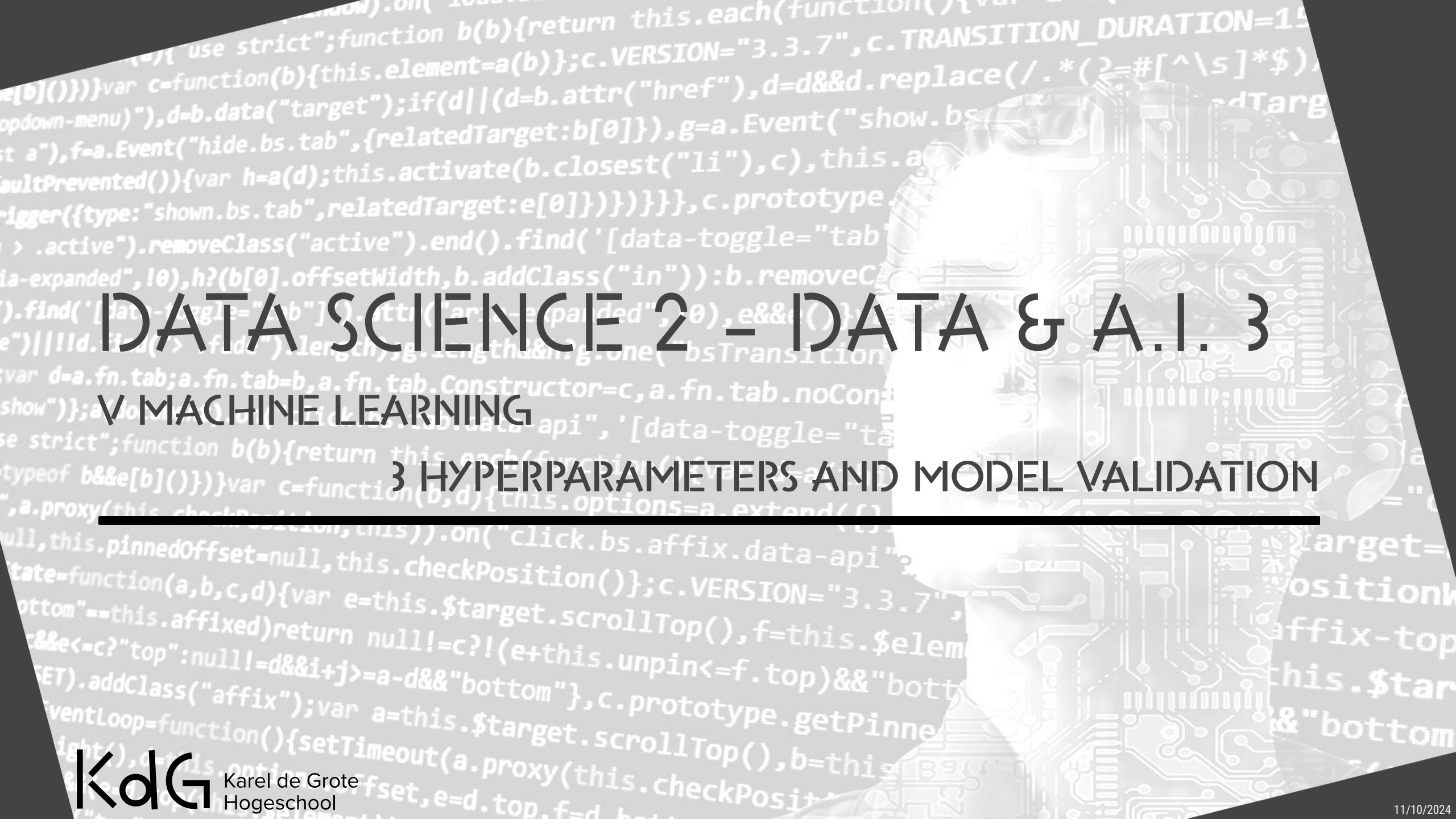
DATA SCIENCE 2 - DATA & A.I. 3

V MACHINE LEARNING

3 HYPERPARAMETERS AND MODEL VALIDATION



KdG Karel de Grote Hogeschool

DATA SCIENCE 2
DATA & A.I. 3

I

PYTHON BASICS

Python for data science

II

WORKING WITH ARRAYS

Numpy

III

DATA ENGINEERING

pandas

IV

DATA VISUALISATION

Matplotlib

V

MACHINE LEARNING

Automatically find patterns

V

MACHINE LEARNING

scikit-learn

MACHINE LEARNING

01 WHAT IS MACHINE LEARNING
Automatically find patterns

02 INTRODUCING SCIKIT-LEARN
Machine learning with Python

03 HYPERPARAMETERS AND CROSS VALIDATION
Holdout samples and cross-validation

04 REGRESSION
Best fitting line

05 DECISION TREES
Best separating lines

06 K-MEANS CLUSTERING
Object grouping

07 ASSOCIATION RULES
Frequent itemsets

08 ARTIFICIAL NEURAL NETWORK
Imitate the human brain

HYPERPARAMETERS AND MODEL VALIDATION

Holdout samples and cross-validation

WHAT KIND OF PROBLEMS CAN YOU SOLVE WITH MACHINE LEARNING

SUPERVISED

- Value estimation / regression: predict a continuous variable
  e.g. sales prediction; car use
- Classification: predict a categorical variable
  e.g. churn prediction; diagnosis

