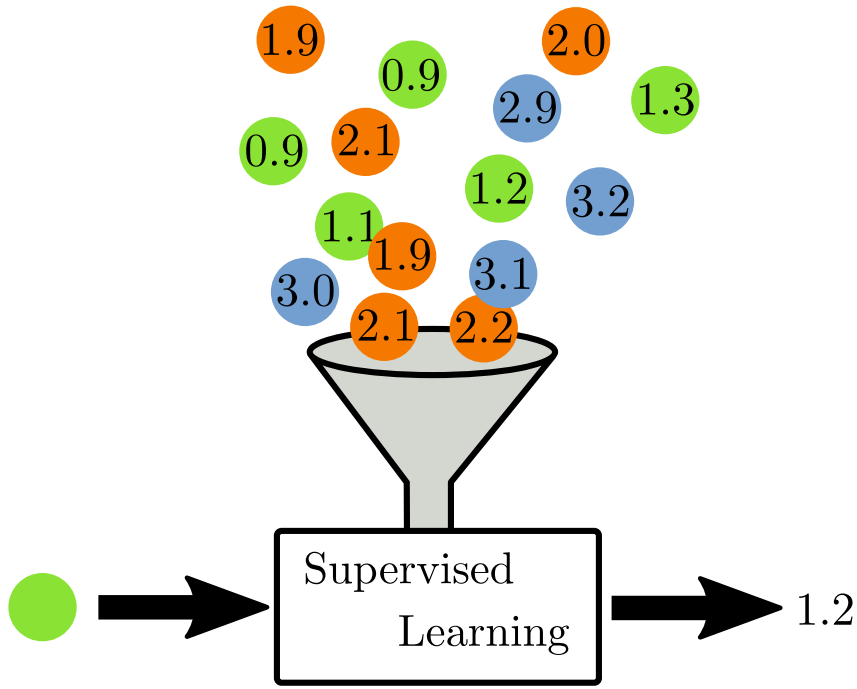
Introduction to Data Science
Supervised Learning

Benoit Gaüzère

INSA Rouen Normandie - Laboratoire LITIS

March 4, 2024

What we are talking about ?



- Learning trough examples
- Mimic human tasks (IA ?)
- Produce outputs given some inputs (functions ?)

Supervised Learning

Purpose

Given a dataset $\{(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}, i = 1, \ldots, N\}$, learn the dependancies between $\mathcal{X}$ and $\mathcal{Y}$.

- Example: Learn the links between cardiac risk and food habits. $\mathbf{x}_i$ is one person describe by d features concerning its food habits; y_i is a binary category (risky, not risky).
- y_i are essential for the learning process.
- Methods : K-Nearest Neighbors, SVM, Decision Tree, ...

How to Encode Data

$\mathbf{X} \in \mathbb{R}^{n \times p}$

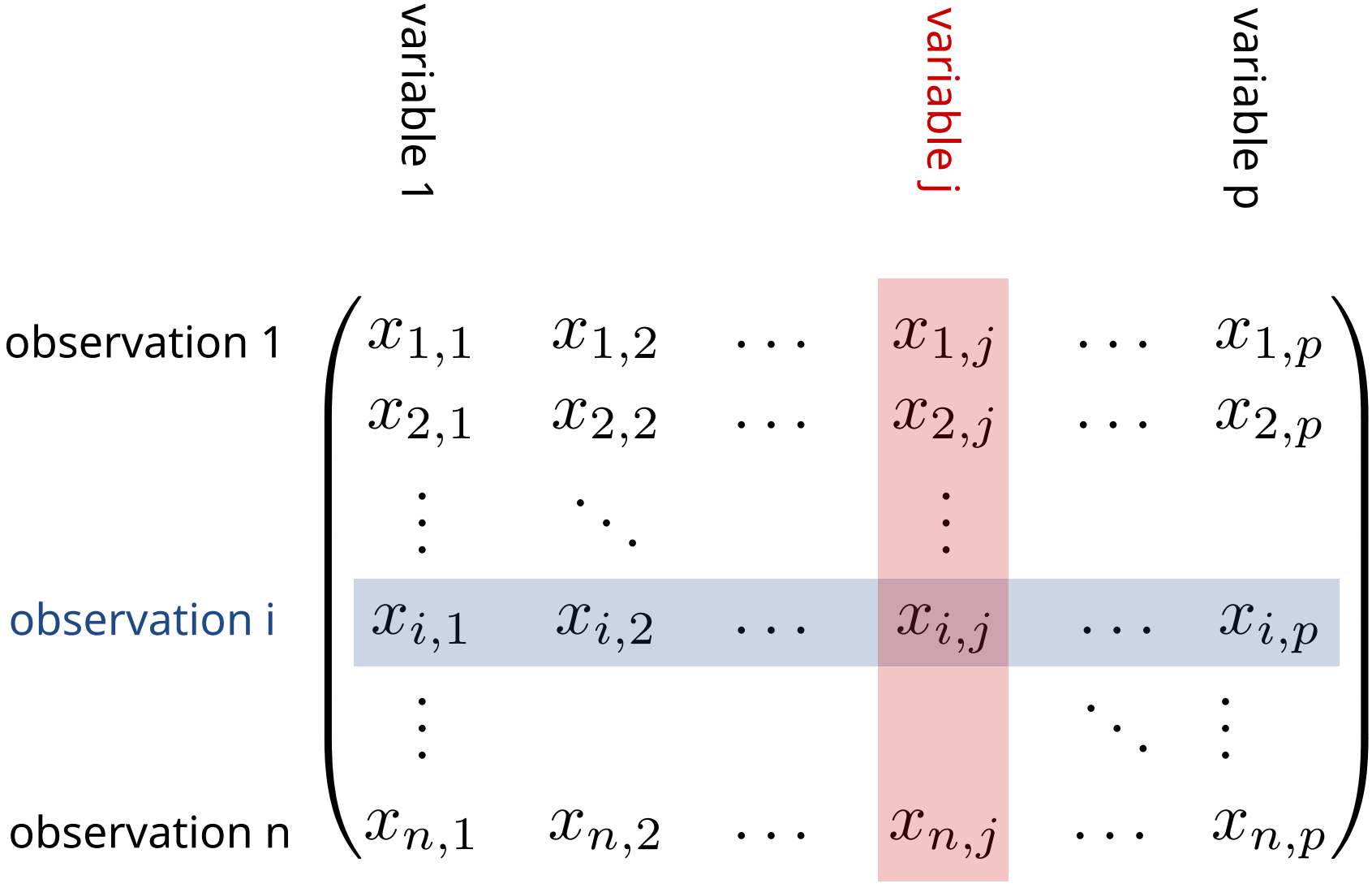
Samples

- n samples (number of lines)
- $\mathbf{X}(i,:) = \mathbf{x}_i^\top$: the i-th sample
- $\mathbf{x}_i \in \mathbb{R}^p$

Features

- p features (number of colons)
- Each sample is described by p features
- $\mathbf{X}(:,j) = \boldsymbol{x}_{\bullet j}$: The j-th feature for all sample

$\mathbf{X}(i,j)$: j-th feature of i-th sample.


variable 1
variable j
variable p
observation 1
observation i
observation n
$\begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,j} & \cdots & x_{1,p} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,j} & \cdots & x_{2,p} \\ \vdots & \ddots & & \vdots & & \\ x_{i,1} & x_{i,2} & \cdots & x_{i,j} & \cdots & x_{i,p} \\ \vdots & & & & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,j} & \cdots & x_{n,p} \end{pmatrix}$

Learning Model

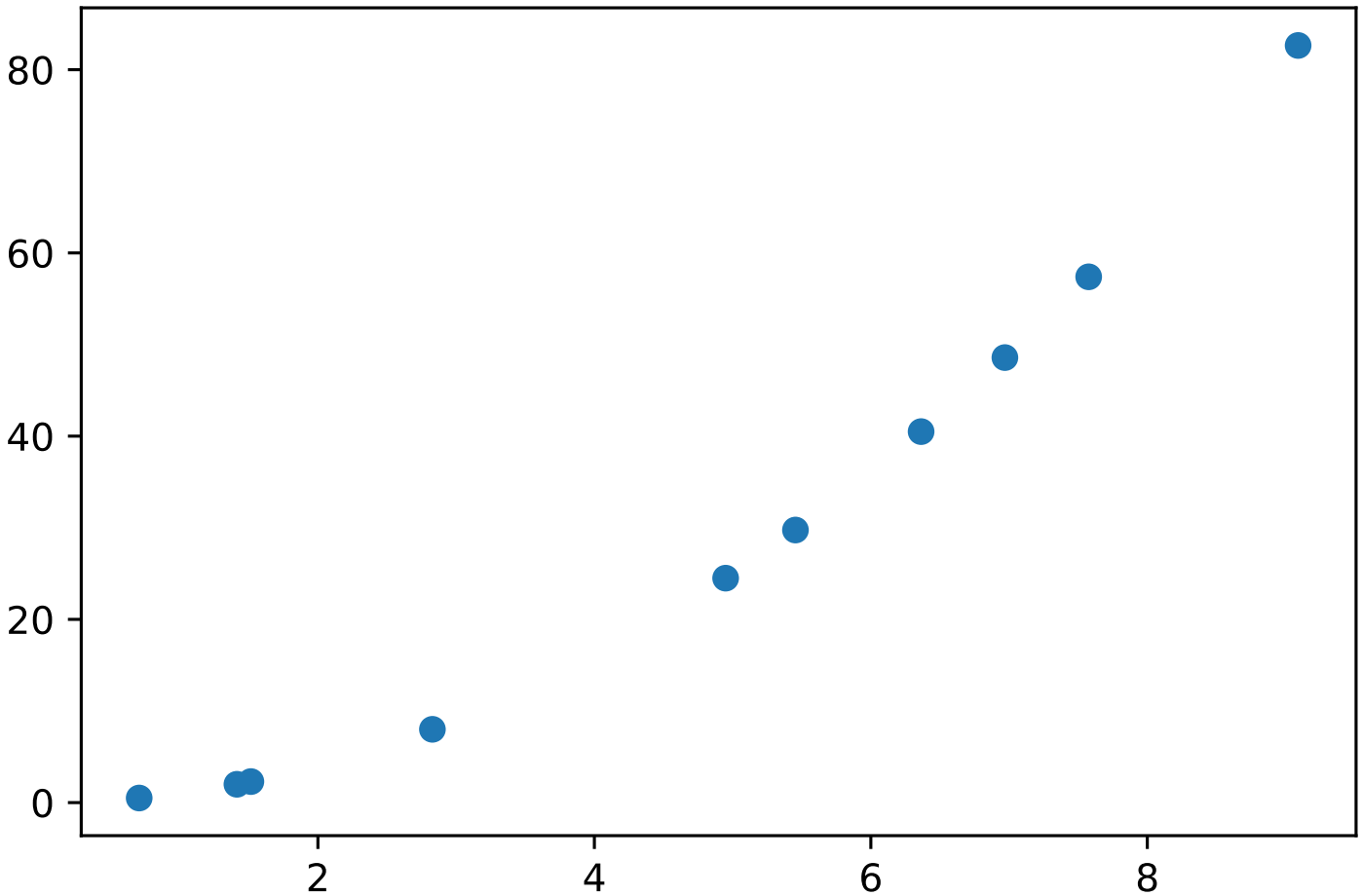
Model

$$f: \quad \mathcal{X} \to \mathcal{Y}$$
$$\mathbf{x}_i \to \hat{y}$$

We want that

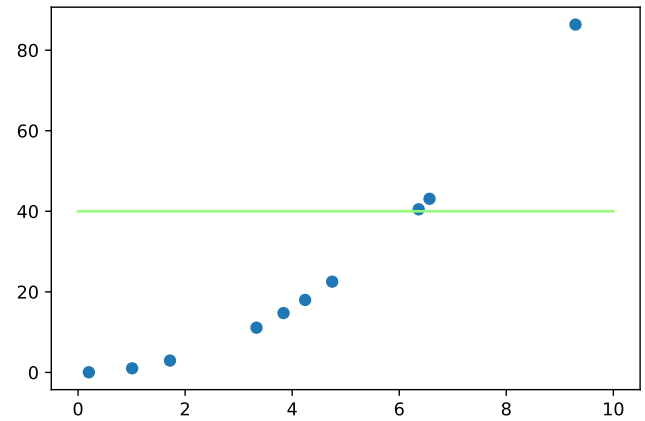
$$f(\mathbf{x}_i) \simeq y_i$$

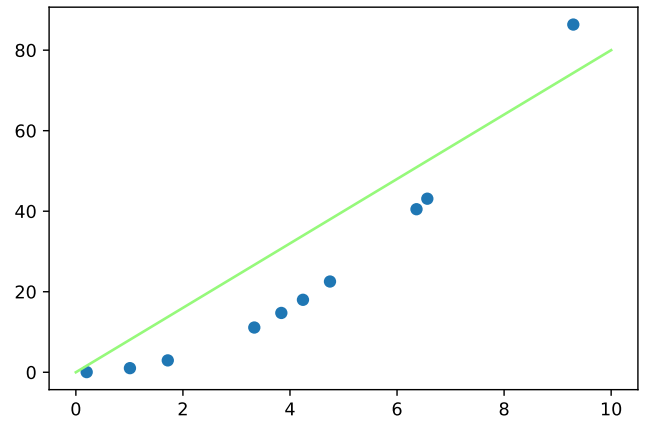
Example

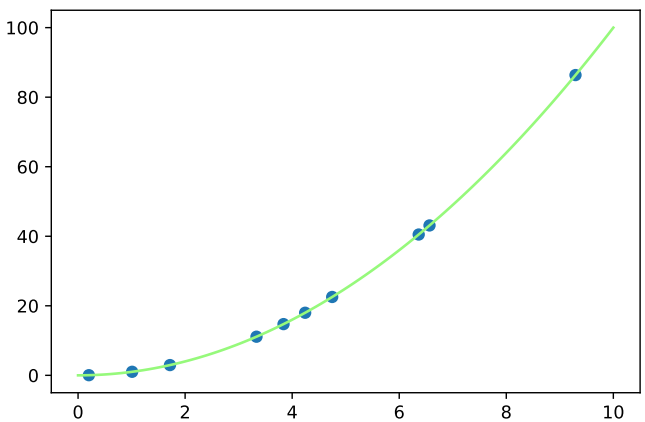


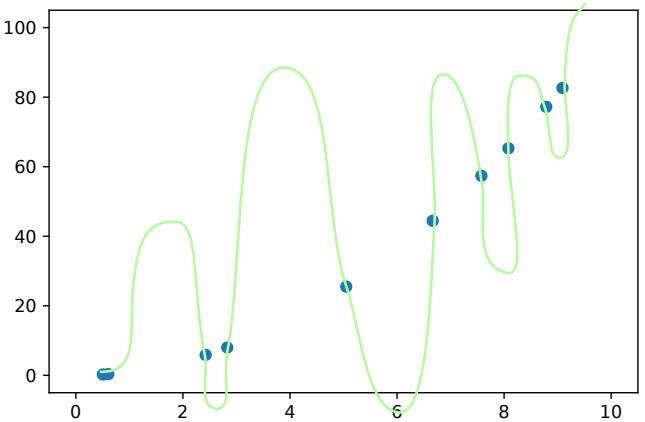
What is the underlying f function ?

