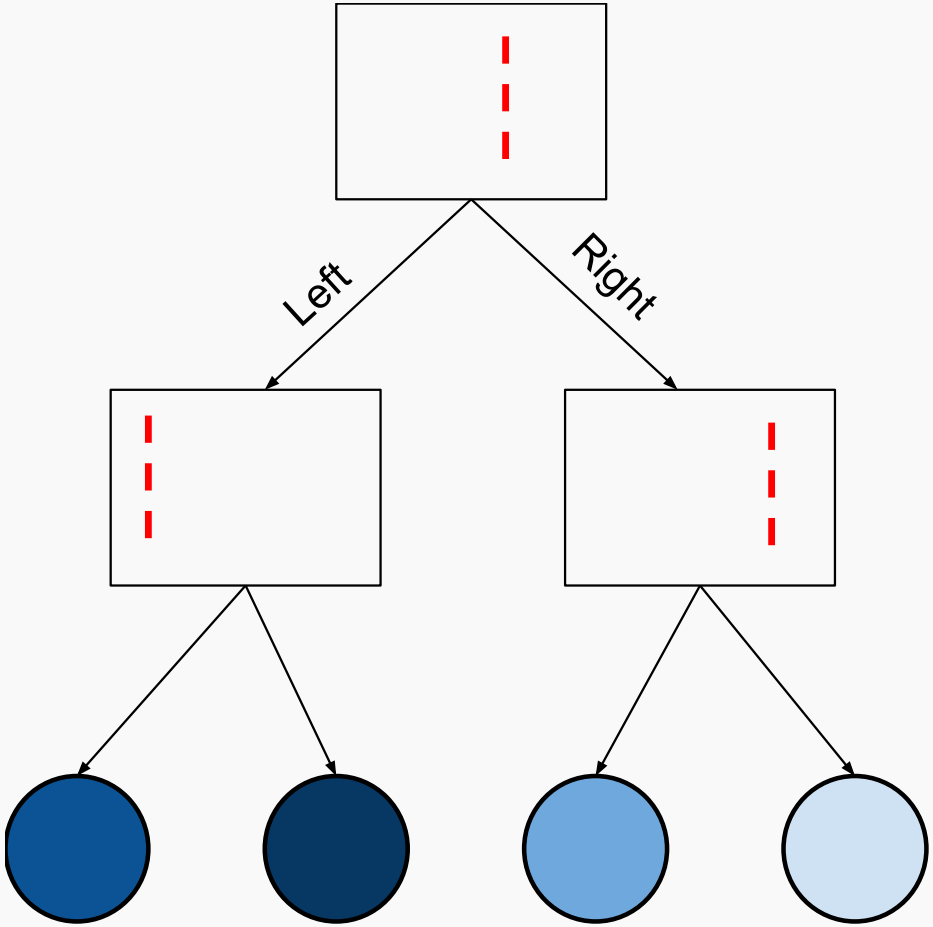


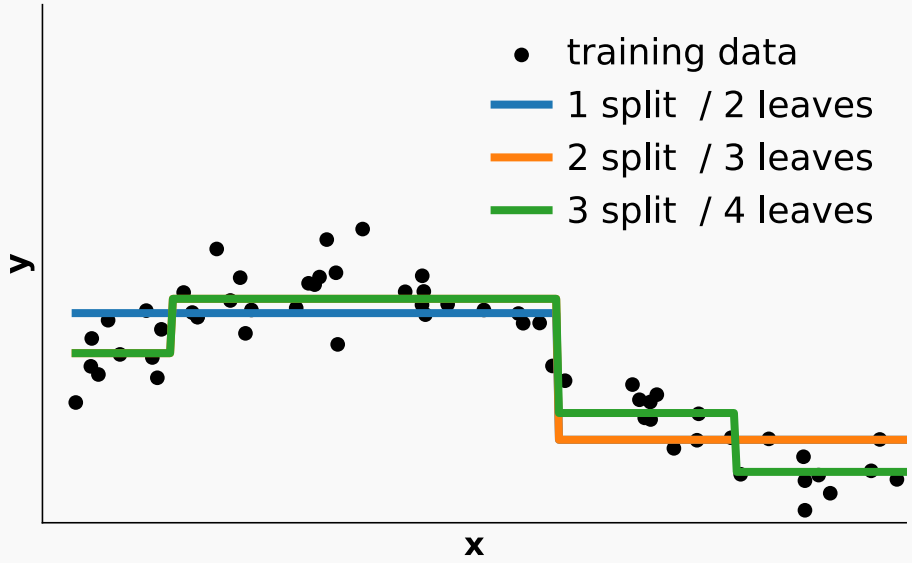
Growing a regression tree





Growing a regression tree





How the best split is chosen?

The best split minimizes an impurity criteria

- For the next left and right nodes
- Over all features
- And all possible splits

How the best split is chosen?

The best split minimizes an impurity criteria

- For the next left and right nodes
- Over all features
- And all possible splits

Formally

Let the data at node m be Q_m with n_m samples. For a candidate split on feature j and threshold t_m $\theta = (j, t_m)$, the split yields:

$$Q_m^{\text{left}}(\theta) = \left\{(x, y) | x_j \leq t_m\right\} \text{ and } Q_m^{\text{right}}(\theta) = Q_m \setminus Q_m^{\text{left}}(\theta)$$

How the best split is chosen?

The best split minimizes an impurity criteria

- For the next left and right nodes
- Over all features
- And all possible splits

Formally

Let the data at node m be Q_m with n_m samples. For a candidate split on feature j and threshold t_m $\theta = (j, t_m)$, the split yields:

$$Q_m^{\text{left}}(\theta) = \left\{(x, y) | x_j \leq t_m\right\} \text{ and } Q_m^{\text{right}}(\theta) = Q_m \setminus Q_m^{\text{left}}(\theta)$$

Then θ^* is chosen to minimize the impurity criteria averaged over the two children nodes:

$\theta^* = \text{argmin}_{j,t_m} \left[\frac{n_m^{\text{left}}}{n_m} H\left(Q_m^{\text{left}}(\theta)\right) + \frac{n_m^{\text{right}}}{n_m} H\left(Q_m^{\text{right}}(\theta)\right)\right]$ with H the impurity criteria.

Classification

Gini impurity

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk}) \text{ with } p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k)$$

Classification

Gini impurity

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk}) \text{ with } p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k)$$

Cross-entropy

$$H(Q_m) = -\sum_{k \in K} p_{mk} \log(p_{mk})$$

Impurity criteria

Classification

Gini impurity

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk}) \text{ with } p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k)$$

Cross-entropy

$$H(Q_m) = -\sum_{k \in K} p_{mk} \log(p_{mk})$$

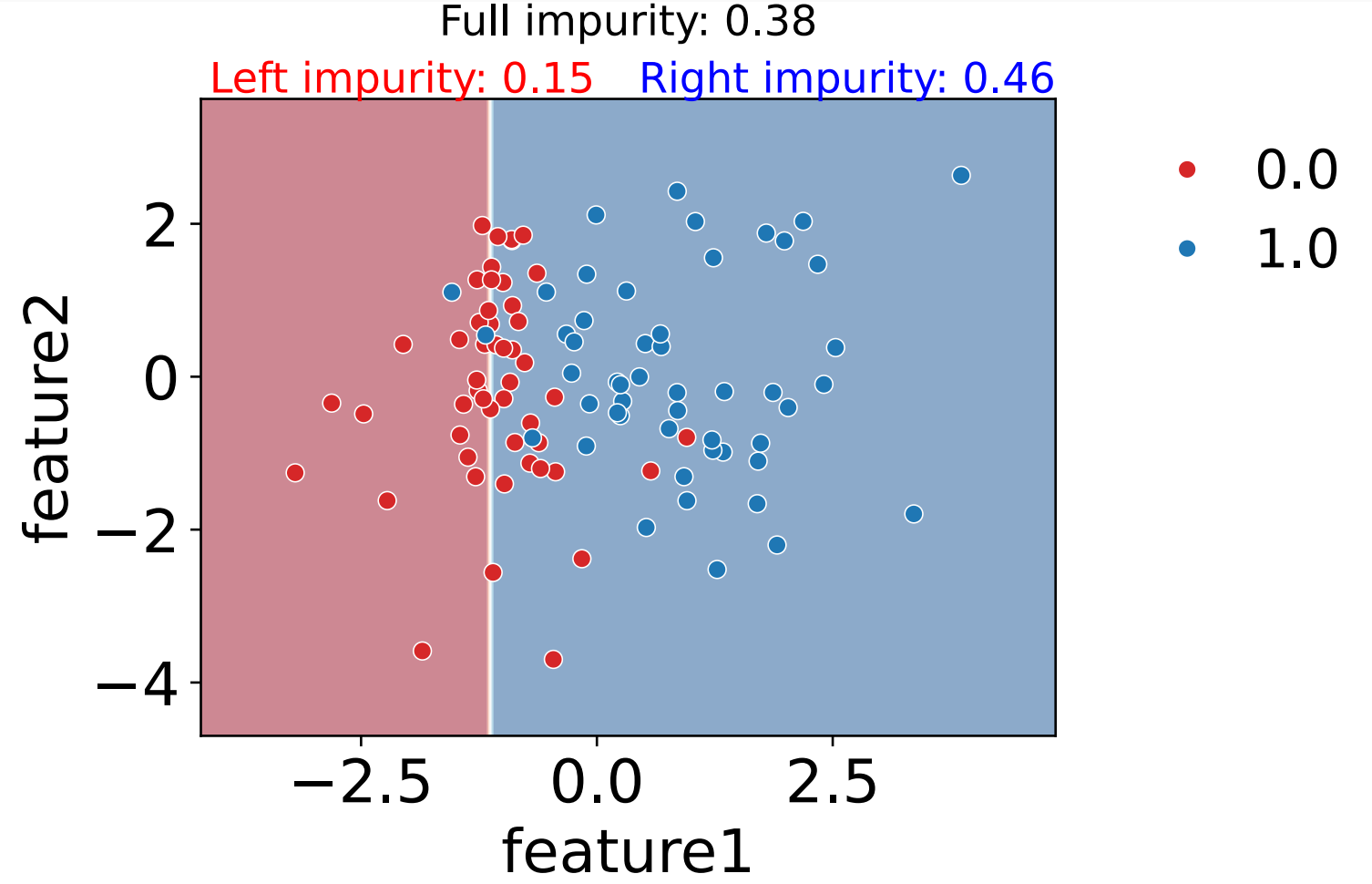
Regression

Mean squared error

$$H(Q_m) = \frac{1}{n_m} \sum_{y \in Q_m} (y - \overline{y_m})^2 \text{ where } \overline{y_m} = \frac{1}{n_m} \sum_{y \in Q_m} y$$

