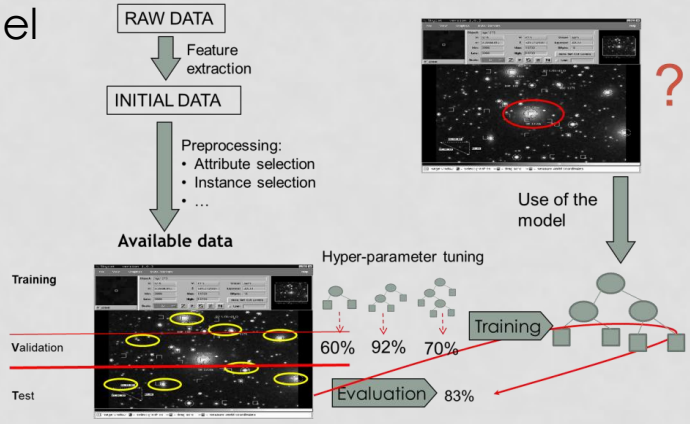
SYLLABUS

1. Introduction to Machine Learning: tasks, algorithms & models
2. Basic methods for training classification and regression models:
3. Methodology (the Machine Learning workflow): model evaluation, hyper-parameter tuning, preprocessing, …
4. Methods for preprocessing
5. Advanced training methods based on ensembles of models
6. Large Scale Machine Learning. Big Data
7. Advanced topics
8. Software tools



HYPERPARAMETER TUNING / HYPER-PARAMETER OPTIMIZATION / **HPO**

HYPER-PARAMETER OPTIMIZATION (**HPO**) INDEX

- **Motivation for HPO (Hyper-Parameter Optimization)**
- HPO and estimation of future performance wih train/validation/test
- HPO and estimation of future performance wih inner = crossvalidation and outer = test (and inner/outer = crossvalidation)
- Standard methods for HPO:
  - Grid-search
  - Random-search
- Improved methods for HPO:
  - Sequential Model Based Optimization / Bayesian Optimization
  - Fixed vs. Non-fixed hyper-parameters search space: Optuna (define-by-run)
  - Successive Halving
- The CASH problem (Combined Algorithm Selection and Hyper-parameter tuning)

HYPER-PARAMETERS

- Every machine learning algorithm / method has one or several hyper-parameters that control the way they generate (train/fit) models
- For instance, KNN has *K* (the number of neighbors)
- For instance, decision trees have:
  - *max_depth*: the maximum depth of the tree
  - *min_samples_split*: the minimum number of instances to split a node (the default value is 2: if a node contains fewer than 2 instances, the node is not split)
- A ML method can train a model, but:
  - Either some default values for the hyper-parameters are used
  - Or they are set in advance by the user.
  - In both cases, they might not be the most appropriate for the problem at hand.

HYPER-PARAMETERS

- Finding a good value of a hyper-parameter may result in improved performance of the model
- Most hyperparameters control, directly or indirectly, the complexity of models trained by the method, which is related to overfitting
- Overfitting means obtaining models that memorize the training data rather than models that generalize well

OVERFITTING

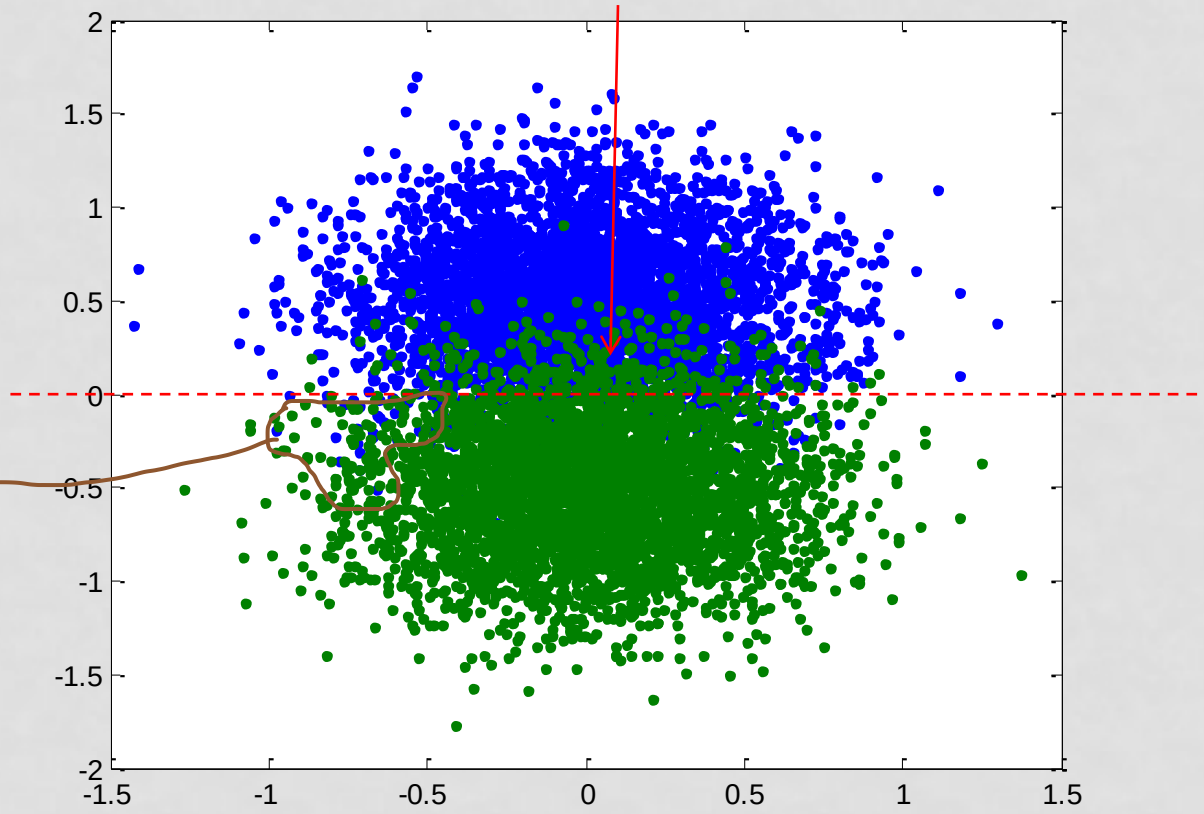
- Overfitting typically happens when:
  - Data contains noise (e.g. class overlap in classification)
  - There is little data
  - Model is so complex that it can memorize noise (but if it is not complex enough, it will underfit the data)
- Model complexity can be controlled by means of the hyper-parameters (in some contexts, this is called regularization).