Example


0
20
40
60
80
0
2
4
6
8
10


0
20
40
60
80
0
2
4
6
8
10


0
20
40
60
80
100
0
2
4
6
8
10


0
20
40
60
80
100
0
2
4
6
8
10

How to find a good f ?

$$f^\star = \arg\min_f \mathcal{L}(f(\mathbf{X}), \mathbf{y}) + \lambda\Omega(f)$$

- $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$
- $\lambda \in \mathbb{R}^+$
- $\Omega : (\mathcal{X} \to \mathcal{Y}) \to \mathbb{R}^+$

Fit to data term

$$\mathcal{L}(f(\mathbf{X}), \mathbf{y})$$

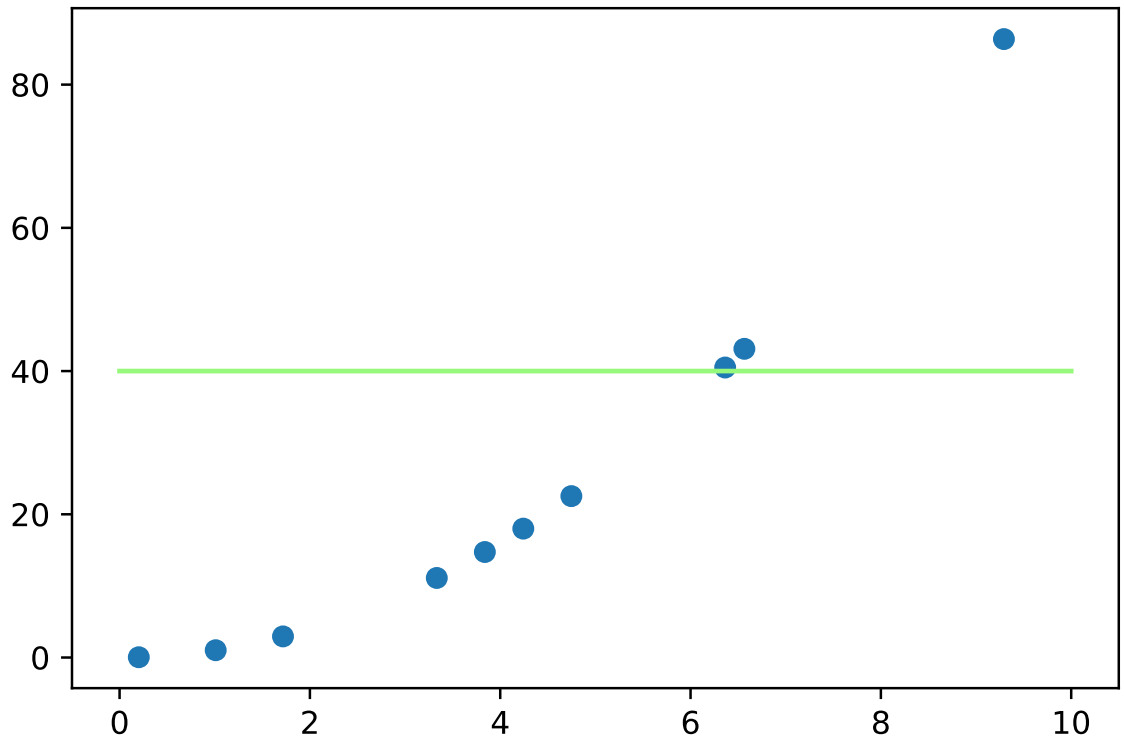
- Guarantees that the model fits the data
- Penalizes when the predicted value of $f(\mathbf{x}_i)$ is far from y_i

Regularization term

$$\Omega(f)$$

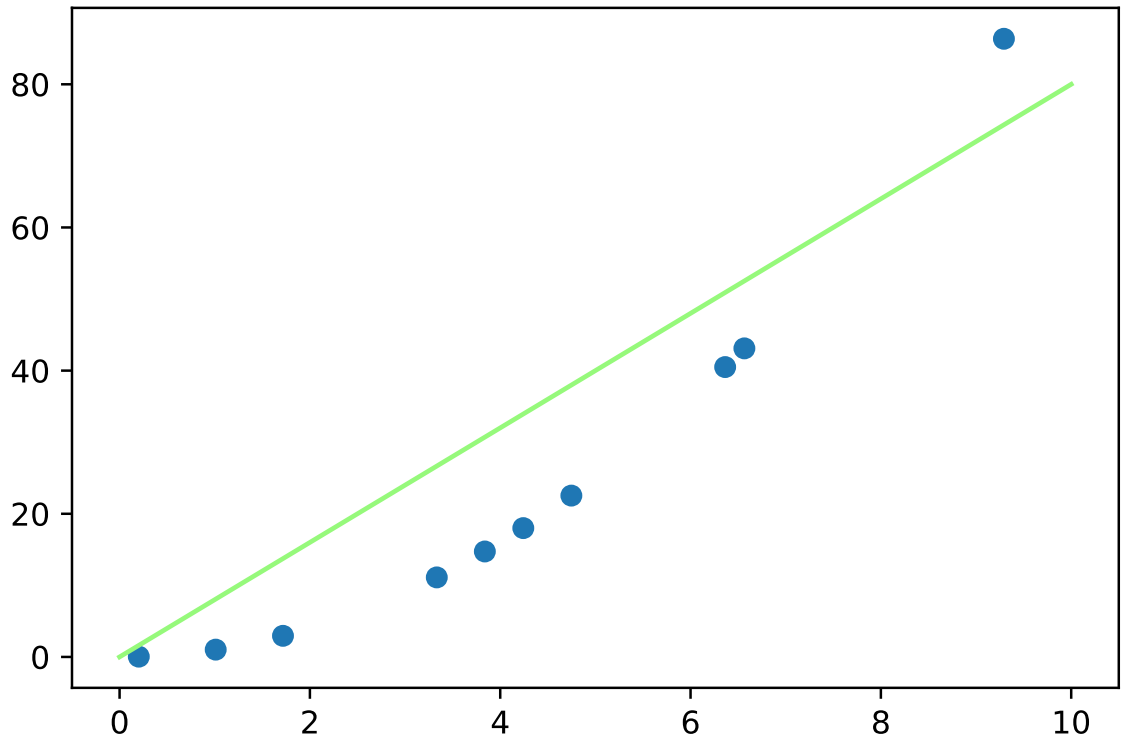
- Constrain the complexity of function f
- Occam's razor : simpler is better
- λ : Weight the balance between the two terms

Illustrations I



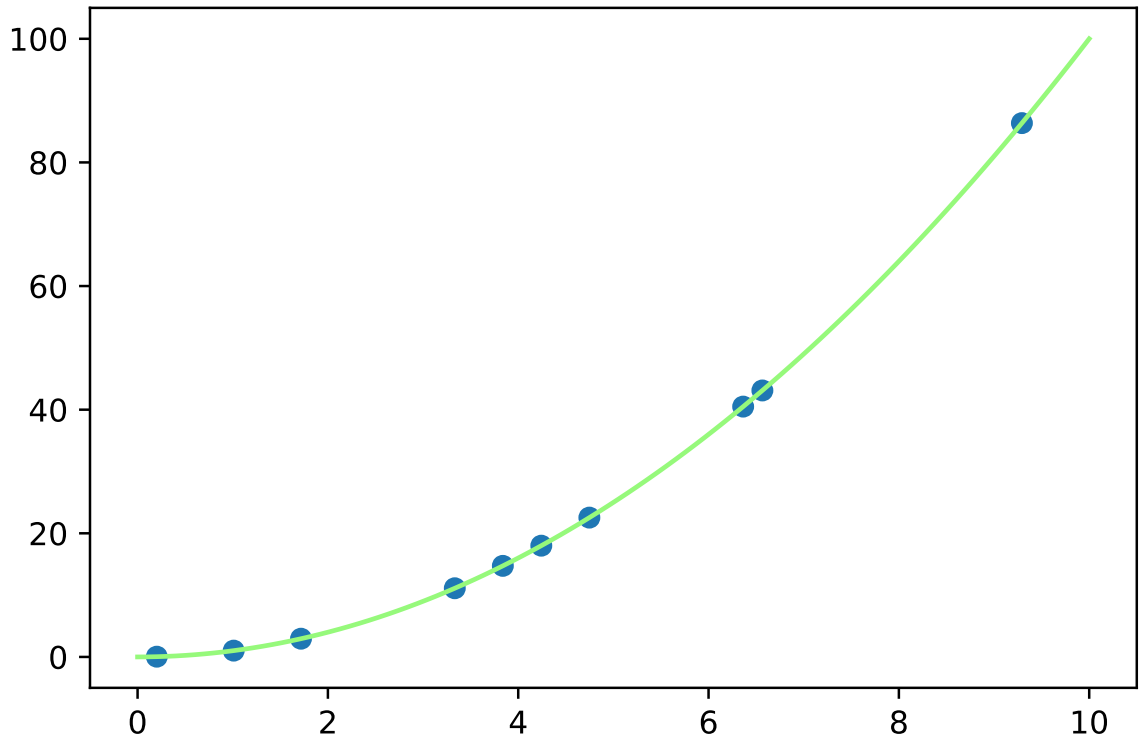
- Very high $\mathcal{L}(f(\mathbf{X}), \mathbf{y})$
- Very low $\Omega(f)$

Illustrations II



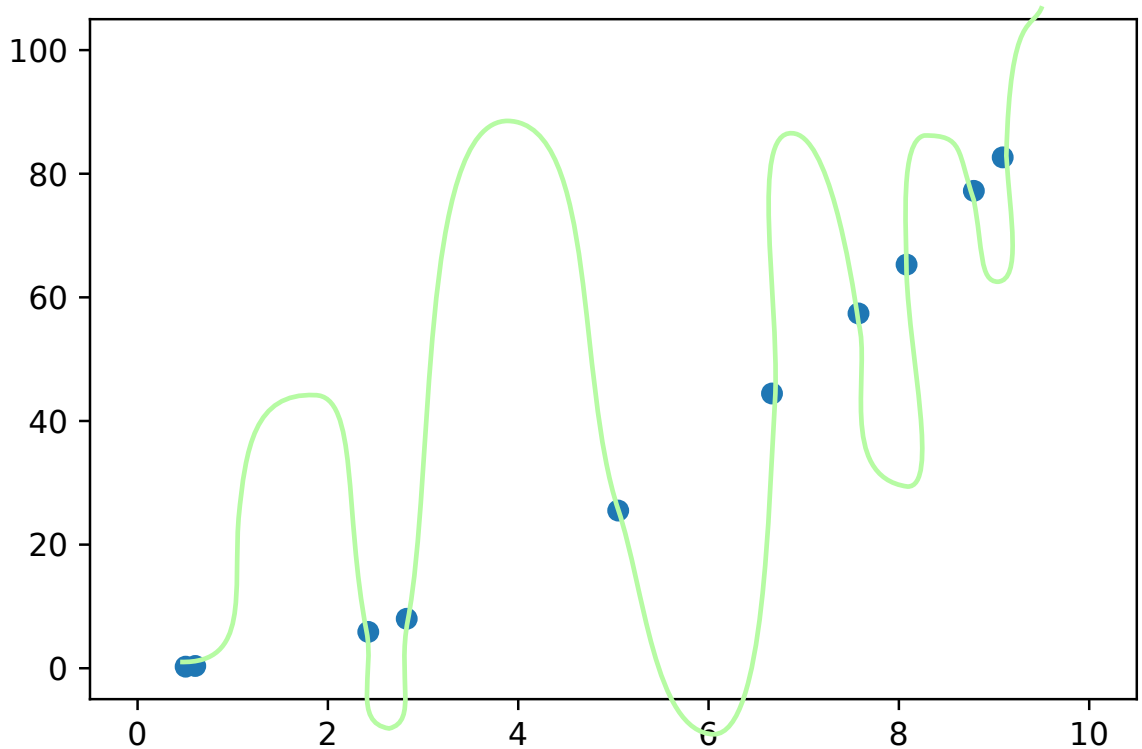
- High $\mathcal{L}(f(\mathbf{X}), \mathbf{y})$
- Low $\Omega(f)$

