Machine Learning for econometrics

Flexible models for tabular data

Matthieu Doutreligne

February 18th, 2025

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?

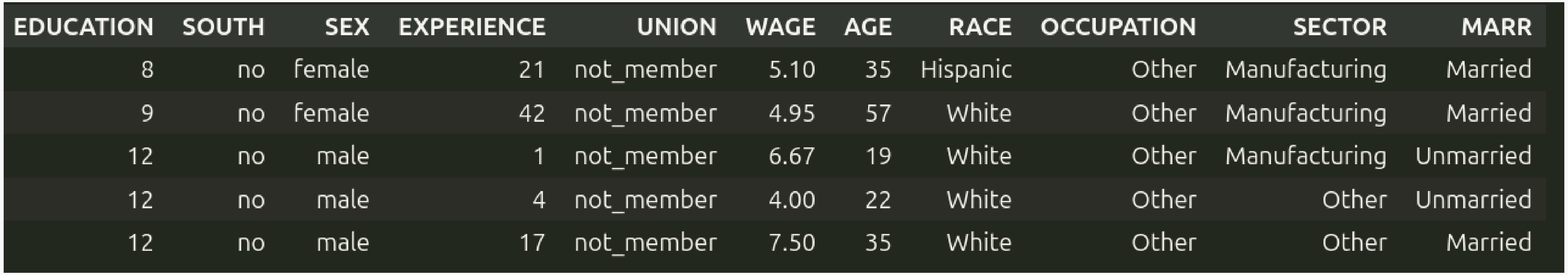
Table of contents

Model evaluation and selection with cross-validation

A closer look at model evaluation: Wage example

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

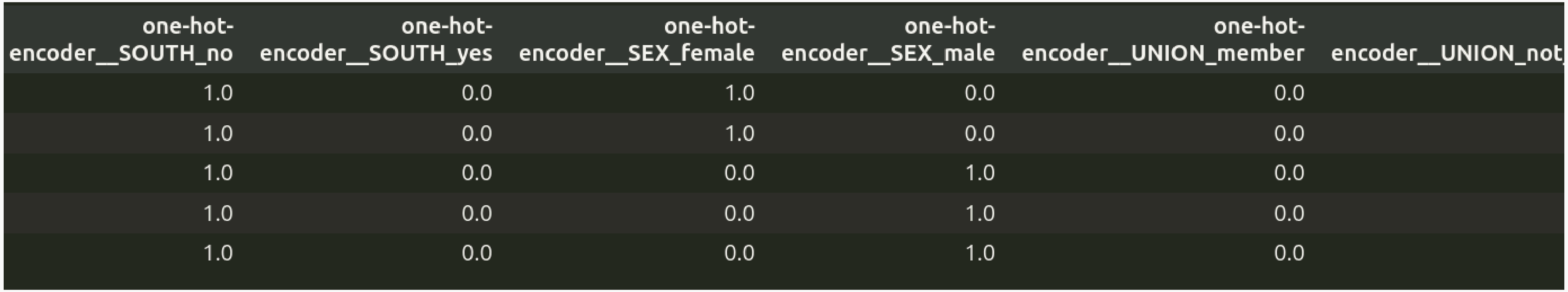
•

•

A closer look at model evaluation: Wage example

Example with the Wage dataset

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

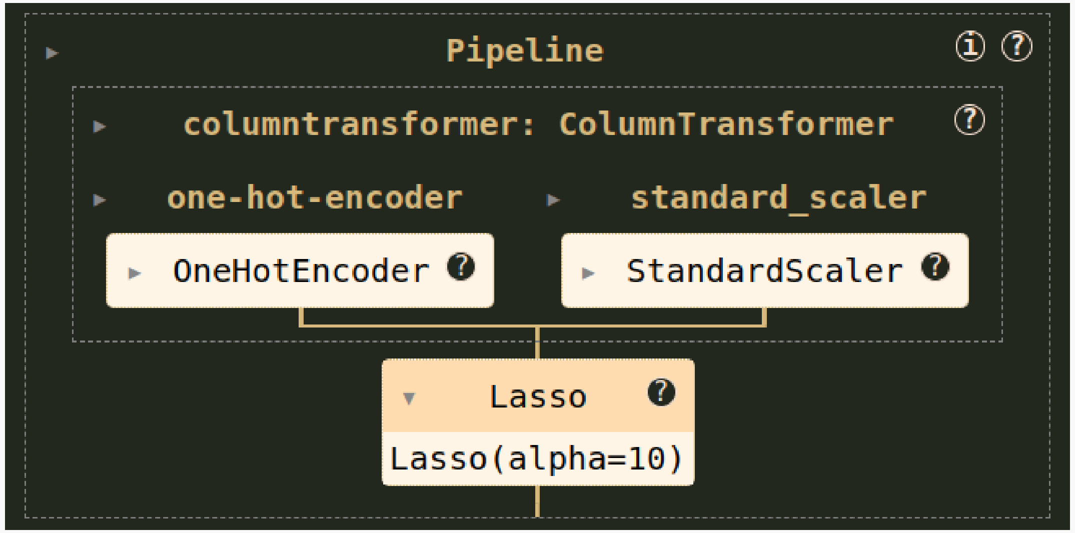


one-hot-encoder__SOUTH_no	one-hot-encoder__SOUTH_yes	one-hot-encoder__SEX_female	one-hot-encoder__SEX_male	one-hot-encoder__UNION_member	encoder__UNION_not
1.0	0.0	1.0	0.0	0.0	
1.0	0.0	1.0	0.0	0.0	
1.0	0.0	0.0	1.0	0.0	
1.0	0.0	0.0	1.0	0.0	
1.0	0.0	0.0	1.0	0.0	