- Segmentation / clustering: split or group cases/observations
  e.g. customer segmentation; document topic search
- Co-occurence / association rule discovery: events happening together
  e.g. market basket analysis ; recommendation system

UNSUPERVISED

DATA ANALYTICS PIPELINE

DATA PREPARATION

- Load (labeled) source data
- Compile feature matrix
- Compile target array (for supervised methods)

MODEL SELECTION AND HYPERPARAMETER SELECTION (MODEL SPECIFIC)

- Decide on the method to use (linear regression, decision tree, K-means clustering, …)
- Decide on the hyperparameter to use (degree of polynomial, tree depth, number of clusters, …)

  Hyperparameters are parameters that the algorithm uses to derive a model

  Hyperparameters depend on the method (every methods has it's on kind of hyperparameters)

DERIVE MODEL FROM LABELED DATA (TRAIN MODEL/FIT MODEL)

- Apply the method/algorithm on the labeled data to derive the model

DISPLAY MODEL (MODEL SPECIFIC)

- Display the resulting model

  The resulting model depends on the method used (equation of regression line with intercept and slope, decision tree with nodes and split conditions, groups of observations with centroid, …)

DATA ANALYTICS PIPELINE

VALIDATE MODEL USING LABELED DATA (SUPERVISED LEARNING)

- Predict target feature for the labeled data
- Calculate the difference between predicted and real values for the labeled data / calculate confusion matrix (classification)
- Calculate validation metrics for the labeled data

  Regression : MAE, MAPE, RMSE, …

  Classification : accuracy, precision, recall, f-score, AUC, ROC

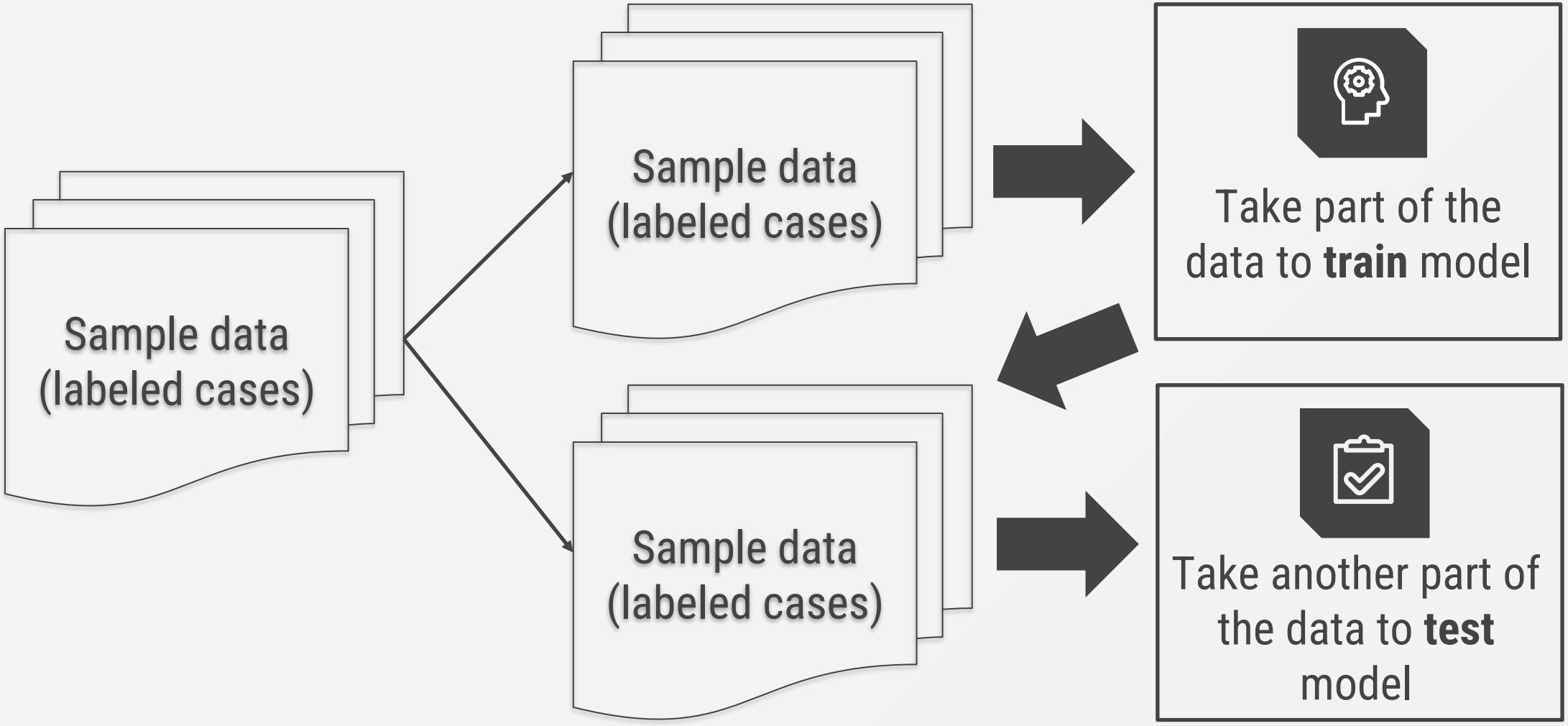
APPLY MODEL ON NEW DATA

- Apply the model on new data

  In case of supervised machine learning methods, this will predict the target feature for new data