EXAMPLE OF A CLASSIFICATION MODEL NOT GENERALIZING WELL

Let's suppose we want to learn a model that is able to separate class "blue" from class "green". Below we can see instance space with thousands of instances. Notice that both classes overlap in the middle

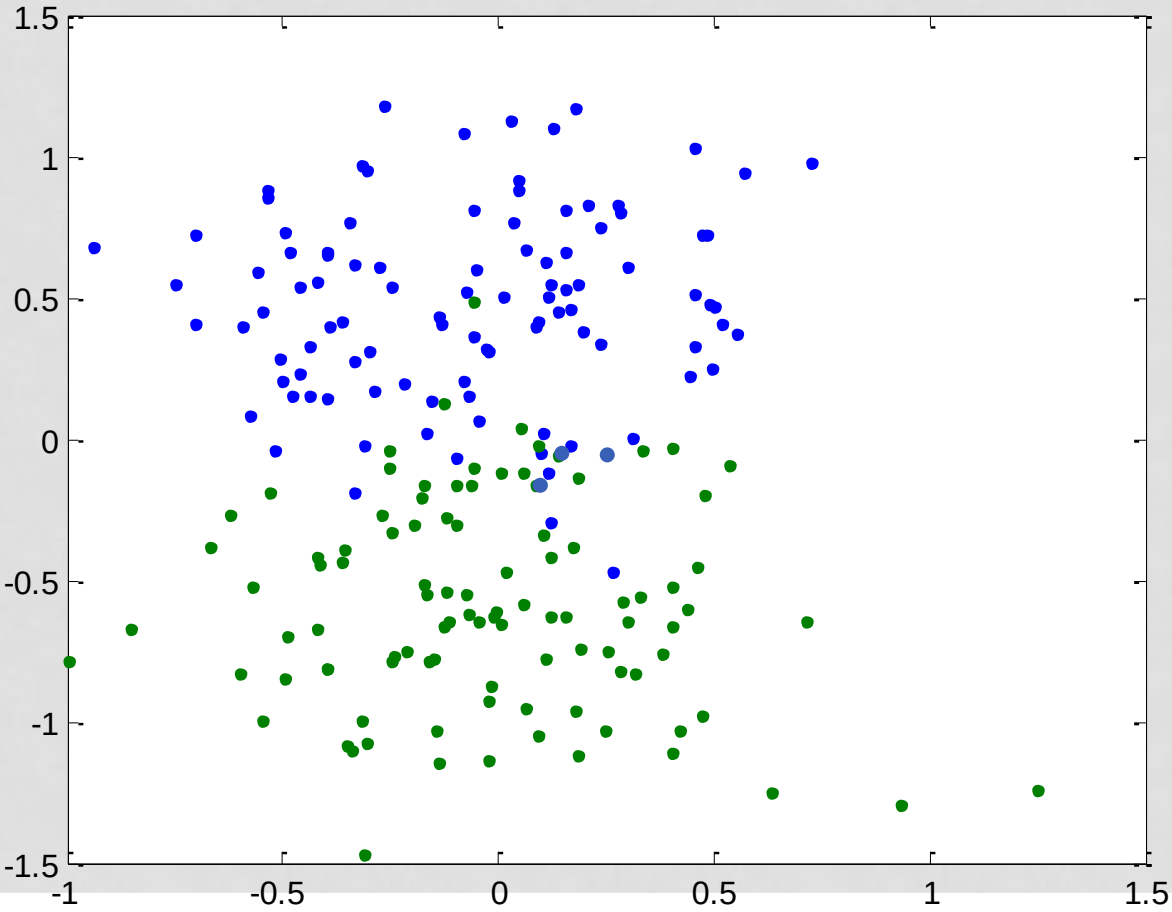
Both classes are generated by two symmetric gaussians distributions (containing noise)