A closer look at model evaluation: Wage example

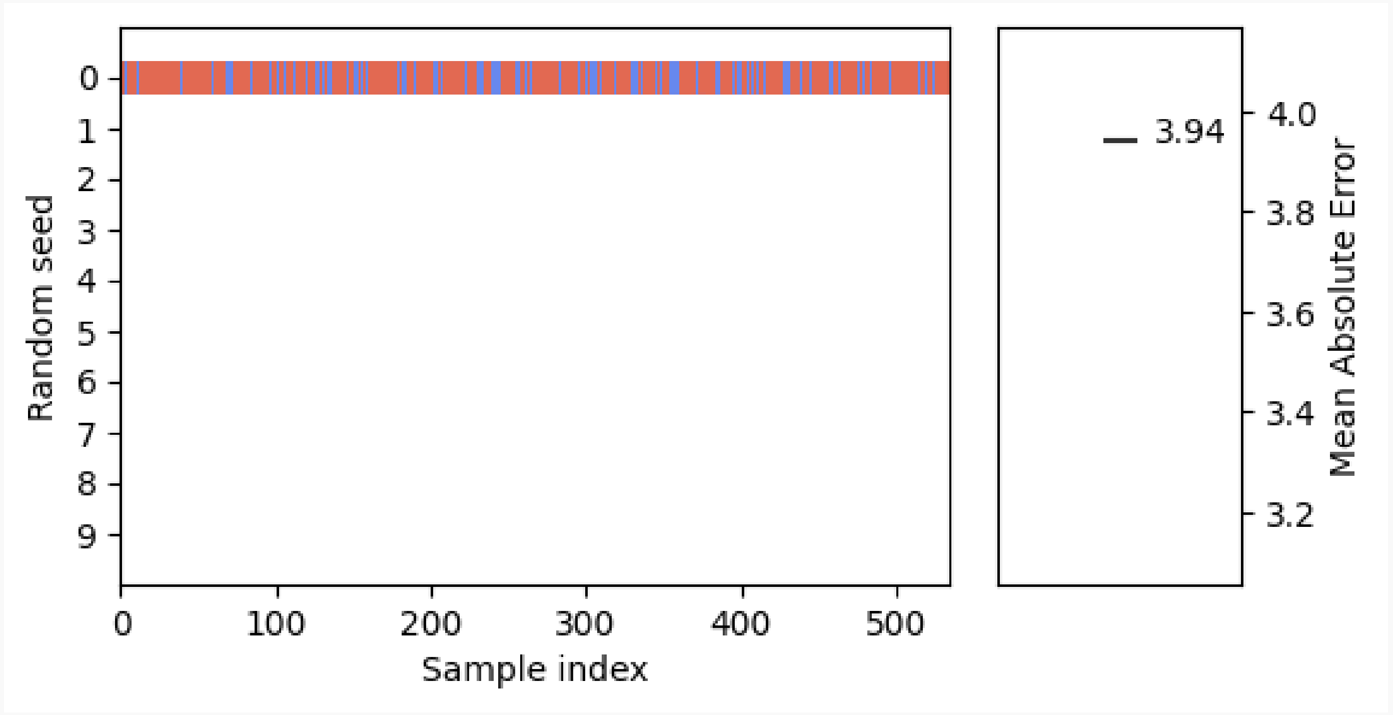
Example with the Wage dataset

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



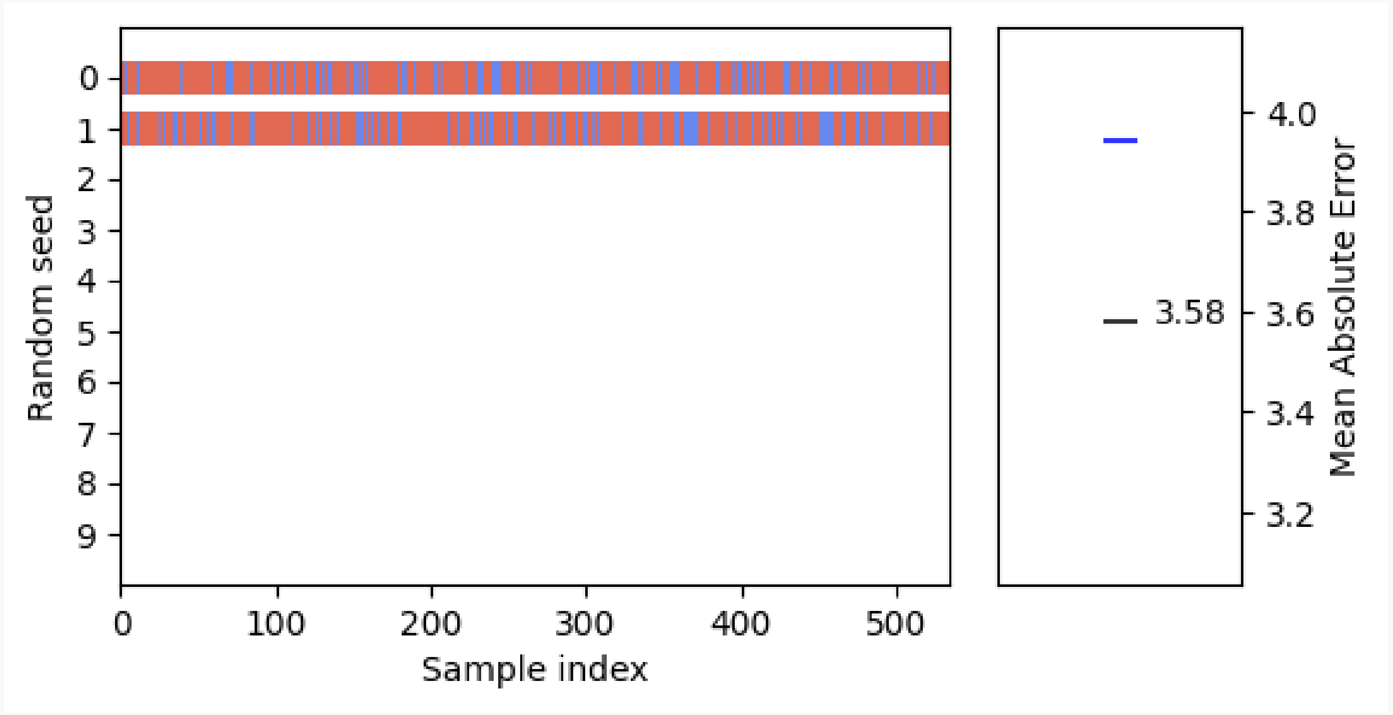
Repeated train/test splits

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



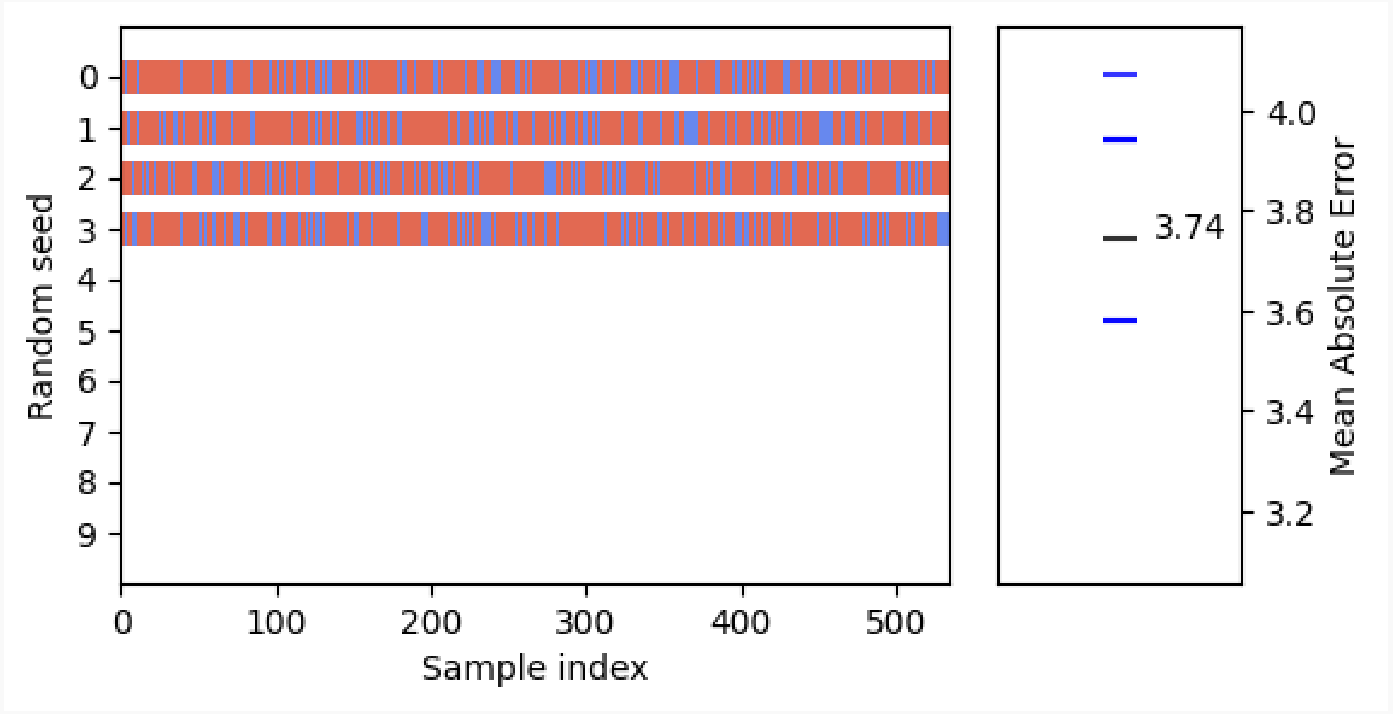
Repeated train/test splits

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



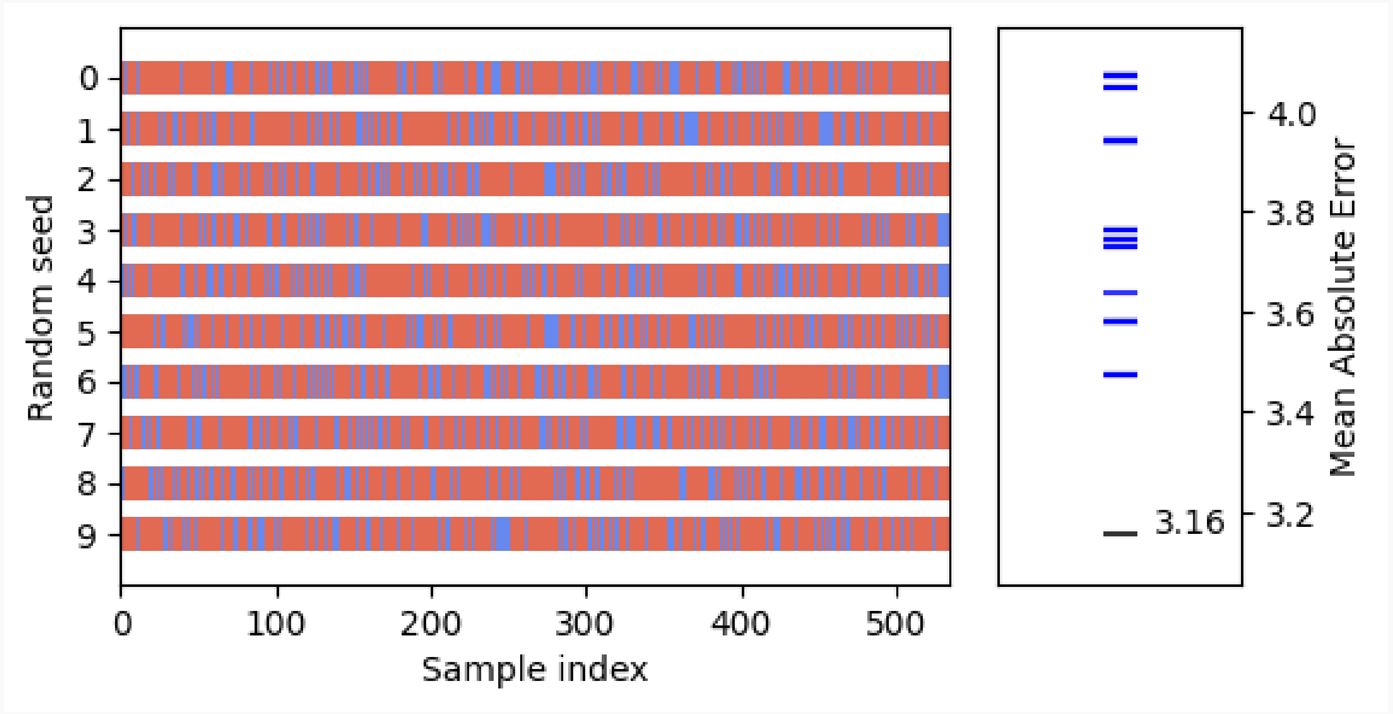
Repeated train/test splits

And another split...



Repeated train/test splits

Splitting ten times



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

Repeated train/test splits = Cross-validation

Cross-validation

- In sklearn, it can be instantiated with `cross_validate`.

Python

```python
from sklearn.model_selection import cross_validate
from sklearn.model_selection import ShuffleSplit

cv = ShuffleSplit(n_splits=40, test_size=0.3, random_state=0)
cv_results = cross_validate(
    regressor, data, target, cv=cv, scoring="neg_mean_absolute_error"
)
```

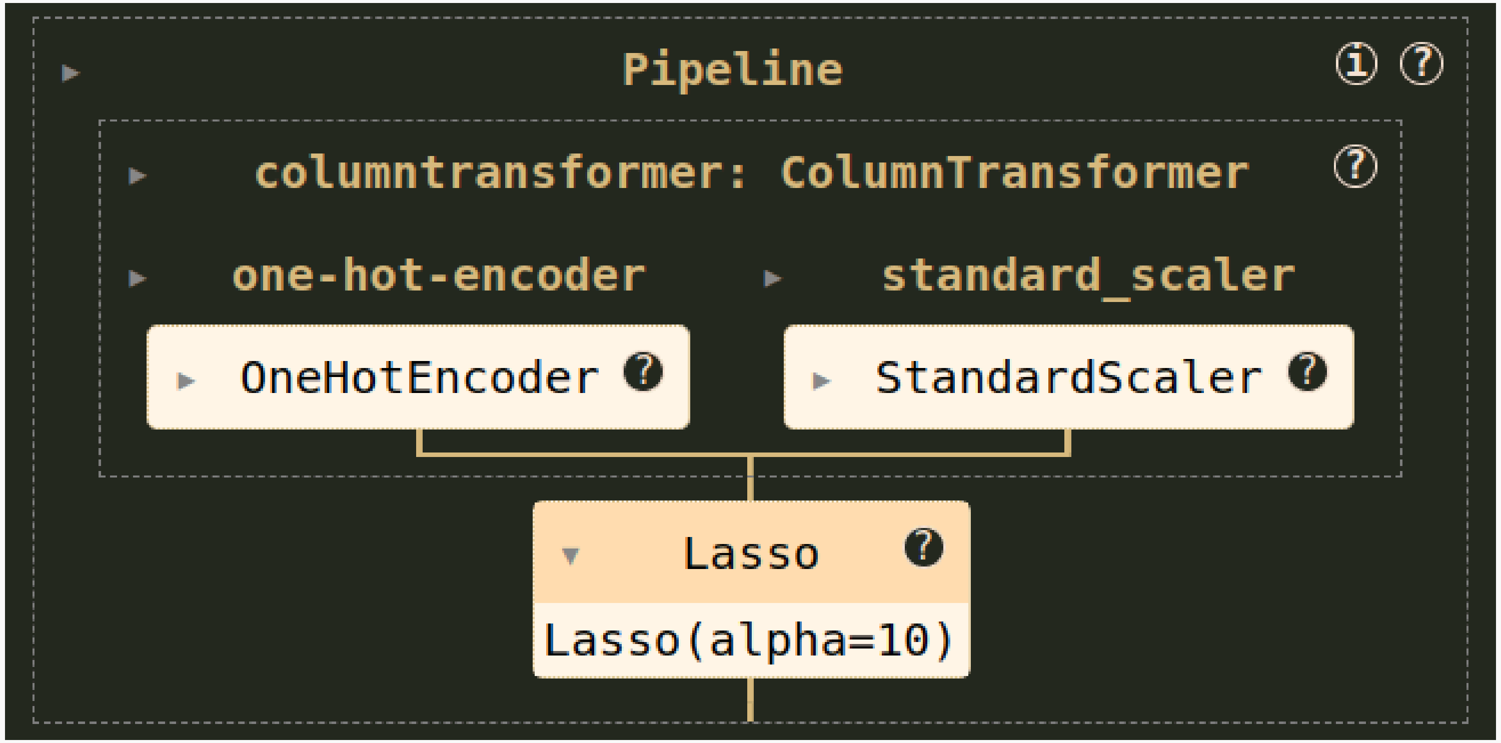
Repeated train/test splits = Cross-validation

Cross-validation

- In sklearn, it can be instantiated with `cross_validate`.
- 🙂 Robustly estimate generalization performance
- 🤩 Estimate variability of the performance: similar to bootstrapping (but different).
- 🚀 Let’s use it to select the best models among several canditates!

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

- Wage pipeline



```
Lasso(alpha=10)
```

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

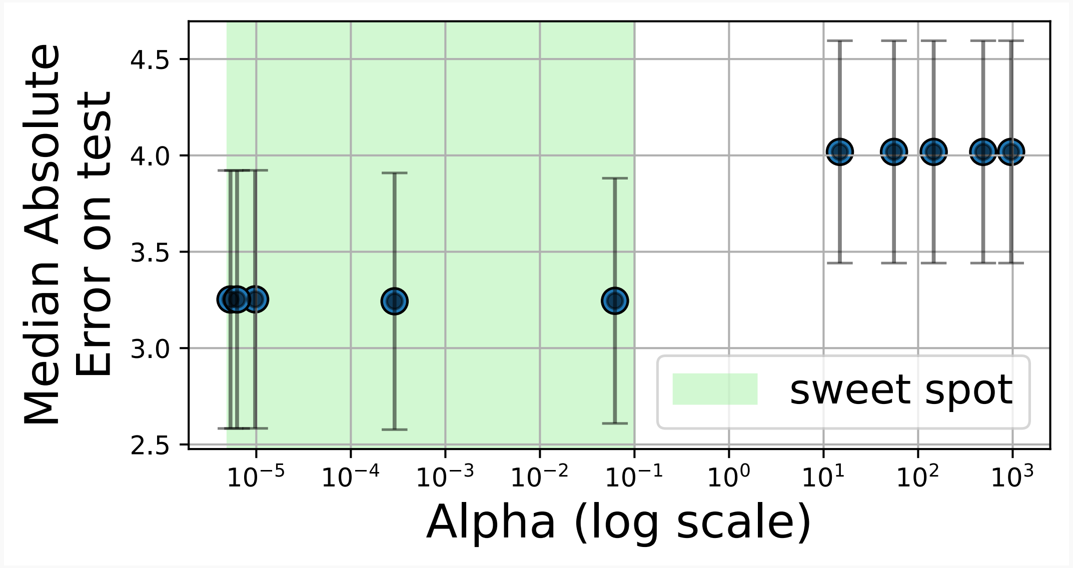
- Wage pipeline
- Random search over a distribution of α values

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
- Identify the best α value(s)



What final model to use for new prediction?

- Either refit on full data the model with the best hyper-parameters on the full data: often used in pratice.
- Or use the aggregation of outputs from the cross-validation 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.

- Proof that cross-validation selects the best model asymptotically among a family of models (averaging on the folds): (Lecué & Mitchell, 2012)