  In case of unsupervised machine learning methods, this will restructure the feature data (clusters, association rules, new feature matrix with reduced dimensions)

TRAIN - TEST SPLIT



CROSS-VALIDATION

ALTERNATIVE FOR SPLITTING LABELED DATA INTO TRAINING SET AND TEST SET

The problem with splitting labeled example data into a training set and test set is that less labeled examples are available for training (as they are set aside for validation). In general, the more labeled examples for training, the better the model. But labeling a dataset can be very costly, hence to 'loose' valuable labelled examples to testing might not be preferable, especially for smaller labeled dataset.

The solution might be cross validation.

CROSS-VALIDATION

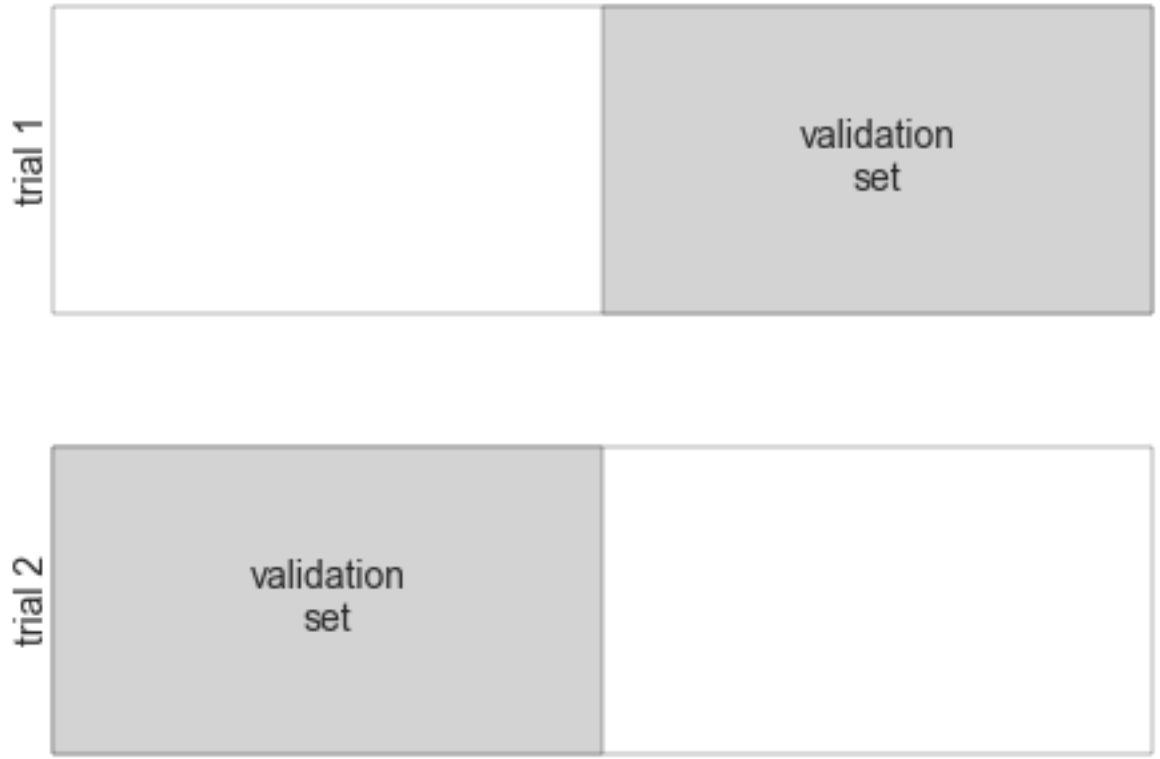
Cross-validation validation means that the labeled examples are split into multiple partitions and that multiple models are trained on different partitions. Every time, one partition of the labelled data is hold out for testing, and all other partitions are used for training (and the hold out partition changes every time). So, for N partitions, N different models are trained, with one of the N partitions as hold out sample for testing, and the N-1 other partitions for training. As a result, we have N models, all trained on slightly different data, covering all the examples in the labeled dataset (all examples are used N-1 times for training, and once for testing), resulting in N validation metrics. Those validation metrics can be combined, e.g. by calculating the mean over the N validation metric values.

The advantage of this method is that all data is used both for training and testing, and that the combined validation metrics over all parts of the data give a better measure for global performance.