noise



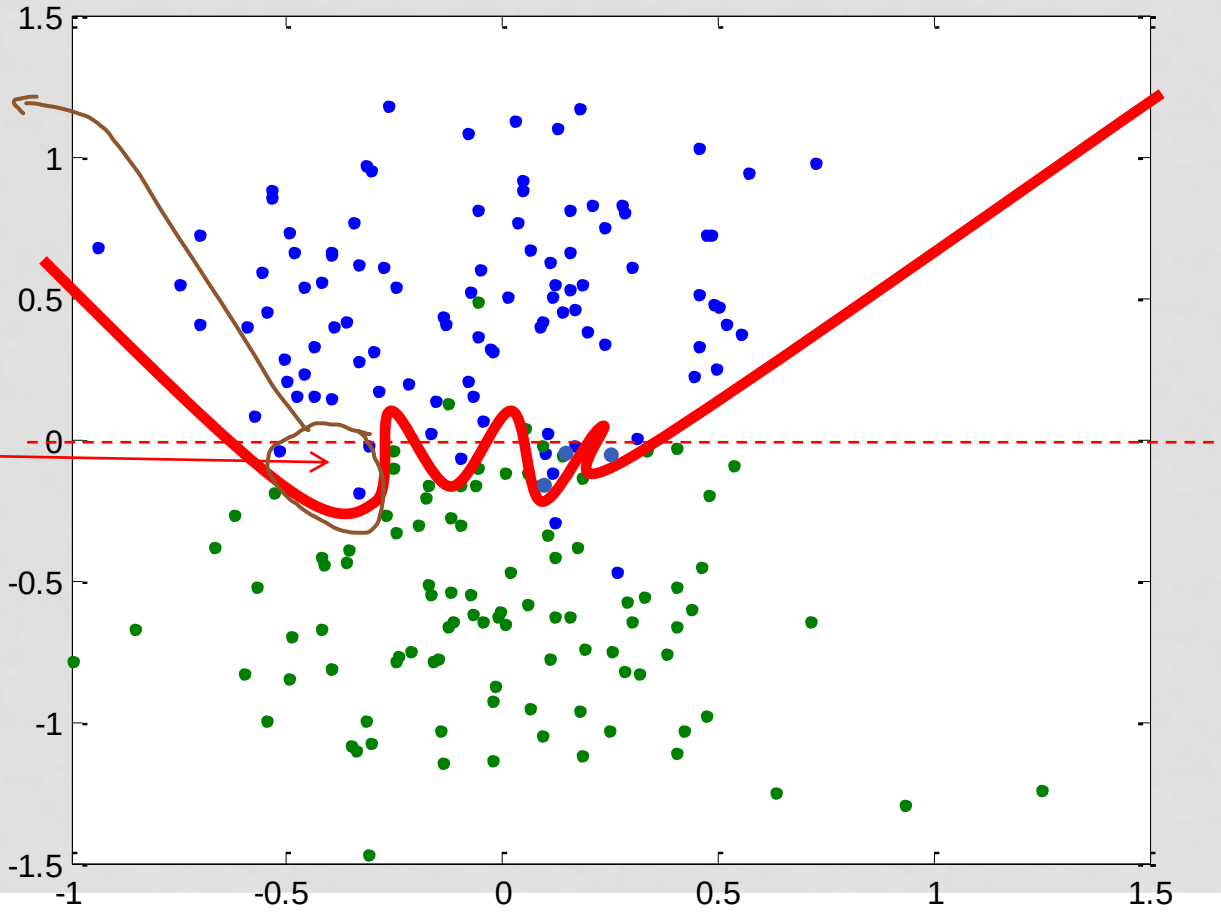
EXAMPLE OF A CLASSIFICATION MODEL NOT GENERALIZING WELL

- But usually, we have few data for training:



EXAMPLE OF A CLASSIFICATION MODEL NOT GENERALIZING WELL

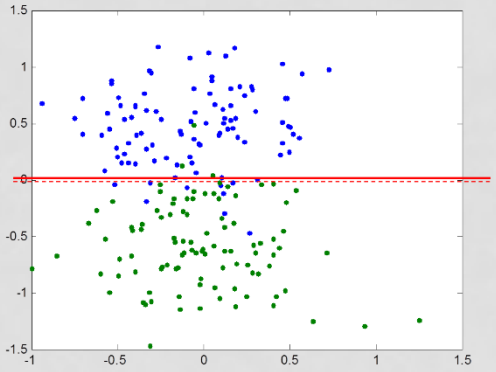
- But if we have few data for training, the following model might be learned
- The model is obviously not generalizing well. It is memorizing the noise, or **overfitting** the data

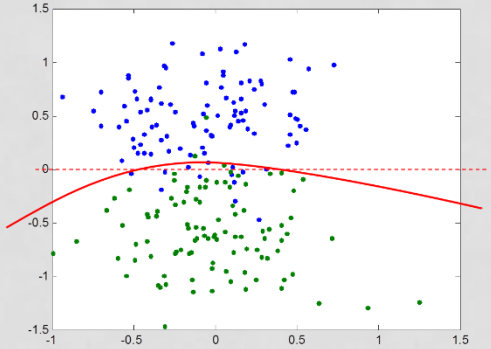


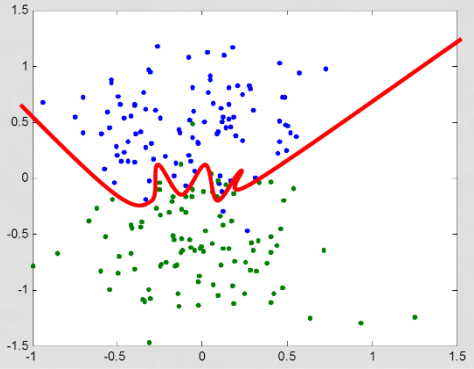
HYPER-PARAMETERS

- Hyperparameters control, directly or indirectly, the complexity of models trained by the method
- In general, the more complex a model is, the more likely it will overfit the data (but if it is not complex enough, it will underfit the data)