Naive cross-validation to select AND estimate the best performances

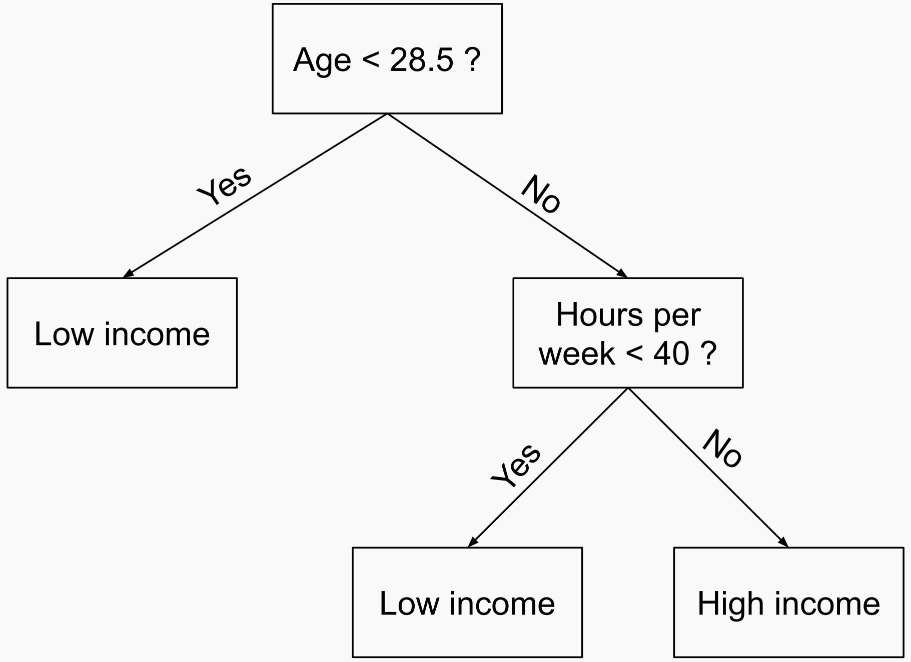
TODO

Nested cross-validation to select the best model

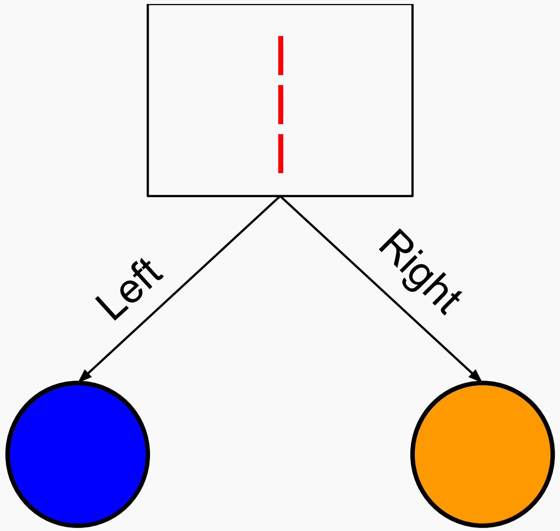
TODO

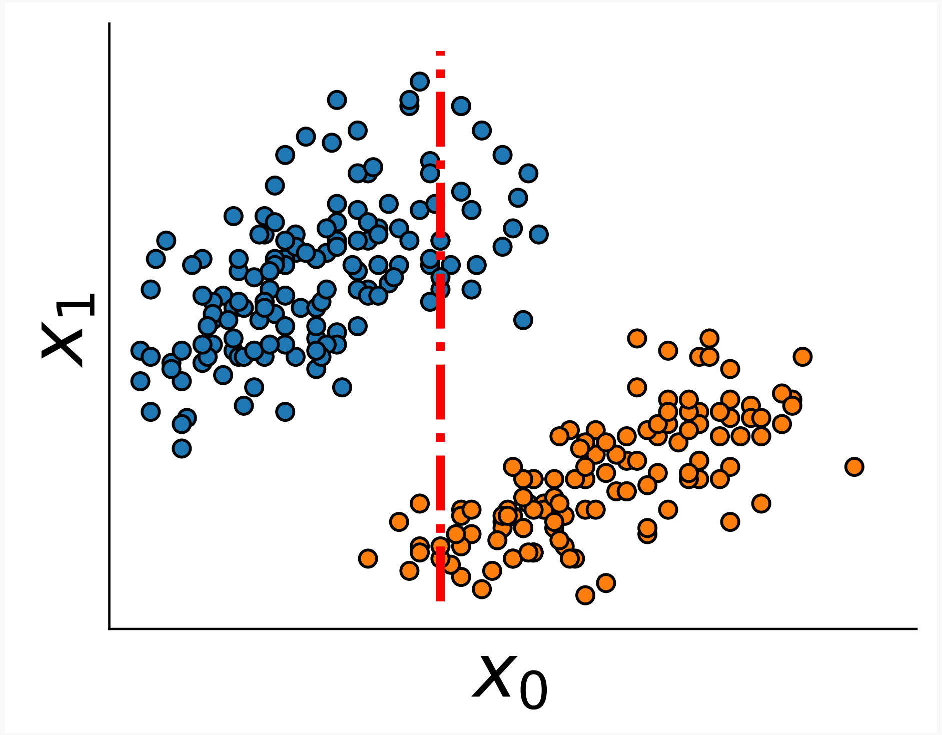
Flexible models: Tree, random forests and boosting

What is a decision tree? An example.