Illustrations III



- $\mathcal{L}(f(\mathbf{X}), \mathbf{y}) = 0$
- High $\Omega(f)$

Illustrations IV



- $\mathcal{L}(f(\mathbf{X}), \mathbf{y}) = 0$
- Very high $\Omega(f)$

Generalization

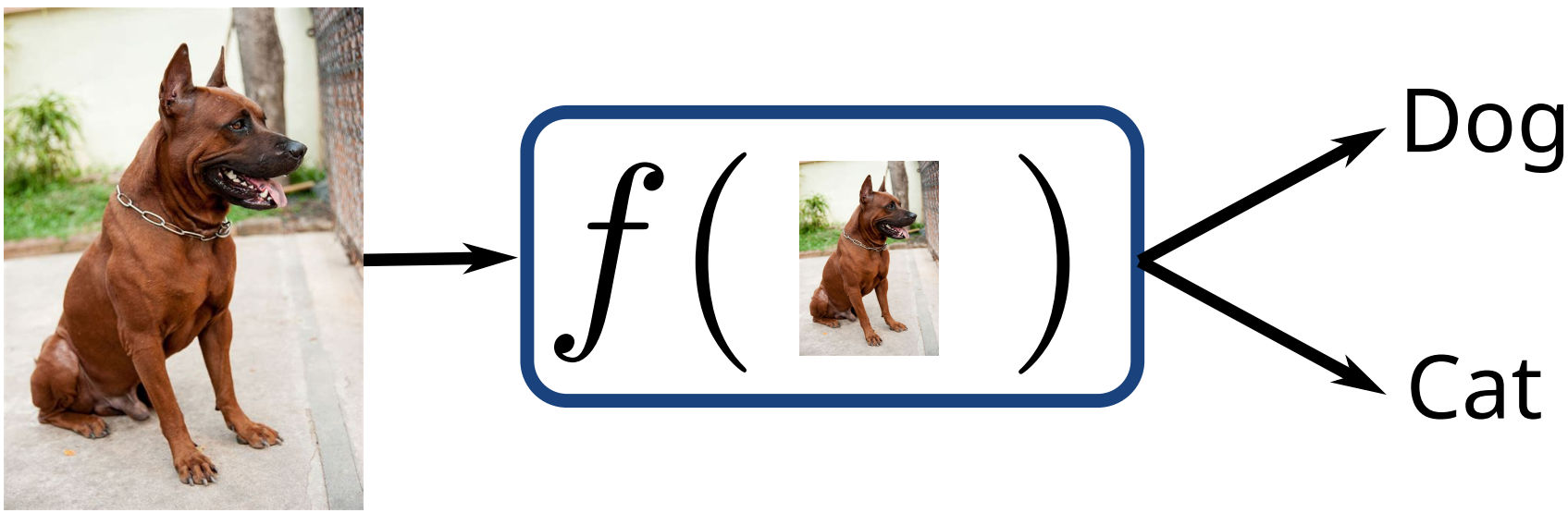
A good model generalizes well

- Good generalization : good prediction on unseen data
- Hard to evaluate without bias
-  Overfitting
- Regularization term prevents from overfitting

Supervised Learning Tasks I

Binary Classification

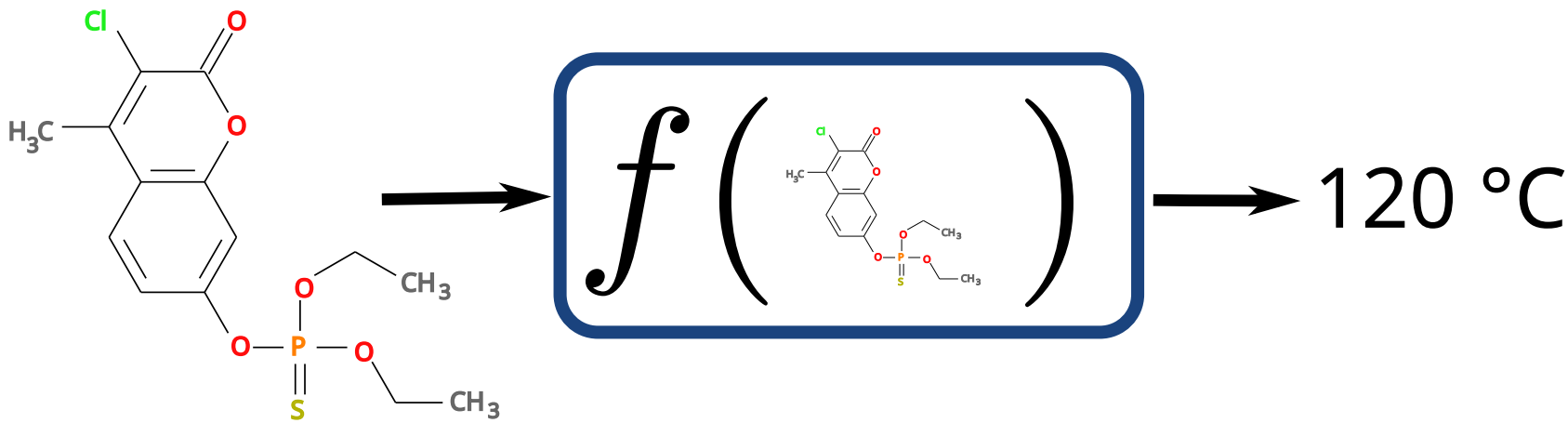
- $\mathcal{Y} = \{0, 1\}$
- Dog or cat ? Positive or Negative ?
- Performance : Accuracy, recall, precision, ...



Supervised Learning Tasks II

Regression

- $\mathcal{Y} = \mathbb{R}$
- Stock market, House price, boiling points of molecules, ...
- Performance: RMSE, MSE, MAE, ...



Supervised Learning Tasks III

And many others

- Ranking
- MultiClass classification
- Multi Labeling
- ...

Machine learning Methods for Classification

- K-nearest neighbors
- Random forests
- SVM & consorts
- Multi Layer Perceptron

k Nearest Neighbors

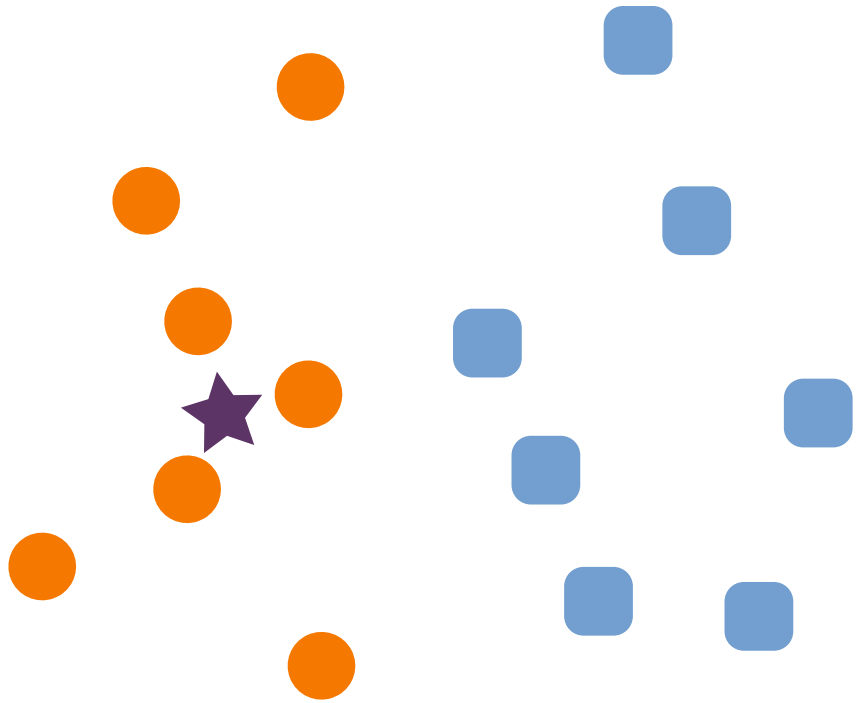
Principle

Determine property from the property of similar data

k Nearest Neighbors

Principle

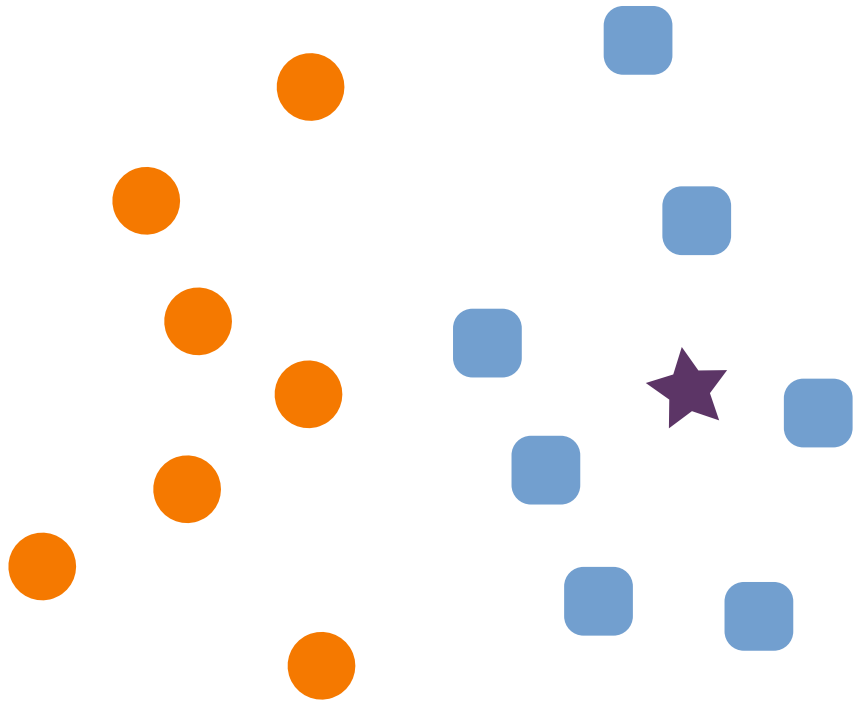
Determine property from the property of similar data



k Nearest Neighbors

Principle

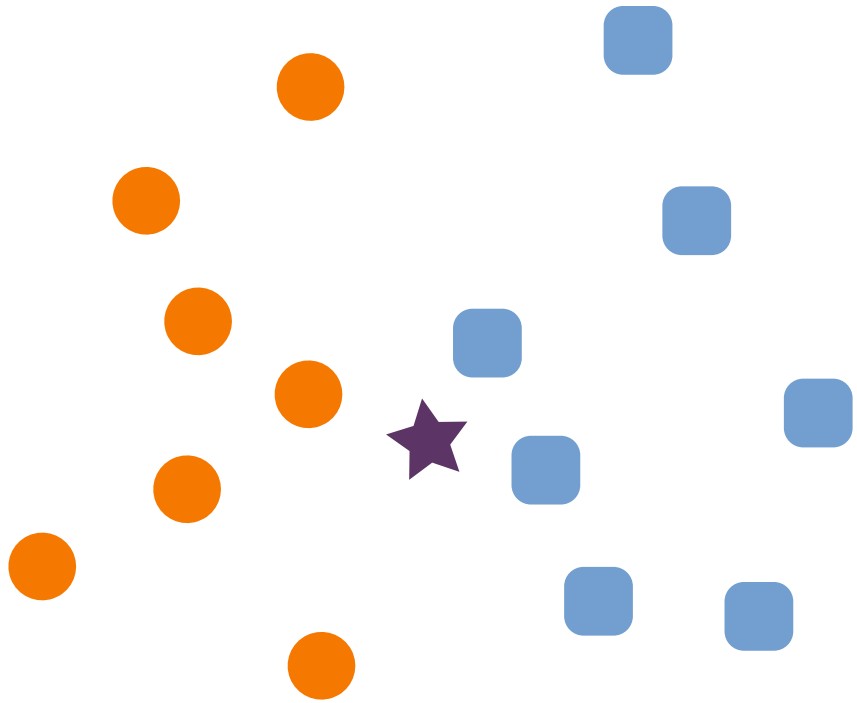
Determine property from the property of similar data



k Nearest Neighbors

Principle

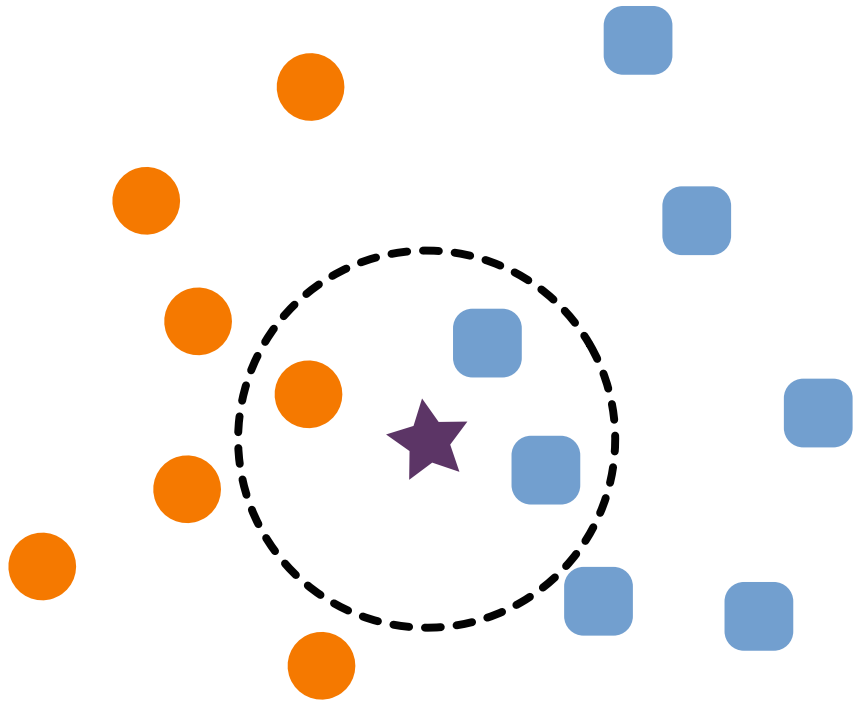
Determine property from the property of similar data



k Nearest Neighbors

Principle

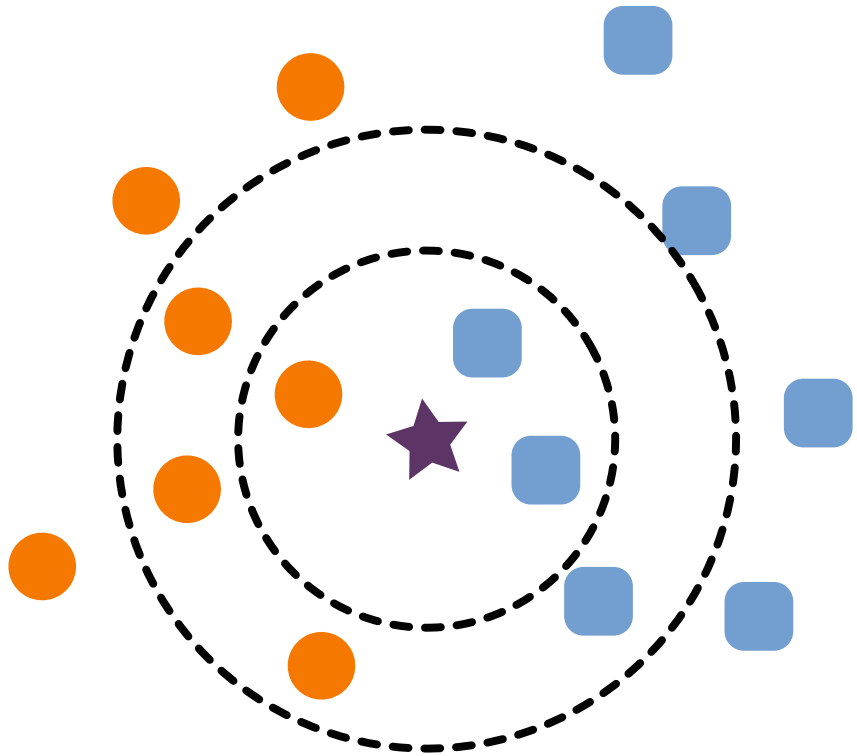
Determine property from the property of similar data