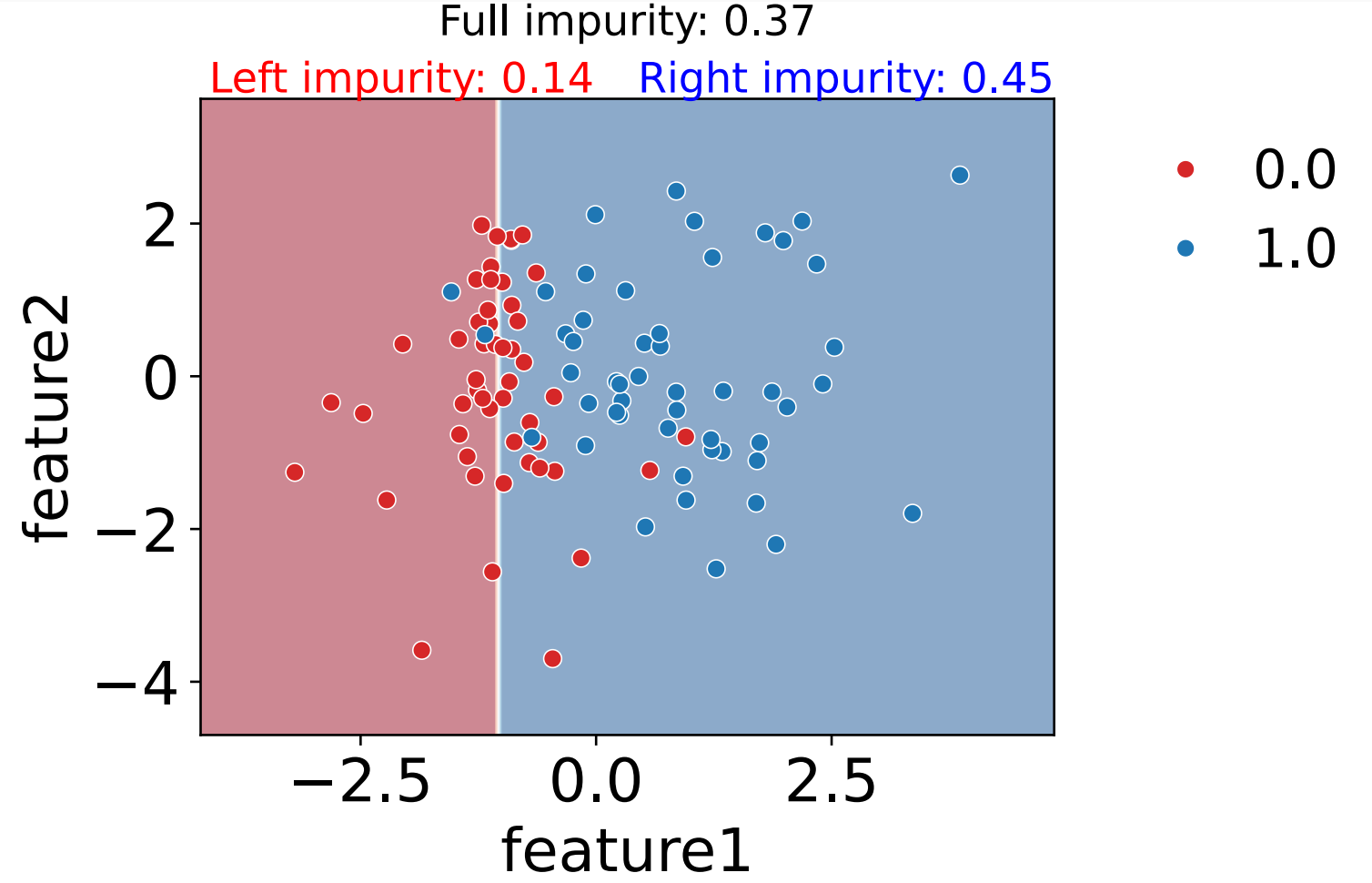
Chose the best split: example

Random split



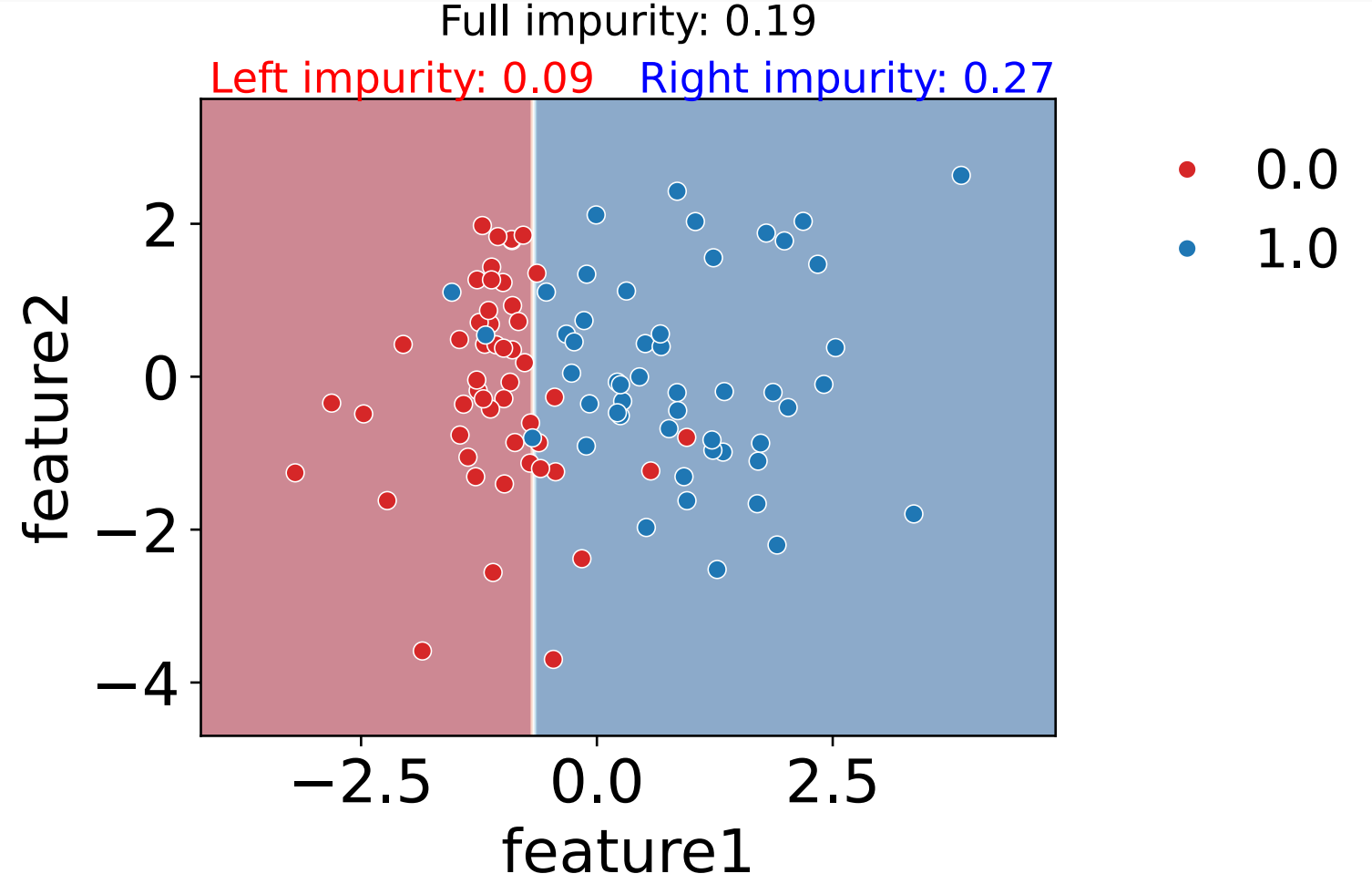
Chose the best split: example

Moving the split to the right from one point



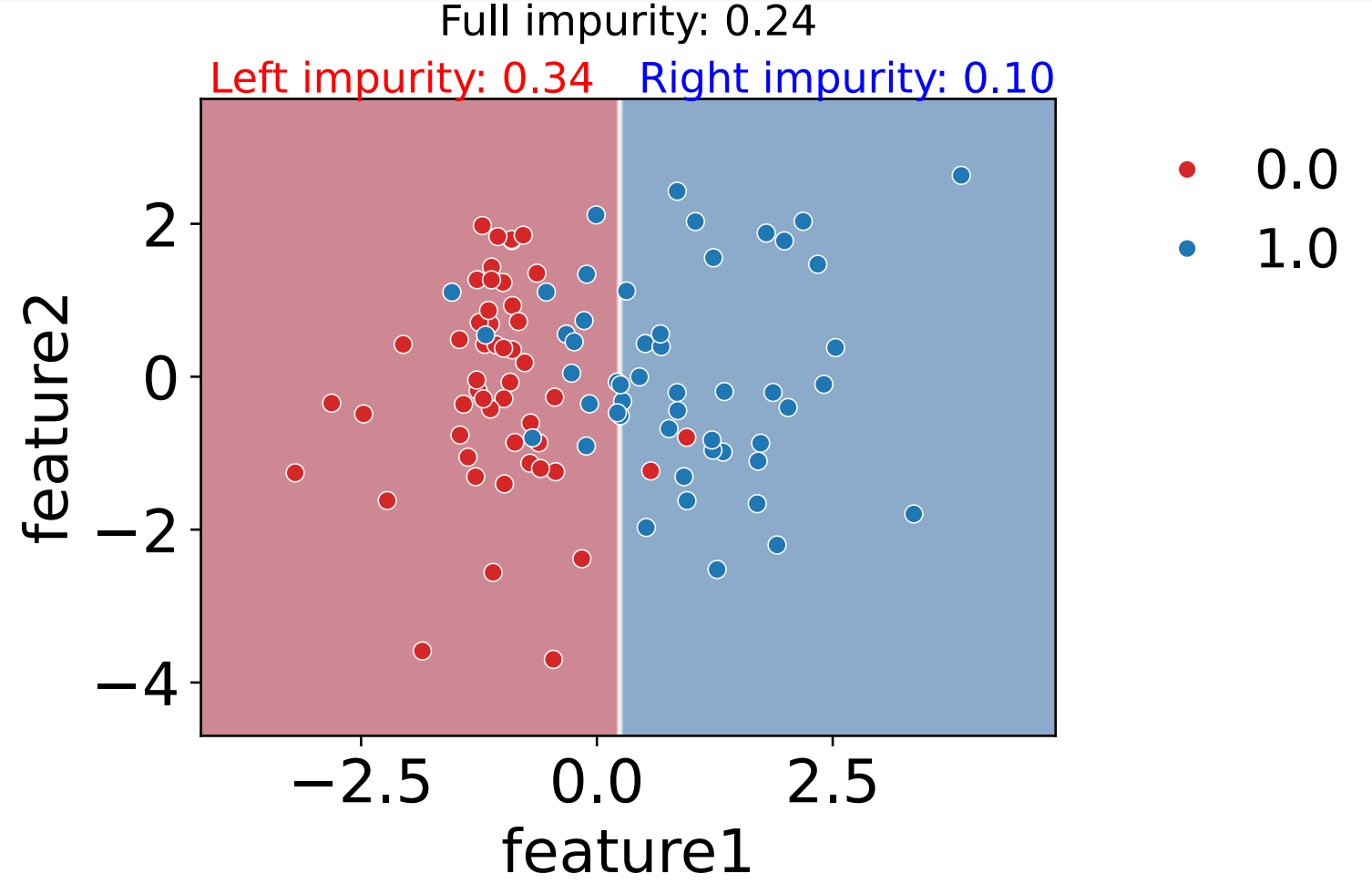
Chose the best split: example

Moving the split to the right from 10 points



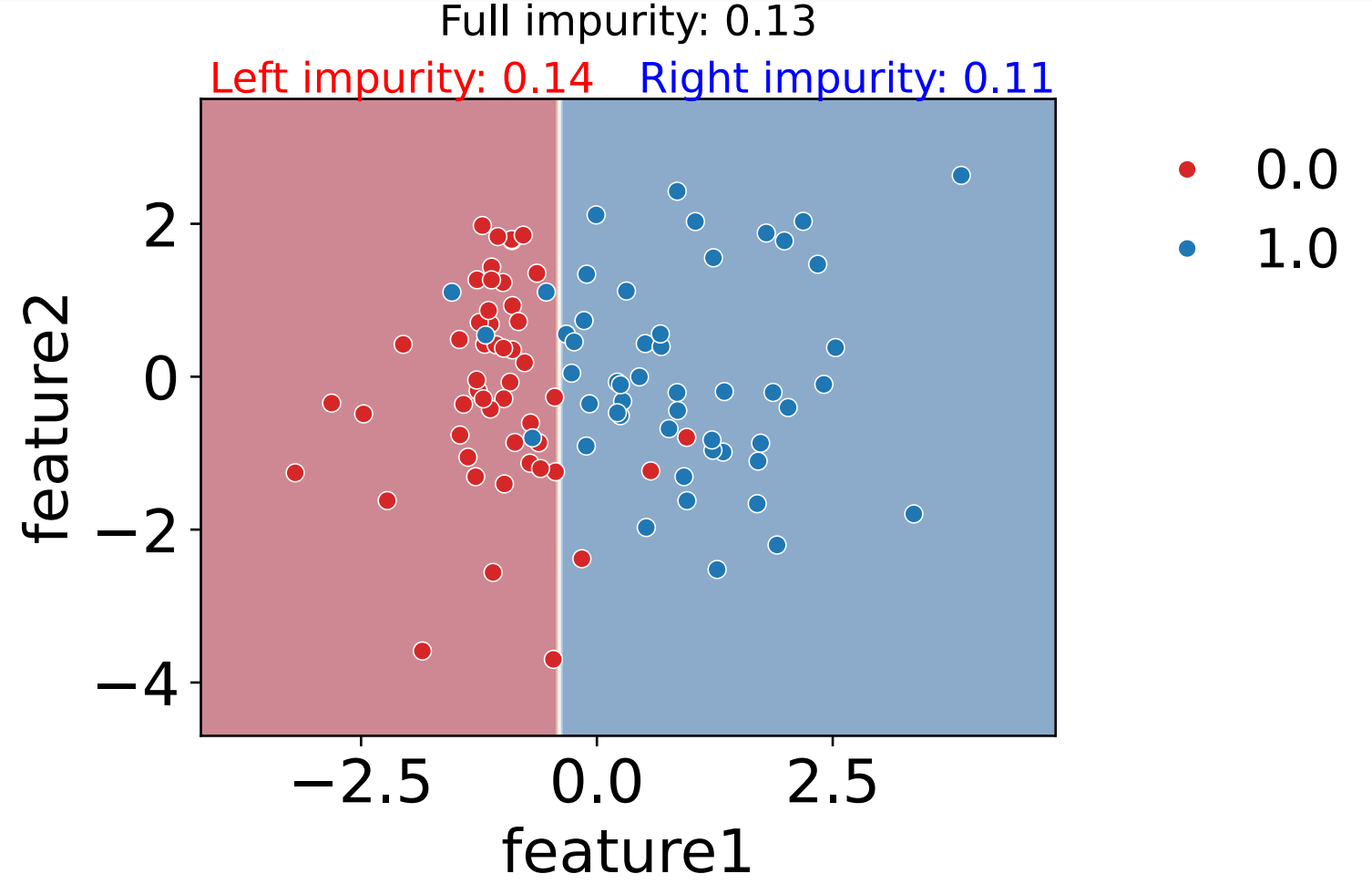
Chose the best split: example

Moving the split to the right from 20 points

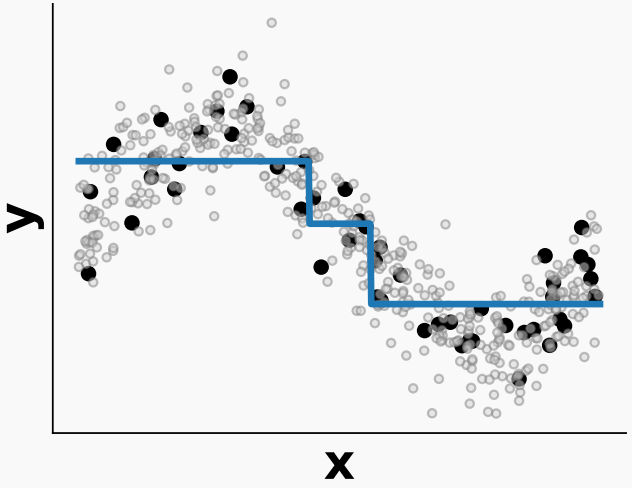


Chose the best split: example

Best split



Tree depth and overfitting



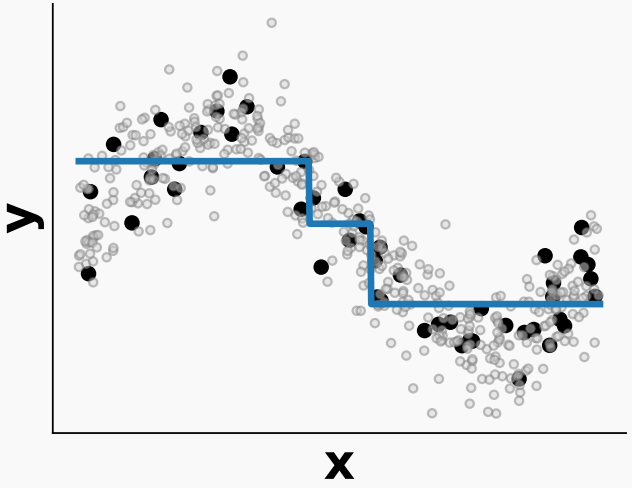
Underfitting

`max_depth` or

`max_leaf_nodes`

too small

Tree depth and overfitting

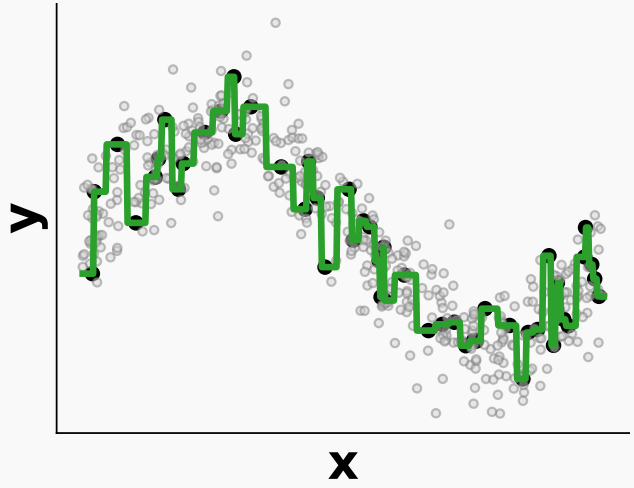


Underfitting

`max_depth` or

`max_leaf_nodes`

too small



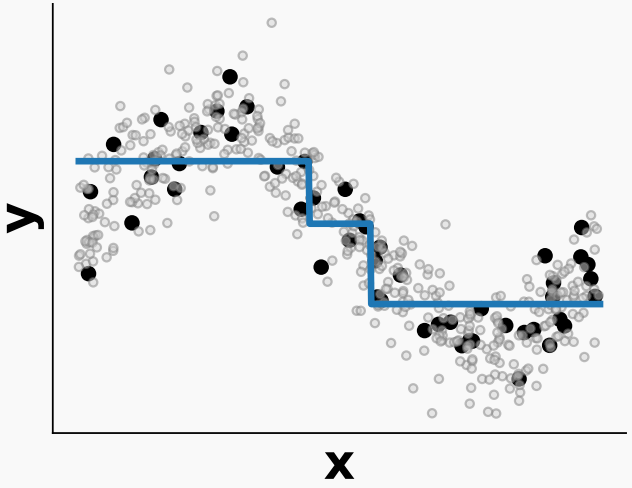
Overfitting

`max_depth` or

`max_leaf_nodes`

too large

Tree depth and overfitting

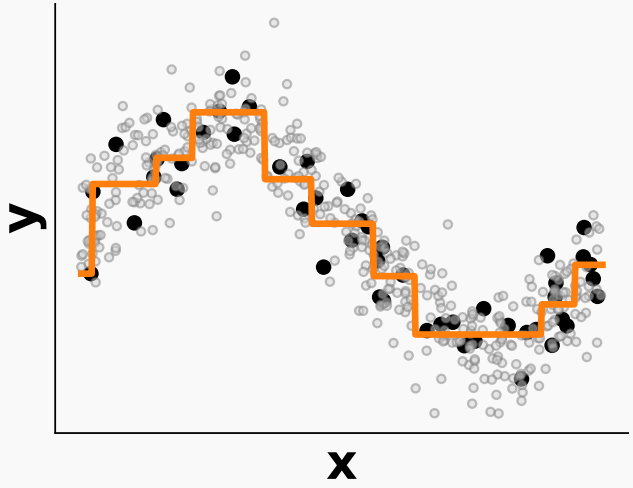


Underfitting

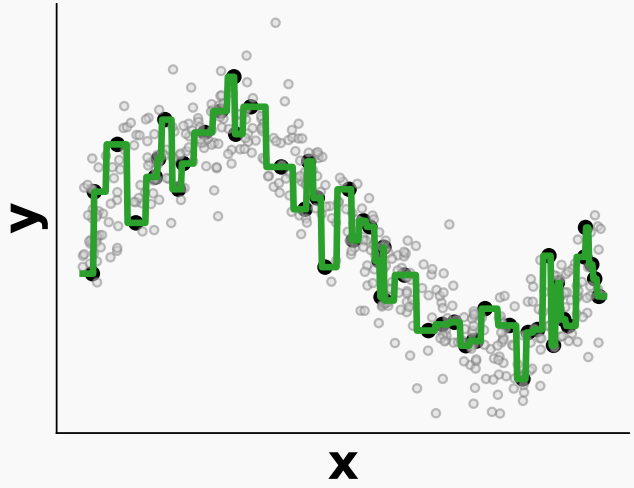
`max_depth` or

`max_leaf_nodes`

too small



Best trade-off



Overfitting

`max_depth` or

`max_leaf_nodes`

too large

Main hyper-parameters of tree models

Python

```python
DecisionTreeRegressor(
    criterion="squared error",
    max_depth=None, # Tree depth (assume symetric trees)
    min_samples_split=2, # Tree depth (allowing asymetric trees)
    min_samples_leaf=1, # Tree depth (allowing asymetric trees)
    max_leaf_nodes=None, # Tree depth (allowing asymetric trees)
    min_impurity_decrease=0.0, # Tree depth (allowing asymetric trees)
)
```