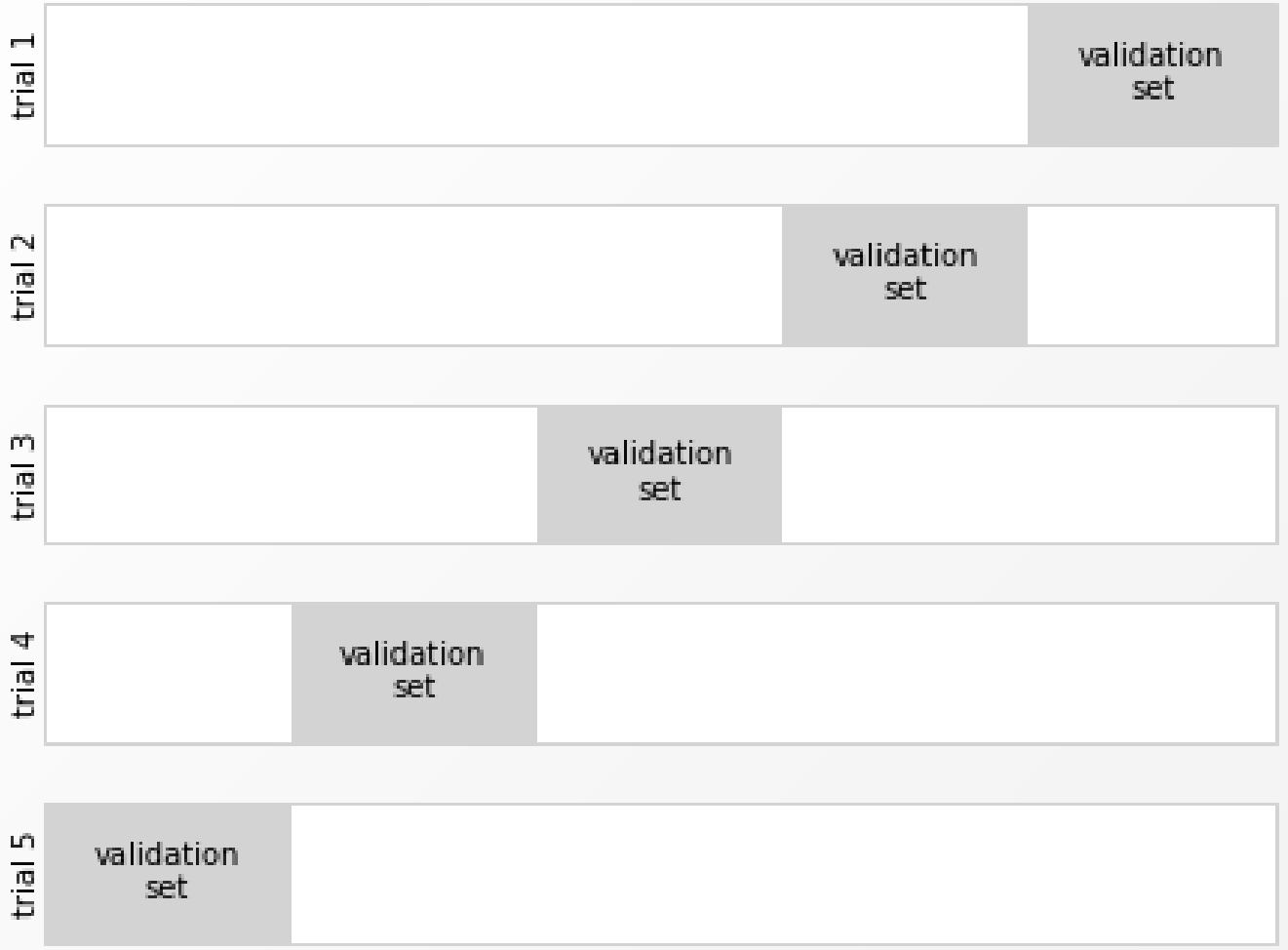
The disadvantage is that we end up with multiple models. That means there is no single model to use for predicting new data. One can select the best model, but that choice might be biassed, because the difference in performance is due to difference in training and validation data, so is based on coincidence. There is no guarantee that that model will also perform best on new data (we come back to that later).

Cross-validation is commonly used in model selection and hyperparameter tuning (see later).

CROSS-VALIDATION
(2-FOLD CROSS VALIDATION)



CROSS-VALIDATION
(K-FOLD CROSS VALIDATION)