k Nearest Neighbors

Principle

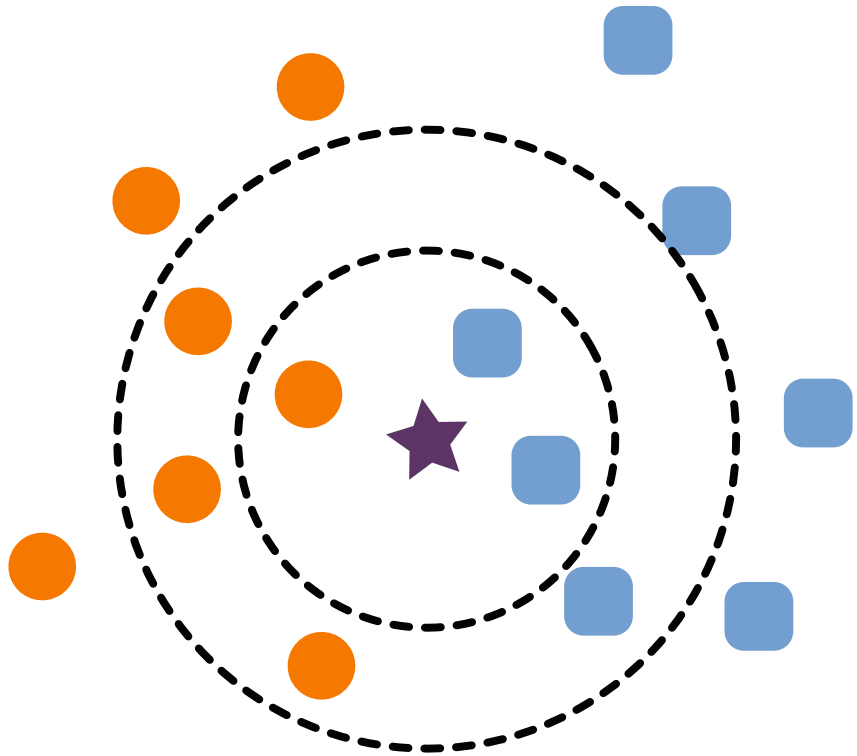
Determine property from the property of similar data



k Nearest Neighbors

Principle

Determine property from the property of similar data



K-NN Hyperparameters

Number of neighboors k

Quantify the number of neighbors used for the decision

- Low number : high variability, high accuracy
- High number : more smooth

Distance

How the similarity is determined

- Similarity depends on data, structure, task
- Euclidean, Manhattan, ad hoc distances

KNN : the code !

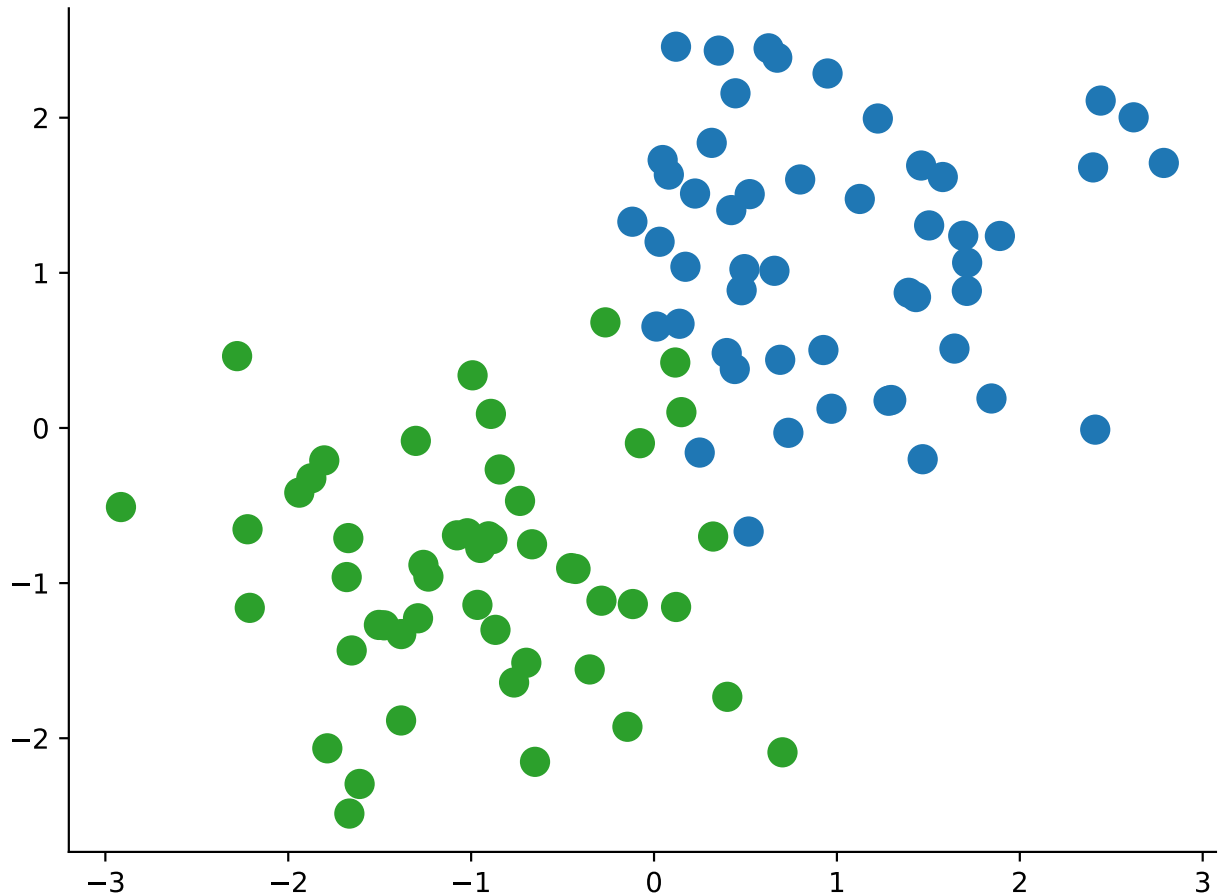
```
1  from sklearn.neighbors import KNeighborsClassifier
2  k=5
3  metric = 'manhattan'
4  knn = KNeighborsClassifier(n_neighbors=k,metric=metric)
5  knn.fit(X,y) #apprentissage
6  y_pred = knn.predict(X) # prediction
```

- Metrics : link
- $\Rightarrow$ Notebook
- the documentation

Decision Tree

Principle

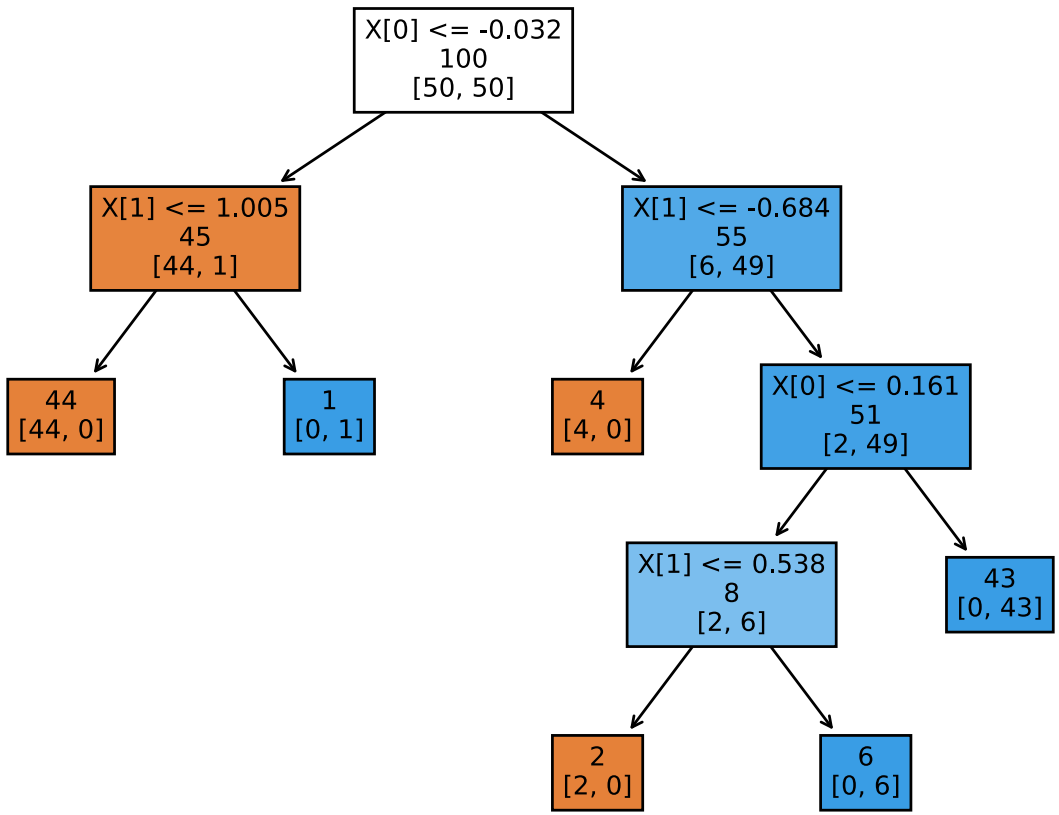
Learn decision rules to separate the data.



Decision Tree

Principle

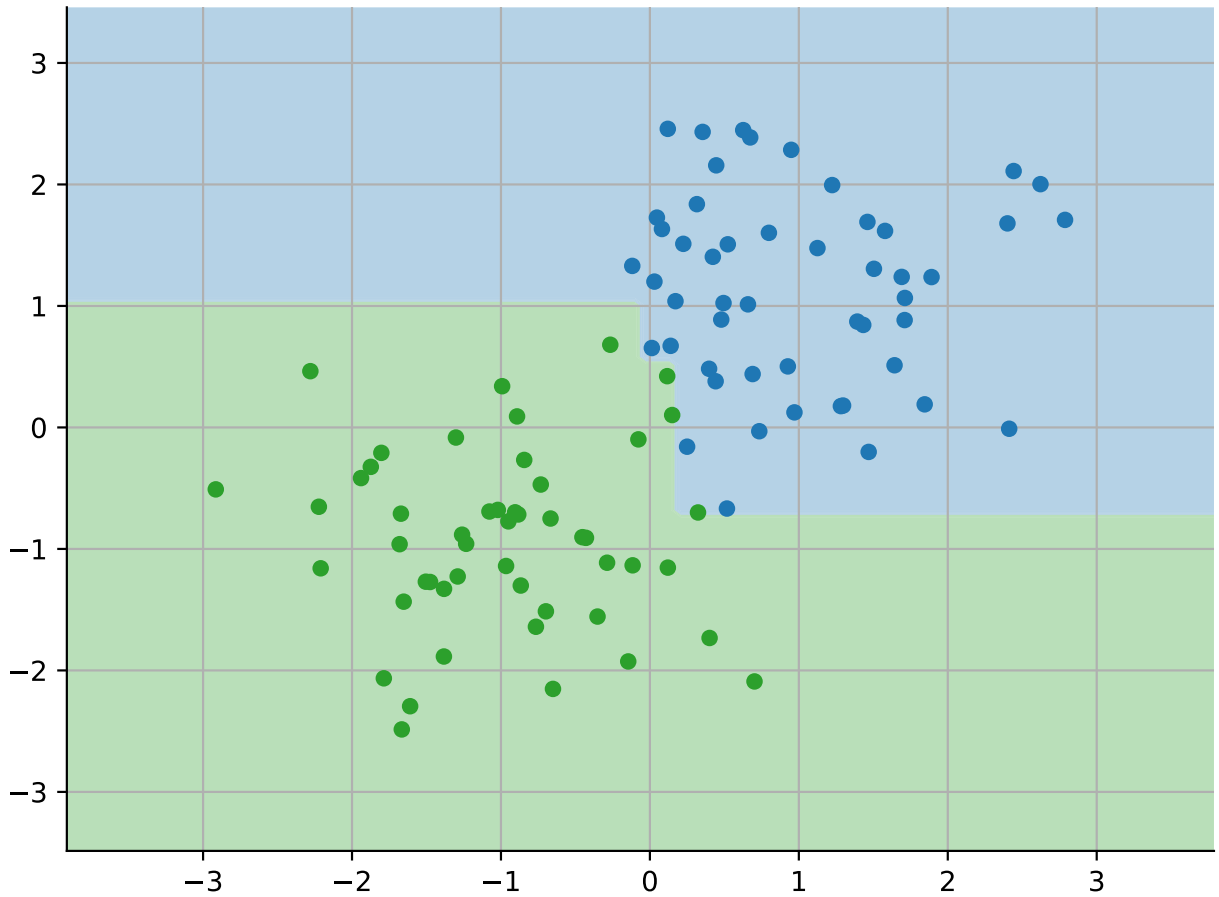
Learn decision rules to separate the data.