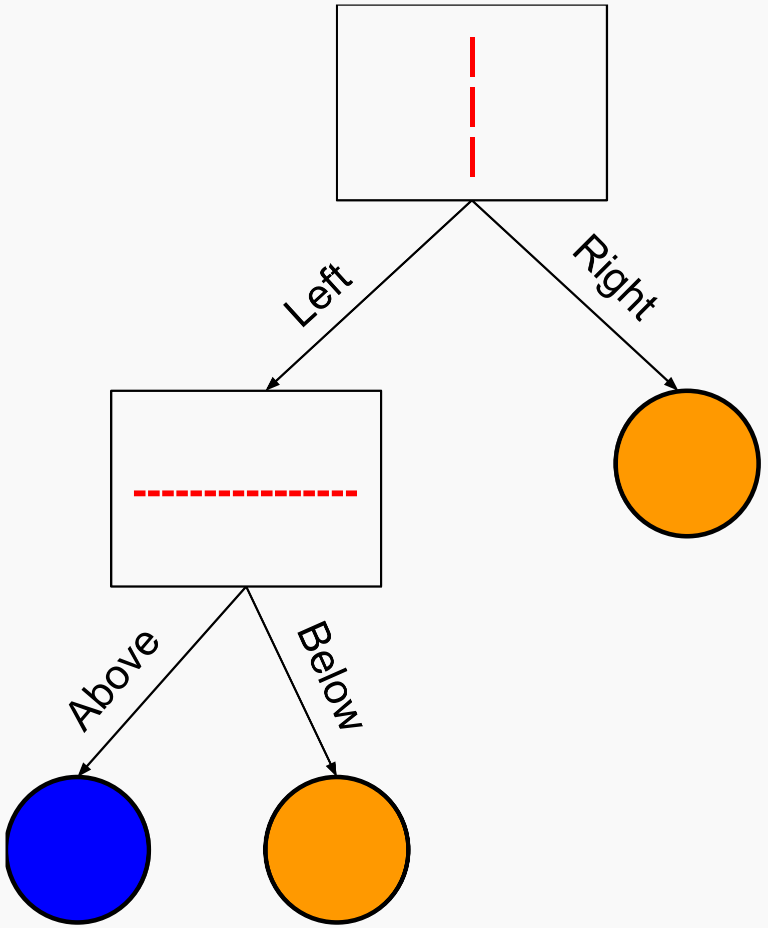


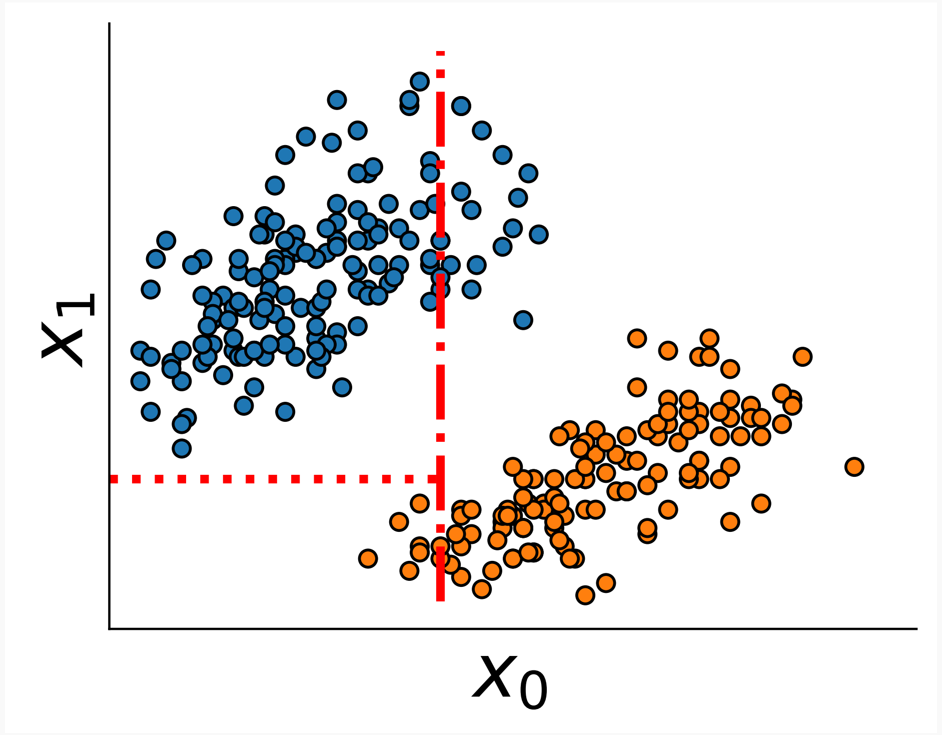
Growing a classification tree



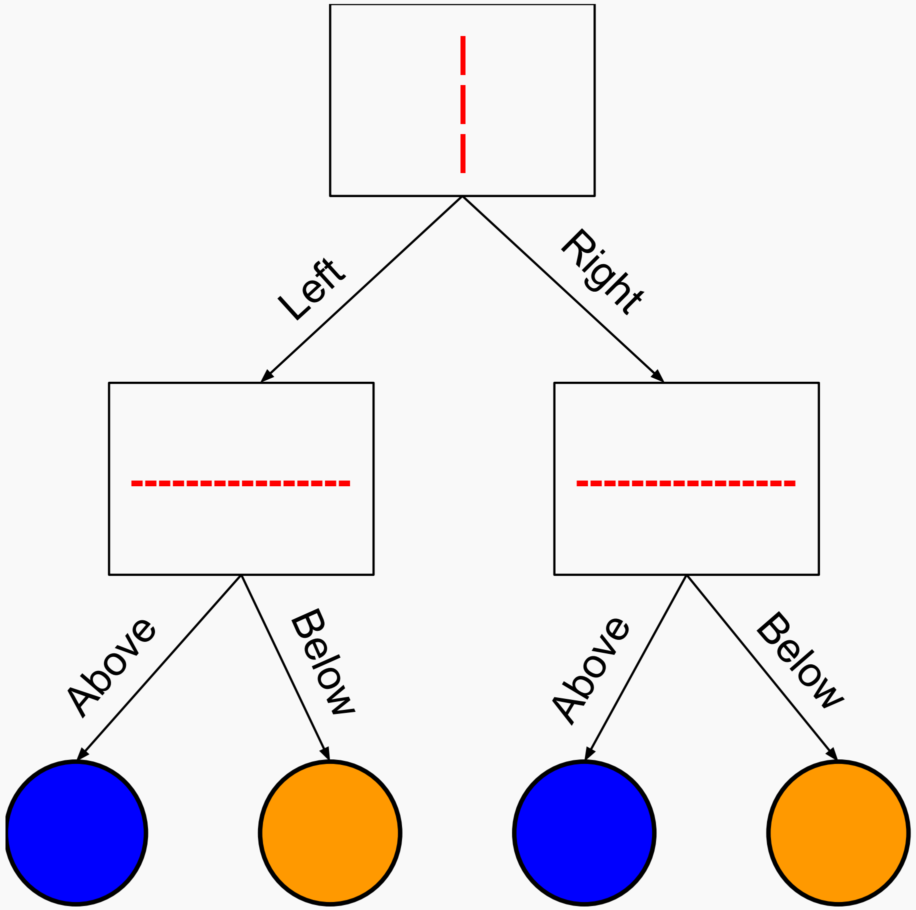


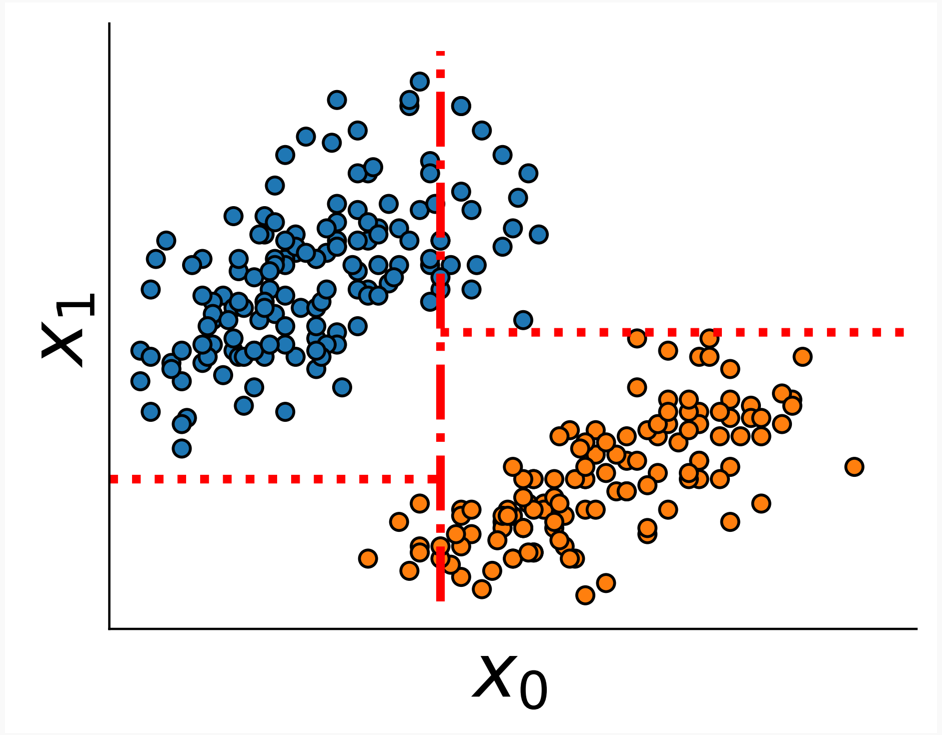
Growing a classification tree



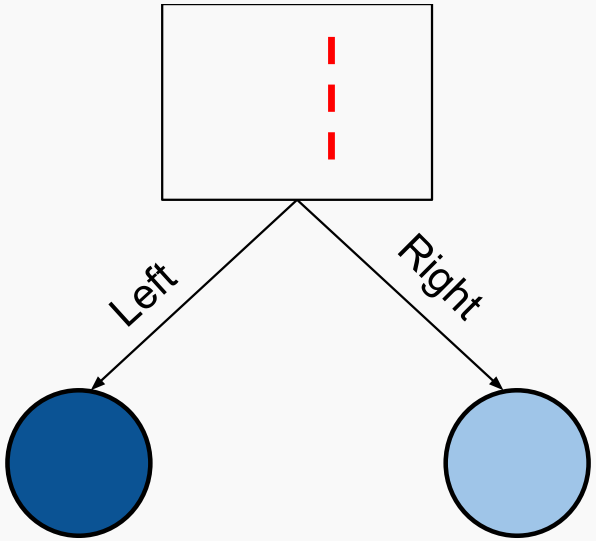


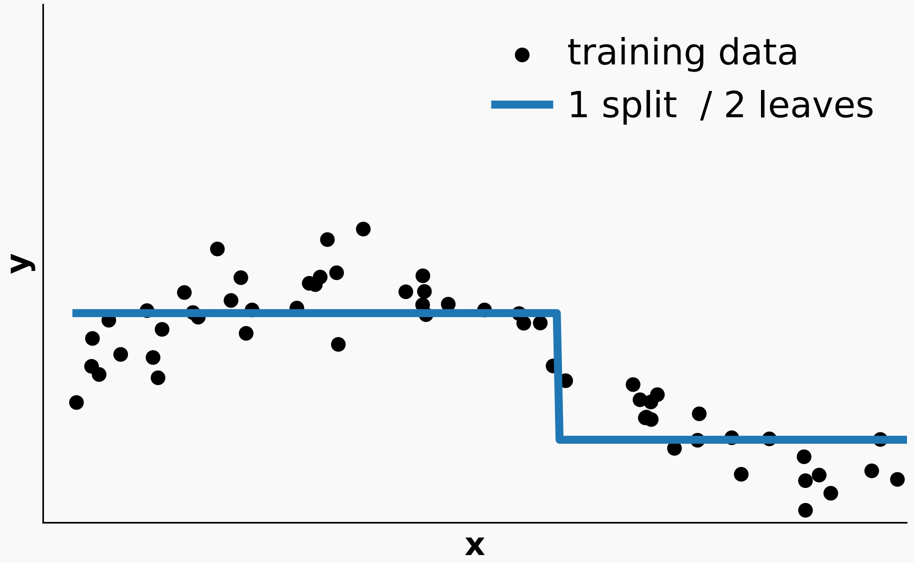
Growing a classification tree



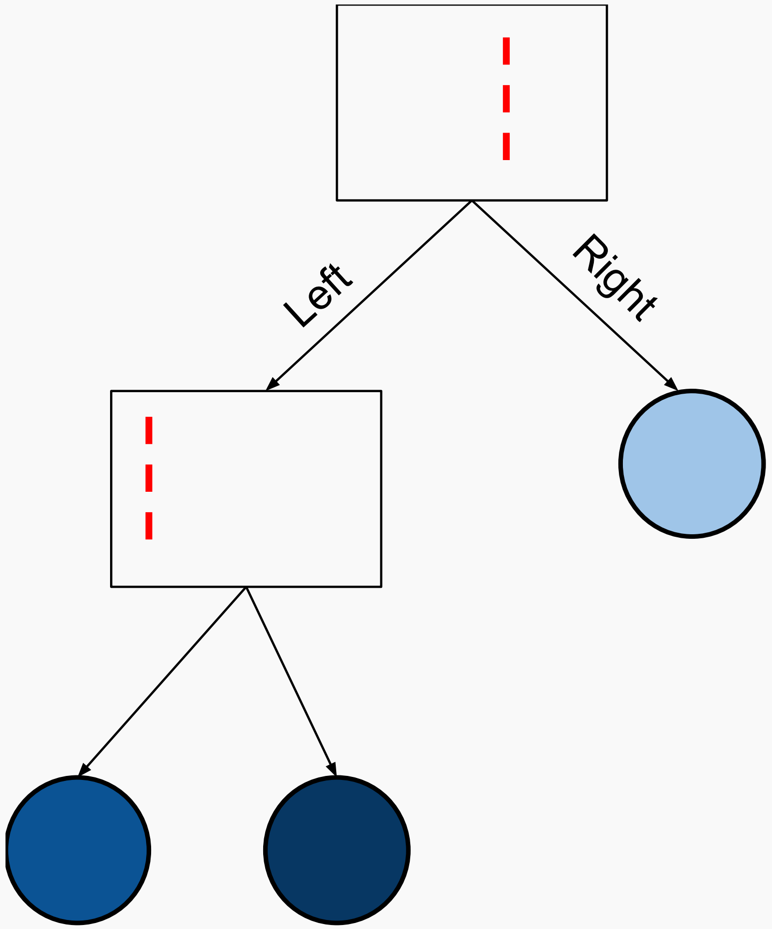


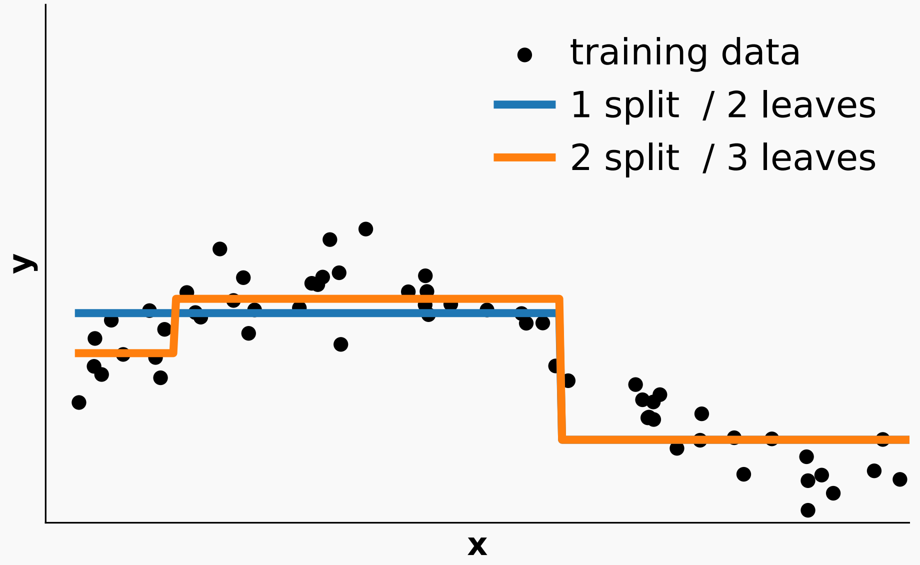
Growing a regression tree



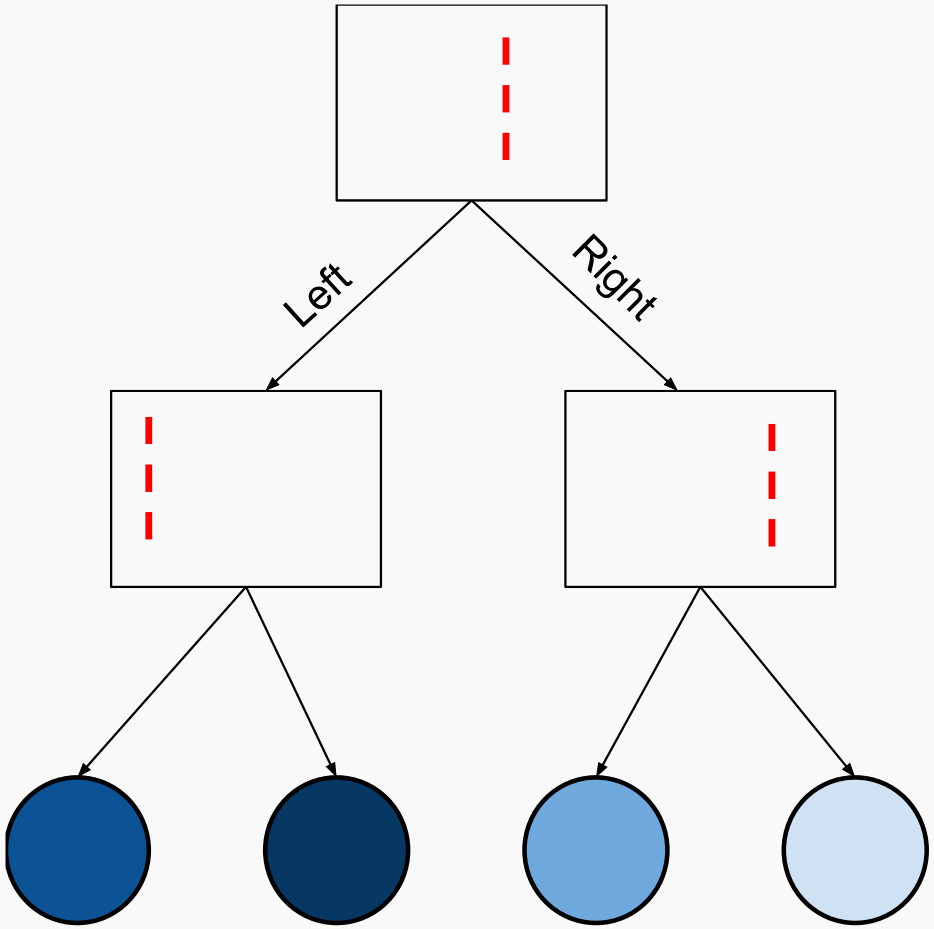


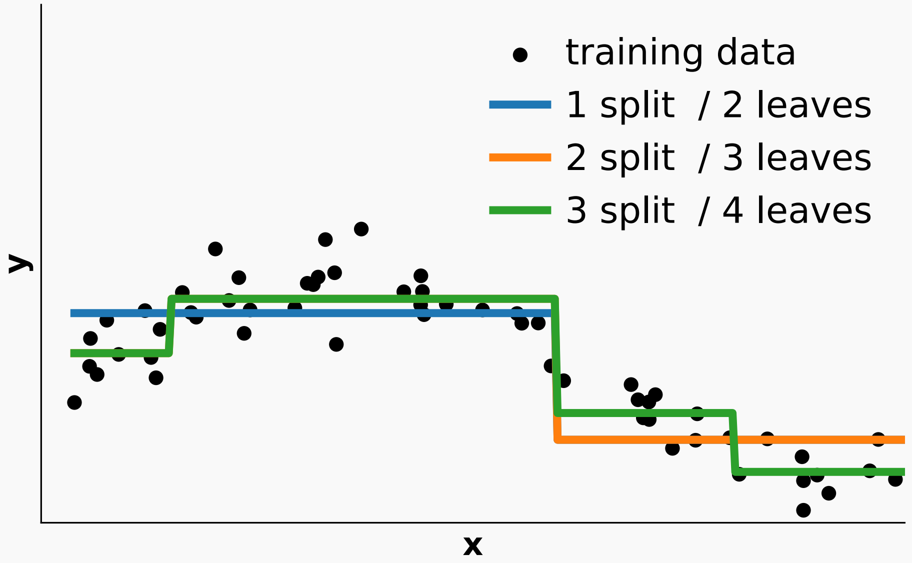
Growing a regression tree





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

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:

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

Impurity criteria

For classification

Gini impurity

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

Cross-entropy

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

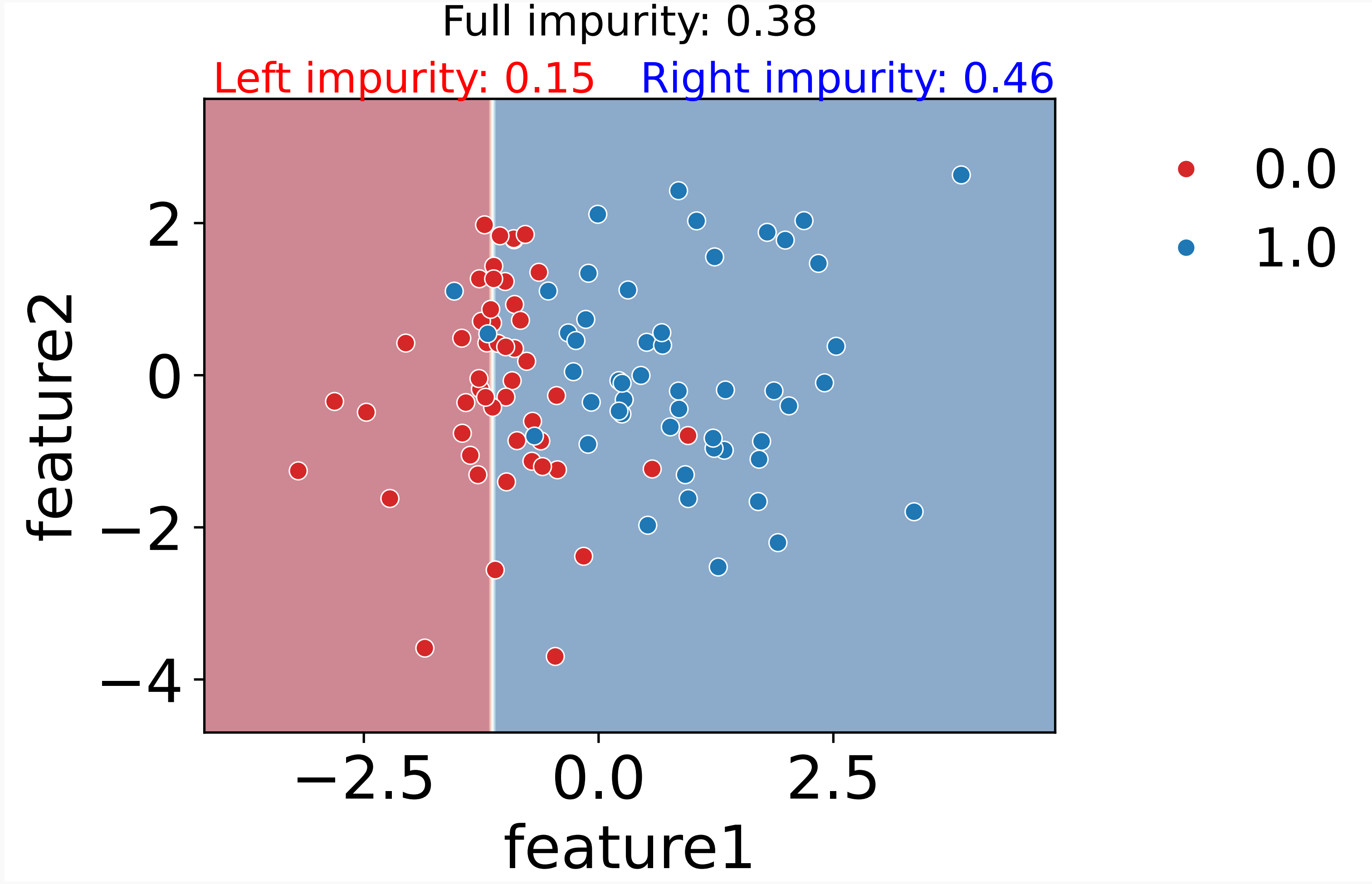
For regression

Mean squared error

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

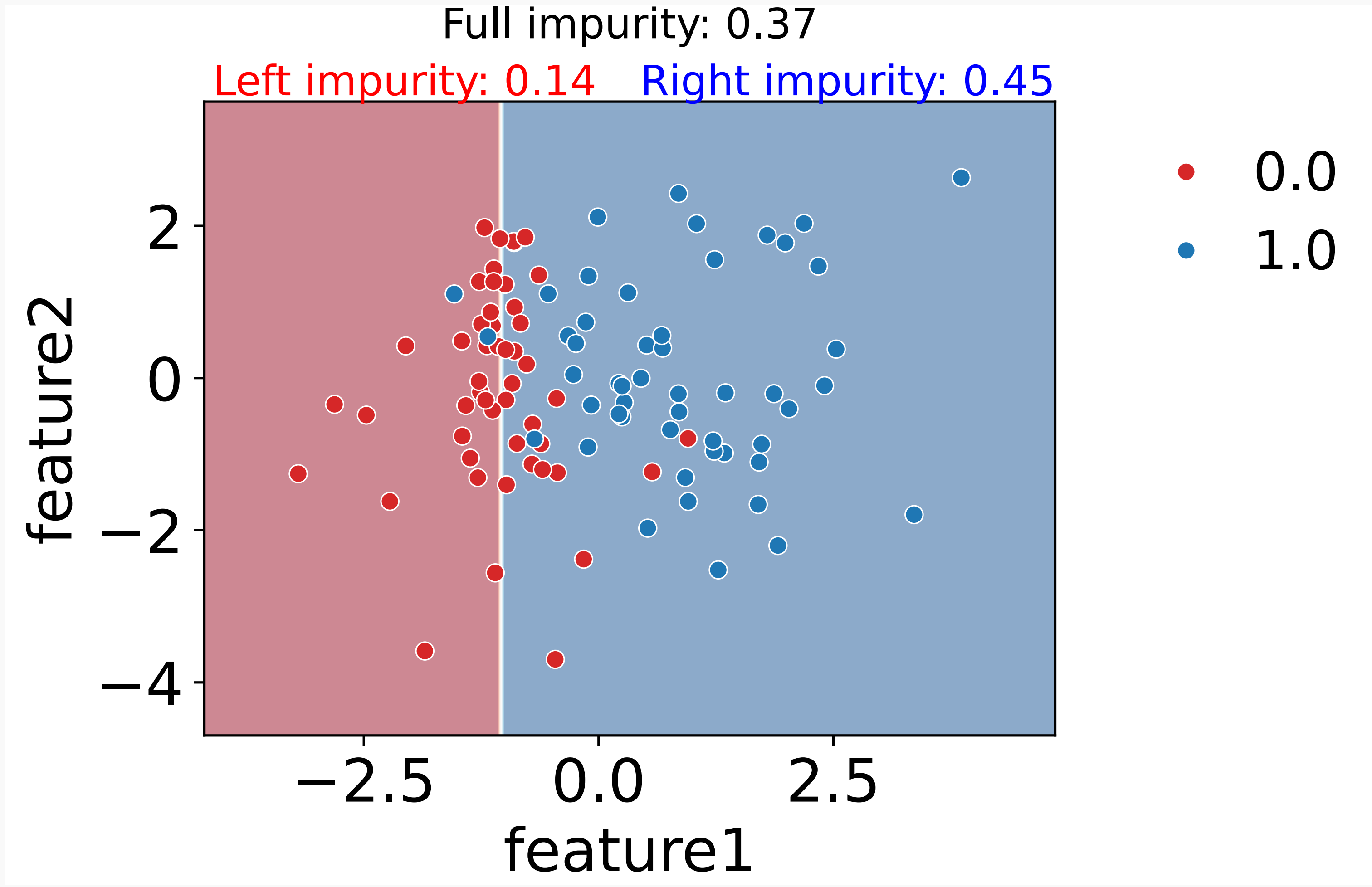
Chose the best split: example

Random split



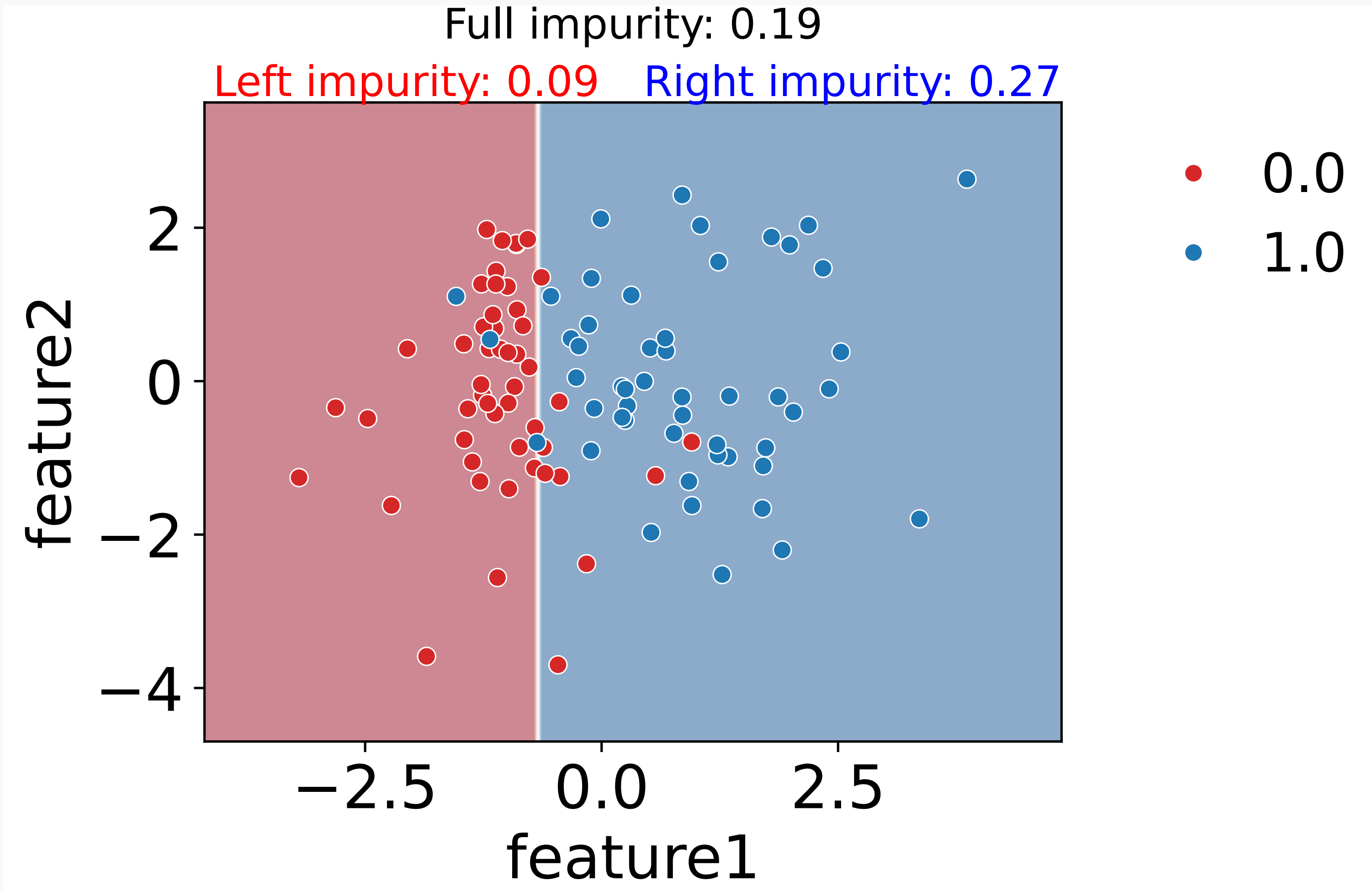
Chose the best split: example

Moving the split to the right from one point



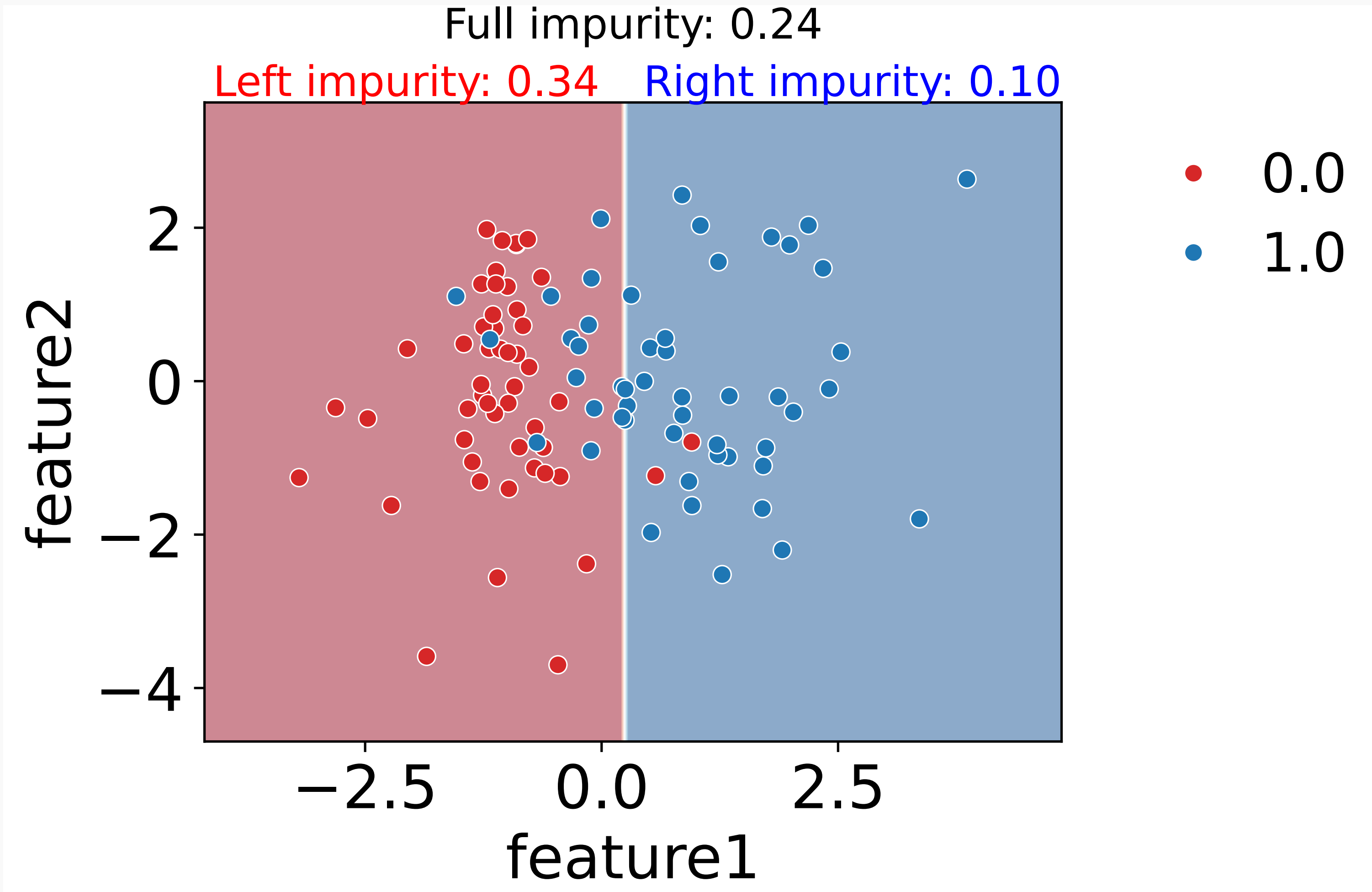
Chose the best split: example

Moving the split to the right from 10 points



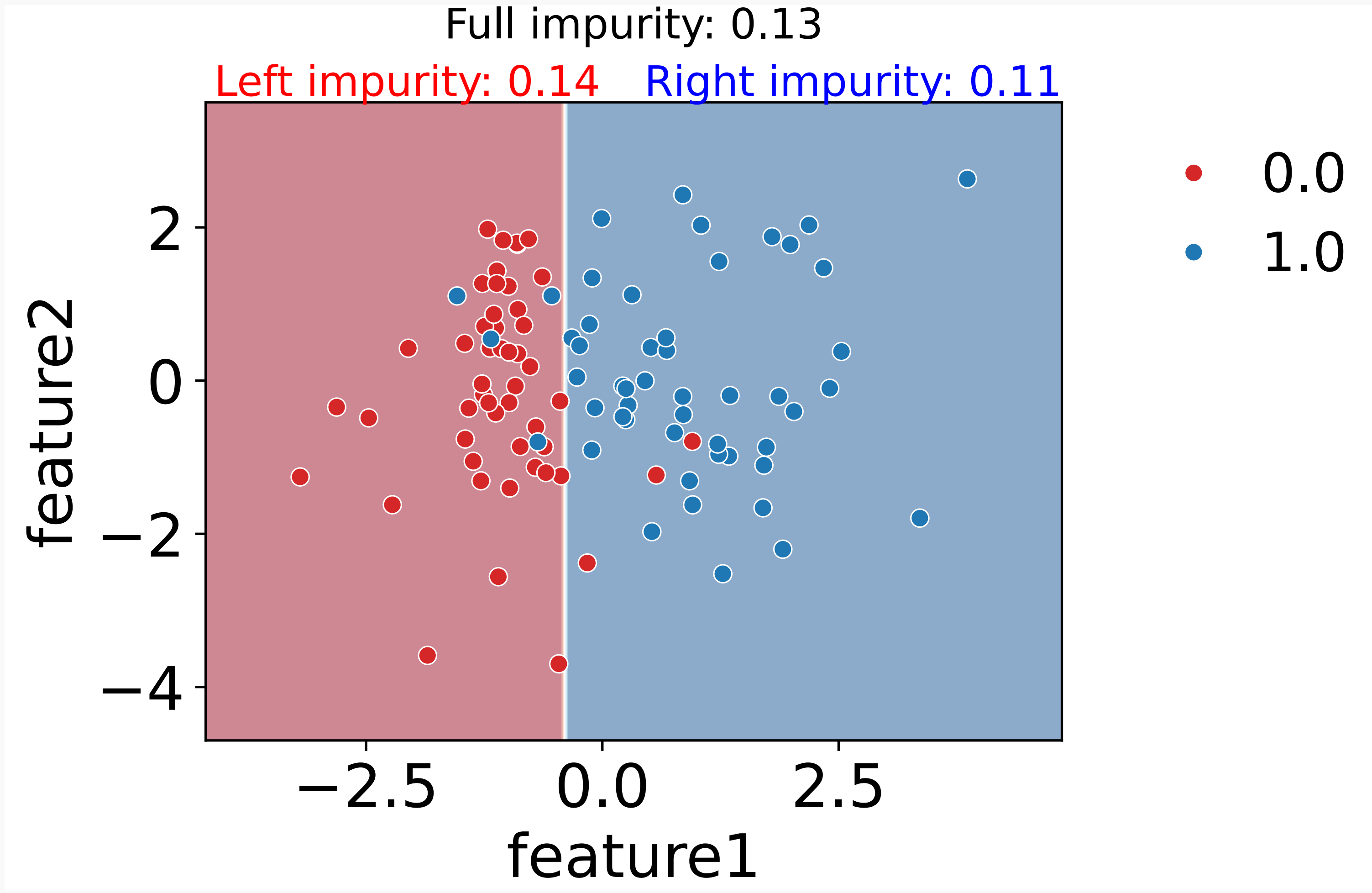
Chose the best split: example

Moving the split to the right from 20 points

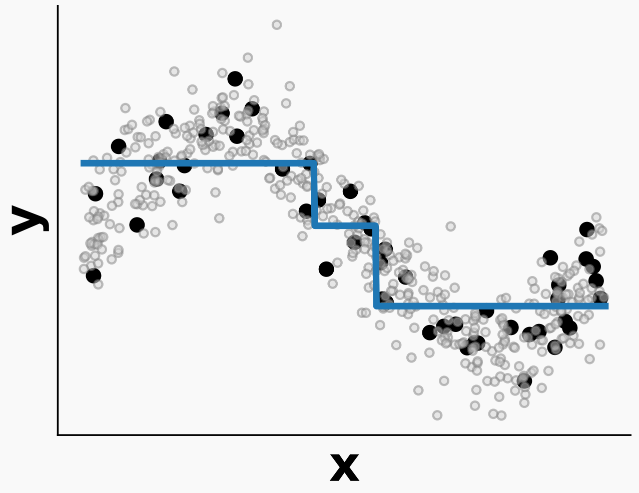


Chose the best split: example