This is a limitation of the complex from the model

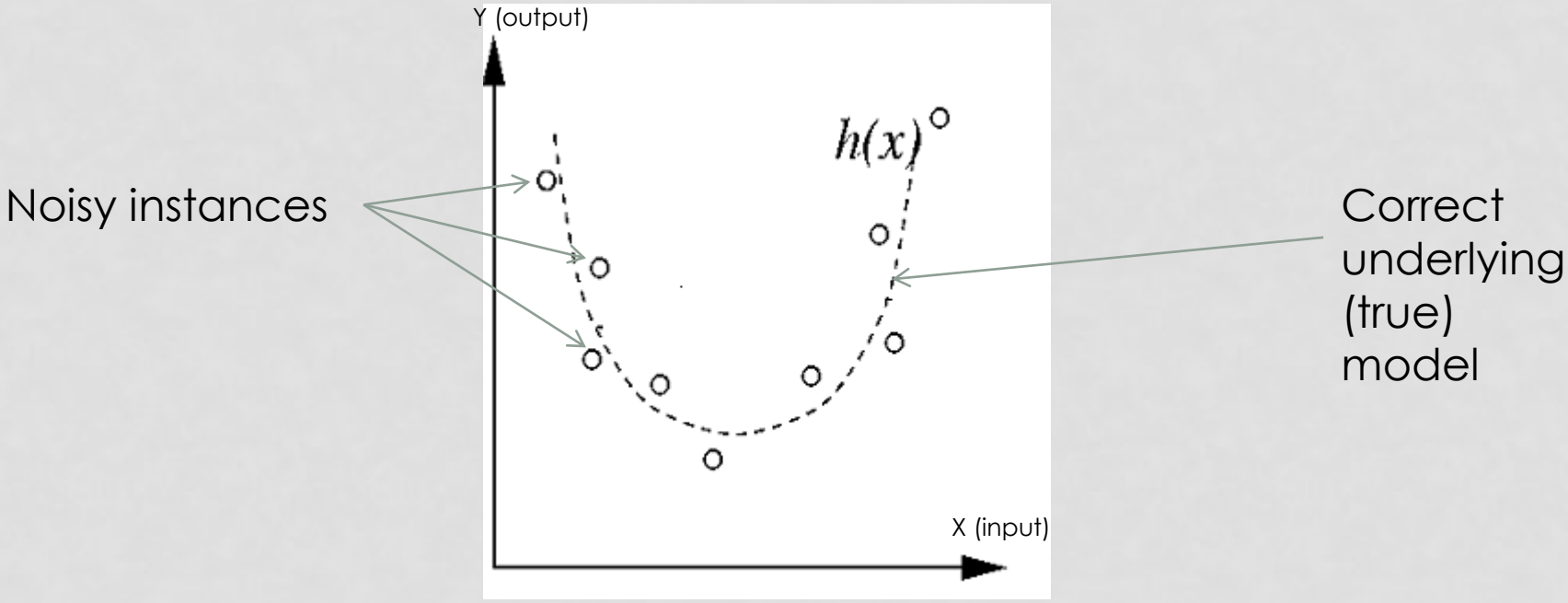






EXAMPLE OF A REGRESSION MODEL NOT GENERALIZING WELL

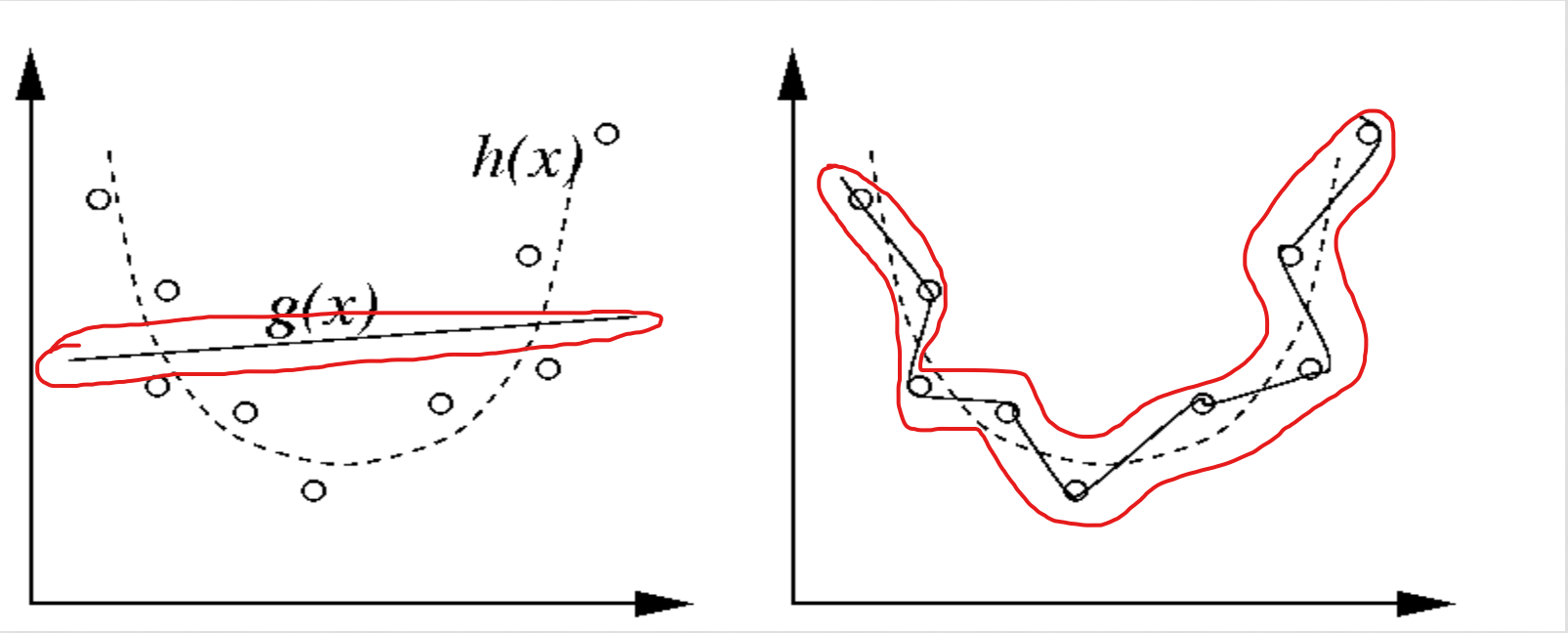
- Let's suppose that the underlying model is a parabola, but instances have some noise
  - For example, **y** might be "temperature", but the thermometer used to measure it is not very accurate