Decision Tree

Principle

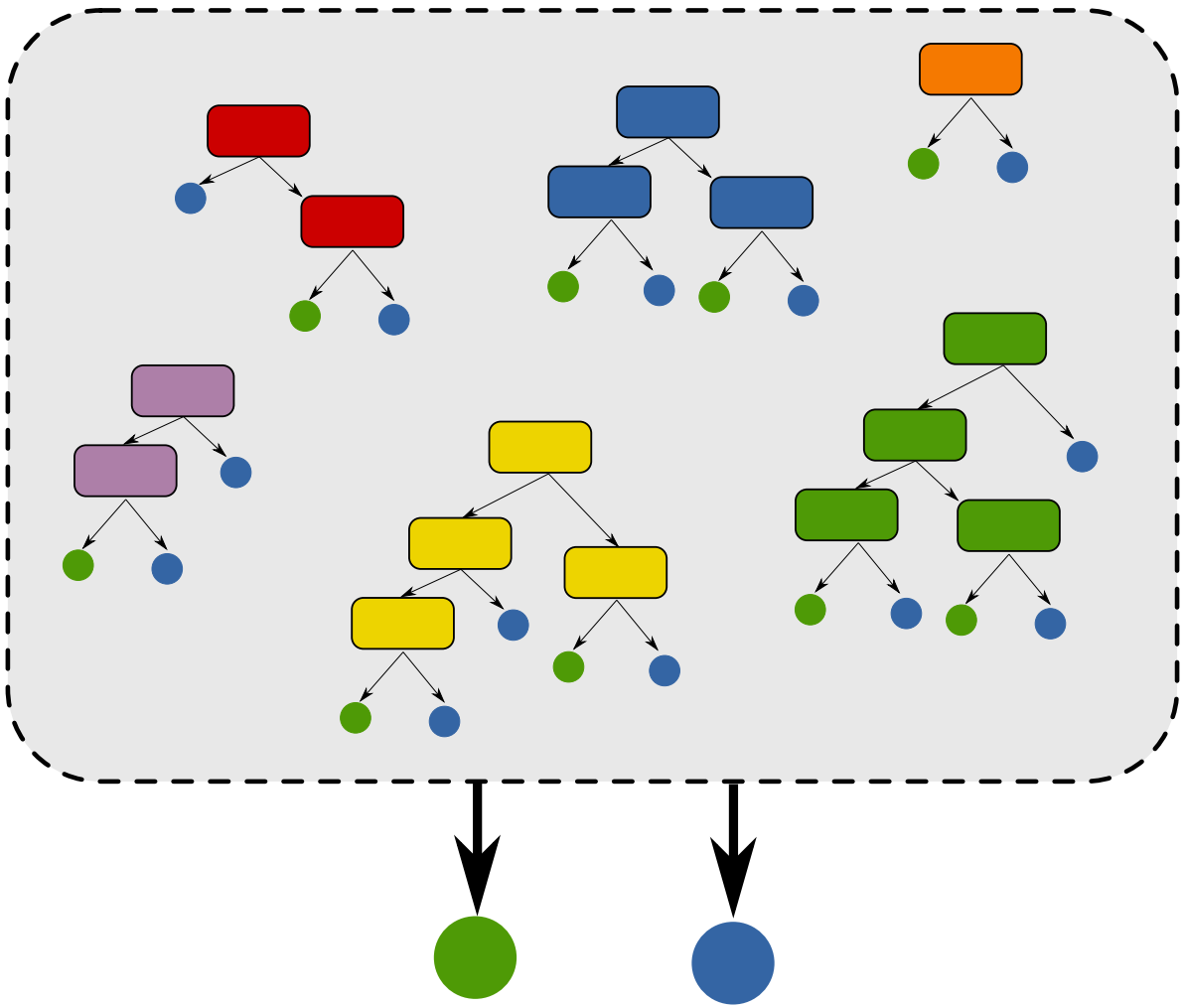
Learn decision rules to separate the data.



Random Forests

Principle

- Combine many decision trees to learn complex functions
- Ensemble methods, majority voting



Random Forests Hyperparameters

Number of trees

Adjust the number of trees composing the forests

- low number : fast to compute, but less accurate
- high number : slower to compute, but more accurate up to some number

Number of features

Determine the number of features to be used when splitting the data

- See the guidelines of `scikit-learn`

Tree depth

Specify the maximal depth of tree. An higher depth will make dedicate categories, but less generalizable.

Random Forests : the code !

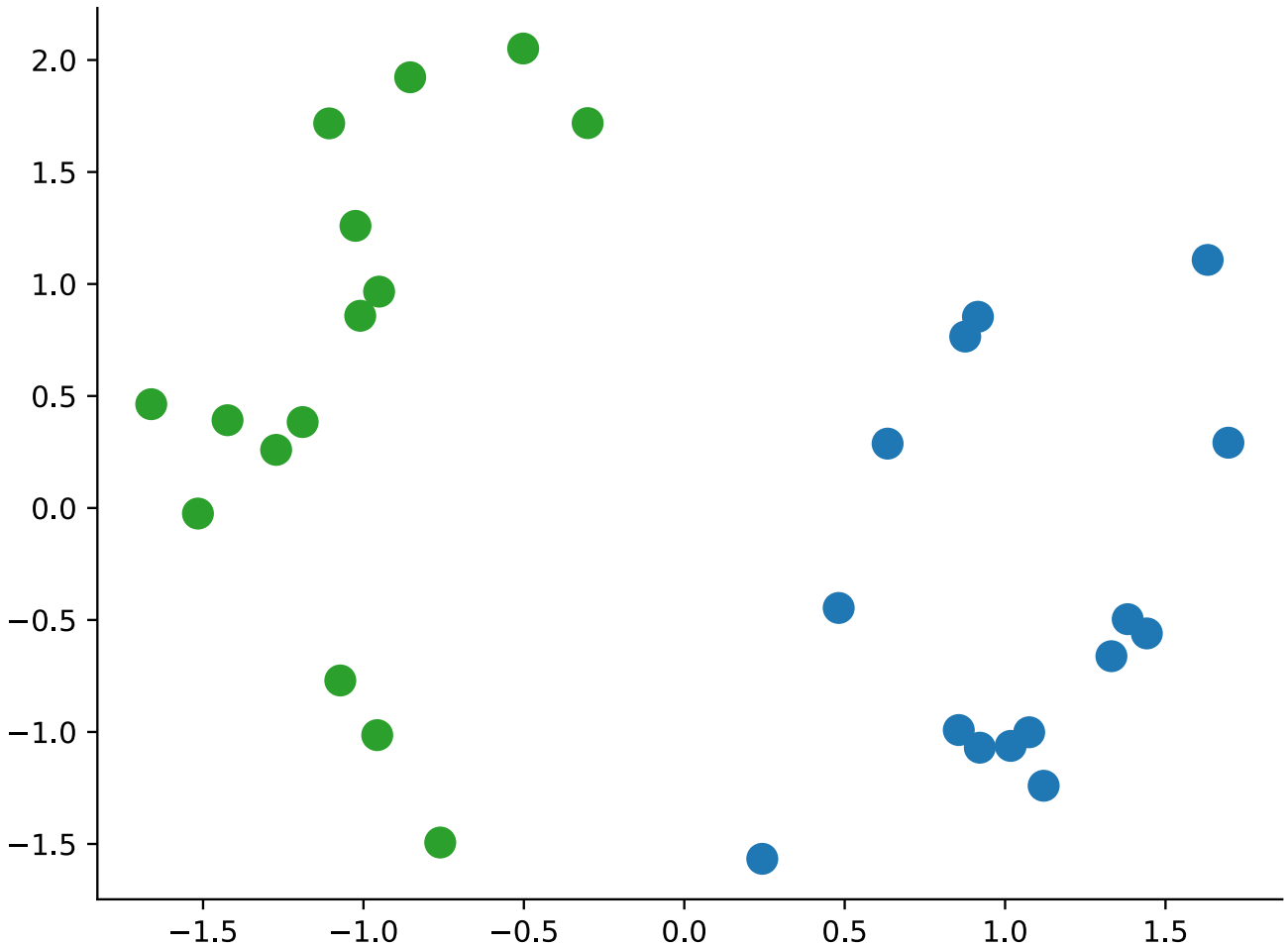
```python
from sklearn.ensemble import RandomForestClassifier
n_estimators = 20 # the number of trees in the forest
max_depth = None # expand as you can
max_features = "sqrt" # RTFM
clf = RandomForestClassifier(n_estimators=n_estimators,
                             max_depth=max_depth,
                             max_features=max_features)
clf.fit(X,y)
ypred = clf.predict(X)
```

- User guide for hyperparameters : link
- $\Rightarrow$ Notebook
- the documentation

Support Vector Machines

Principle

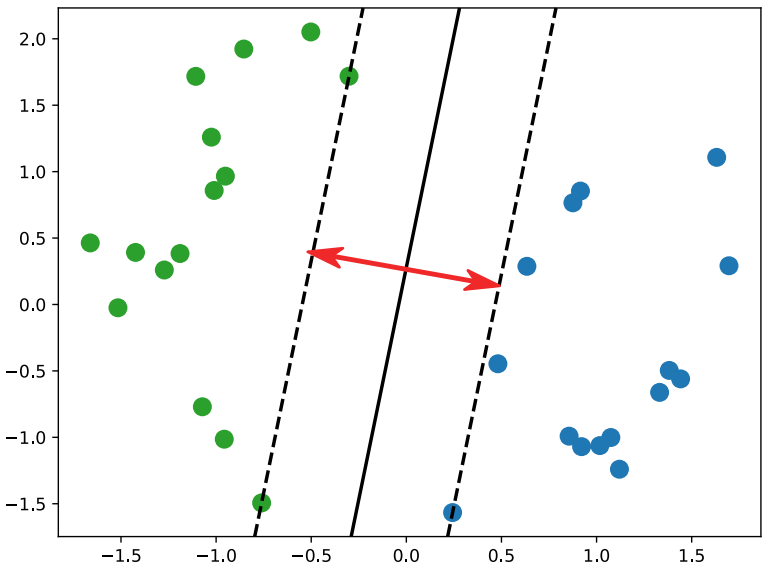
Find the best line which separates the data

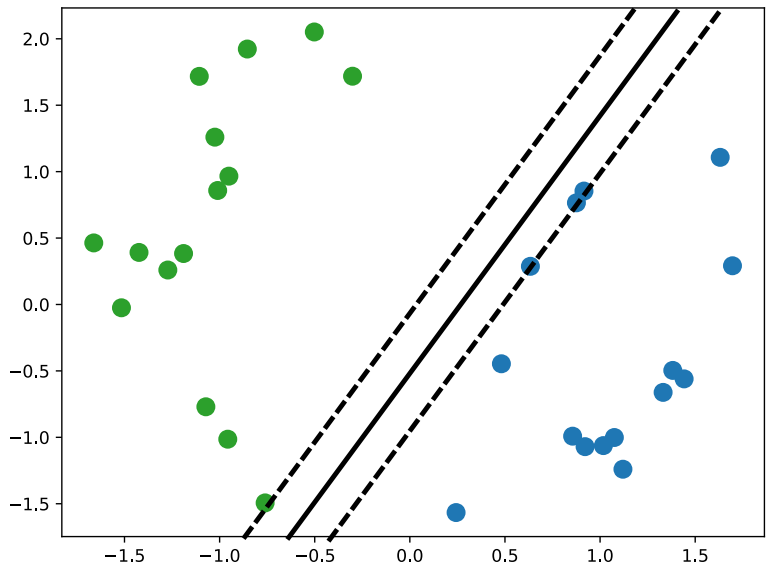


Support Vector Machines

Principle

Find the best line which separates the data



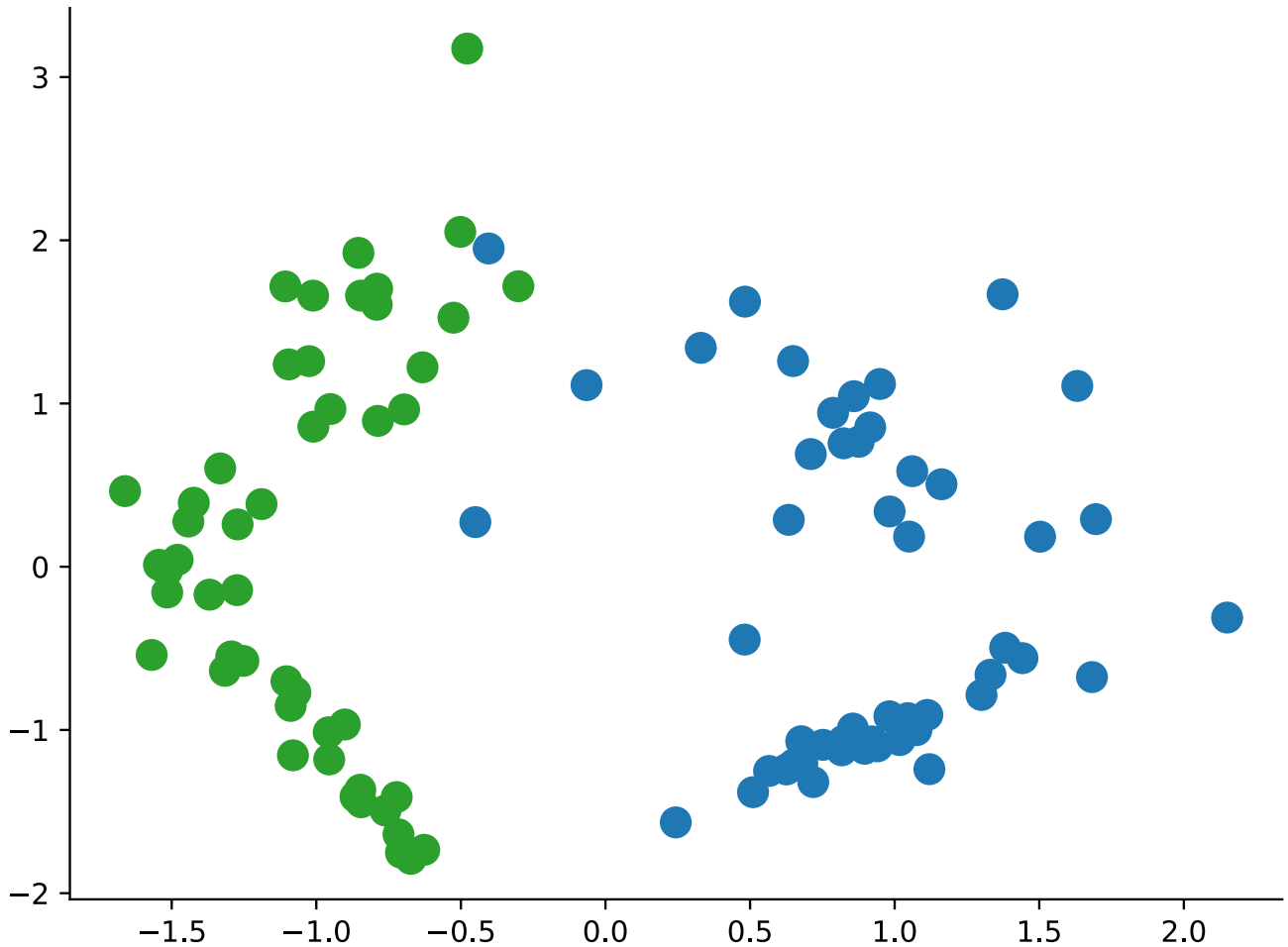


- Best separation $\Rightarrow$ points far away the separation
- Points on margin : support vectors