Pros

- Easy to interpret
- Handle mixed types of data: numerical, categorical and missing data
- Handle interactions
- Fast to fit

Pros and cons of trees

Pros

- Easy to interpret
- Handle mixed types of data: numerical, categorical and missing data
- Handle interactions
- Fast to fit

Cons

- Prone to overfitting
- Unstable: small changes in the data can lead to very different trees
- Mostly useful as a building block for ensemble models: random forests and boosting trees

Ensemble models: Bagging ie. Bootstrap AGGregatING

Bootstrap resampling (random sampling with replacement) proposed by (Breiman, 1996)

Built upon Bootstrap, introduced by (Efron, 1992) to estimate the variance of an estimator.

Ensemble models: Bagging ie. Bootstrap AGGregatING

Bootstrap resampling (random sampling with replacement) proposed by (Breiman, 1996)

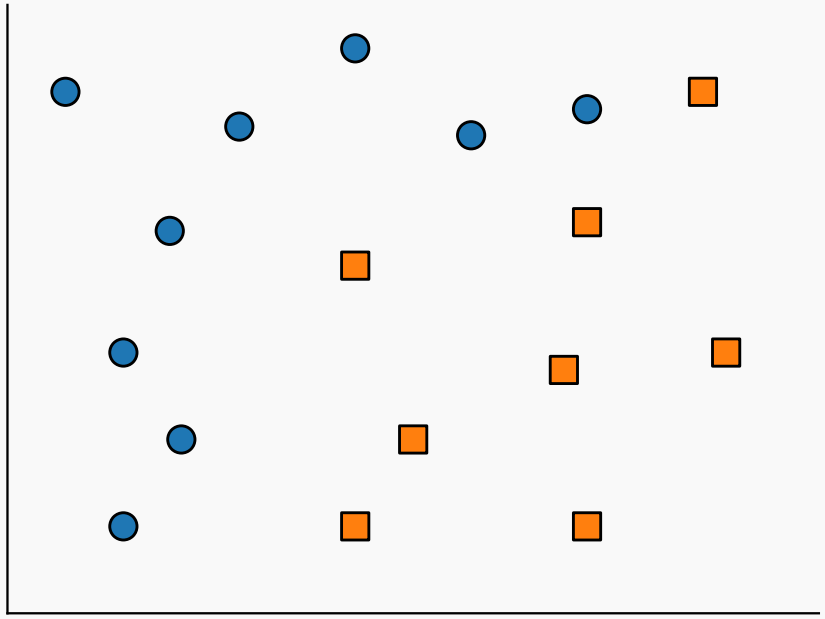
Built upon Bootstrap, introduced by (Efron, 1992) to estimate the variance of an estimator.

Bagging is used in machine learning to reduce the variance of a model prone to overfitting

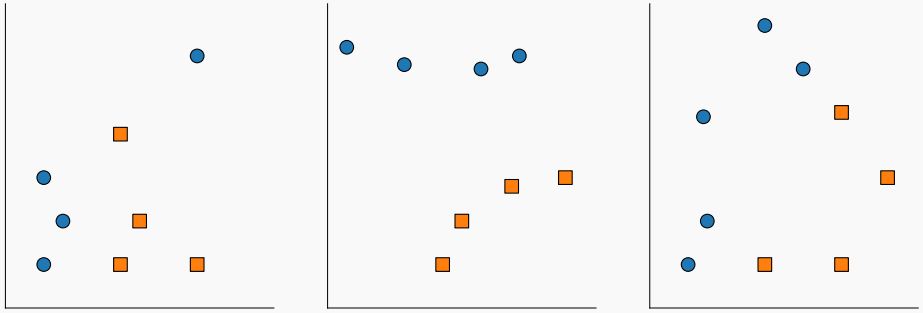
Can be used with any model!

Random forests: Bagging with classification trees

Full dataset

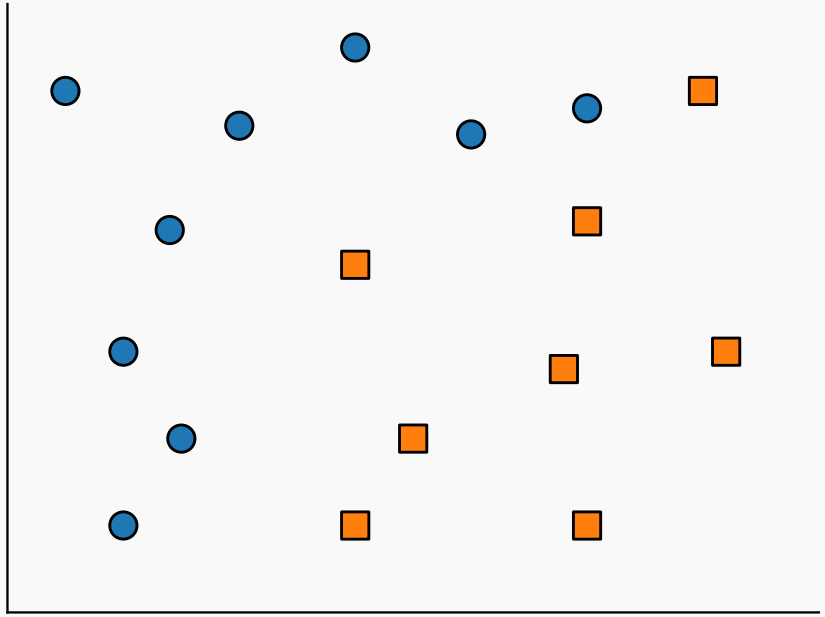


Three bootstrap samples

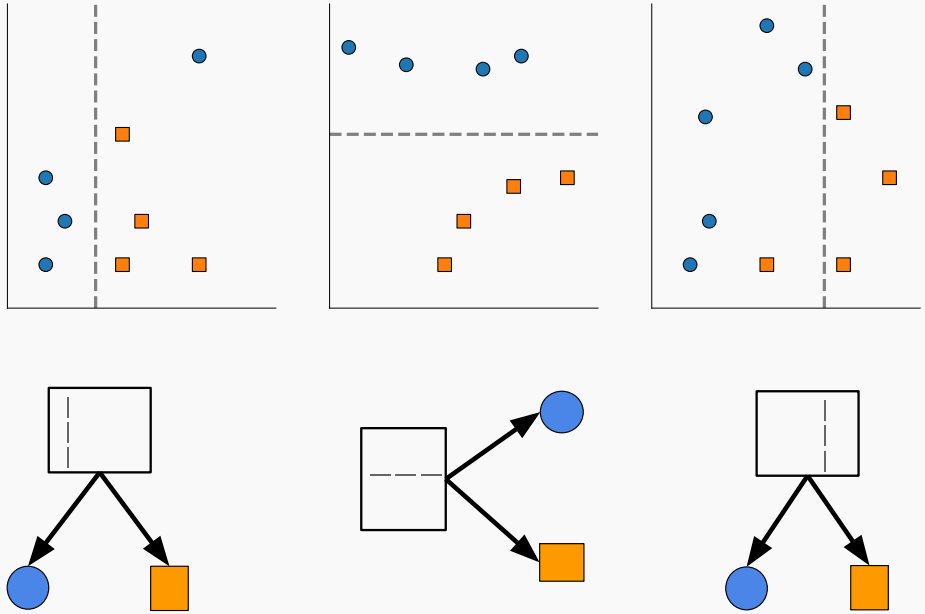


Random forests: Bagging with classification trees

Full dataset

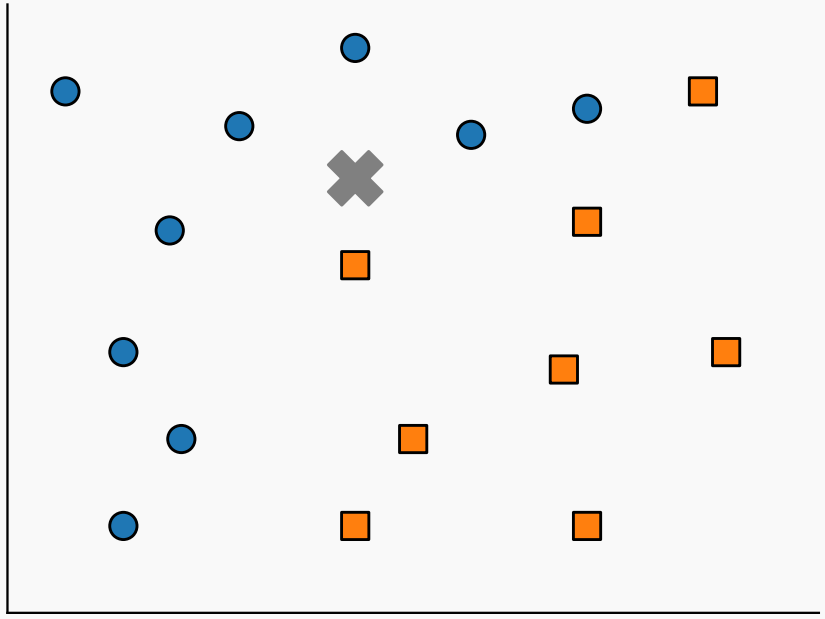


Three bootstrap samples

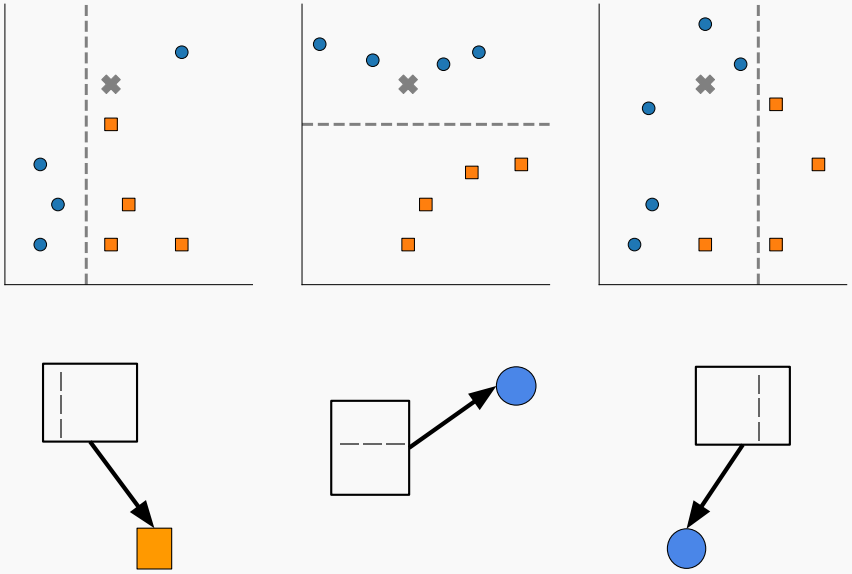


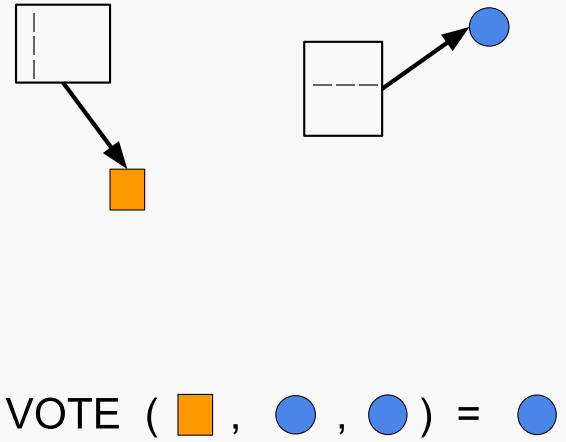
Random forests: Bagging with classification trees