EXAMPLE OF A REGRESSION MODEL NOT GENERALIZING WELL

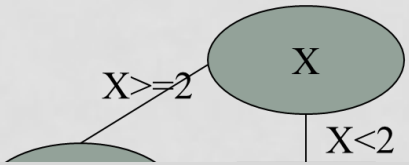
Underfitting

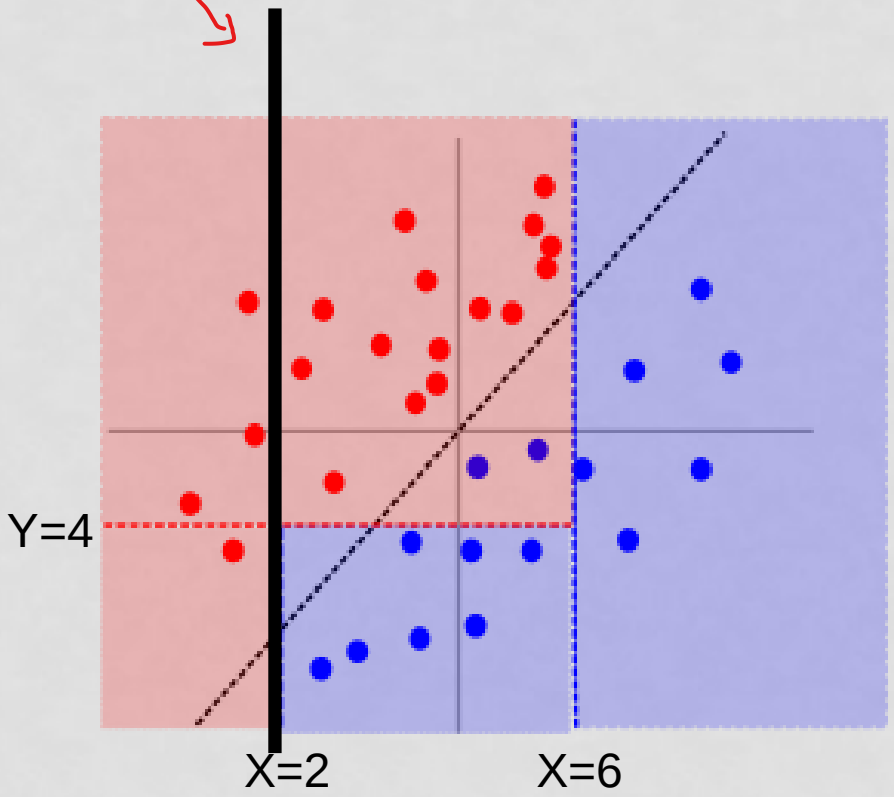
Overfitting



MAX-DEPTH HYPER-PARAMETER FOR DECISION TREES

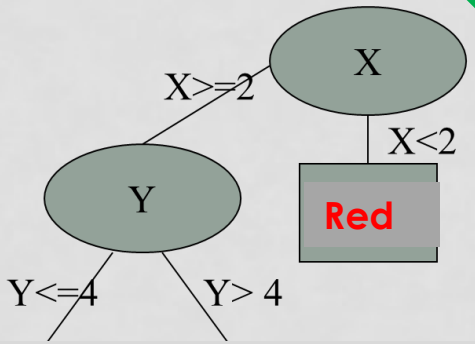
- With max_depth = 1, boundary is a line.