Support Vector Machines

General Principle

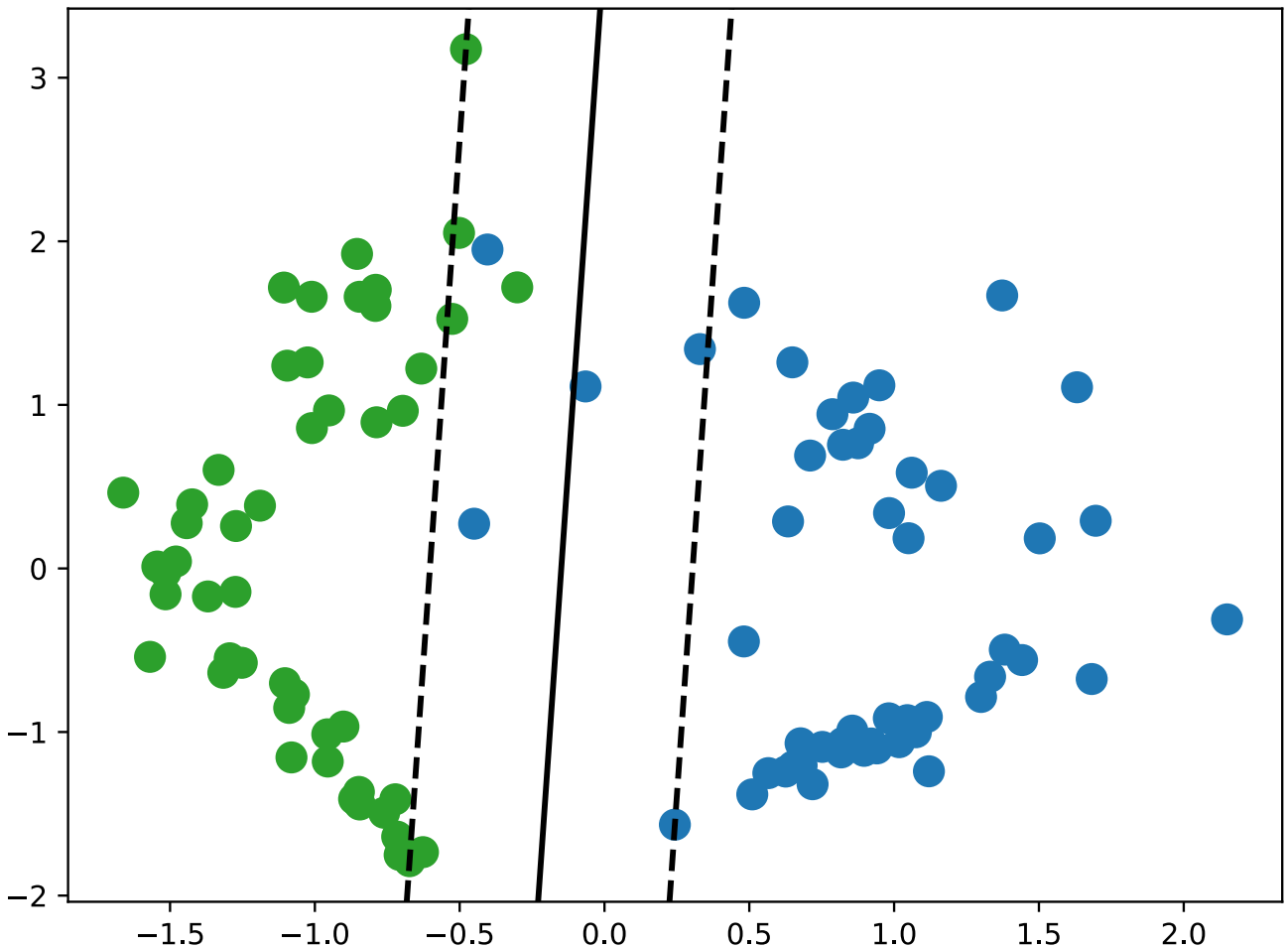
What happens when there is no separation line ?



Support Vector Machines

General Principle

What happens when there is no separation line ?

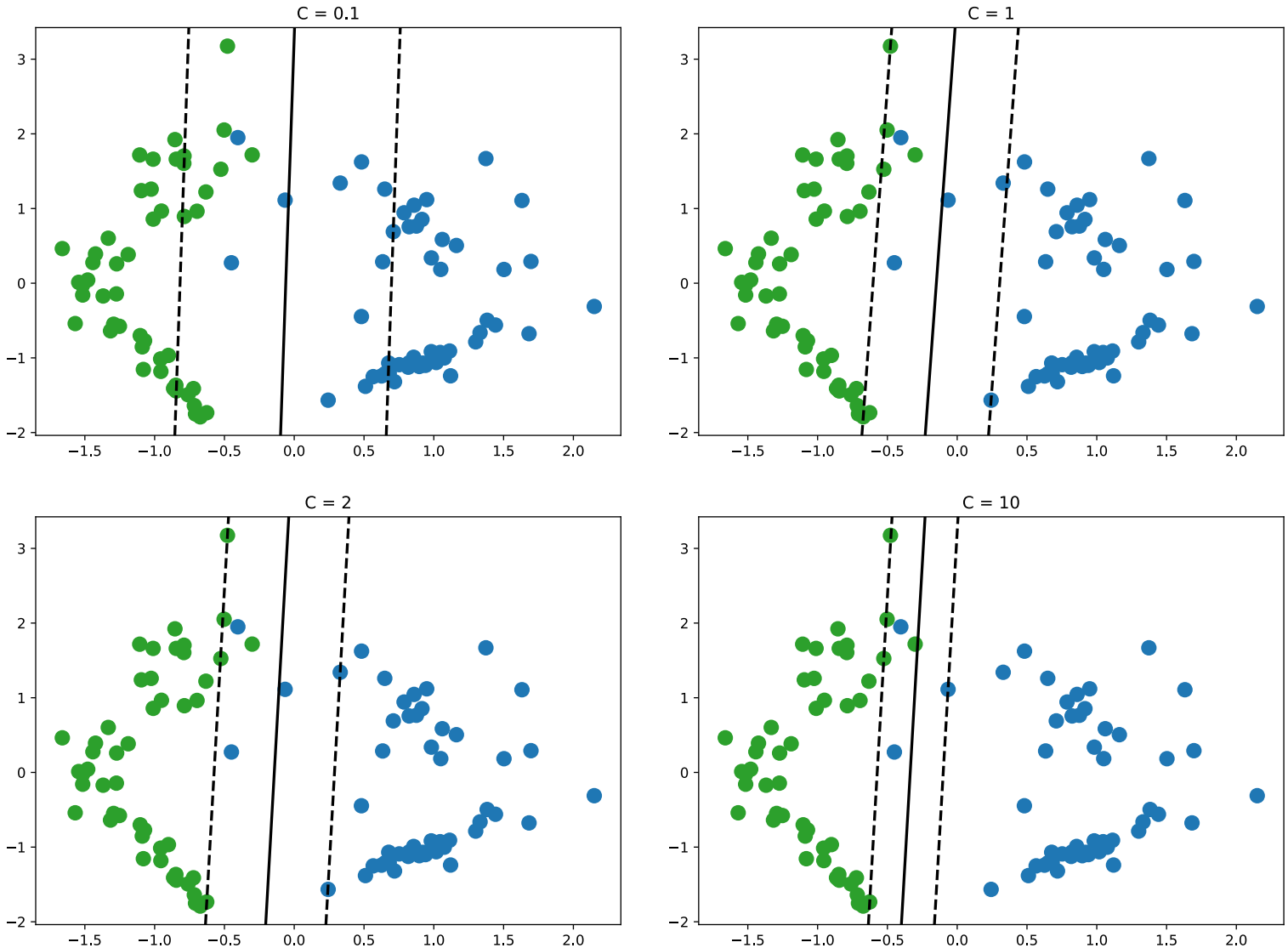


- $\Rightarrow$ We allow errors !

Support Vector Machines

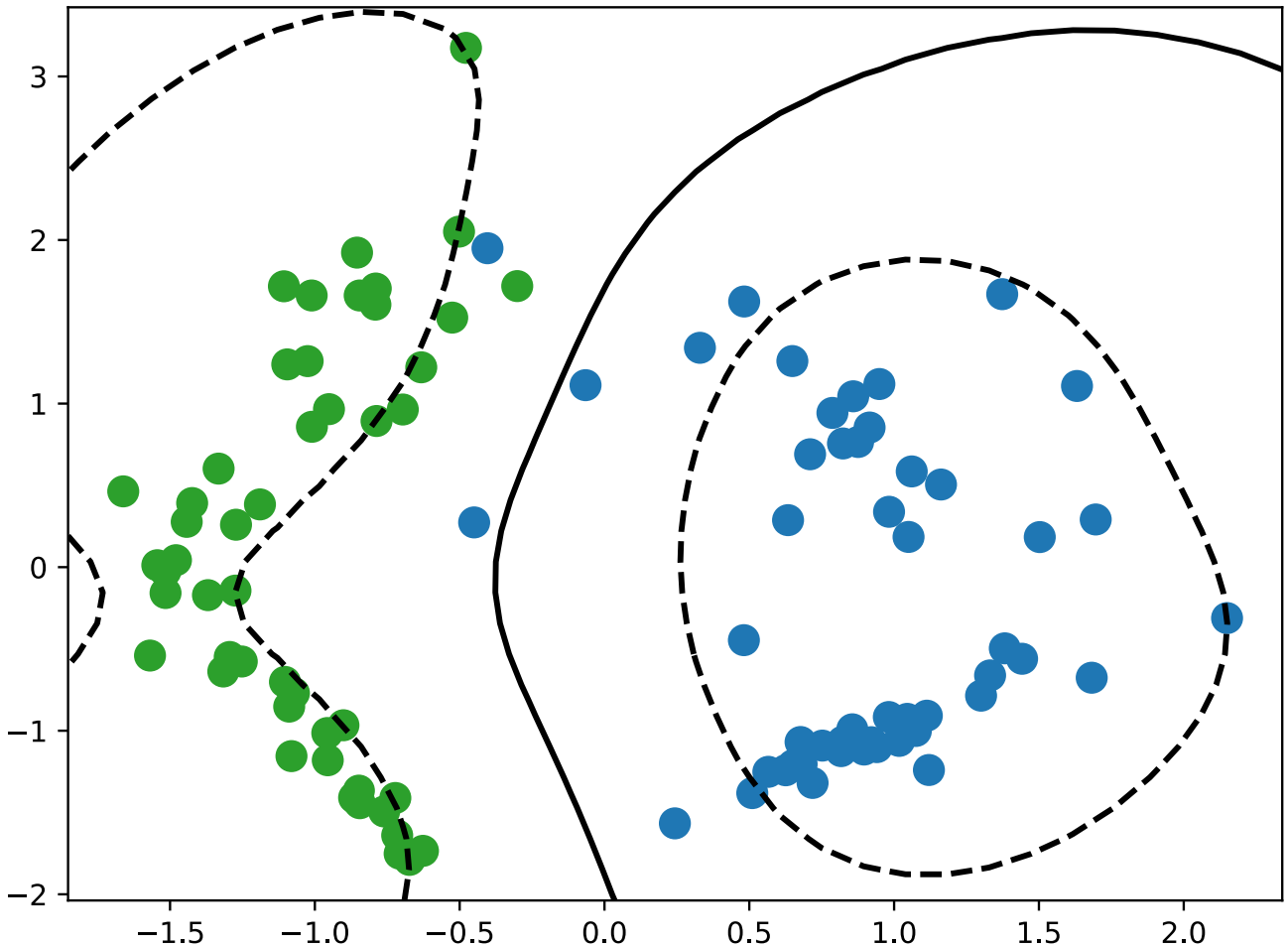
Error control

C controls the trade off between errors and separation



Extension to non linear separation

Thanks to kernel trick, SVM can compute any kind of separation line



- Depends on kernel

SVM Hyperparameters

C

Adjust the importance of errors

- low C : fast to compute, more simple separation, more errors
- high number : slower to compute, less errors, maybe too complex separation line

kernel

Determine how the separation line is build

- linear : straight line
- poly, rbf, sigmoid : complex lines, basic choice is rbf
- precomputed : provide a similarity matrix (more difficult)

SVM : the code !