Full dataset

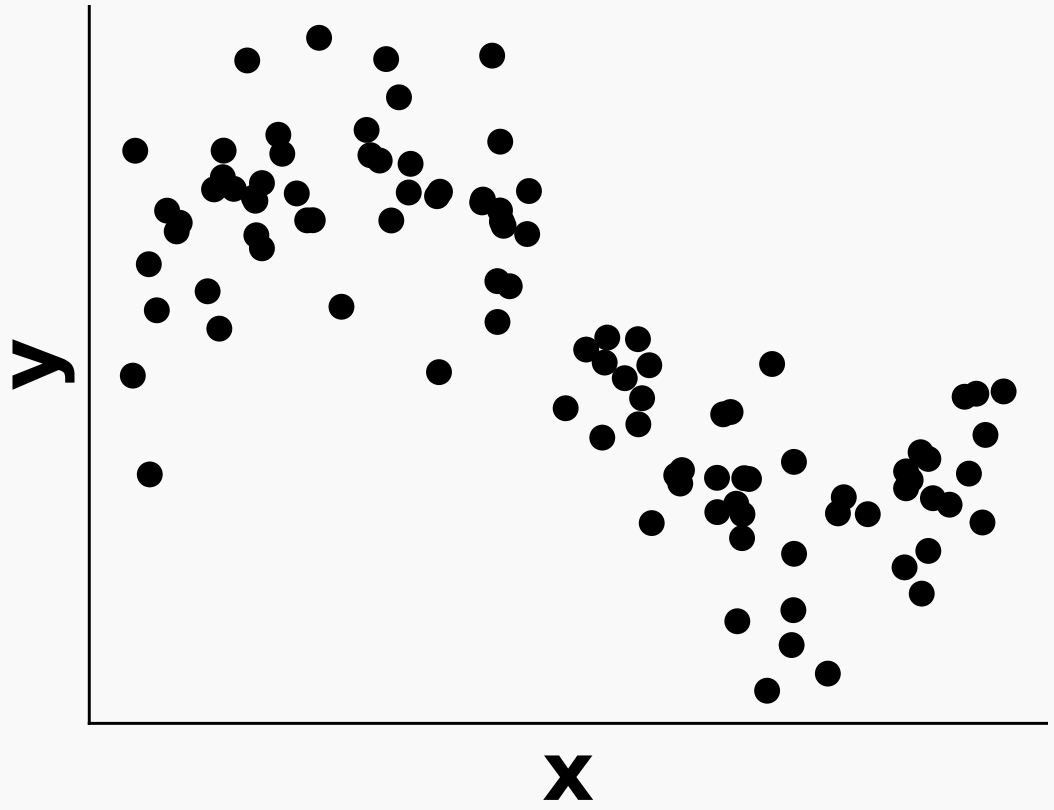


Three bootstrap samples

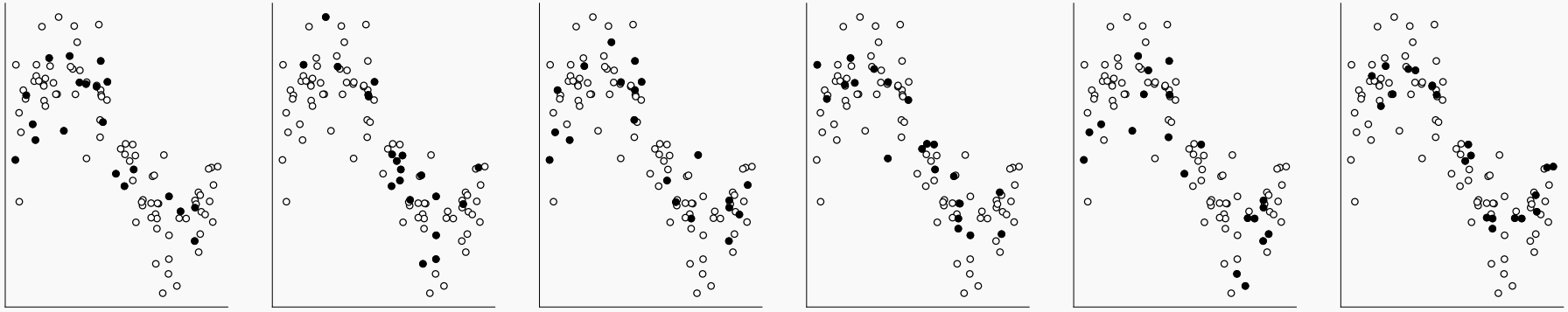




Random forests: Bagging with regression trees

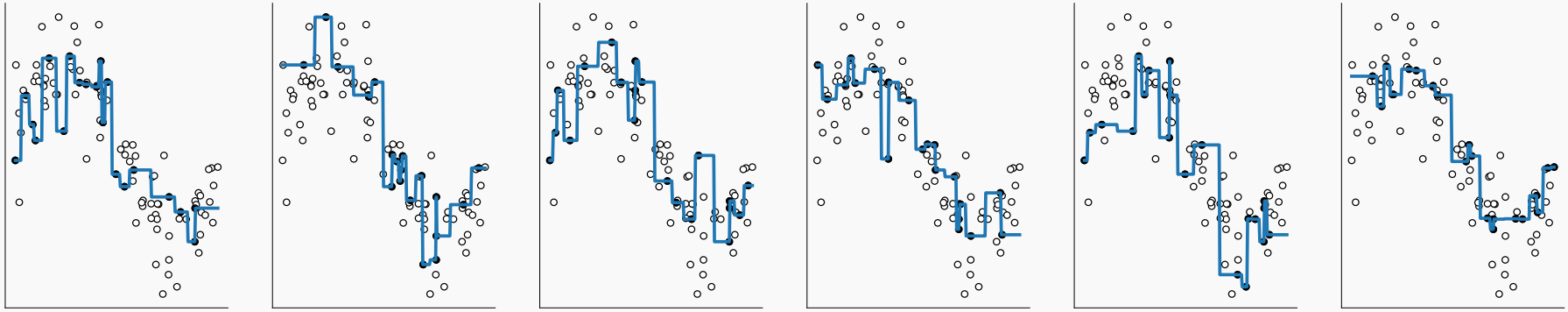


Random forests: Bagging with regression trees



Bootstrap multiple subsets

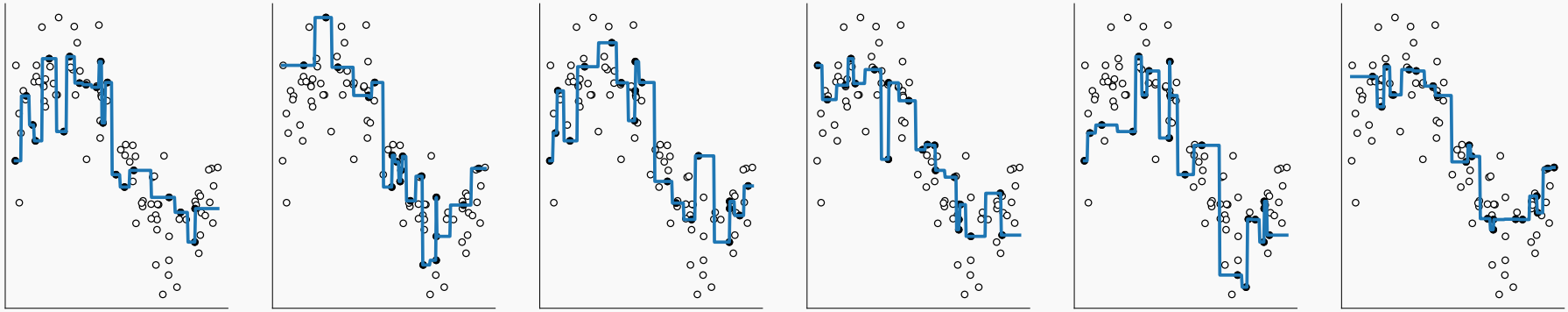
Random forests: Bagging with regression trees



Bootstrap multiple subsets

Fit one model to each subset

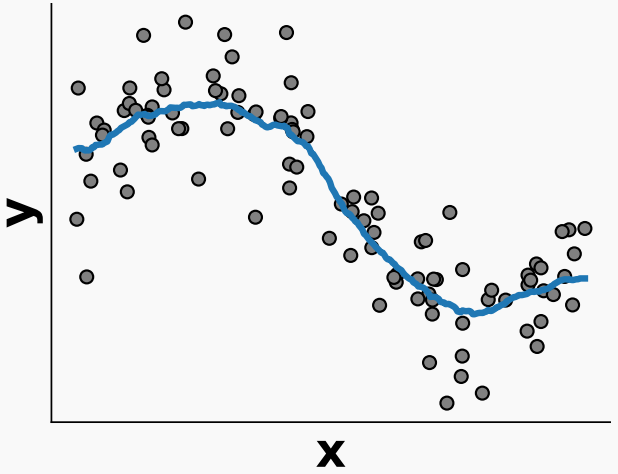
Random forests: Bagging with regression trees



Bootstrap multiple subsets

Fit one model to each subset

Average the predictions



Main hyper-parameters of random forests

```python
sklearn.ensemble.RandomForestRegressor(
  n_estimators=100, # Number of trees to fit (sample randomization): not useful to tune in practice
  criterion='squared_error',
  max_depth=None, # tree regularization
  min_samples_split=2, # tree regularization
  min_samples_leaf=1, # tree regularization
  min_impurity_decrease=0.0, # tree regularization
  n_jobs=None, # Number of jobs to run in parallel
  random_state=None, # Seed for randomization
  max_features=1.0, # Number/ratio of features at each split (feature randomization)
  max_samples = None # Number of sample to draw (with replacement) for each tree
 )
```

Random forests

- For each tree a random subset of samples are selected
- At each split a random subset of features are selected (more randomization)
- The best split is taken among the restricted subset
- Feature randomization decorrelates the prediction errors
- Uncorrelated errors make bagging work better

Random forests

- For each tree a random subset of samples are selected
- At each split a random subset of features are selected (more randomization)
- The best split is taken among the restricted subset
- Feature randomization decorrelates the prediction errors
- Uncorrelated errors make bagging work better

Take away

- Bagging and random forests fit trees independently
- Each deep tree overfits individually
- Averaging the tree predictions reduces overfitting

Boosting use multiple iterative models

- Use of simple underfitting models: eg. shallow trees
- Each model corrects the errors of the previous one

Boosting: Adaptive boosting

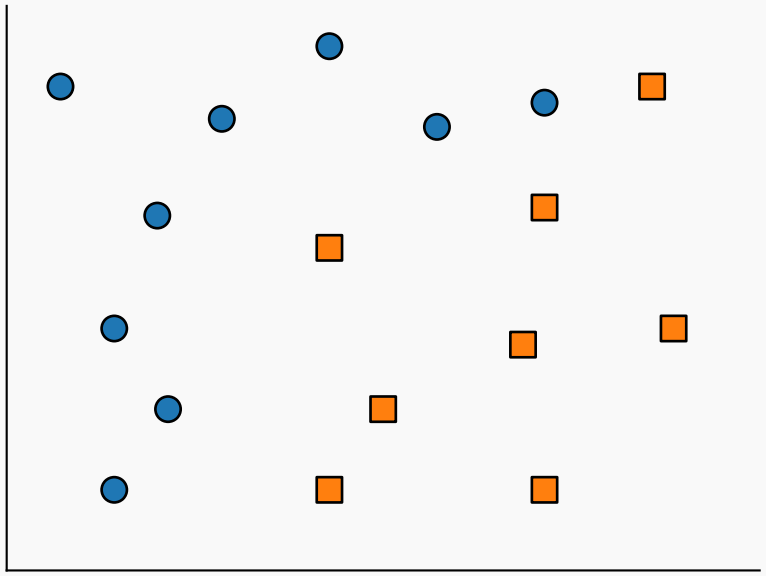
Boosting use multiple iterative models

- Use of simple underfitting models: eg. shallow trees
- Each model corrects the errors of the previous one

Two examples of boosting

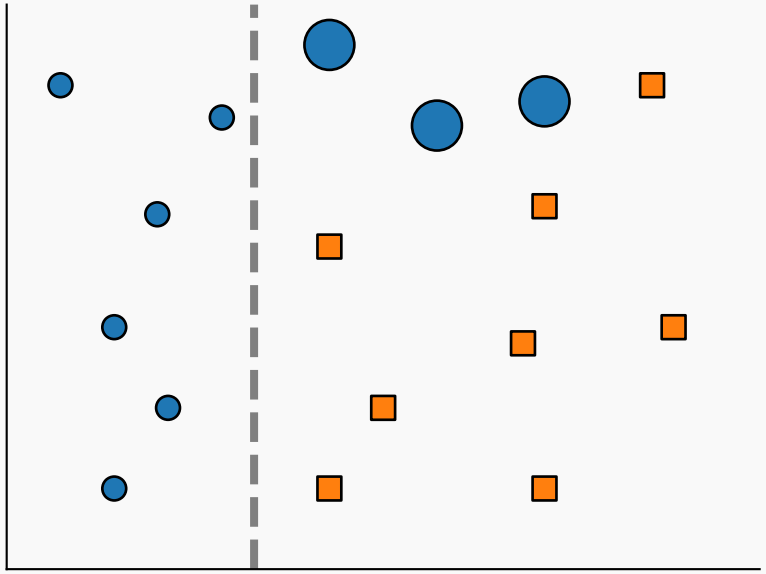
- Adaptive boosting (AdaBoost): reweight mispredicted samples at each step (Friedman et al., 2000)
- Gradient boosting: predict the negative errors of previous models at each step (Friedman, 2001)

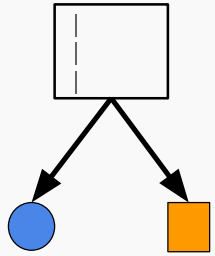
Boosting: Adaptive boosting, classification example



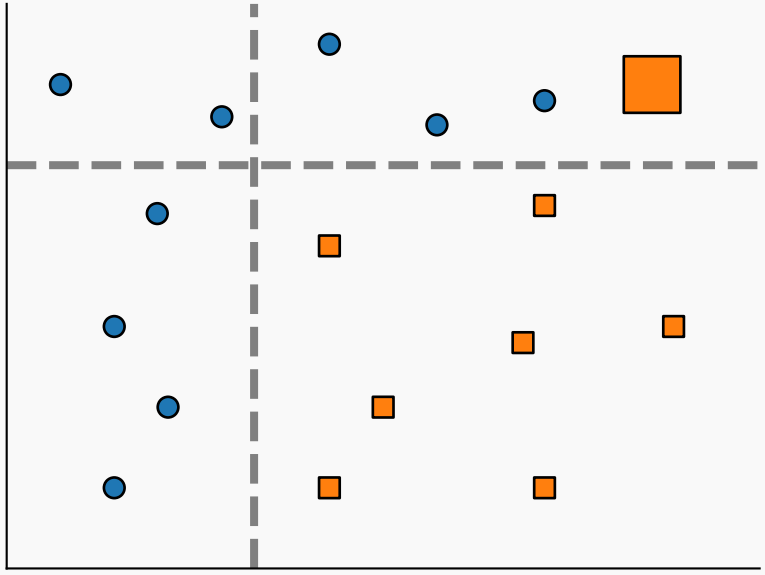
Boosting: Adaptive boosting, classification example

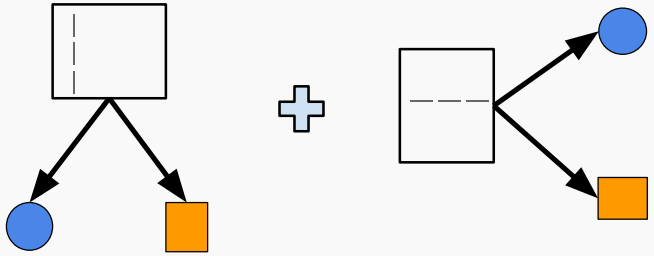
First prediction:



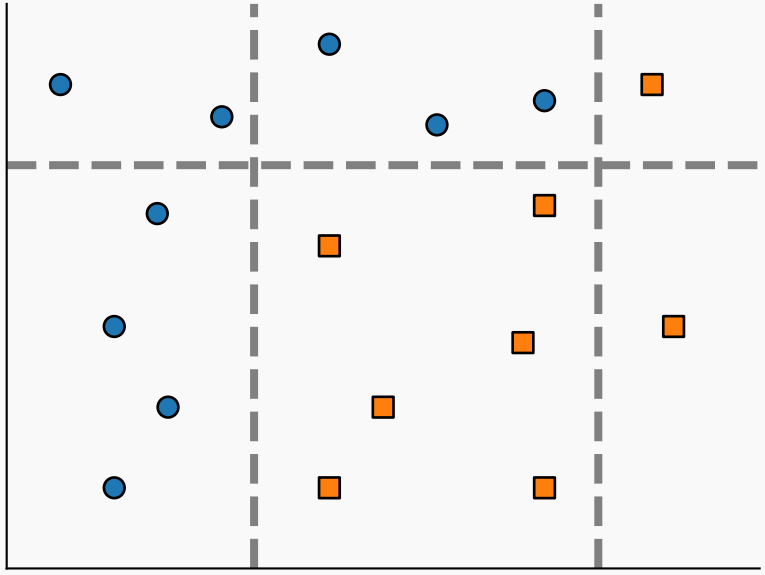


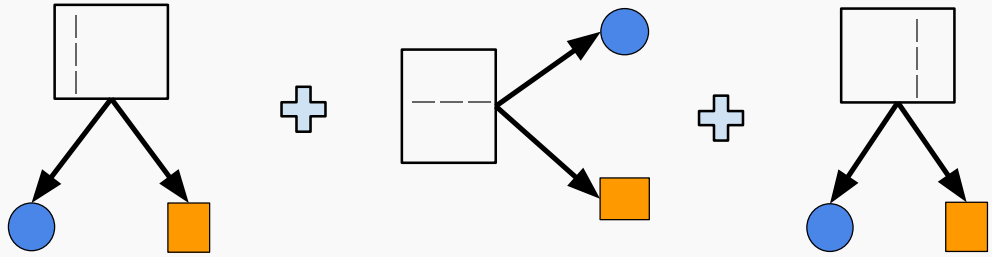
Boosting: Adaptive boosting, classification example





Boosting: Adaptive boosting, classification example





At each step, AdaBoost weights mispredicted samples

Adaboost for classification: choice of the weight

🔎 **Motivation in (Murphy, 2022)**

1. Initialize the observation weights $w_i = \frac{1}{N}, i = 1..N$
2.

Adaboost for classification: choice of the weight

🔎 **Motivation in (Murphy, 2022)**

1. Initialize the observation weights $w_i = \frac{1}{N}, i = 1..N$
2. For m = 1 to M (iterate):

Adaboost for classification: choice of the weight

🔎 **Motivation in (Murphy, 2022)**

1. Initialize the observation weights $w_i = \frac{1}{N}, i = 1..N$
2. For m = 1 to M (iterate):

- Fit a classifier $F_m(x)$ to the training data using weights w_i

Adaboost for classification: choice of the weight

🔎 **Motivation in (Murphy, 2022)**

1. Initialize the observation weights $w_i = \frac{1}{N}, i = 1..N$
2. For m = 1 to M (iterate):

- Fit a classifier $F_m(x)$ to the training data using weights w_i
- Compute $\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbb{1}[y_i \neq F_m(x_i)]}{\sum_{i=1} w_i}$

Adaboost for classification: choice of the weight

🔎 **Motivation in (Murphy, 2022)**

1. Initialize the observation weights $w_i = \frac{1}{N}, i = 1..N$
2. For m = 1 to M (iterate):

- Fit a classifier $F_m(x)$ to the training data using weights w_i
- Compute $\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbb{1}[y_i \neq F_m(x_i)]}{\sum_{i=1} w_i}$
- Compute $\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$

Adaboost for classification: choice of the weight

🔎 **Motivation in (Murphy, 2022)**

1. Initialize the observation weights $w_i = \frac{1}{N}, i = 1..N$
2. For m = 1 to M (iterate):

- Fit a classifier $F_m(x)$ to the training data using weights w_i
- Compute $\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbb{1}[y_i \neq F_m(x_i)]}{\sum_{i=1} w_i}$
- Compute $\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$
- Set $w_i \rightarrow w_i \exp[\alpha_m \mathbb{1}[y_i \neq F_m(x_i)], i = 1..N$

Adaboost for classification: choice of the weight

🔎 **Motivation in (Murphy, 2022)**

1. Initialize the observation weights $w_i = \frac{1}{N}, i = 1..N$
2. For m = 1 to M (iterate):

- Fit a classifier $F_m(x)$ to the training data using weights w_i
- Compute $\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbb{1}[y_i \neq F_m(x_i)]}{\sum_{i=1} w_i}$
- Compute $\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$
- Set $w_i \rightarrow w_i \exp[\alpha_m \mathbb{1}[y_i \neq F_m(x_i)], i = 1..N$
- Output $F(x) = \text{sign}\left(\sum_{i=1}^{M} \alpha_m G_{m(x)}\right)$

Adaboost for classification: choice of the weight

🔎 **Motivation in (Murphy, 2022)**

1. Initialize the observation weights $w_i = \frac{1}{N}, i = 1..N$
2. For m = 1 to M (iterate):

- Fit a classifier $F_m(x)$ to the training data using weights w_i
- Compute $\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbb{1}[y_i \neq F_m(x_i)]}{\sum_{i=1} w_i}$
- Compute $\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$
- Set $w_i \rightarrow w_i \exp[\alpha_m \mathbb{1}[y_i \neq F_m(x_i)], i = 1..N$
- Output $F(x) = \text{sign}\left(\sum_{i=1}^{M} \alpha_m G_{m(x)}\right)$

Adaboost: Take-away

- Sequentially fit weak learners (eg. shallow trees)
- Each new learner corrects the errors of the previous one thanks to sample weights
- The final model is a weighted sum of the weak learners
- The weights are learned by the algorithm to given more importance to errors
- Any weak learner can be used

Adaboost: Take-away

- Sequentially fit weak learners (eg. shallow trees)
- Each new learner corrects the errors of the previous one thanks to sample weights
- The final model is a weighted sum of the weak learners
- The weights are learned by the algorithm to given more importance to errors
- Any weak learner can be used

Adaboost is tailored to a specific loss function (exponential loss)

🤔 Can we exploit the boosting idea for any loss function?

Gradient boosting: how to choose the iterative learners?

Boosting formulation

$F_{m(x)} = F_{m-1}(x) + h_{m(x)}$ with F_{m-1} the previous estimator, h_m, new week learner.

Minimization problem

$h_m = \text{argmin}_h(L_m) = \text{argmin}_h \sum_{i=1}^{n} l(y_i, F_{m-1}(x_i) + h(x_i))$

Expand the loss inside the sum using a Taylor expansion.

Gradient boosting: how to choose the iterative learners?

Boosting formulation

$F_{m(x)} = F_{m-1}(x) + h_{m(x)}$ with F_{m-1} the previous estimator, h_m, new week learner.

Minimization problem

$h_m = \text{argmin}_h(L_m) = \text{argmin}_h \sum_{i=1}^{n} l(y_i, F_{m-1}(x_i) + h(x_i))$

Expand the loss inside the sum using a Taylor expansion.

🔔 Taylor expansion

For $l(\cdot)$ differentiable: $l(y + h) \approx l(y) + h\frac{\partial l}{\partial y}(y)$

Gradient boosting: how to choose the iterative learners?

Boosting formulation

$F_{m(x)} = F_{m-1}(x) + h_{m(x)}$ with F_{m-1} the previous estimator, h_m, new week learner.

Minimization problem

$h_m = \text{argmin}_h(L_m) = \text{argmin}_h \sum_{i=1}^{n} l(y_i, F_{m-1}(x_i) + h(x_i))$

Expand the loss inside the sum using a Taylor expansion.

$$l(y_i, F_{m-1}(x_i) + h(x_i)) = l(y_i, F_{m-1}(x_i)) + h(x_i) \left[\frac{\partial l(y_i, F(x_i))}{\partial F(x_i)} \right]_{F=F_{m-1}}$$

🔔 Taylor expansion

For $l(\cdot)$ differentiable: $l(y + h) \approx l(y) + h \frac{\partial l}{\partial y}(y)$

Gradient boosting: how to choose the iterative learners?

Boosting formulation

$F_{m(x)} = F_{m-1}(x) + h_{m(x)}$ with F_{m-1} the previous estimator, h_m, new week learner.

Minimization problem

$h_m = \text{argmin}_h(L_m) = \text{argmin}_h \sum_{i=1}^{n} l(y_i, F_{m-1}(x_i) + h(x_i))$

Expand the loss inside the sum using a Taylor expansion.

$$l(y_i, F_{m-1}(x_i) + h(x_i)) = \underbrace{l(y_i, F_{m-1}(x_i))}_{\text{constant in } h(x_i)} + h(x_i) \underbrace{\left[\frac{\partial l(y_i, F(x_i))}{\partial F(x_i)}\right]_{F=F_{m-1}}}_{\stackrel{\text{def}}{=} g_i, \text{ the gradient}}$$

Gradient boosting: how to choose the iterative learners?

Boosting formulation

$F_{m(x)} = F_{m-1}(x) + h_{m(x)}$ with F_{m-1} the previous estimator, h_m, new week learner.

Minimization problem

$h_m = \text{argmin}_h(L_m) = \text{argmin}_h \sum_{i=1}^{n} l(y_i, F_{m-1}(x_i) + h(x_i))$

Expand the loss inside the sum using a Taylor expansion.

$$l(y_i, F_{m-1}(x_i) + h(x_i)) = \underbrace{l(y_i, F_{m-1}(x_i))}_{\text{constant in } h(x_i)} + h(x_i) \underbrace{\left[\frac{\partial l(y_i, F(x_i))}{\partial F(x_i)}\right]_{F=F_{m-1}}}_{\stackrel{\text{def}}{=} g_i, \text{ the gradient}}$$

Finally: $h_m = \text{argmin}_h \sum_{i=1}^{n} h(x_i) g_i$ -> kind of an inner product $< g, h >$

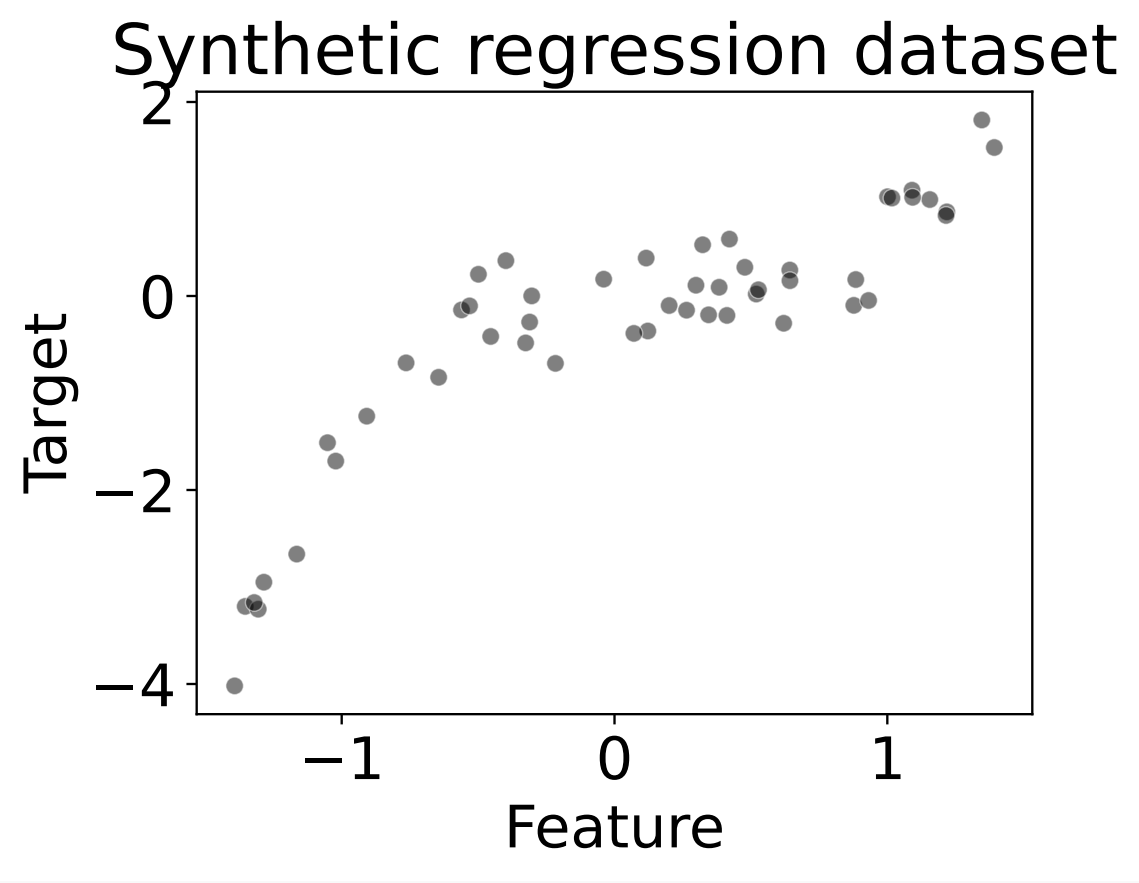
So $h_{m(x_i)}$ should be proportional to $-g_i$, so fit h_m to the negative gradient.