CROSS VALIDATION WITH SCIKIT-LEARN

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, cv=5)
```

Parameters

model : e.g. LinearRegression, DecisionTreeClassifier, …

X : feature matrix

y : target vector

cv :

- int: number of cross validation folds (default = 5-fold)
- splitter: e.g. LeaveOneOut() (from sklearn.model_selection import LeaveOneOut)

Returns

scores : array with scores (for every cv fold), these are model dependent, e.g. accuracy for decision trees and R2 for linear regression

SIMPLE MODELS VERSUS COMPLEX MODELS
UNDERFITTING VERSUS OVERFITTING

MULTIPLE MODELING TECHNIQUES

When using supervised machine learning, multiple techniques can be used to derive a predictive model

- regression: linear regression, polynomial regression, artificial neural network, …;
- classification: decision tree, support vector machine, artificial neural network, ….

The challenge is to select the technique that is most appropriate for your problem or data. That does not necessarily mean the most advanced or complex method. The best choice has to do with underfitting and overfitting.

SIMPLE METHODS

A simple method is a method that uses a limited set of parameters to create a predictive model. The limited number of parameters reduces the flexibility to capture patterns in data.

The problem with simple models is UNDERFITTING, i.e. the model is not flexible enough to capture the patterns in the data.

A good example of a simple method is linear regression, which simply tries to fit a straight line through the data, and hence only uses an intercept parameter and one slope parameter for every dimension (every independent feature or predictor). That works very well if the data has more or less the shape of a cloud around a linear line. But that will not work if the data does not have a linear shape (e.g. U-shaped data). The only thing linear regression can do is to fit a straight line, nothing more, nothing less.

SIMPLE MODELS VERSUS COMPLEX MODELS
UNDERFITTING VERSUS OVERFITTING

COMLEX METHODS

A complex method is a method that uses an extended set of parameters to create a predictive model. The extended number of parameters increases the flexibility to capture patterns in data.

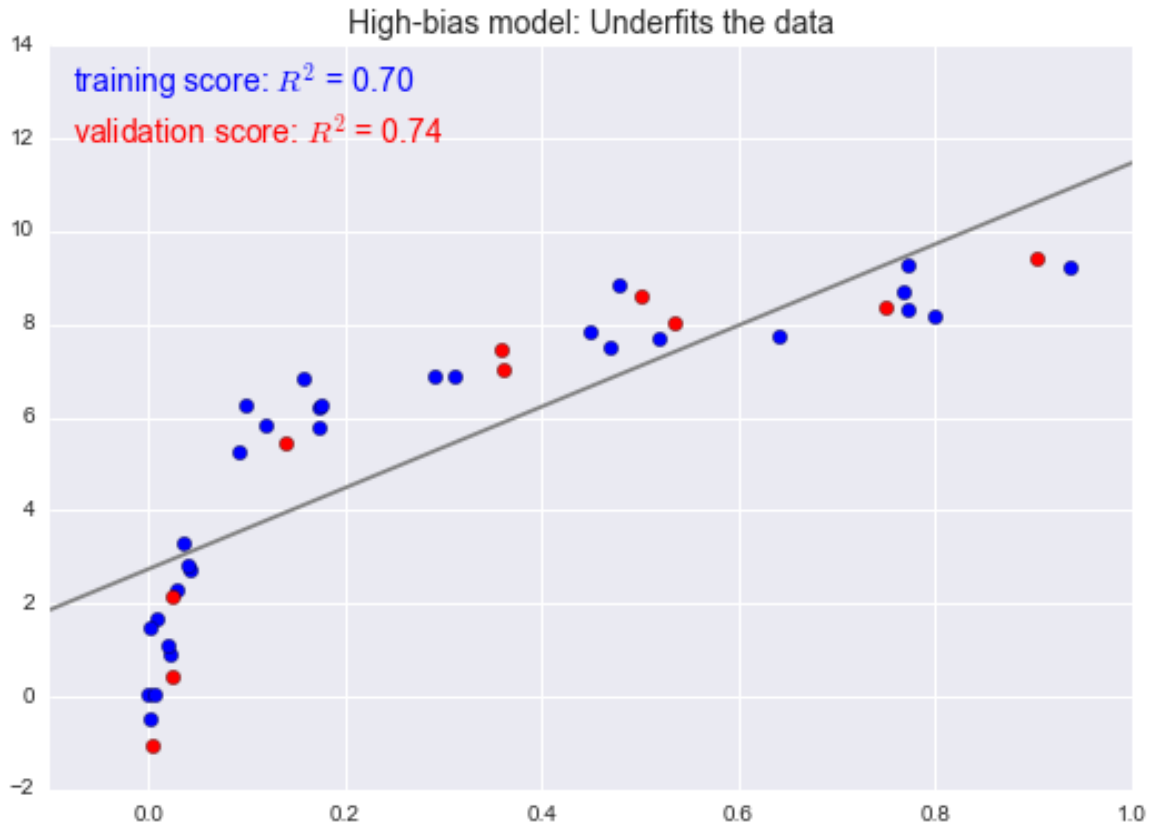
The problem with complex models is OVERFITTING, i.e. the model is too flexible and does not grasp the general pattern but just grasps the characteristics of individual cases.

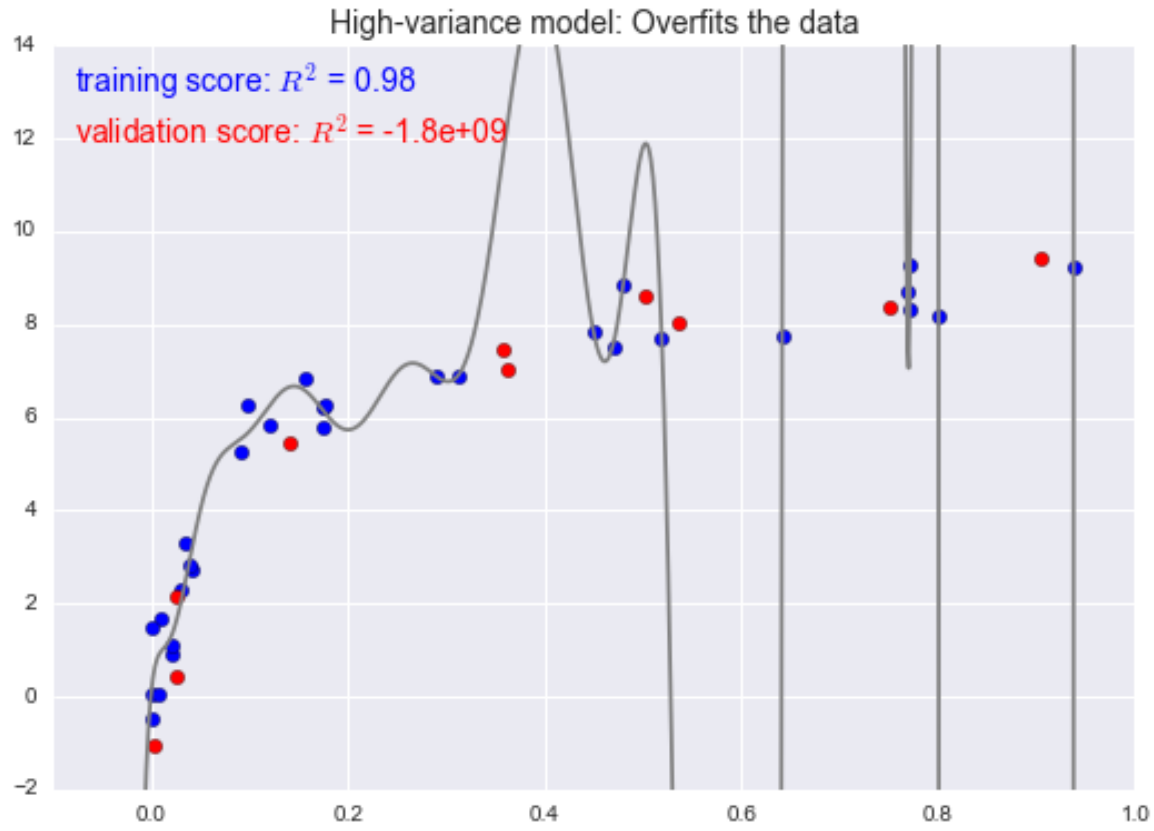
A good example of a complex method is high order polynomial regression, which tries to fit a high order polynomial through the data, and hence requires much more parameters besides intercept and slope ($c_1 x^1$, $c_2 x^2$, $c_3 x^3$, … $c_n x^n$). That works very well if the data has more or less the shape of the polynomial, but can go wrong if the data has a different shape, because the polynomial will start predicting individual cases instead of the general pattern.

HOW TO DETECT UNDERFITTING AND OVERFITTING

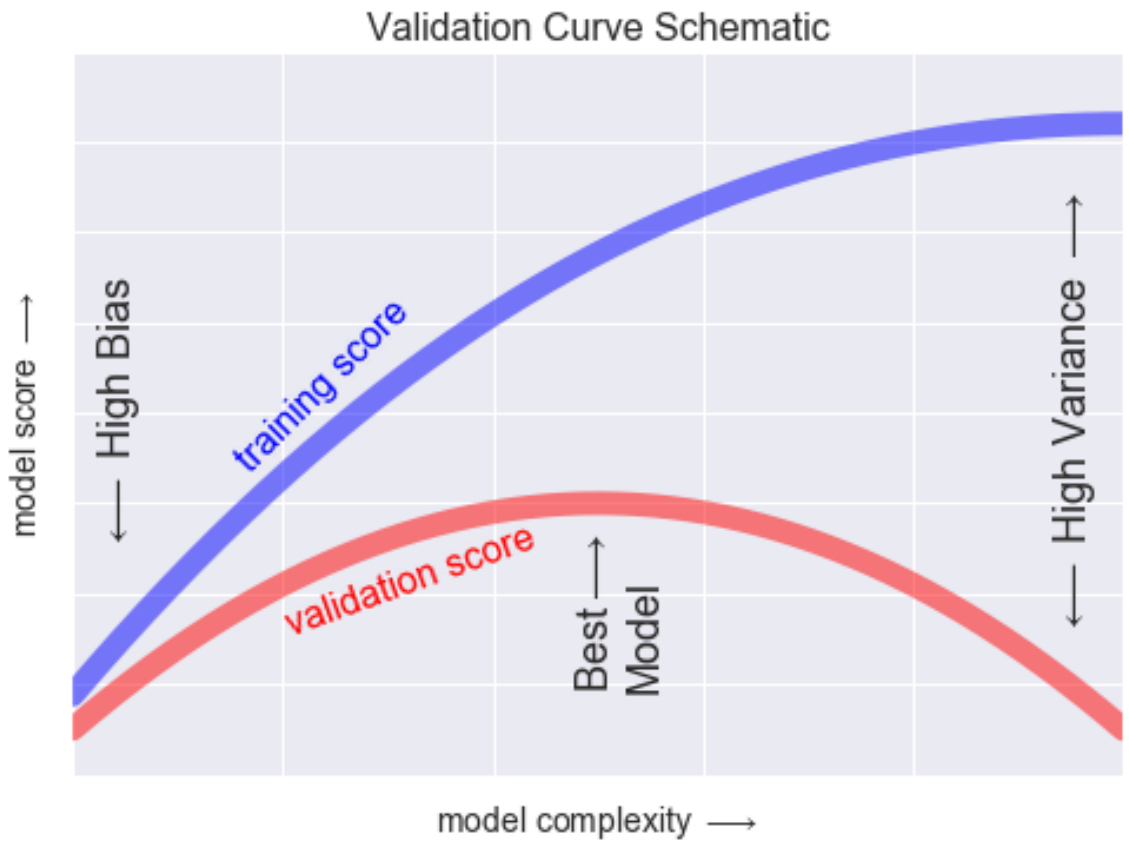
- Underfitting : the model will score bad on both training set and test set (the model is not able to capture any relevant pattern)
- Overfitting : the model will score good on the training set but bad on the test set (the model got tailored to the examples in the training set, but is not able to generalize, not able to predict new data well)

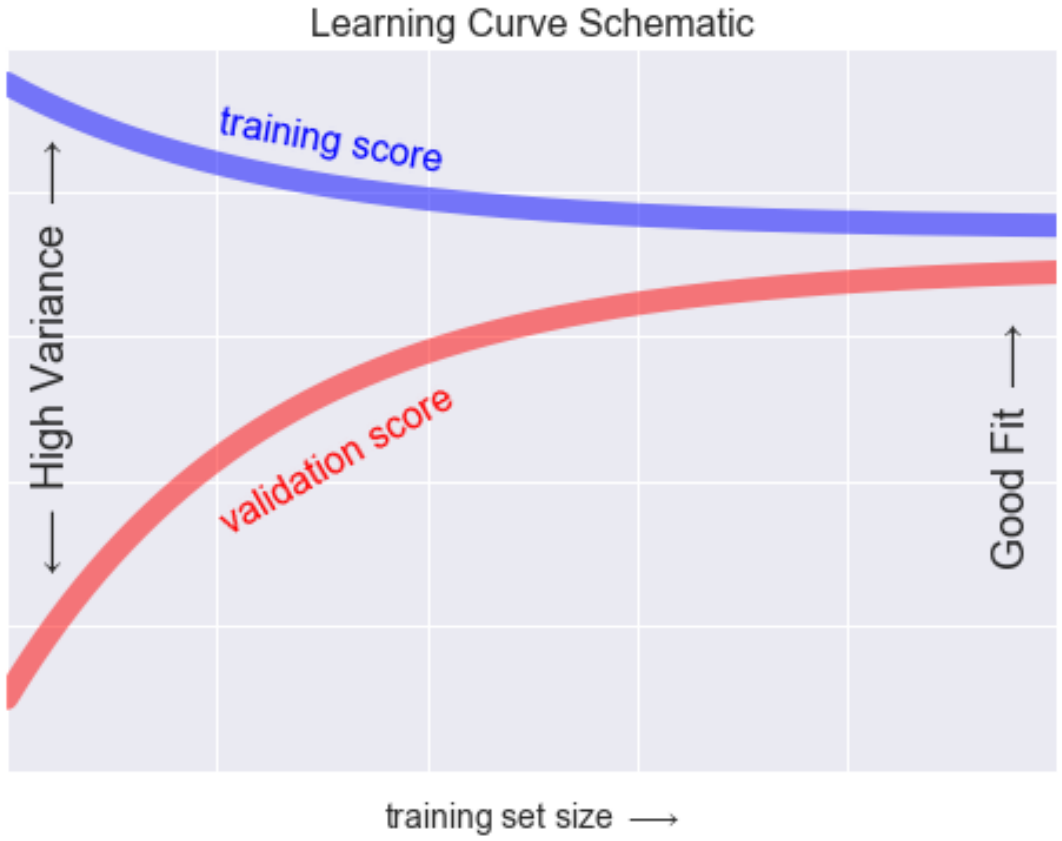
SIMPLE MODELS VERSUS COMPLEX MODELS
UNDERFITTING VERSUS OVERFITTING





SIMPLE MODELS VERSUS COMPLEX MODELS
UNDERFITTING VERSUS OVERFITTING





MODEL SELECTION AND HYPERPARAMETER TUNING
TRAIN – VALIDATE - TEST SPLIT

MODEL SELECTION AND HYPERPARAMETER TUNING