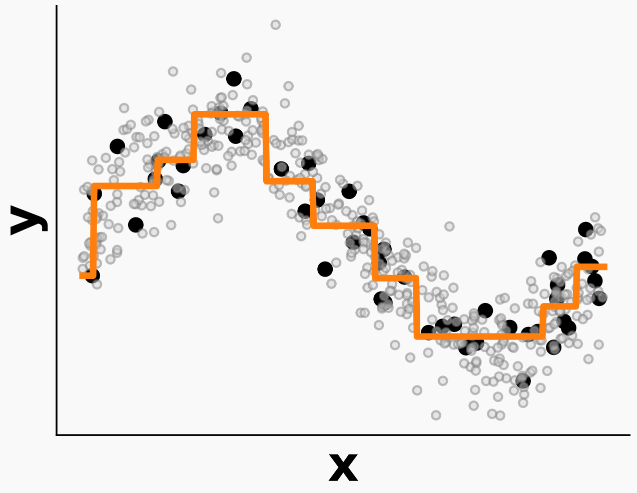
Best split



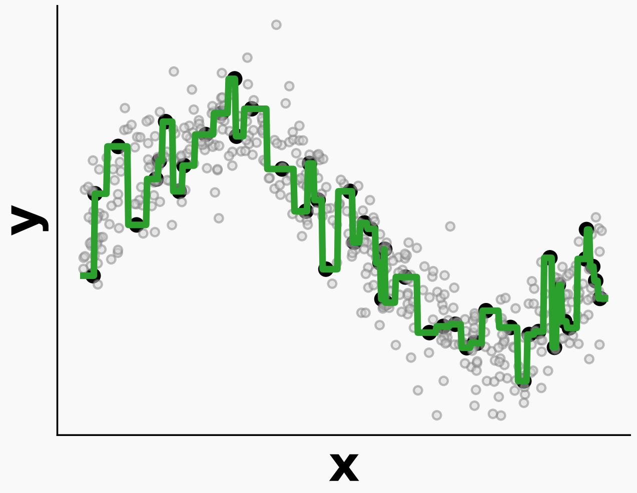
Tree depth and overfitting



Underfitting
max depth or
max_leaf_nodes
too small



Best trade-off



Overfitting
max depth or
max_leaf_nodes
too large

Main hyper-parameters of tree models

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 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

Bagging: Bootstrap AGGregatING

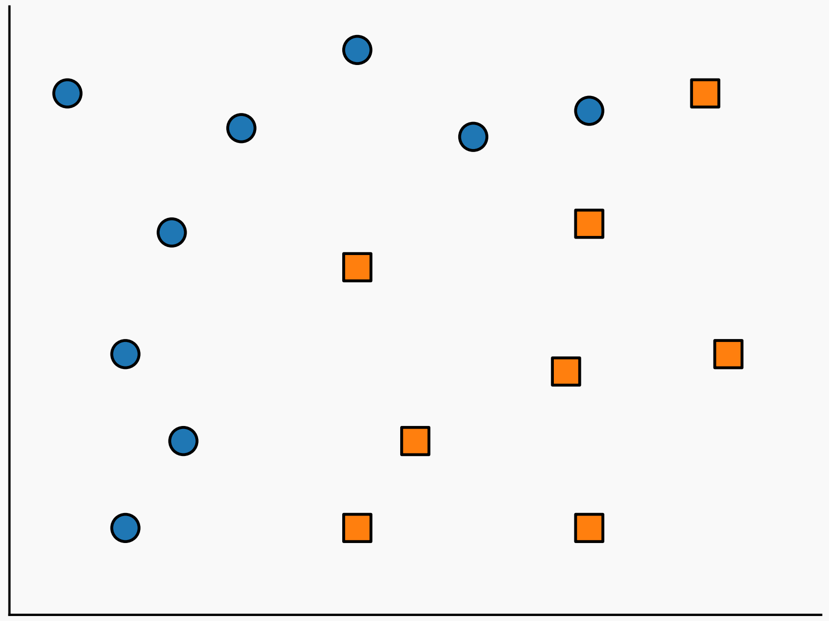
Bootstrap resampling (random sampling with replacement) proposed by (Breiman, 1996)

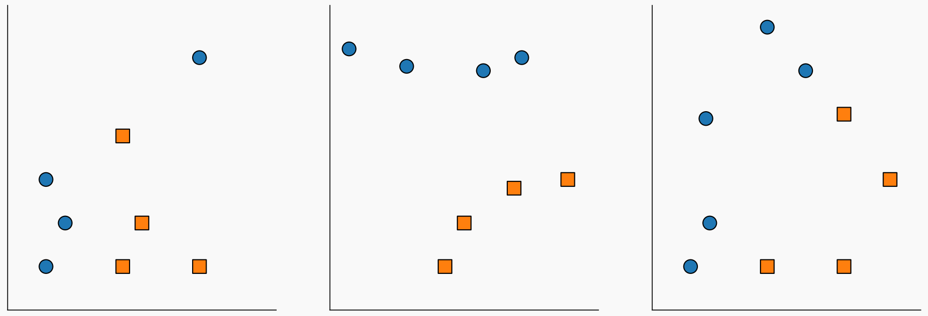
Built upon Bootstrap, introduced by (Efron, 1992) to estimate the variance of an estimator.

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

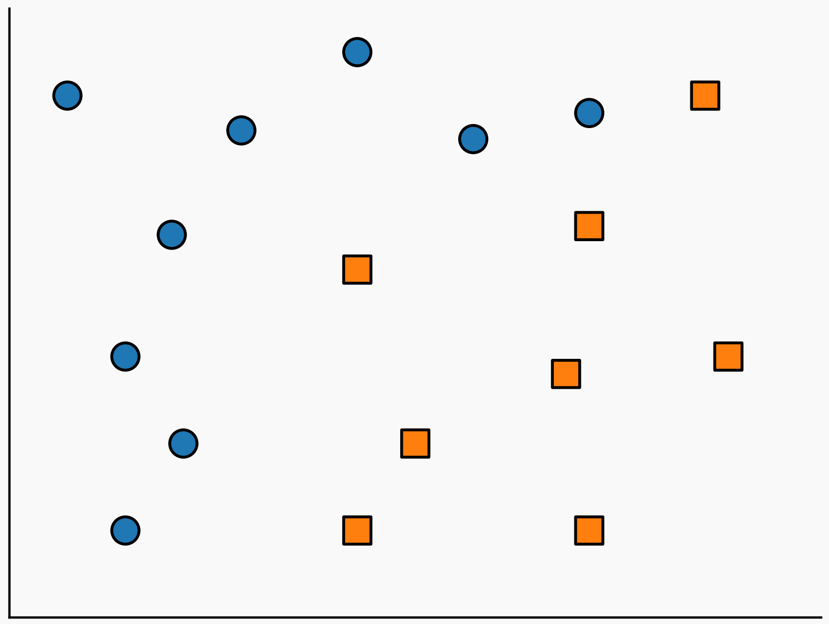
Can be used with any model

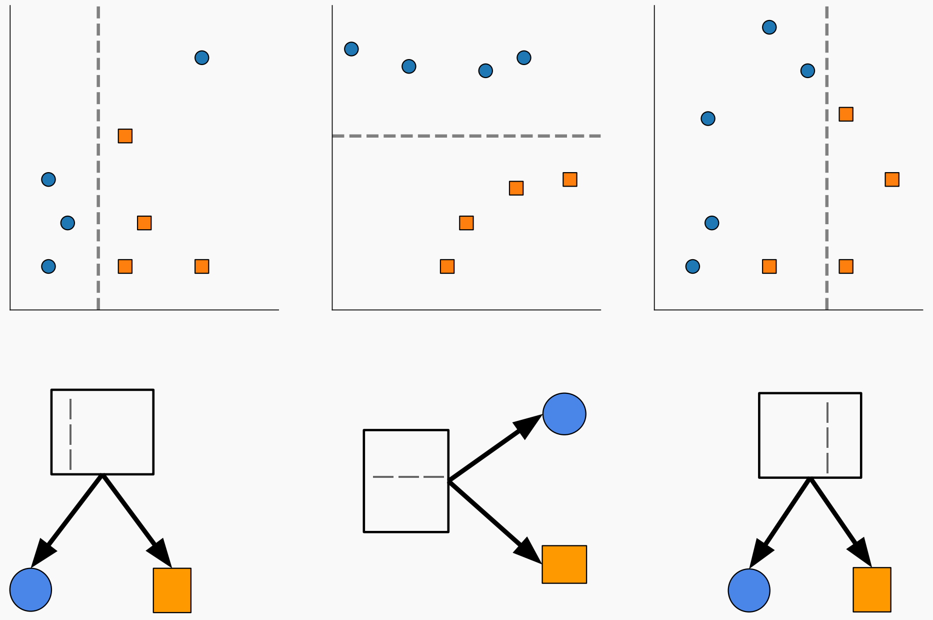
Random forests: Bagging with classification trees