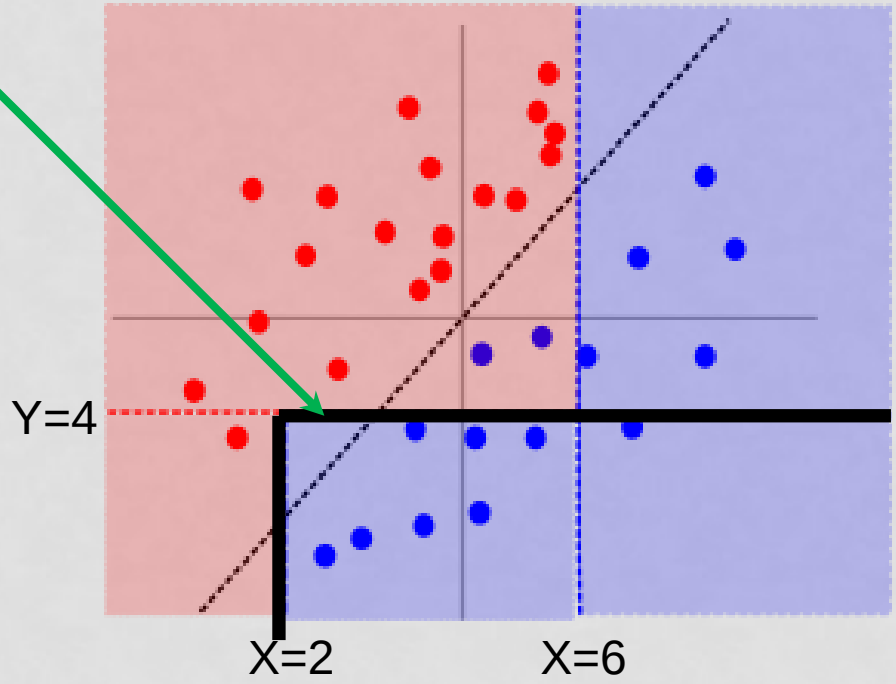




MAX-DEPTH HYPER-PARAMETER FOR DECISION TREES

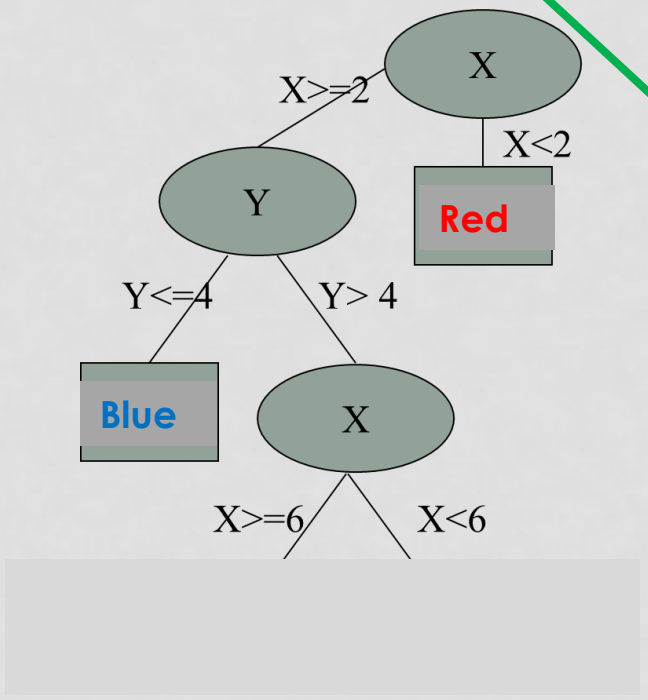
- With max_depth = 2, boundary is non-linear

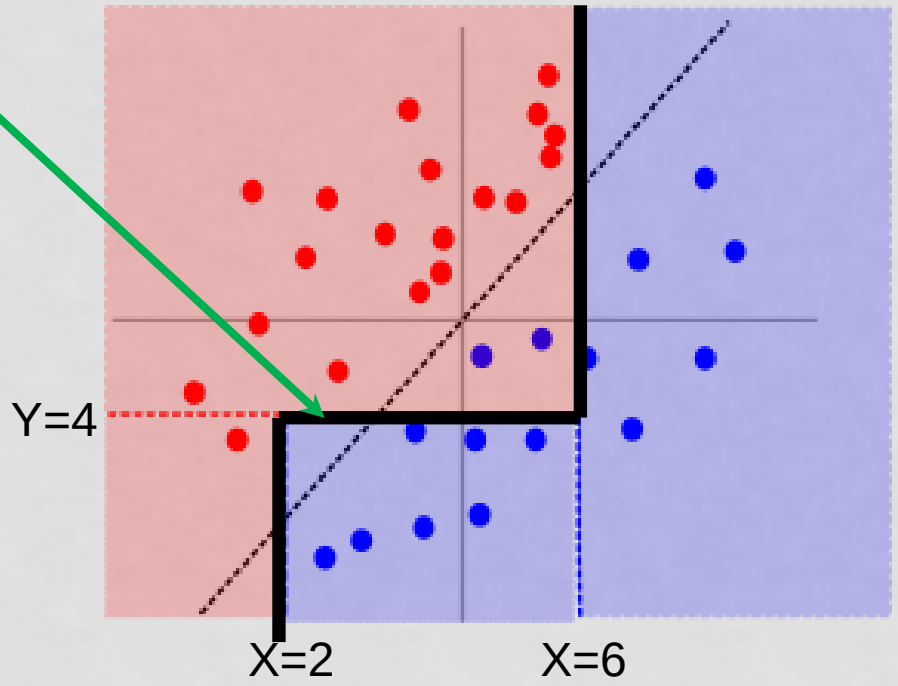




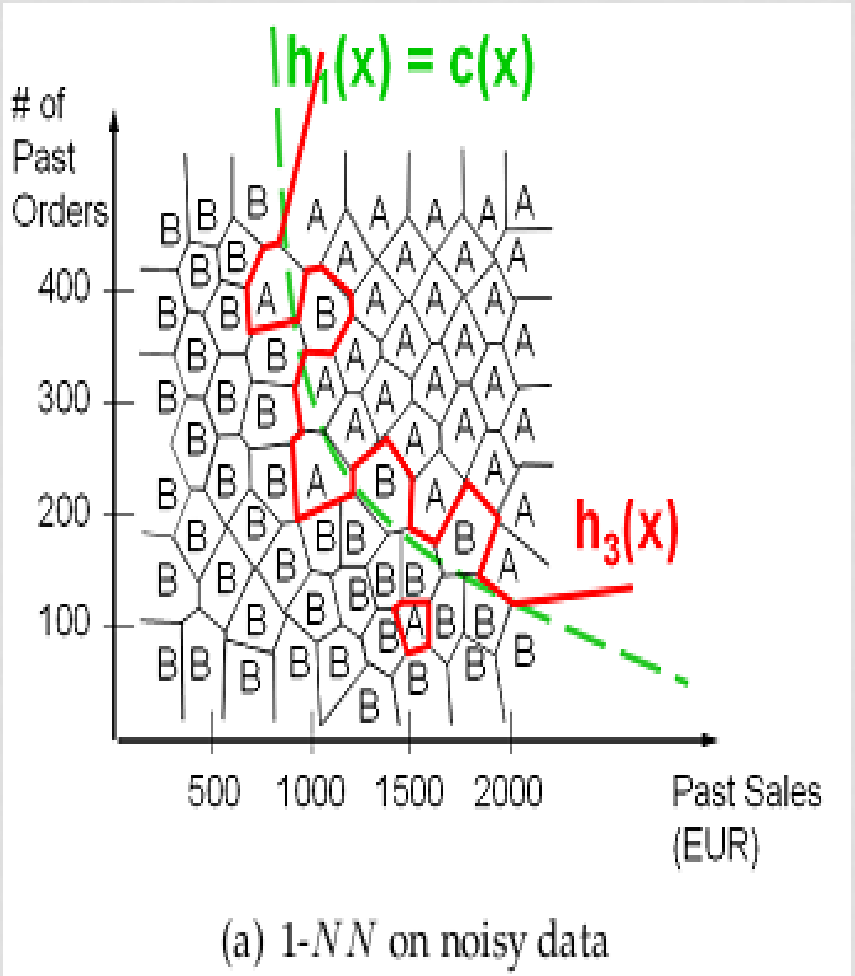
MAX-DEPTH HYPER-PARAMETER FOR DECISION TREES