The challenge is to select the modeling technique that is most appropriate for your problem or data. This is called model selection (e.g. decision tree, support vector machine, artificial neural network).

But every method also has hyperparameters that also influence the model and hence can have a big impact on the model quality:

- Decision trees : tree depth, minimum samples split, maximum leaf nodes, . .
- Support vector machine : kernel function, C, gamma, …
- Artificial neural network : number of layers, number of neurons per layer, activation function, …

So it is not only about choosing the right modeling technique, but also the right hyperparameters. This is called hyperparameter tuning.

Mostly, it is only possible to find the best modeling technique and set of hyperparameters by trial and error. But doing this right requires a train-validate-test setup (see below), meaning that more labeled examples are lost for training.

TRAIN – VALIDATE - TEST SPLIT

Instead of a train-test split, model selection and hyperparameter tuning requires 3 subsets of labeled examples :

- Train dataset : dataset used to train multiple models with multiple hyperparameter sets. The result is a set of models with a set of hyperparameters, all trained on the same training set.
- Validation dataset : dataset to validate all models and hyperparameter sets generated in the previous step. All models and hyperparameter sets are validated using the same validation set. Select the best scoring model and hyperparameter set.
- Test dataset : dataset for the ultimate, independent test of the one selected best model and hyperparameter set from the previous step to check whether the selected model is general enough to predict new data.

MODEL SELECTION AND HYPERPARAMETER TUNING WITH CROSS-VALIDATION

USING CROSS-VALIDATION FOR MODEL SELECTION AND HYPERPARAMETER TUNING

Cross-validation comes in handy for model selection and hyperparameter tuning because it allows multiple training and testing runs on multiple labeled subsets making use of all information in the labeled dataset and excluding the coincidence factor of data selection.

Using cross-validation for model selection and hyperparameter tuning only requires the labelled example dataset to be split into two labeled subsets instead of three labeled subsets (train-validate-test subsets)

PRACTICAL SETUP : TRAIN/VALIDATE – TEST SPLIT

- Split the labeled data into two subsets: a train/validate subset and a holdout test subset
- Use the train/validate subset to train multiple modeling techniques with multiple hyperparameter sets using cross-validation

  Every modeling technique/hyperparameter set is trained and validated multiple times, each time validated by one partition and trained using all other partitions. For every modeling technique/hyperparameter set, the average of all validation scores over all runs is calculated
- Select the modeling technique and hyperparameter set with the highest average score (average of all scores over all runs of the cross-validation) in the previous step
- Train a new model using the best modeling technique and hyperparameter set as selected in the previous step. Use all data in the train/validate labeled subset for the training

  So the train/validate labeled subset is used twice, once to find the best modeling technique/hyperparameter set using cross-validation, once to train a new model based on the best modeling technique/hyperparameter set
- Test the model trained in the previous step using the holdout test set for an independent test (test generalization on new data)

HYPERPARAMETER TUNING WITH SCIKIT-LEARN

One parameter