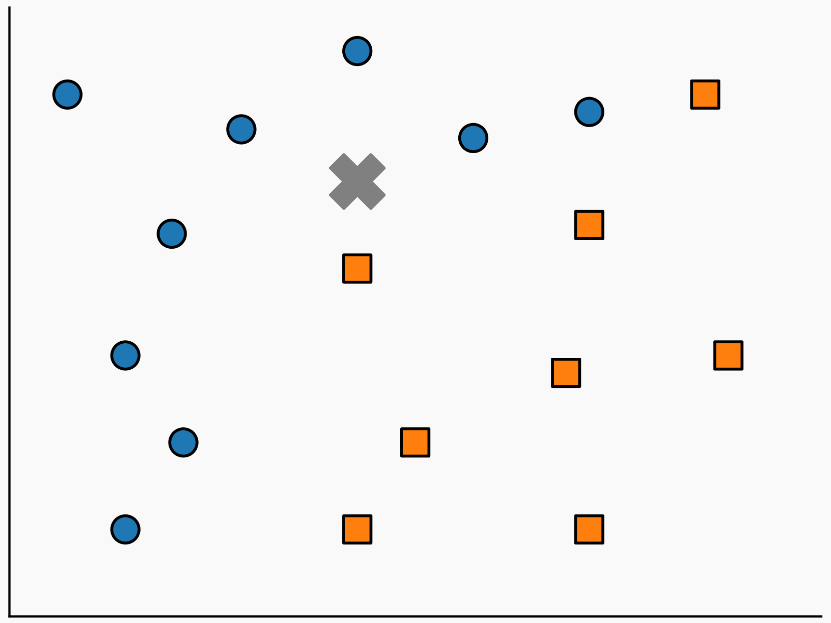


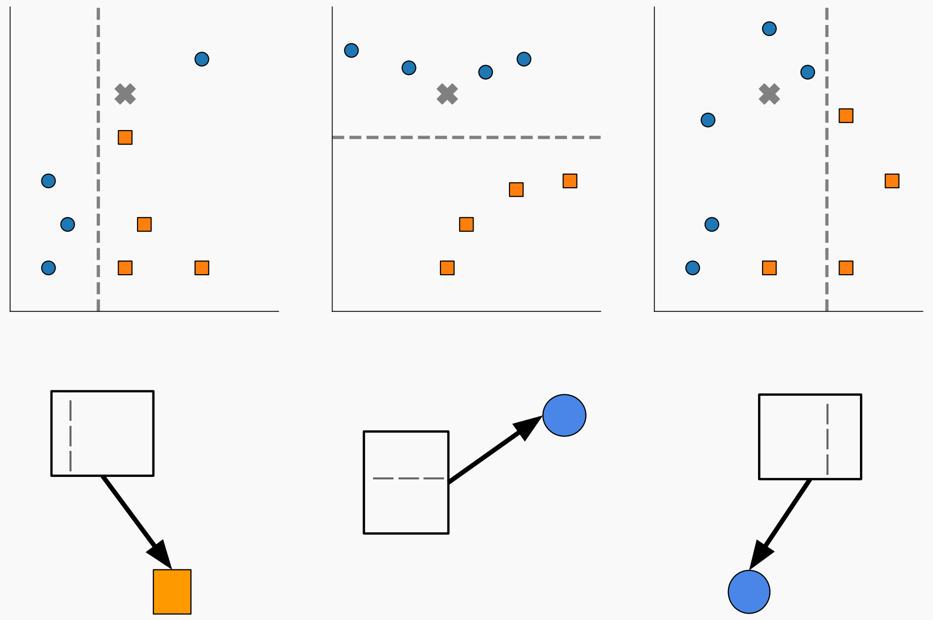
Random forests: Bagging with classification trees





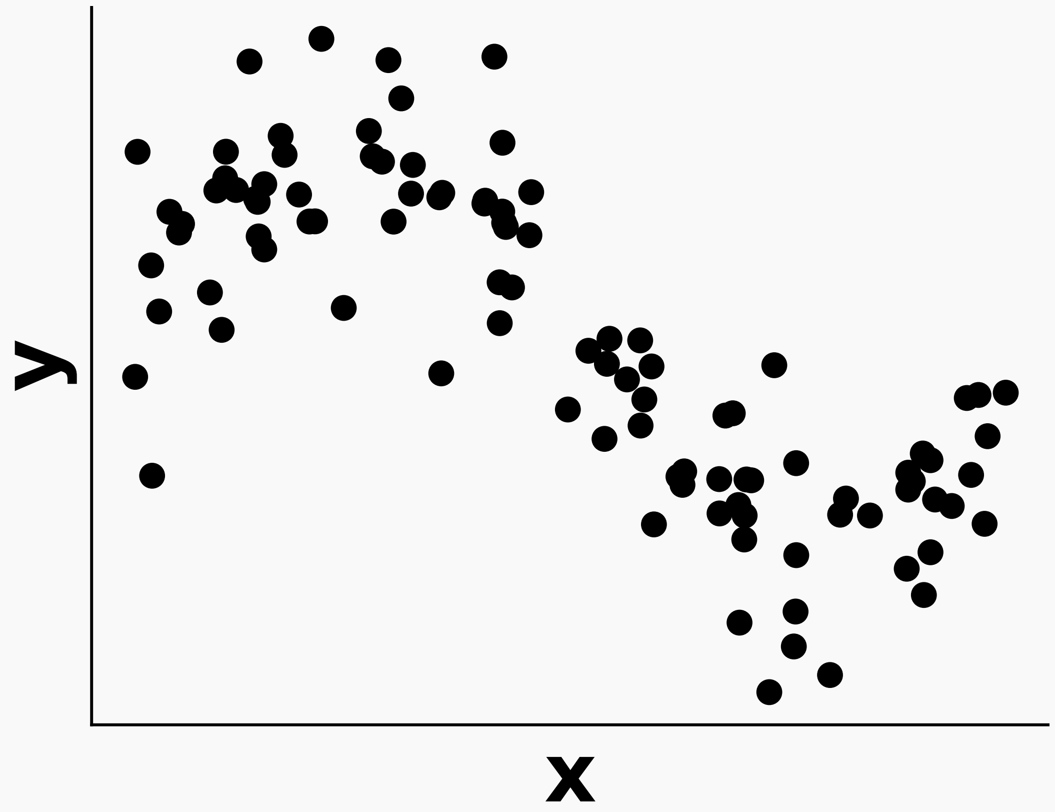
Random forests: Bagging with classification trees



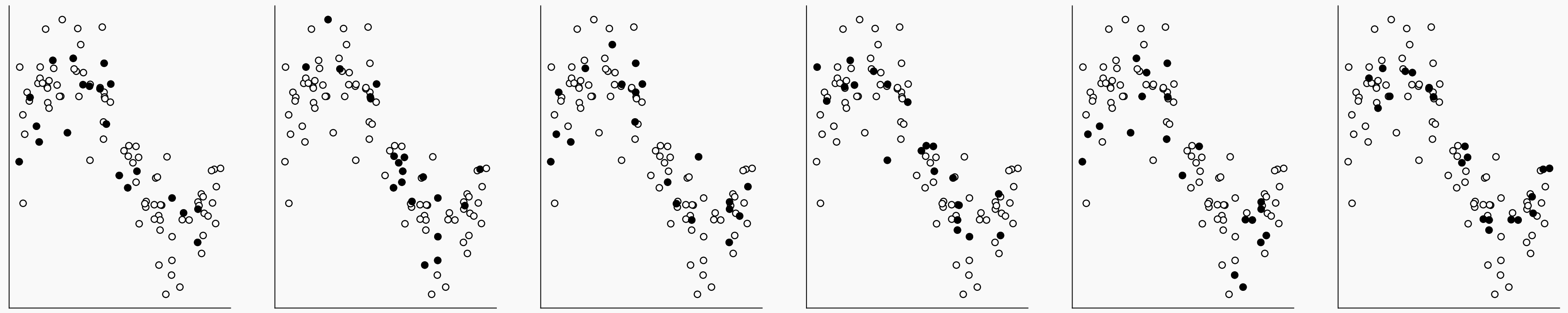


VOTE (■ , ● , ●) = ●

Random forests: Bagging with regression trees

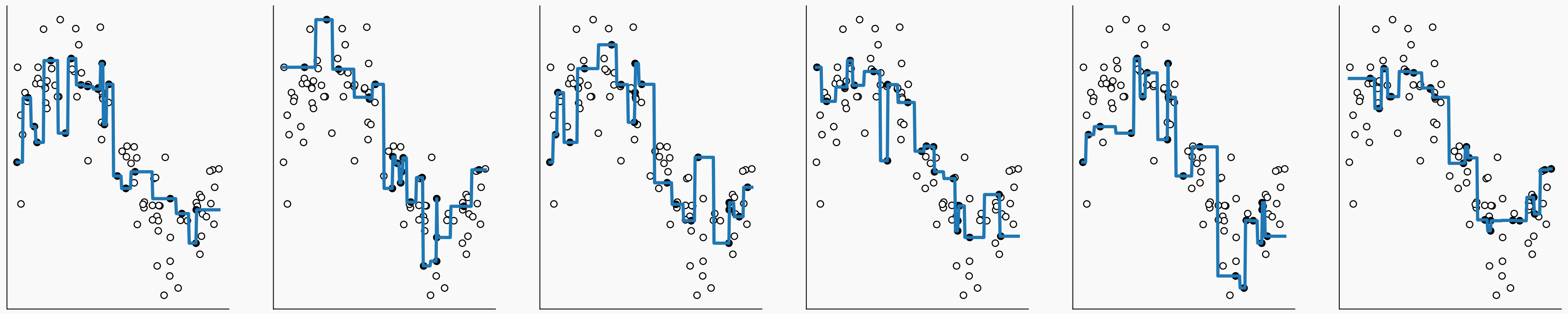


Random forests: Bagging with regression trees



- Select multiple subsets of the data

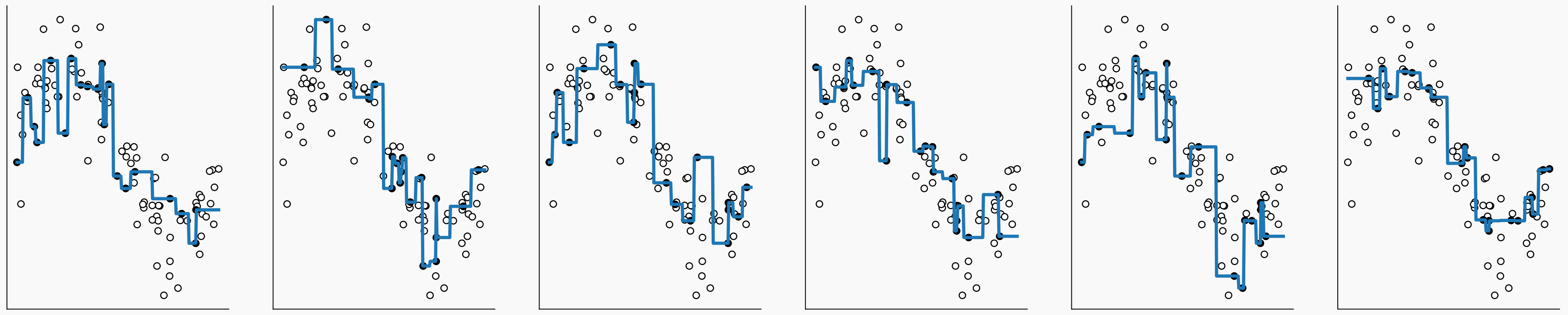
Random forests: Bagging with regression trees



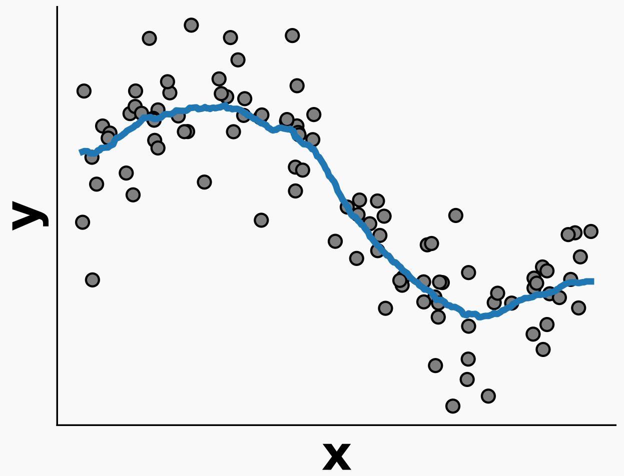
- **Select multiple subsets of the data**

- **Fit one model on each**

Random forests: Bagging with regression trees



- **Select multiple subsets of the data**
- **Fit one model on each**
- **Average the predictions**