- With max_depth = 3, boundary is non-linear and more complex than with max_depth = 2
- And so on

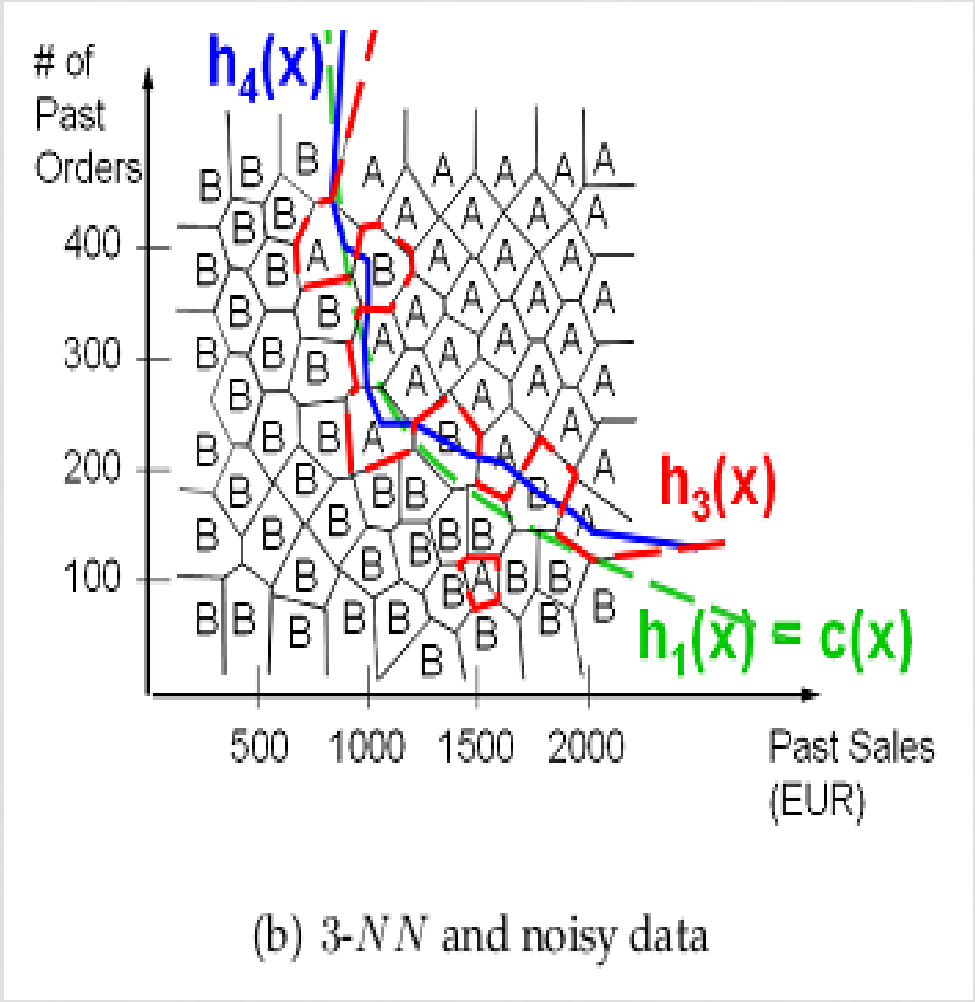




K IN KNN