```
from sklearn.model_selection import validation_curve
train_scores, val_scores = validation_curve(model, X=X_train, y=y_train, cv=5,
                                            param_name=name, param_range=range)
```

Parameters

model : e.g. LinearRegression, DecisionTreeClassifier, …

X : feature matrix

y : target vector

cv : int: number of cross validation folds (default = 5-fold) or splitter (e.g. LeaveOneOut())

param_name :

- name of model parameter, e.g. fit_intercept for LinearRegression or max_depth for DecisionTreeClassifier
- model.get_params() can be used to list the parameters

param_range :

- values of the parameters to be validated, e.g. [True, False] or np.arrange(1,6)

Returns

train_scores, val_scores : 2D-arrays with scores (e.g. accuracy for classification or R2 for regression), shape is (nbr. in param range, nbr. of folds)

HYPERPARAMETER TUNING WITH GRID SEARCH

Multiple parameters

GRID SEARCH

Grid search is way to test a method for multiple hyperparameter combinations. First a set of hyperparameters to be checked is compiled. Next, for every selected hyperparameter, a set of parameter values to be checked is compiled. The combination of all parameter values of interest forms a grid (e.g. a set of 2 hyperparameters, one with 4 parameter values of interest, one with 3 parameter values of interest, form a grid of 4 by 3, resulting in 12 hyperparameter value combinations). Next every cell of the grid is searched, i.e. a model with the combination of hyperparameter values present in the grid cell is trained and validated. In the end, the best scoring model can be used to select the best combination of hyperparameter values.

FULL GRID SEARCH AND RANDOM GRID SEARCH

The total number of hyperparameter value combination can become very high (e.g. 5 hyperparameters with an average of 4 hyperparameter values results in 1024 combinations; for neural networks, far more combinations are possible as the number of layers and the amount of neurons per layer can be high).

A full grid search with train and test a model for every cell in the grid, i.e. every combination of hyperparameter values.

For large datasets, and complex methods, training can become very long, making it practically impossible to do a full grid search. In that case, a random grid search can be used, where grid cells are randomly chosen and only those combination are trained and tested.

More advanced optimisation methods exist apart from pure random selection.

HYPERPARAMETER TUNING WITH GRID SEARCH

Multiple parameter