Main hyper-parameters of random forests

Python

```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 are bagged randomized decision trees

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: Adaptive boosting

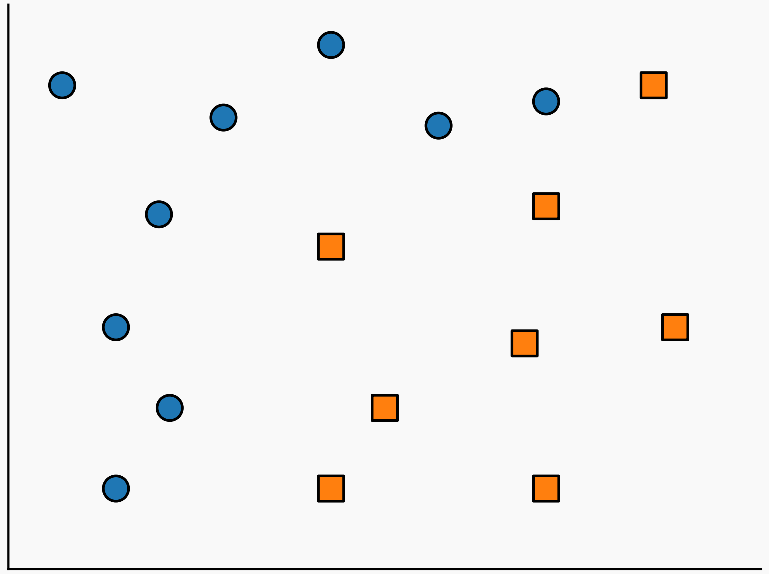
Boosting use multiple iterative models

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

Two examples of boosting

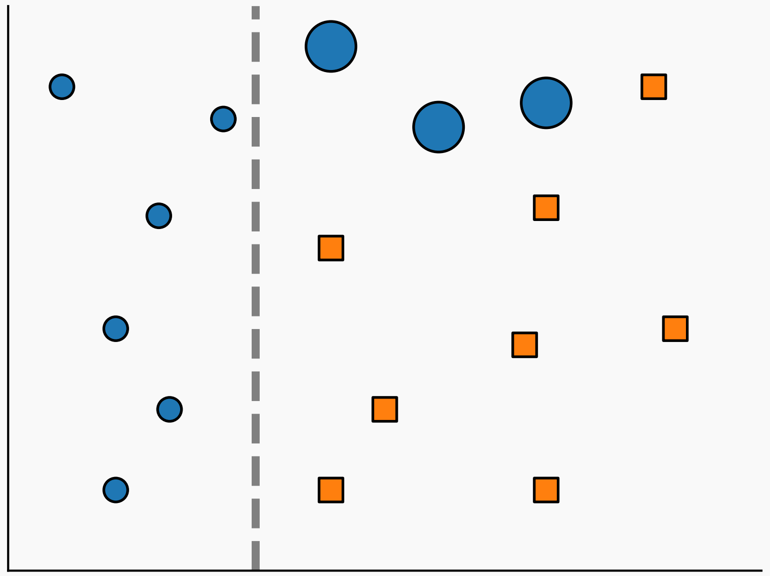
- Adaptive boosting (AdaBoost): reweight mispredicted samples at each step (Friedman et al., 2000)
- Gradient boosting: predict the negative errors of previous models at each step (Friedman, 2001)

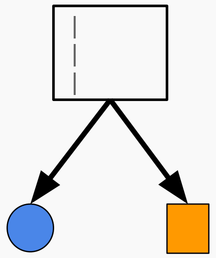
Boosting: Adaptive boosting



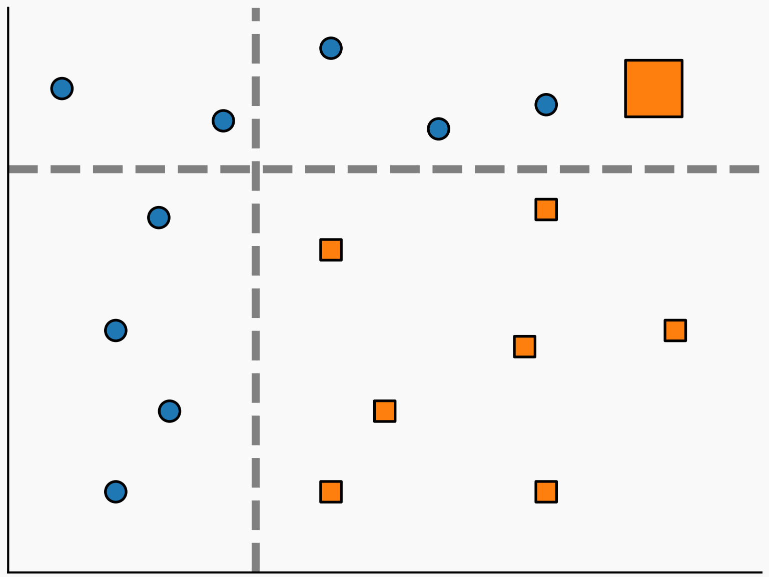
Boosting: Adaptive boosting

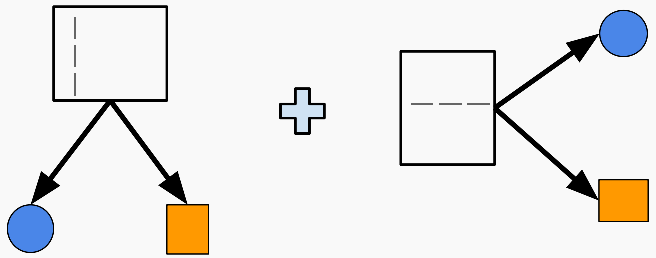
First prediction:



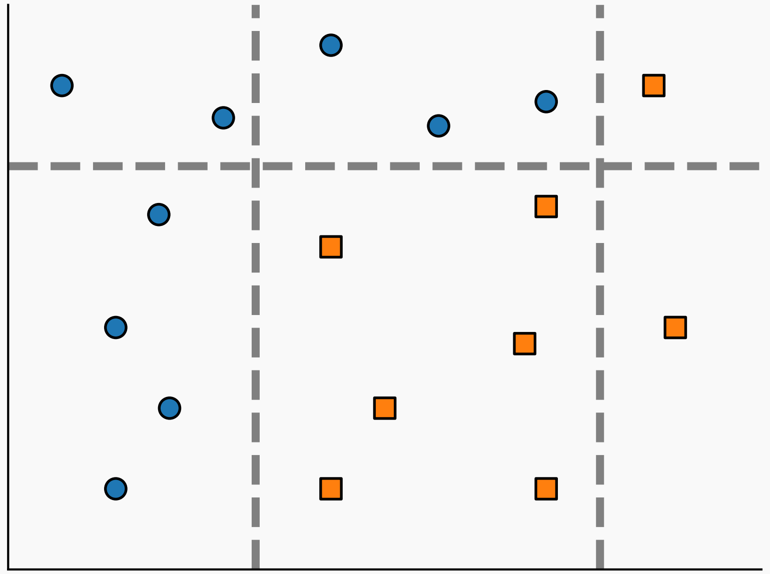


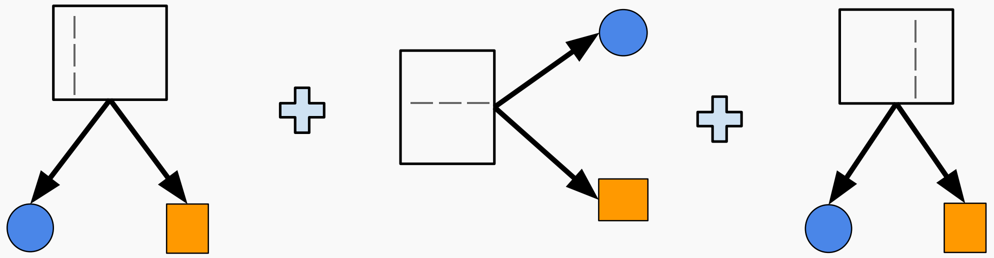
Boosting: Adaptive boosting





Boosting: Adaptive boosting





At each step, AdaBoost weights mispredicted samples

Gradient boosting

Gradient boosting: how are the iterative learners chosen?

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

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

Gradient boosting: how are the iterative learners chosen?

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

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

💡 Taylor expansion

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

Gradient boosting: how are the iterative learners chosen?

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

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

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

💡 Taylor expansion

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

]

Faster gradient boosting with binned features

😭 Gradient boosting is slow when N>10,000

🚀 HistGradientBoosting

- Discretize numerical features into 256 bins: less costly for tree splitting
- Multi core implementation
- Much much faster

Take away for ensemble models

[strong("Bagging"), strong("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

Other well known families of models

Generalized linear models

Kernel methods: Support vector machines, Gaussian processes

Deep neural networks

Why not use deep learning everywhere?

- Success of deep learning (aka deep neural networks) in image, speech recognition and text
- 🤔 Why not so used in econometrics?

Why not use deep learning everywhere?

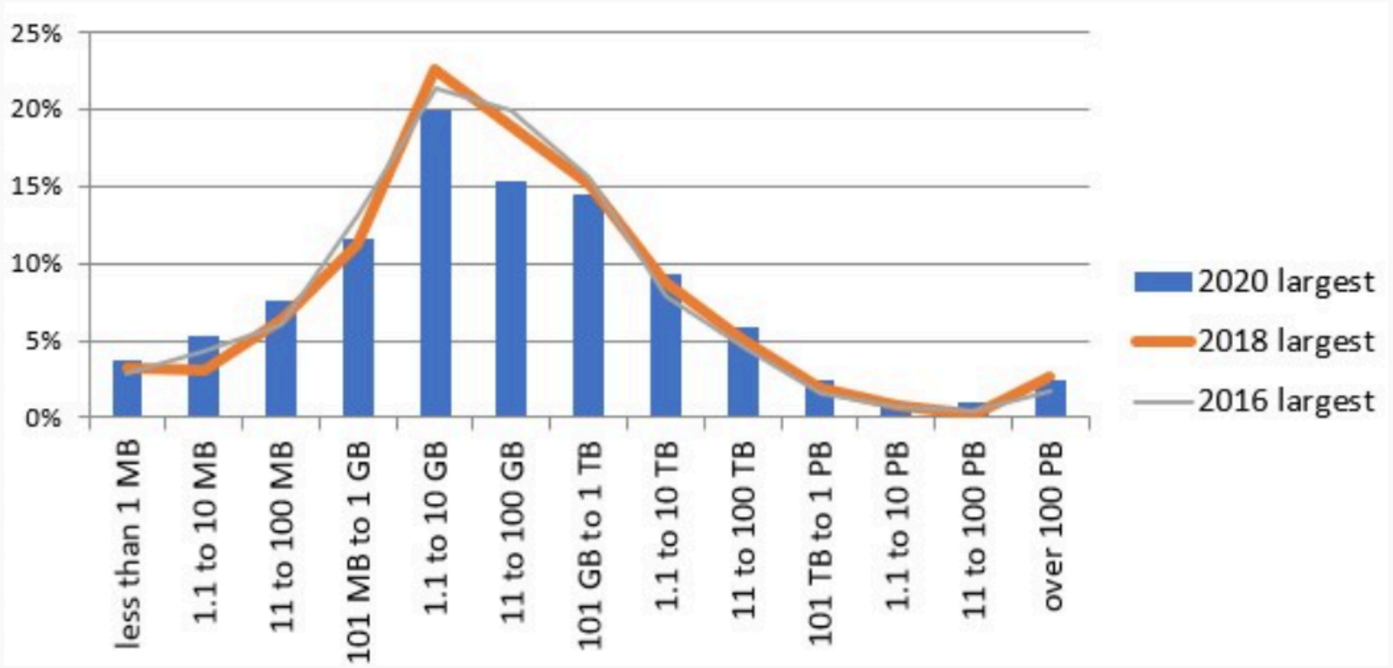
- Success of deep learning (aka deep neural networks) in image, speech recognition and text
- 🤔 Why not so used in econometrics?

Deep learning needs a lot of data (typically $N \approx 1$ million)

Do we have this much data in econometrics?

Answer 1: Limited data settings

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

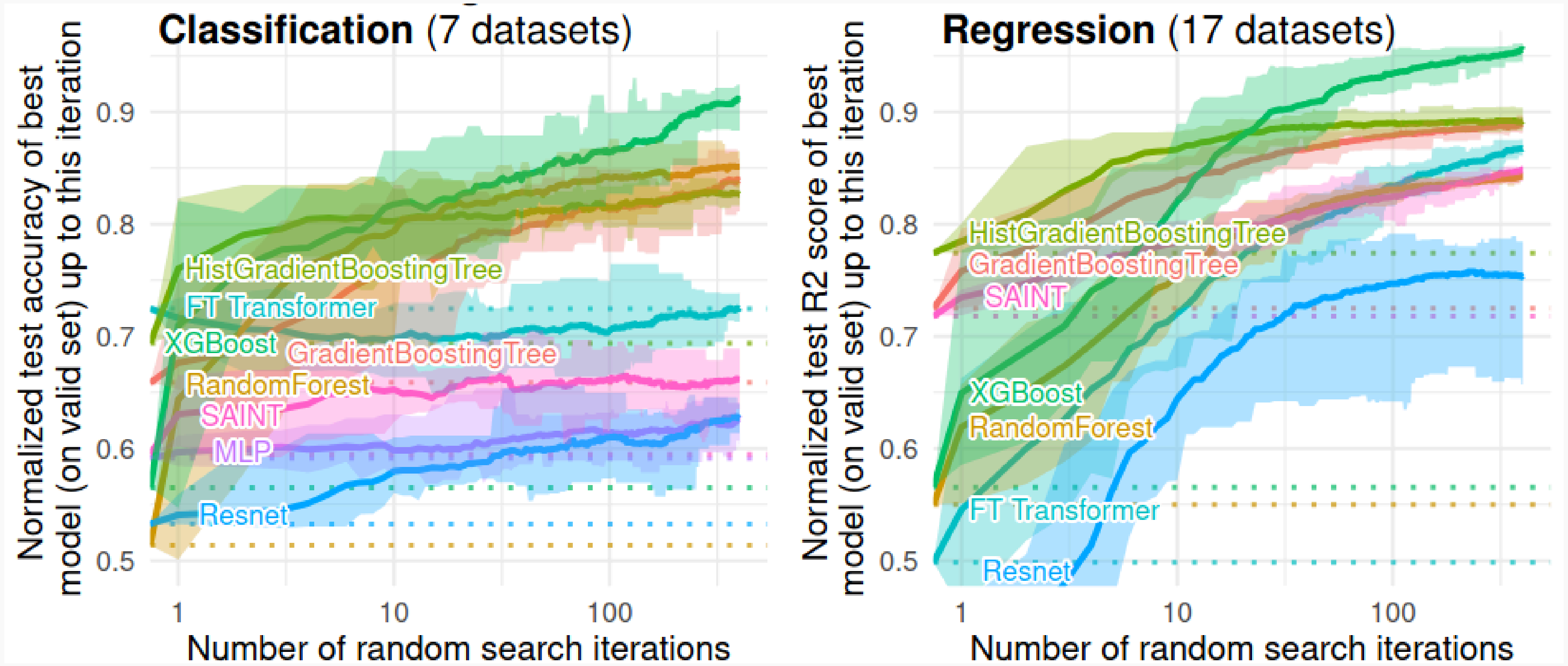


Typical dataset are mid-sized. This does not change with time.[1]

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

Answer 2: Deep learning underperforms on data tables

Tree-based methods outperform tailored deep learning architectures (Grinsztajn et al., 2022)



Nuance: recent work on LLM and pre-trained techniques for tabular learning

Some references:

- Skrub python library: data-wrangling and encoding (same people than sklearn)
- (Kim et al., 2024): CARTE: pretraining and transfer for tabular learning
- (Grinsztajn et al., 2023) : Vectorizing string entries for data processing on tables: when are larger language models better?

Bibliography

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

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.

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.

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.

Grinsztajn, L., Oyallon, E., Kim, M. J., & Varoquaux, G. (2023). Vectorizing string entries for data processing on tables: when are larger language models better?. Arxiv Preprint Arxiv:2312.09634.

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.