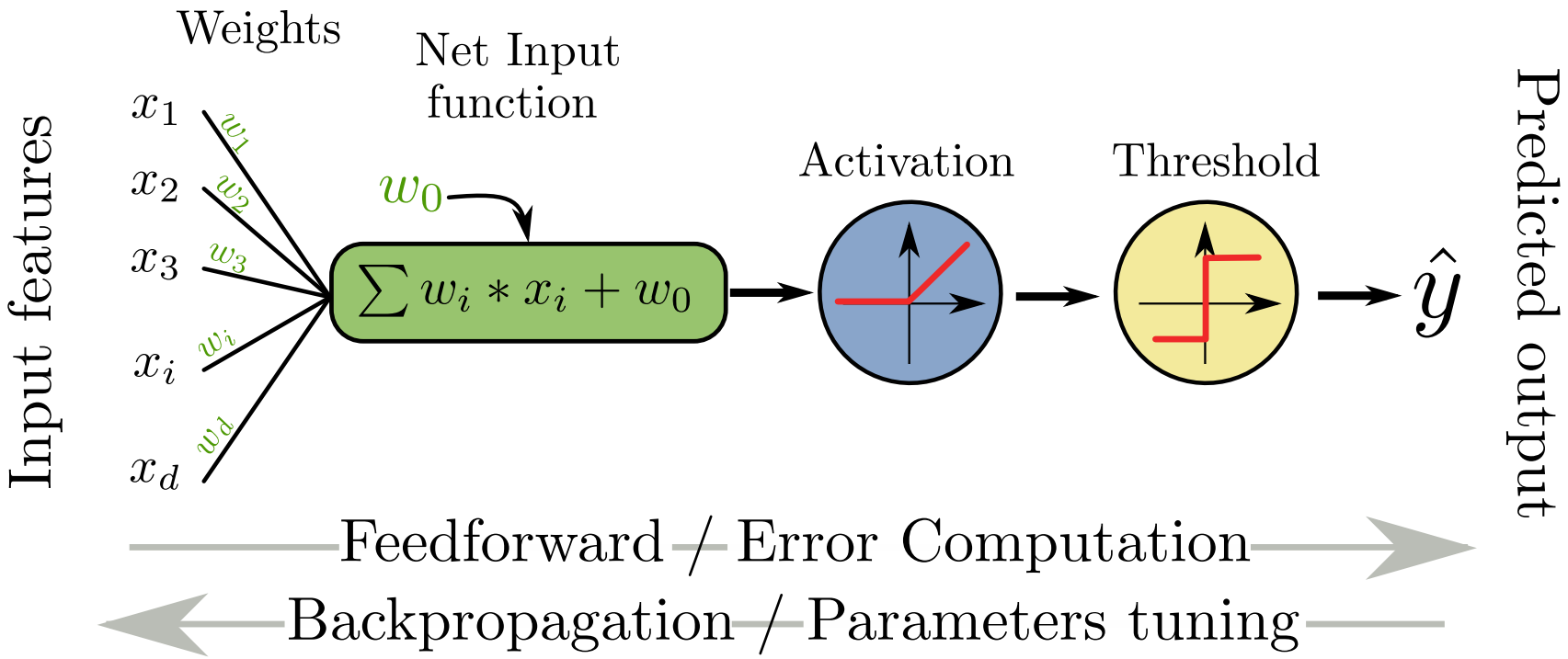
```
1  from sklearn import svm
2  C = 1
3  kernel = 'rbf'
4  clf = svm.SVC(C=C,kernel=kernel)
5  clf.fit(X,y)
6  ypred = clf.predict(X)
```

- User guide for hyperparameters : link
- $\Rightarrow$ Notebook
- the documentation

Multi Layer Perceptron

Principle

Learn the best representation of data



- Weights $\boldsymbol{w}$ are optimized by gradient descent
- Sequence of layers

MLP Hyperparameters

Hidden layers

Define the architecture of your MLP

- Number of layers : a high number tends to deep networks
- Number of neurons per layer : a high number tends to wide networks

The model will be more complex if more neurons are used

Activation function

Determine how the non linearity is brought to the model

- identity : linear model
- tanh, relu, logistic : non linears. ReLU is a very popular choice

MLP : the code !

```
1  from sklearn.neural_network import MLPClassifier
2  activation = 'relu' # default
3  layers = [32,64,128,64,32] #5 layers avec différentes tailles
4  clf = MLPClassifier(hidden_layer_sizes=layers,max_iter=500)
5  clf.fit(X,y)
6  ypred = clf.predict(X)
```

- User guide for tips and help : link
- $\Rightarrow$ Notebook
- the documentation

Pratical

How to learn a "good" model ?

- We want good performance
- Simple as possible
- Able to predict unseen data

Empirical Risk

Error on learning set

- Empirical risk:

$$R_{emp}(f) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(\mathbf{x}_i), y_i)$$

- $\mathcal{L}$ evaluates the performance of prediction $f(\mathbf{x}_i)$
- Error is computed on the training set
- The model can be too specialized on this particular dataset

Generalisation

Tentative of Definition

- Ability of the model to predict well unseen data
- Hard to evaluate
- Real objective of a model

Regularisation

- Regularization term control the model
- Balances between empirical risk and generalization ability
- Need to tune the balance (λ)

How to evaluate to ability to generalize ?

Evaluate on unseen data

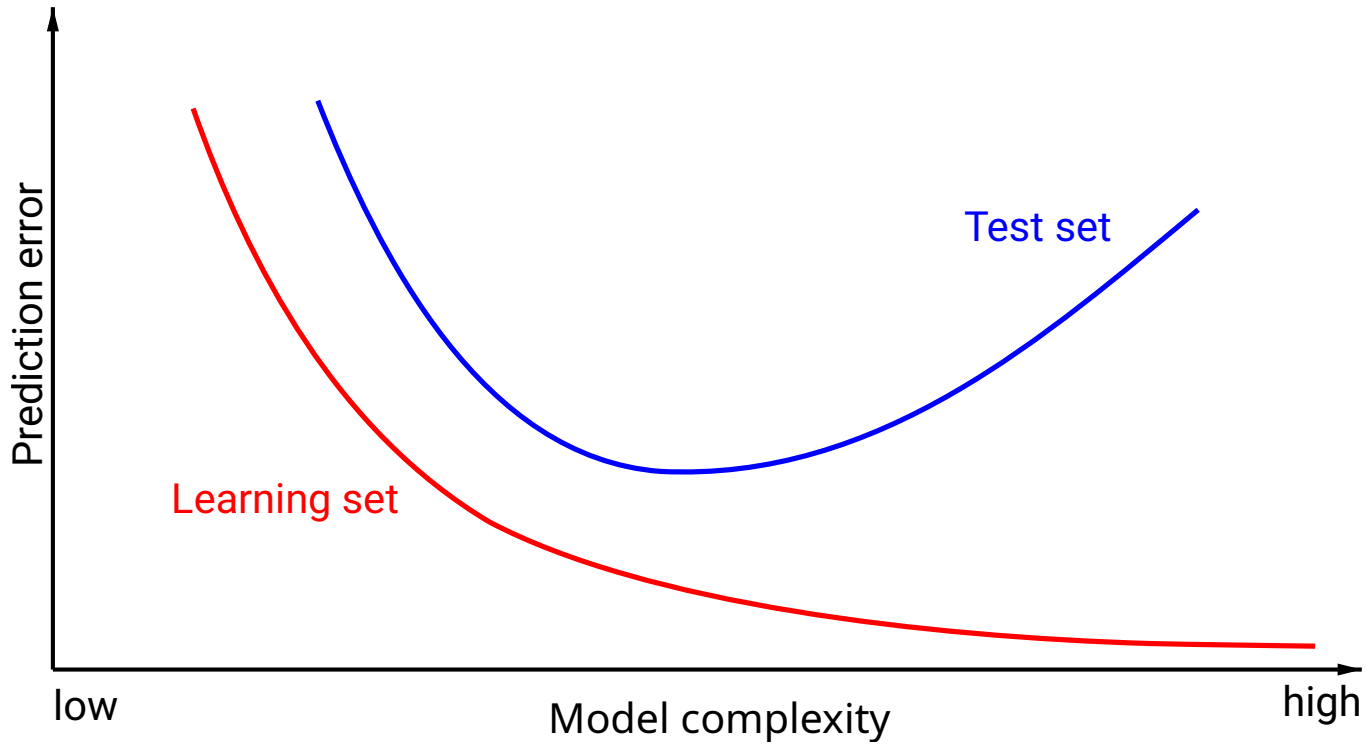
- Define and isolate a test set
- Evaluate on the test set

Bias

- Avoid to use same data in train and test
- Test set must be totally isolated

Overfitting vs Underfitting

- Overfitting: low R_{emp}, high generalization error
- Underfitting: high R_{emp}, medium generalization error



Hyperparameters

Parameters outside the model

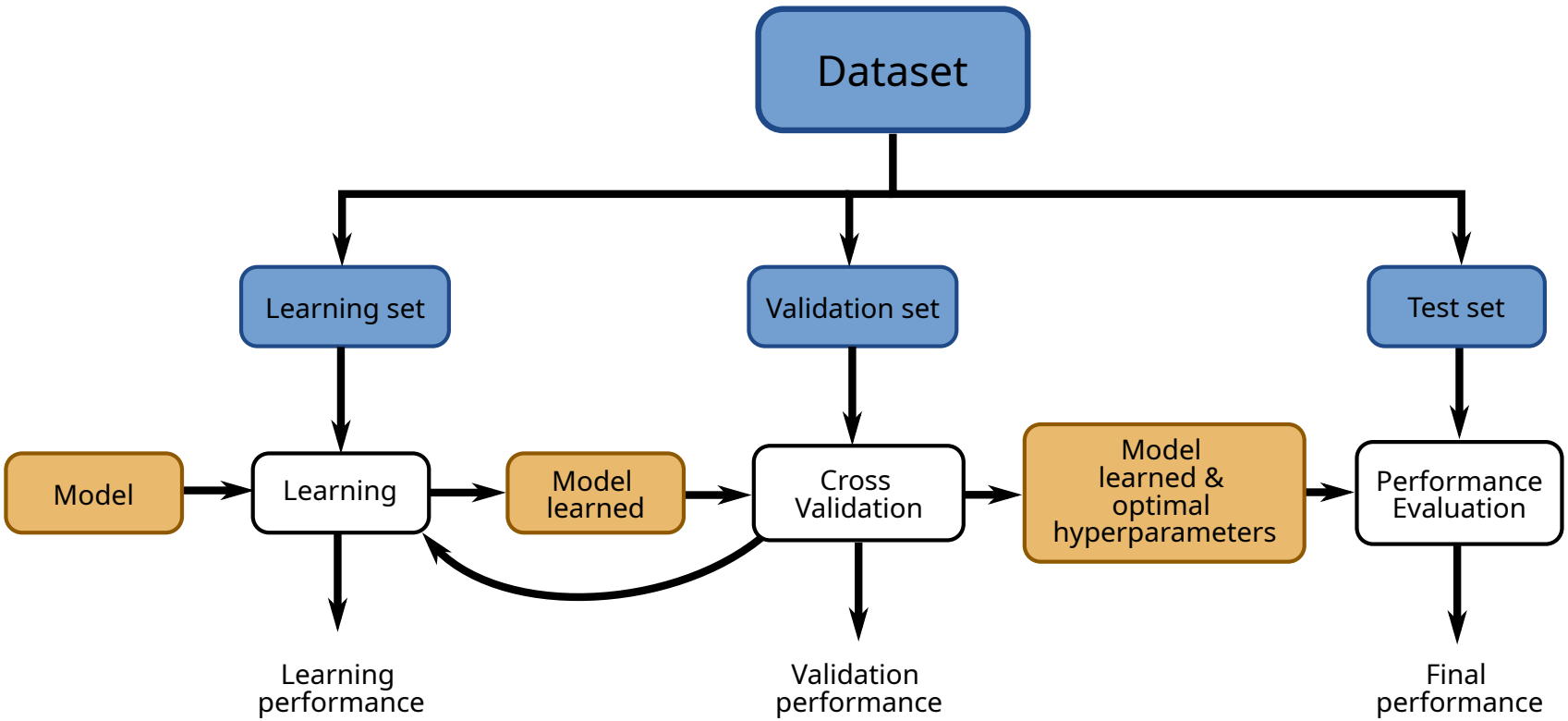
- Some parameters are not learned by the model
- They are "hyperparameters" and must be tuned
-  Tuned on data outside the test set
- Example: λ in Ridge Regression

How to tune the hyperparameters ?

Validation set

- Split train set into validation and learning set
- Learn model parameters using the learning set
- Evaluate the performance on validation set
- Validation set simulates the test set, aka unseen data

General framework



General CV pseudo code

1: **function** EVAL($X, y,$ model, h_params)
2: best_perf $\leftarrow 0$
3: $X_{train}, y_{train}, X_{test}, y_{test} \leftarrow$ split(X, y)
4: **for** h_param $\in$ h_params **do**
5: perf $\leftarrow 0$
6: **for** $i \in 1 \ldots 10$ **do**
7: $X_t, y_t, X_v, y_v \leftarrow$ cv_split$(X_{train}, y_{train}, i)$
8: fitted_model $\leftarrow$ train(model, $X_t, y_t,$ h_param)
9: $y_{pred} \leftarrow$ predict(fitted_model, X_v)
10: perf $\leftarrow$ perf + score(y_pred, y_v)
11: **end for**
12: **if** perf $<$ best_perf **then**
13: best_perf $\leftarrow$ perf
14: best_param $\leftarrow$ h_param
15: **end if**
16: **end for**
17: fitted_model $\leftarrow$ train(model, $X_{train}, y_{train},$ best_param)
18: $y_{pred} \leftarrow$ predict(fitted_model, X_{test})
19: **return** score(y_{pred}, y_{test})
20: **end function**

Still Learned at Top Conf

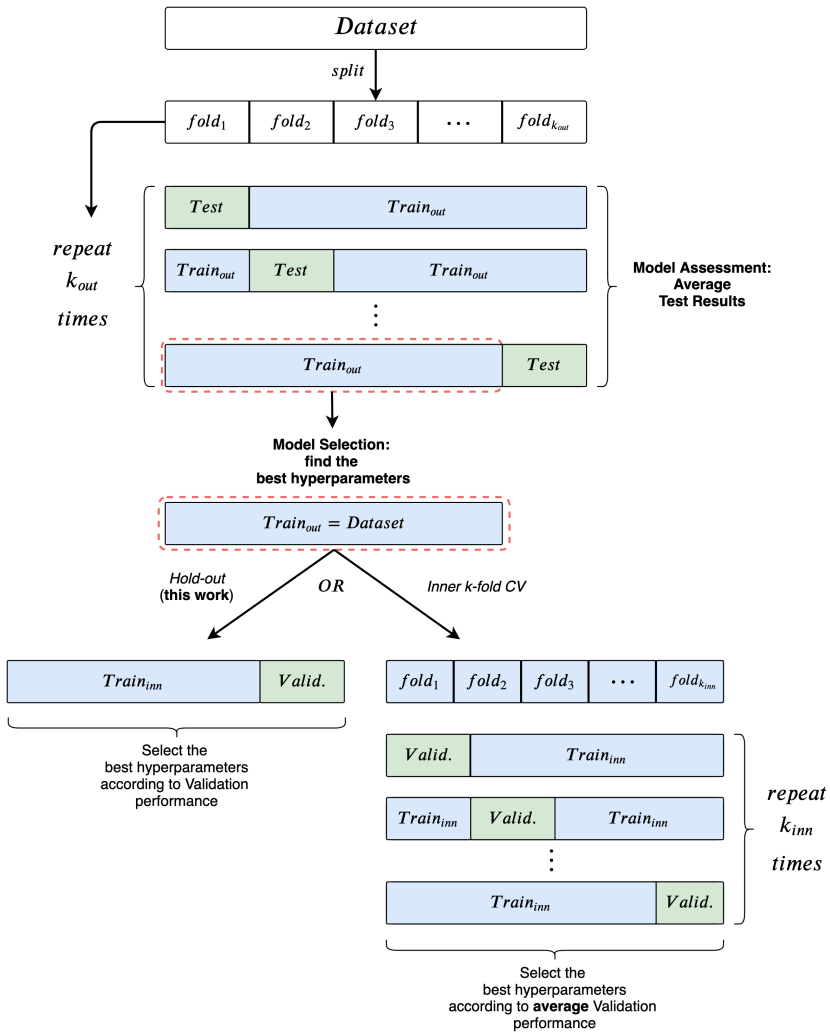
F. Errica, M. Podda, D. Bacciu, and A. Micheli, 'A Fair Comparison of Graph Neural Networks for Graph Classification'. ICLR 2020. http://arxiv.org/abs/1912.09893