Boosting: Gradient boosting, regression example

Regression

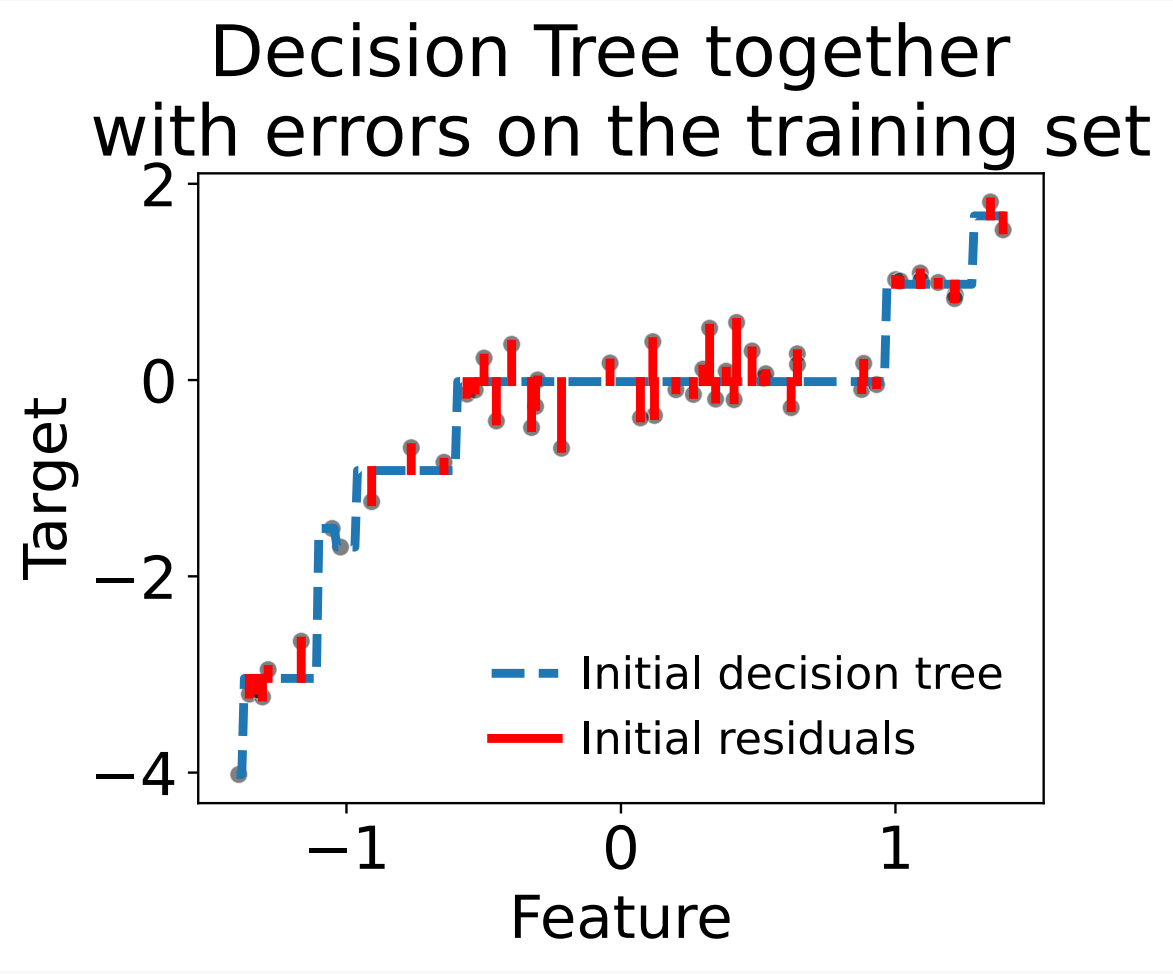
- The loss is: $l(y, F(x)) = (y - F(x))^2$
- The gradient is: $g_i = -2(y_i - F_{m-1}(x_i))$

💡 The new tree should fit the residuals



Boosting: Gradient boosting, regression example

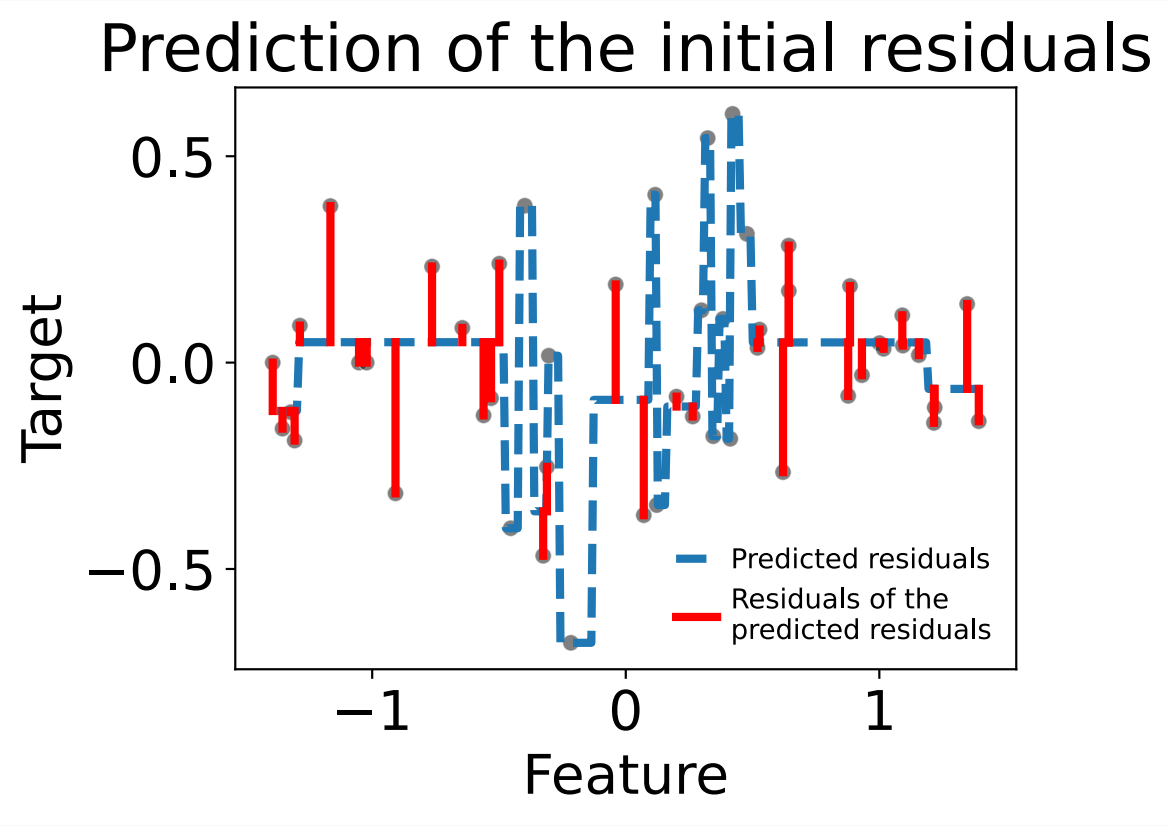
Fit a shallow tree (depth=3)



Boosting: Gradient boosting, regression example

Fit a second tree to the residuals

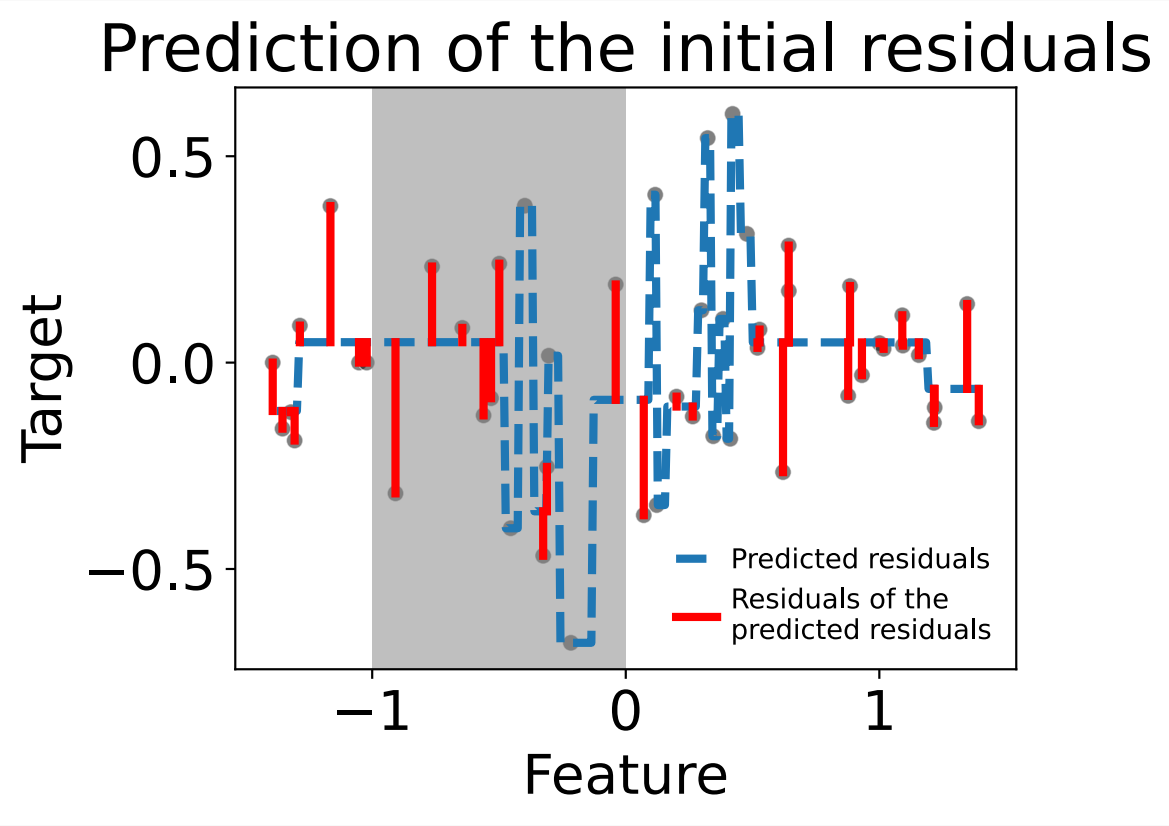
- This tree performs poorly on some samples.



Boosting: Gradient boosting, regression example

Fit a second tree to the residuals

- This tree performs well on some residuals.
- Let's zoom on one of those.



Boosting: Gradient boosting, regression example

Focus on a sample

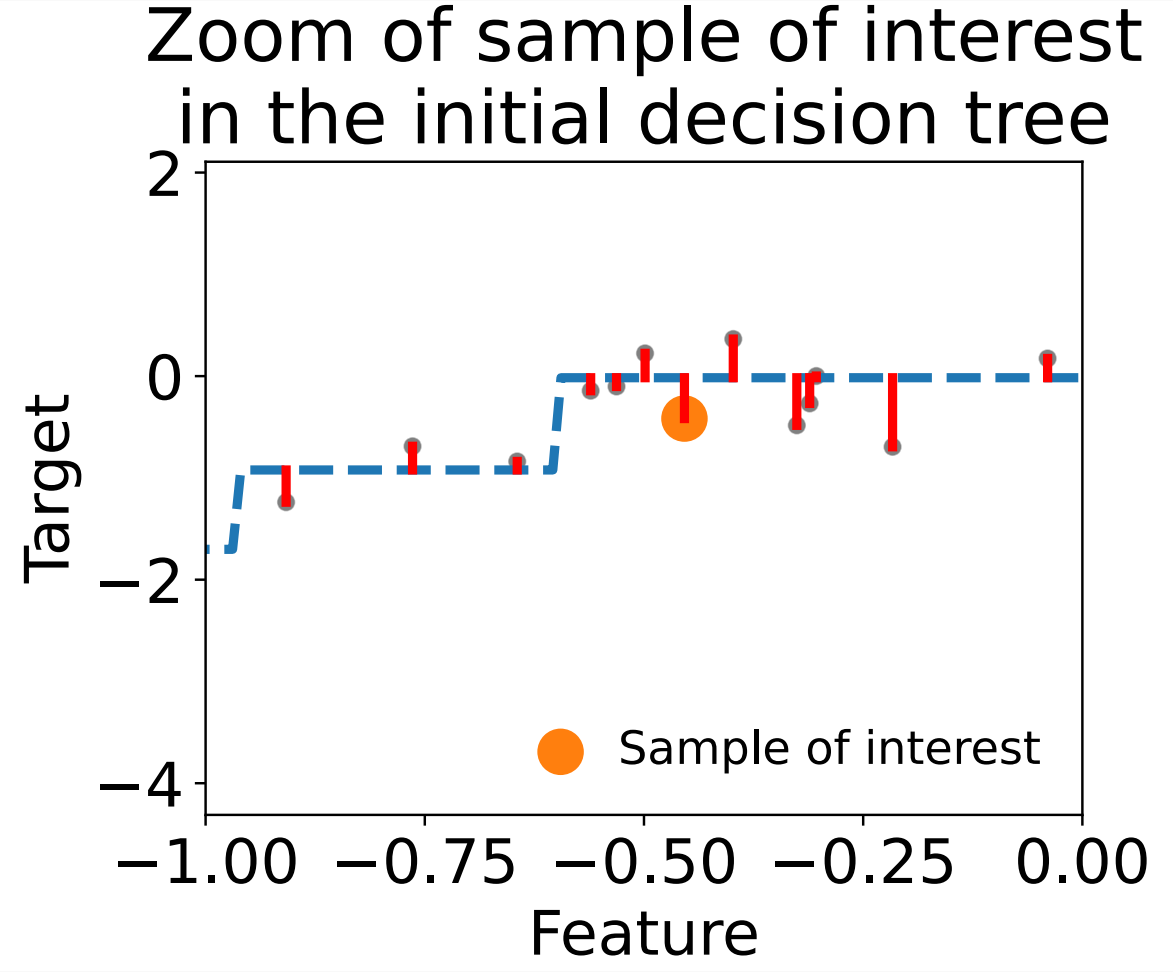
$(x_i, y_i) = (-0.454, -0.417)$

First tree prediction

Prediction: $f_1(x_i) = -0.016$

Residuals:

$y_i - f_1(x_i) = -0.401$



Boosting: Gradient boosting, regression example

Focus on a sample

$(x_i, y_i) = (-0.454, -0.417)$

First tree prediction

Prediction: $f_1(x_i) = -0.016$

Residuals:

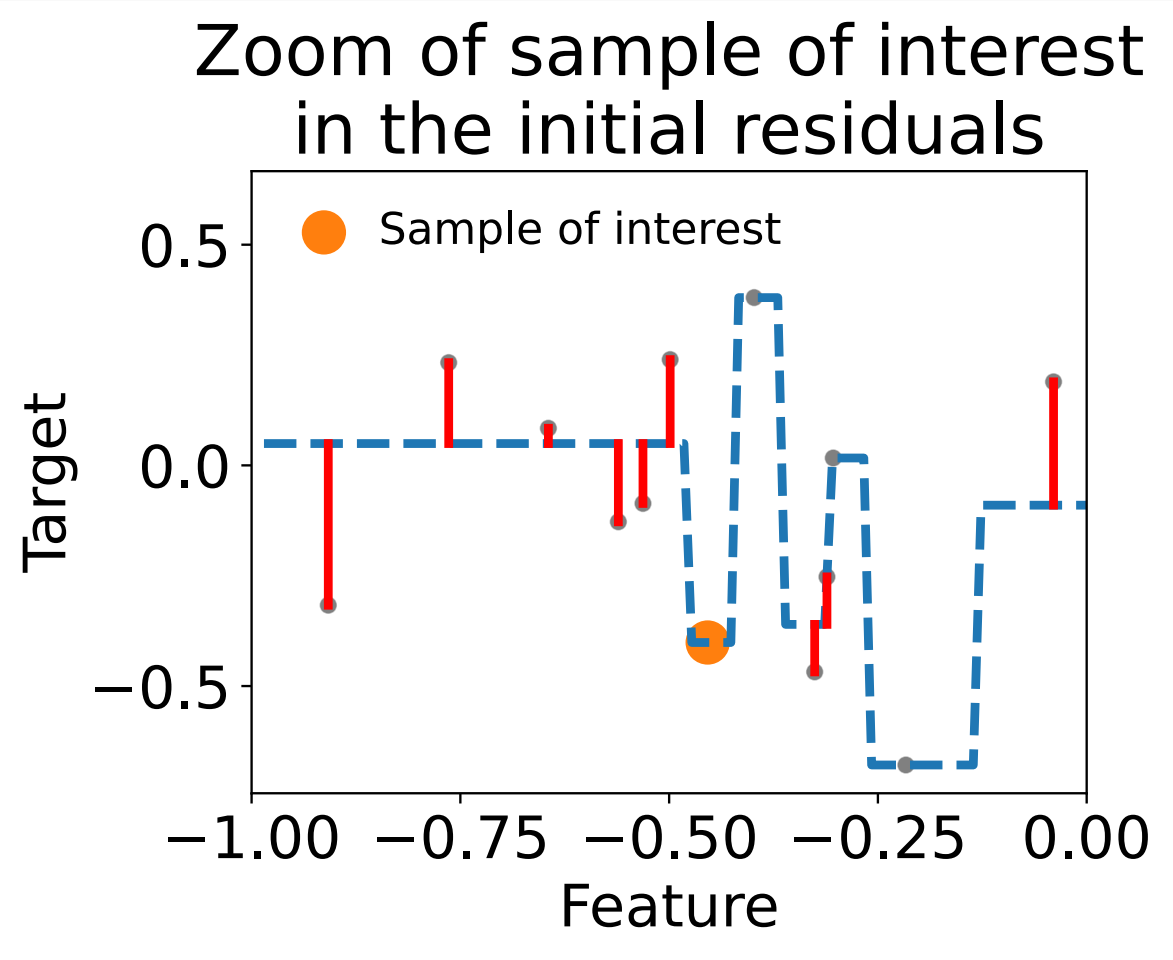
$y_i - f_1(x_i) = -0.401$

Second tree prediction

Prediction: $f_2(x_i) = -0.401$

Residuals:

$y_i - f_1(x_i) - f_2(x_i) = 0$



Faster gradient boosting with binned features

😭 **Gradient boosting is slow when N>10,000**

Fitting each tree is quite slow: $O(pN \log(N))$ operations

Faster gradient boosting with binned features

😭 **Gradient boosting is slow when N>10,000**

Fitting each tree is quite slow: $O(pN\log(N))$ operations

🚀 **XGBoost: eXtreme Gradient Boosting (Chen & Guestrin, 2016)**

- Missing values support
- Parallelization
- Second order Taylor expansion

Faster gradient boosting with binned features

😭 **Gradient boosting is slow when N>10,000**

Fitting each tree is quite slow: $O(pN \log(N))$ operations

🚀 **XGBoost: eXtreme Gradient Boosting (Chen & Guestrin, 2016)**

- Missing values support
- Parallelization
- Second order Taylor expansion

🚀 **HistGradientBoosting: sklearn implementation of lightGBM (Ke et al., 2017)**

- Missing values support
- Parallelization
- Discretize numerical features into 256 bins: less costly for tree splitting

Take away for ensemble models

Bagging (eg. Random forests)	Boosting
Fit trees independently	Fit trees sequentially
Each deep tree overfits	Each shallow tree underfits
Averaging the tree predictions reduces overfitting	Sequentially adding trees reduces underfitting

A word on other families of models

Generalized linear models

Link an OLS ($X\beta$) to the parameters of various probability distributions.

Examples: Poisson regression (for count data), logistic regression.

Other well known families of models

Generalized linear models

Link an OLS ($X\beta$) to the parameters of various probability distributions.

Examples: Poisson regression (for count data), logistic regression.

Kernels: support vector machines, gaussian processes

Local methods with appropriate basis functions (kernels).

Kernels are often chosen with expert knowledge.

Other well known families of models

Generalized linear models

Link an OLS ($X\beta$) to the parameters of various probability distributions.
Examples: Poisson regression (for count data), logistic regression.

Kernels: support vector machines, gaussian processes

Local methods with appropriate basis functions (kernels).
Kernels are often chosen with expert knowledge.

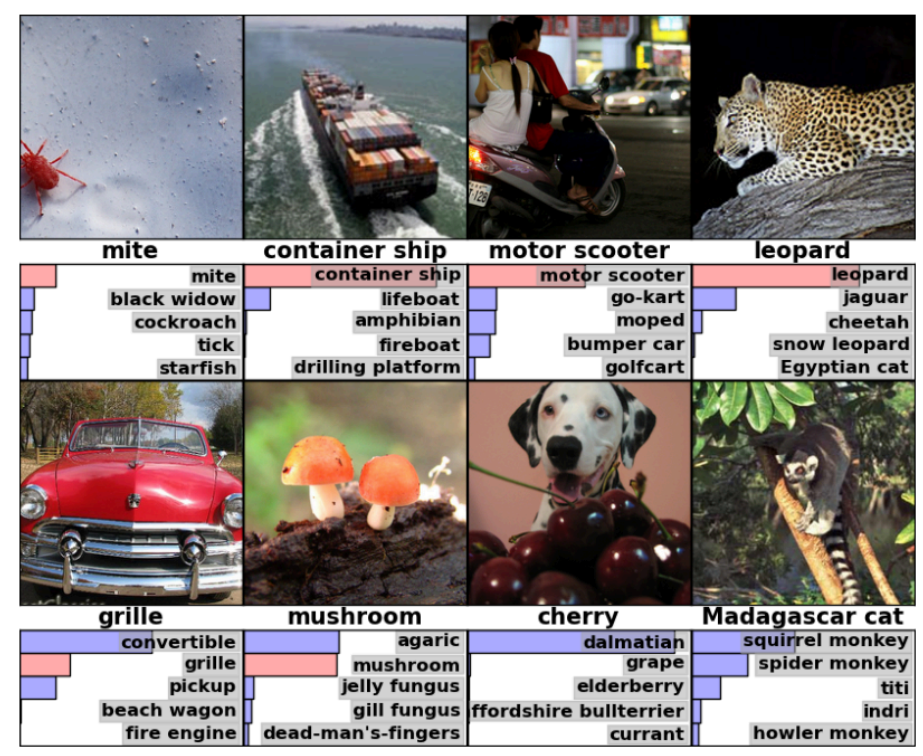
Deep neural networks (deep learning)

Iterative layers of parametrized basis functions: eg. $\mathbb{1}[wX + b \geq 0]$
Trainable by gradient descent: each layer should be differentiable.
Training thanks to backpropagation ie. automatic differentiation and gradient methods.

A word on deep learning

Success of deep learning

- For images: Convolutional Neural Network (CNN) architecture (Russakovsky et al., 2015),
- For text: transformer architecture (Vaswani, 2017),
- For protein folding: transformer architecture (Jumper et al., 2021)



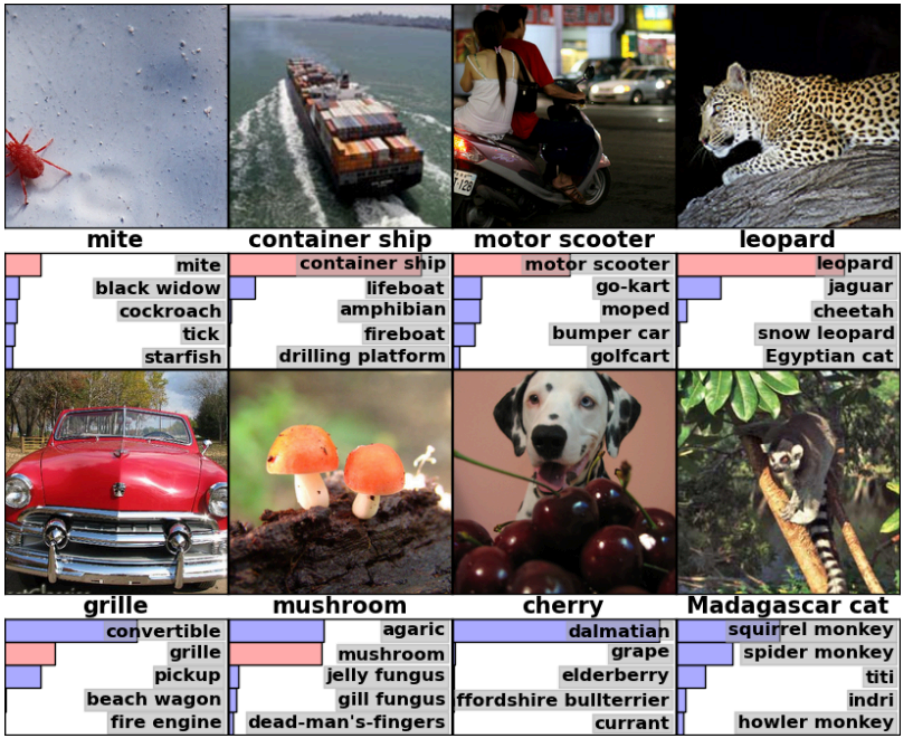
Imagenet challenge (Russakovsky et al., 2015)

A word on deep learning

Success of deep learning

- For images: Convolutional Neural Network (CNN) architecture (Russakovsky et al., 2015),
- For text: transformer architecture (Vaswani, 2017),
- For protein folding: transformer architecture (Jumper et al., 2021)

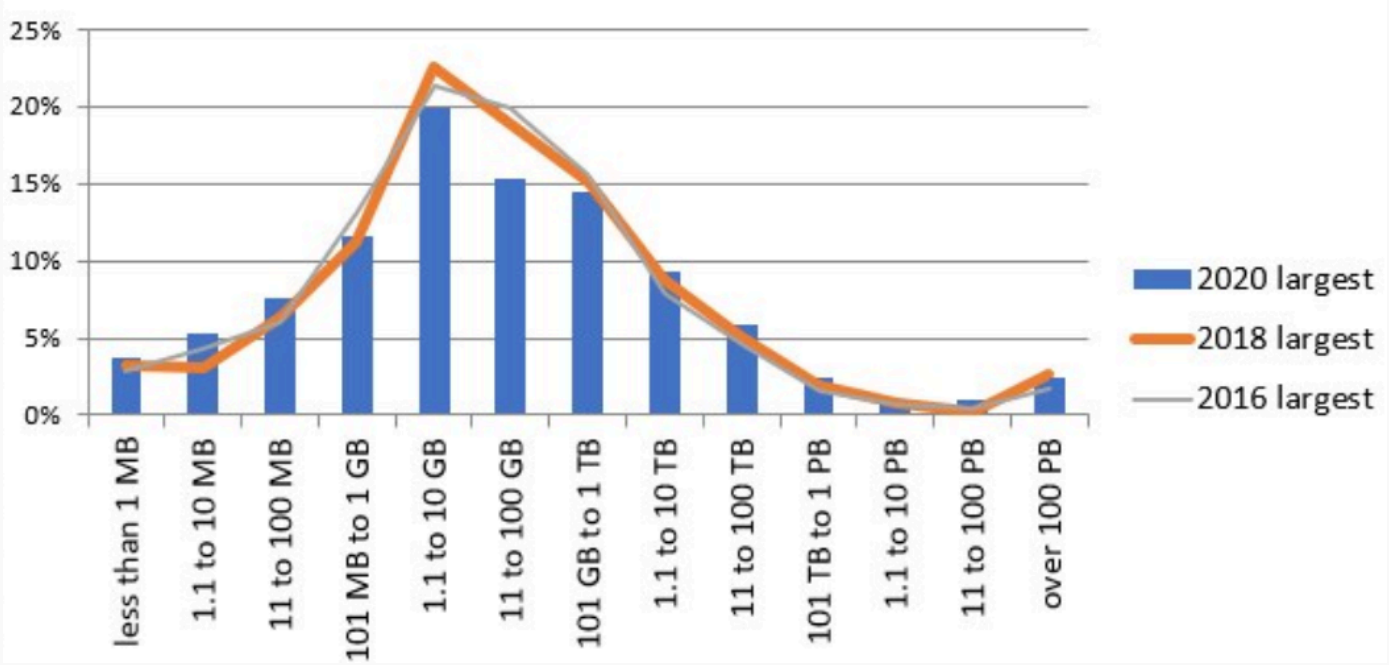
🤔 Why not so used in econometrics?



Imagenet challenge (Russakovsky et al., 2015)

Answer 1: Limited data settings (typically $N \approx 1$ million)

- Typically in economics (but also in industry), we have a limited number of observations

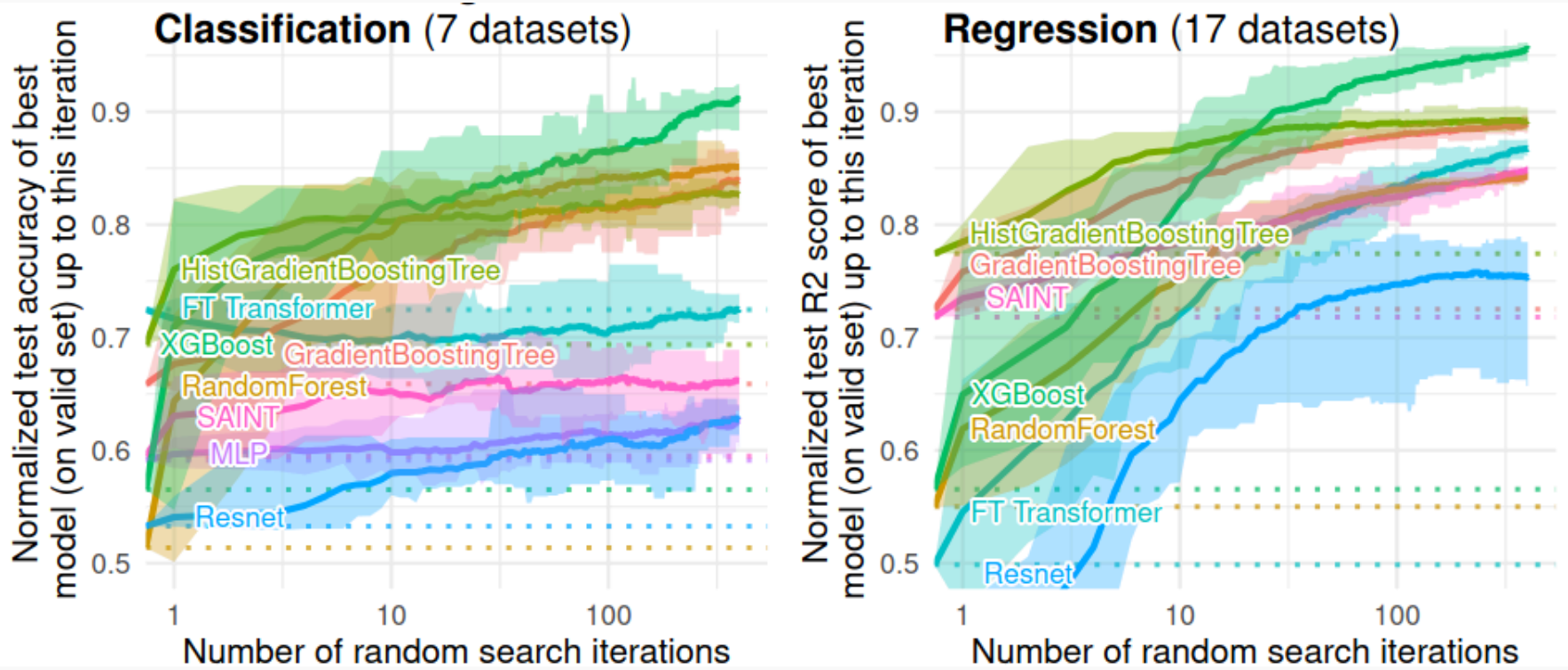


Typical dataset are mid-sized. This does not change with time.[1]