```
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(model, cv=5, param_grid=param)
```

Parameters

model : e.g. LinearRegression, DecisionTreeClassifier, …

cv : int: number of cross validation folds (default = 5-fold) or splitter (e.g. LeaveOneOut())

param_grid :

- dictionary with parameters names and parameter values for the full grid search, e.g. {'criterion': ['gini', 'entropy'], 'max_depth': np.arange(1,6)}

Methods

```
grid.fit(X_train, y_train)
```

Attributes

```
grid.best_params_
grid.best_score_
model = grid.best_estimator_
```

CROSS-VALIDATION AND GRID SEARCH PIPELINE WITH SCIKIT-LEARN (CLASSIFICATION)

```
# MODEL SELECTION AND HYPERPARAMETER TUNING (REPEAT THIS STEP FOR MULTIPLE MODELING TECHNIQUES)
from sklearn.model_selection import GridSearchCV

# Define parameter grid (model specific)
grid_param = {'criterion' : ['gini', 'entropy', 'log_loss'],
              'max_depth' : list(range(2,10)),
              'min_samples_split' : list(range(2,5))}

# Setup grid search with N-fold cross validation (e.g. 5-fold)
grid_search = GridSearchCV(model, grid_param, cv=5)

# Execute full grid search
grid_search.fit(X_train, y_train)

# Display best hyperparameter values and matching validation score
print(f'Best parameters : {grid_search.best_params_}')
print(f'Best score      : {grid_search.best_score_:.3f}')
```