Algorithm 1 Model Assessment (k-fold CV)

Input: Dataset $\mathcal{D}$, set of configurations Θ
Split $\mathcal{D}$ into k folds $F_1, \ldots, F_k$
for $i \leftarrow 1, \ldots, k$ **do**
    $\text{train}_k, \text{test}_k \leftarrow \left(\bigcup_{j \neq i} F_j\right), F_i$
    $\text{best}_k \leftarrow \text{Select}(\text{train}_k, \Theta)$
    for $r \leftarrow 1, \ldots, R$ **do**
        $\text{model}_r \leftarrow \text{Train}(\text{train}_k, \text{best}_k)$
        $\text{p}_r \leftarrow \text{Eval}(\text{model}_k, \text{test}_k)$
    end for
    $\text{perf}_k \leftarrow \sum_{r=1}^{R} \text{p}_r / R$
end for
return $\sum_{i=1}^{k} \text{perf}_i / k$

Algorithm 2 Model Selection

Input: train_k, Θ
Split train_k into *train* and *valid*
$\text{p}_\theta = \emptyset$
for each $\theta \in \Theta$ **do**
    $\text{model} \leftarrow \text{Train}(\text{train}_k, \theta)$
    $\text{p}_\theta \leftarrow \text{p}_\theta \cup \text{Eval}(\text{model}, \textit{valid})$
end for
$\text{best}_\theta \leftarrow \text{argmax}_\theta \, \text{p}_\theta$
return best_θ



Validation strategies

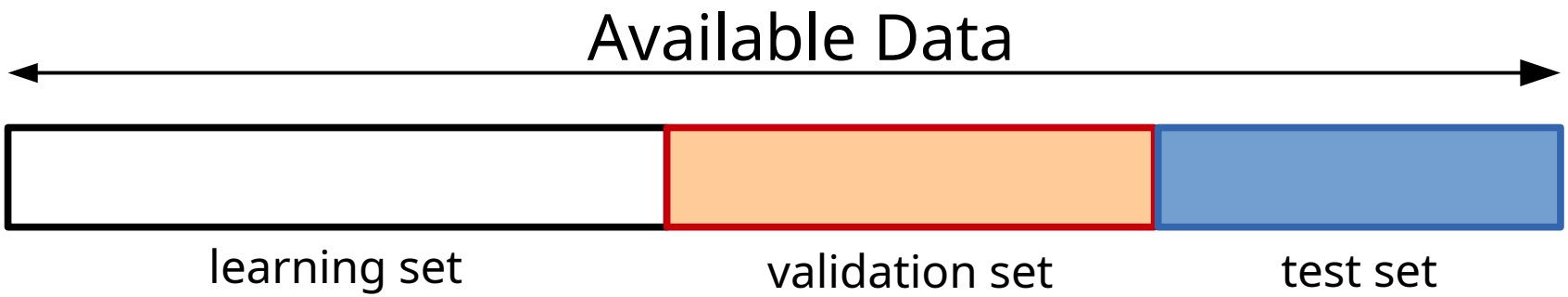
How to split validation/training set

- Need of a strategy to split between training and validation sets
- Training is used to tune the parameters of the model
- Validation is used to evaluate the model according to hyperparameters

Train/Validation/Test

Single split

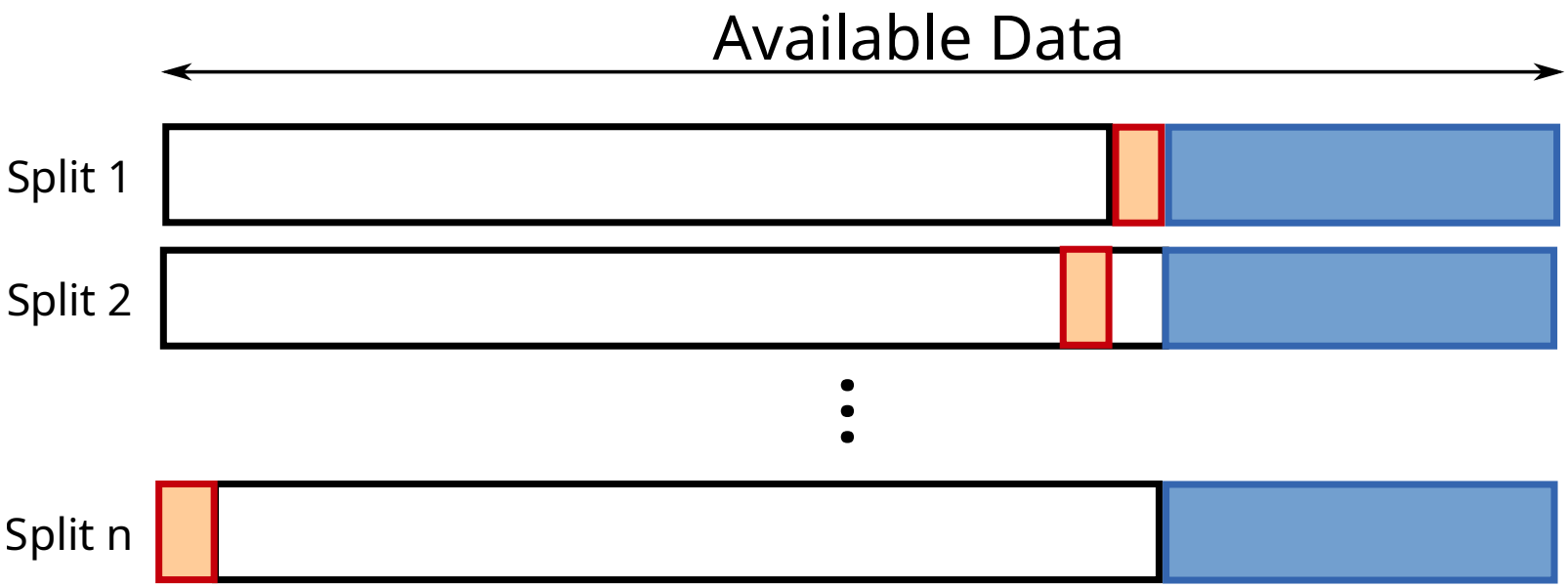
- ✚ An unique model to learn
- ▬ May be subject to split bias
- ▬ Only one evaluation of performance



Leave one out

N splits

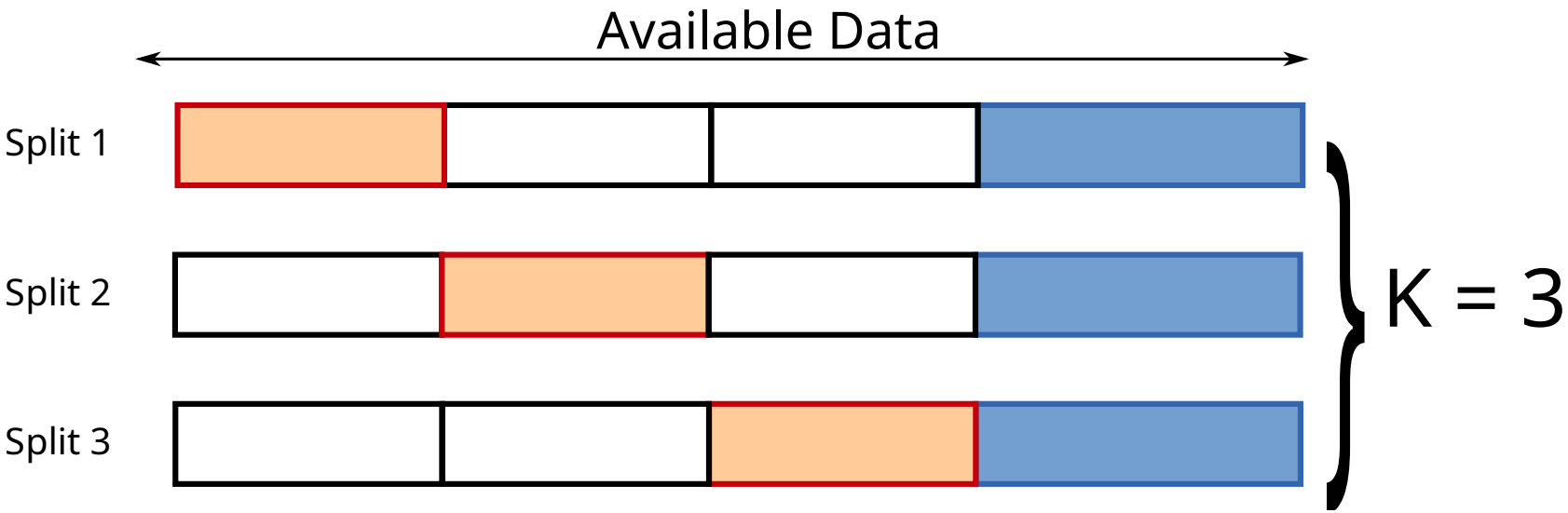
- N models to learn
- Validation error is evaluated on 1 data



KFold Cross validation

K splits

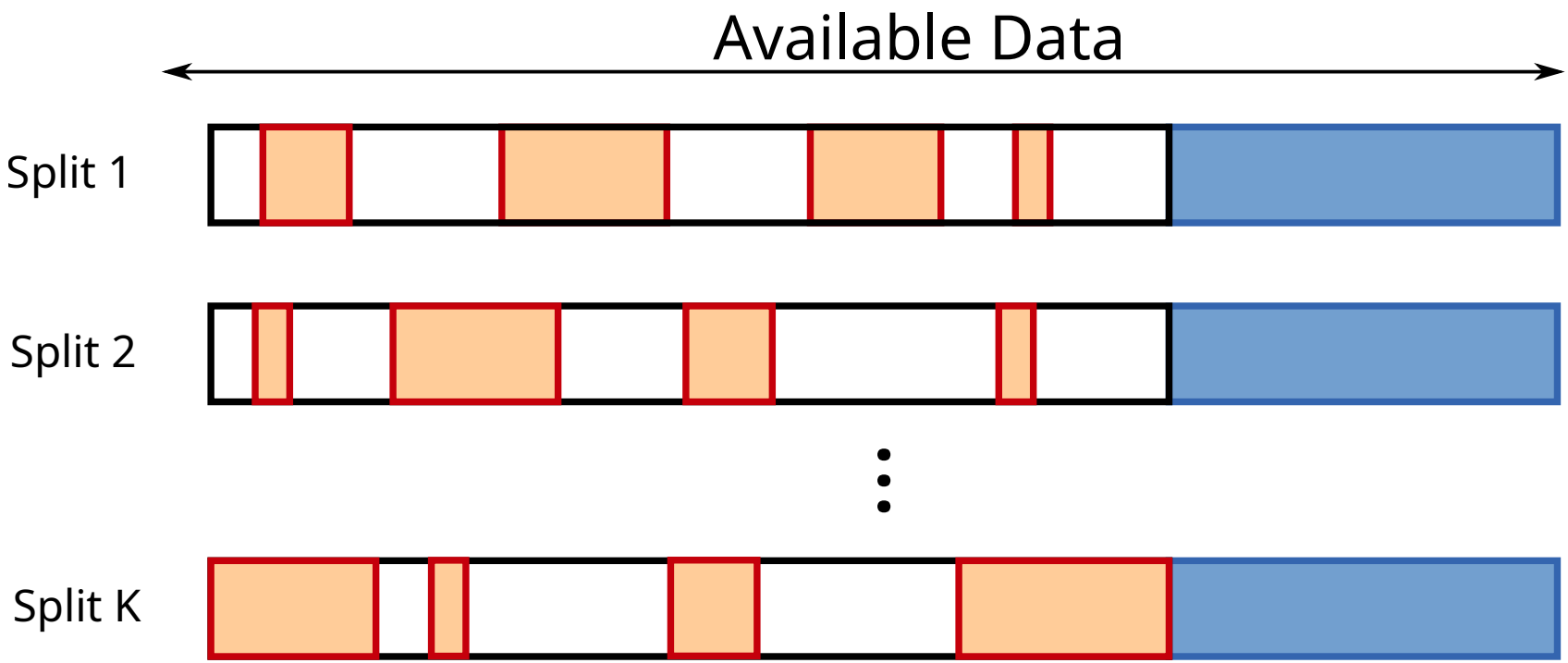
- K models to learn
- Validation error is evaluated on N/K data
- Some splits may be biased



Shuffle Split Cross validation

K splits

- Learn/Valid sets are randomly splited
- ✚ K models to learn
- ✚ Avoid bias
- ▬ Some data may not be evaluated



With scikit-learn

- sklearn.model_selection.train_test_split
- sklearn.model_selection.KFold
- sklearn.model_selection.ShuffleSplit
- sklearn.model_selection.GridSearchCV

Recommandation

Size of splits

- How many splits ?
- How many element by split ?
- Depends on the number of data
- Tradeoff between learning and generalization

Stratified splits

- Splitting may induce to imbalanced datasets
- Take care that the distribution of $\mathbf{y}$ is the same for all sets

Conclusion

- A good protocol avoid bias
- Test is never used during tuning of (hyper)parameters
- Perfect protocol doesn't exists