[1] https://www.kdnuggets.com/2020/07/poll-largest-dataset-analyzed-results.html

Answer 2: Deep learning underperforms on data tables

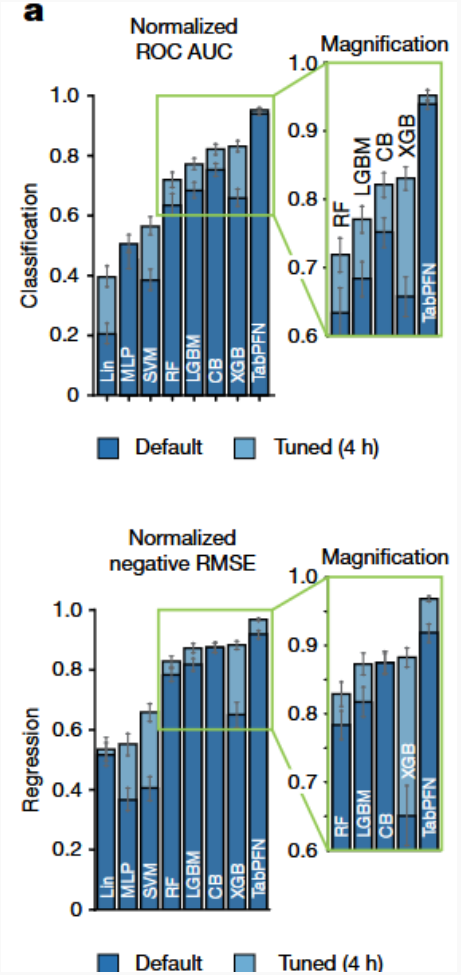
Tailored deep learning architectures lack appropriate prior of tabular data (Grinsztajn et al., 2022)



Nuance: recent work using deep learning for tabular data

Learning appropriate representations (prior) of tabular data

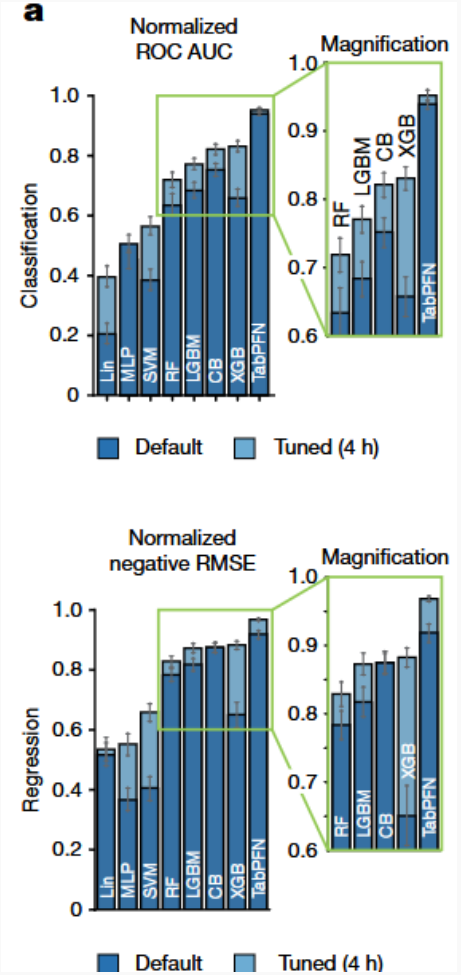
- TabPFN: large scale pretraining a transformer based model on synthetic tabular data (Hollmann et al., 2025)
- Allows In-Context Learning (ICL): learn with few examples.
-



Nuance: recent work using deep learning for tabular data

Learning appropriate representations (prior) of tabular data

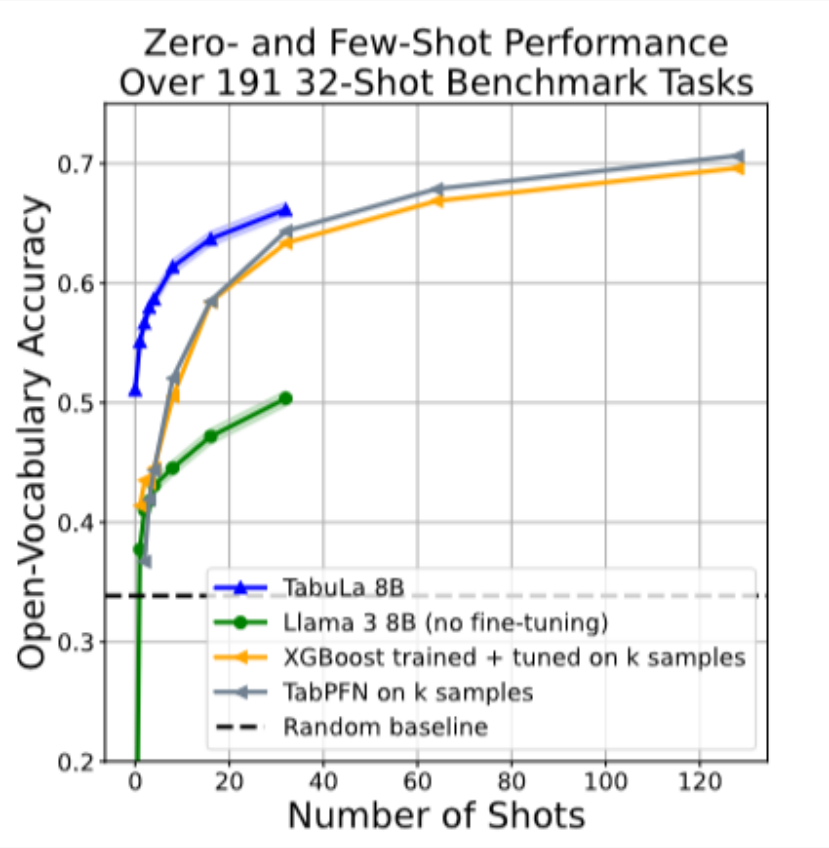
- TabPFN: large scale pretraining a transformer based model on synthetic tabular data (Hollmann et al., 2025)
- Allows In-Context Learning (ICL): learn with few examples.
- (Hollmann et al., 2025) Figure 4a: Comparison on test benchmarks, 29 classification and 28 regression datasets, containing with up to 10,000 samples and 500 features.



Nuance: recent work using deep learning for tabular data

Using Large Language Models (LLM)

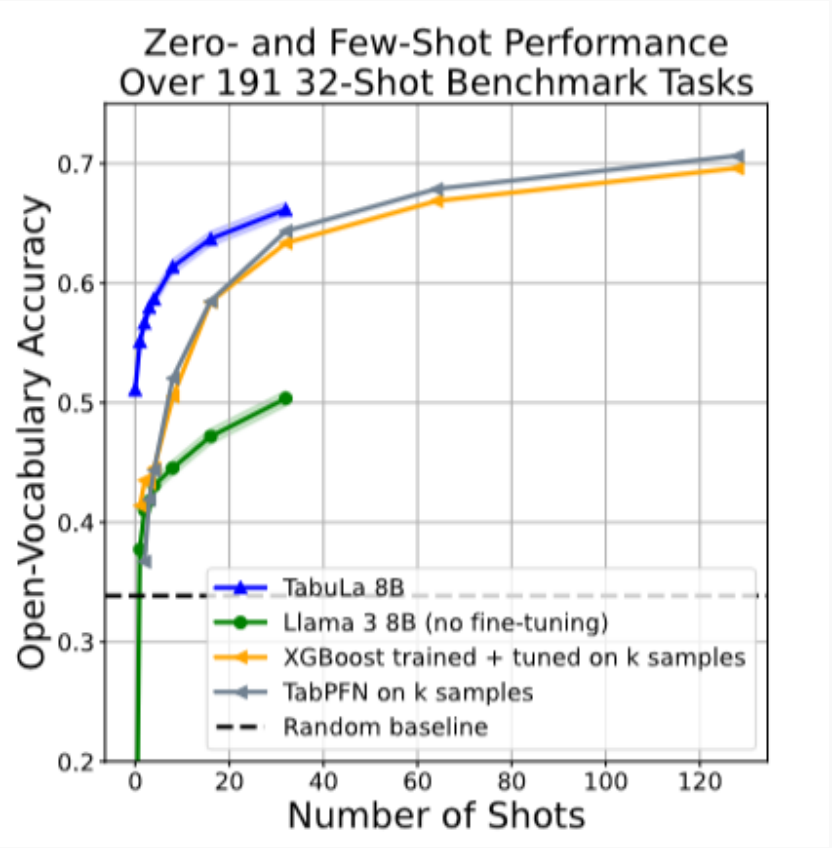
- Tabula 8B Fine-tuning existing LLM (Llama 3-8B) on tabular data (Gardner et al., 2024)
-
-



Nuance: recent work using deep learning for tabular data

Using Large Language Models (LLM)

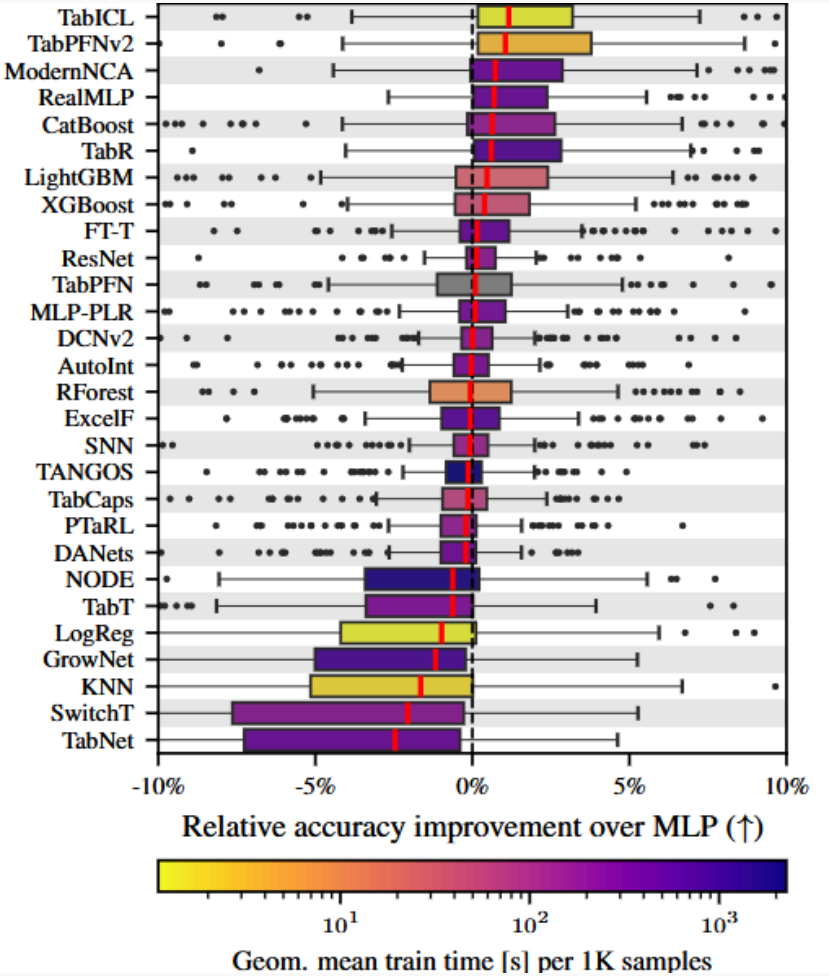
- Tabula 8B Fine-tuning existing LLM (Llama 3-8B) on tabular data (Gardner et al., 2024)
- Allow ICL with few examples.
- But requires large computational resources and is outperform rapidly when number of samples grows.



Nuance: recent work using deep learning for tabular data

Transferable components tailored to tabular data

- CARTE: tailored learning components such as `{key:value}` representations (Kim et al., 2024)
- TABICL: Combine tailored components and pretraining on synthetic data (Qu et al., 2025)
- (Qu et al., 2025) Figure 5: Benchmark accuracy results and train times on 200 classification datasets.



Python hands-on

To your notebooks 👩‍💻 !

- url: https://github.com/strayMat/causal-ml-course/tree/main/notebooks

Bibliography

Bouthillier, X., Delaunay, P., Bronzi, M., Trofimov, A., Nichyporuk, B., Szeto, J., Mohammadi Sepahvand, N., Raff, E., Madan, K., Voleti, V., & others. (2021). Accounting for variance in machine learning benchmarks. Proceedings of Machine Learning and Systems, 3, 747–769.

Breiman, L. (1996). Bagging predictors. Machine Learning, 24, 123–140.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining, 785–794.

Efron, B. (1992). Bootstrap methods: another look at the jackknife. In Breakthroughs in statistics: Methodology and distribution: Breakthroughs in statistics: Methodology and distribution (pp. 569–593). Springer.

Estève, L., Lemaitre, G., Grisel, O., Varoquaux, G., Amor, A., Lilian, Rospars, B., Schmitt, T., Liu, L., Kinoshita, B. P., hackmd-deploy, ph4ge, Steinbach, P., Boucaud, A., Muite, B., Boisberranger, J.

du, Notter, M., Pierre, P, S., … parmentelat. (2022). INRIA/scikit-learn-mooc: Third MOOC session. Zenodo. https://doi.org/10.5281/zenodo.7220307

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. Annals of Statistics, 1189–1232.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). The Annals of Statistics, 28(2), 337–407.

Gardner, J., Perdomo, J. C., & Schmidt, L. (2024). Large Scale Transfer Learning for Tabular Data via Language Modeling. Arxiv Preprint Arxiv:2406.12031.

Grinsztajn, L., Oyallon, E., & Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on typical tabular data?. Advances in Neural Information Processing Systems, 35, 507–520.

Hollmann, N., Müller, S., Purucker, L., Krishnakumar, A., Körfer, M., Hoo, S. B., Schirrmeister, R. T., & Hutter, F. (2025). Accurate predictions on small data with a tabular foundation model. Nature, 637(8045), 319–326.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Ž\idek, A., Potapenko, A., & others. (2021). Highly accurate protein structure prediction with AlphaFold. Nature, 596(7873), 583–589.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. Advances in Neural Information Processing Systems, 30.

Kim, M. J., Grinsztajn, L., & Varoquaux, G. (2024). CARTE: pretraining and transfer for tabular learning. Arxiv Preprint Arxiv:2402.16785.

Lecué, G., & Mitchell, C. (2012). Oracle inequalities for cross-validation type procedures.

Murphy, K. P. (2022). Probabilistic machine learning: an introduction. MIT press.

Qu, J., Holzmüller, D., Varoquaux, G., & Morvan, M. L. (2025). TabICL: A Tabular Foundation Model for In-Context Learning on Large Data. Arxiv Preprint Arxiv:2502.05564.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., & others. (2015). Imagenet large scale visual recognition challenge. International Journal of Computer Vision, 115, 211–252.

Varoquaux, G., Raamana, P. R., Engemann, D. A., Hoyos-Idrobo, A., Schwartz, Y., & Thirion, B. (2017). Assessing and tuning brain decoders: cross-validation, caveats, and guidelines. Neuroimage, 145, 166–179.

Vaswani, A. (2017). Attention is all you need. Advances in Neural Information Processing Systems.