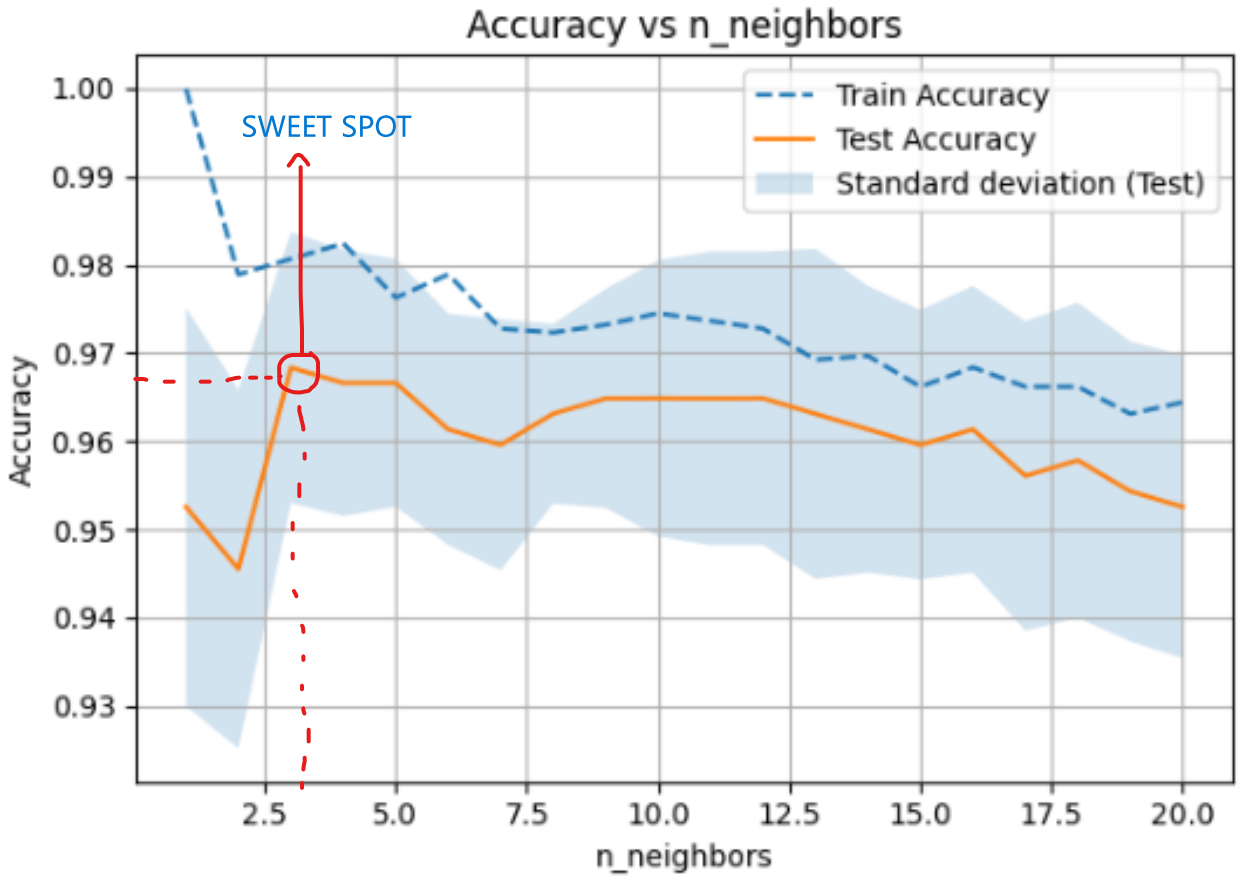
(a) 1-NN on noisy data



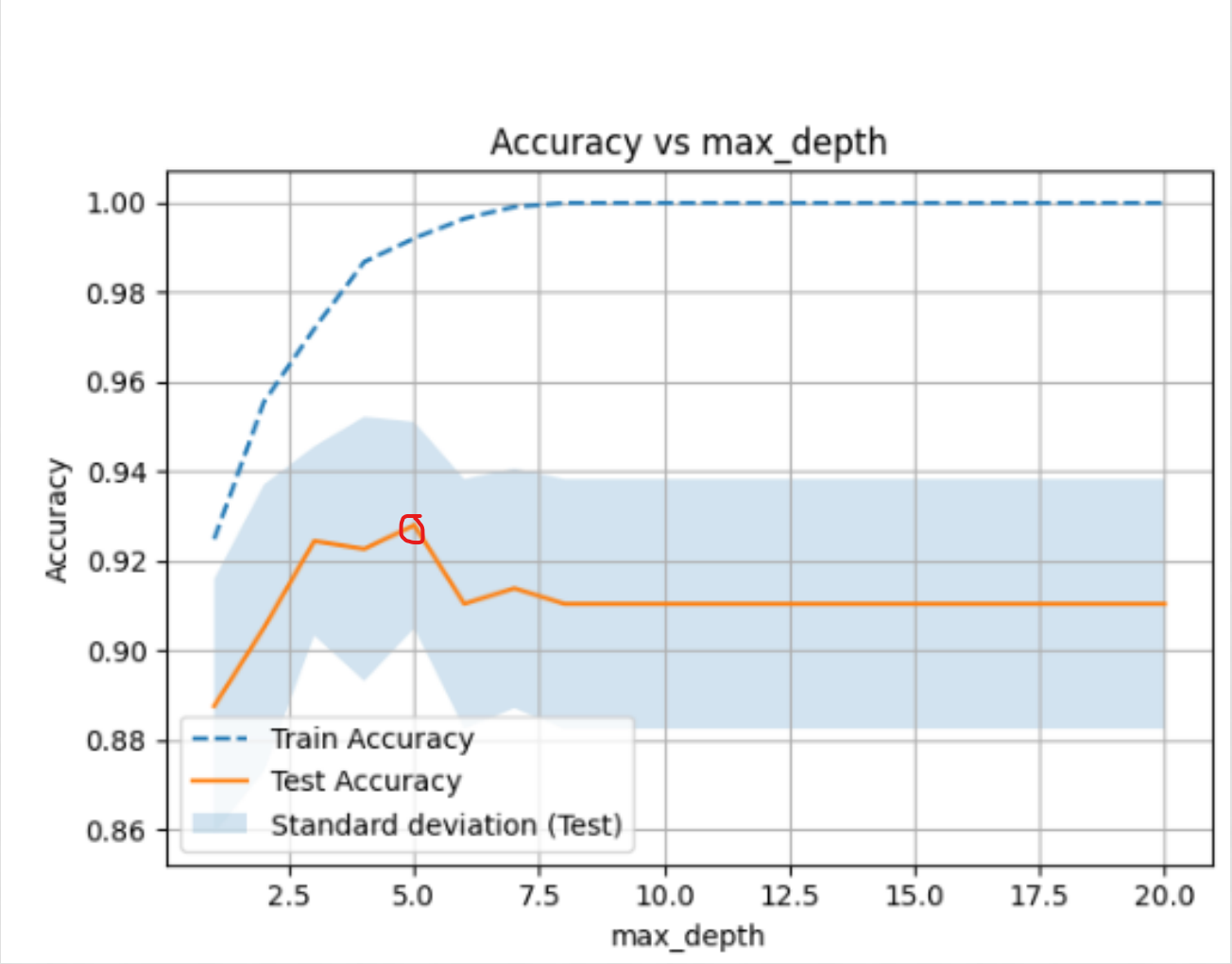
(b) 3-NN and noisy data

KNN: NEIGHBORS VS. TEST ACCURACY

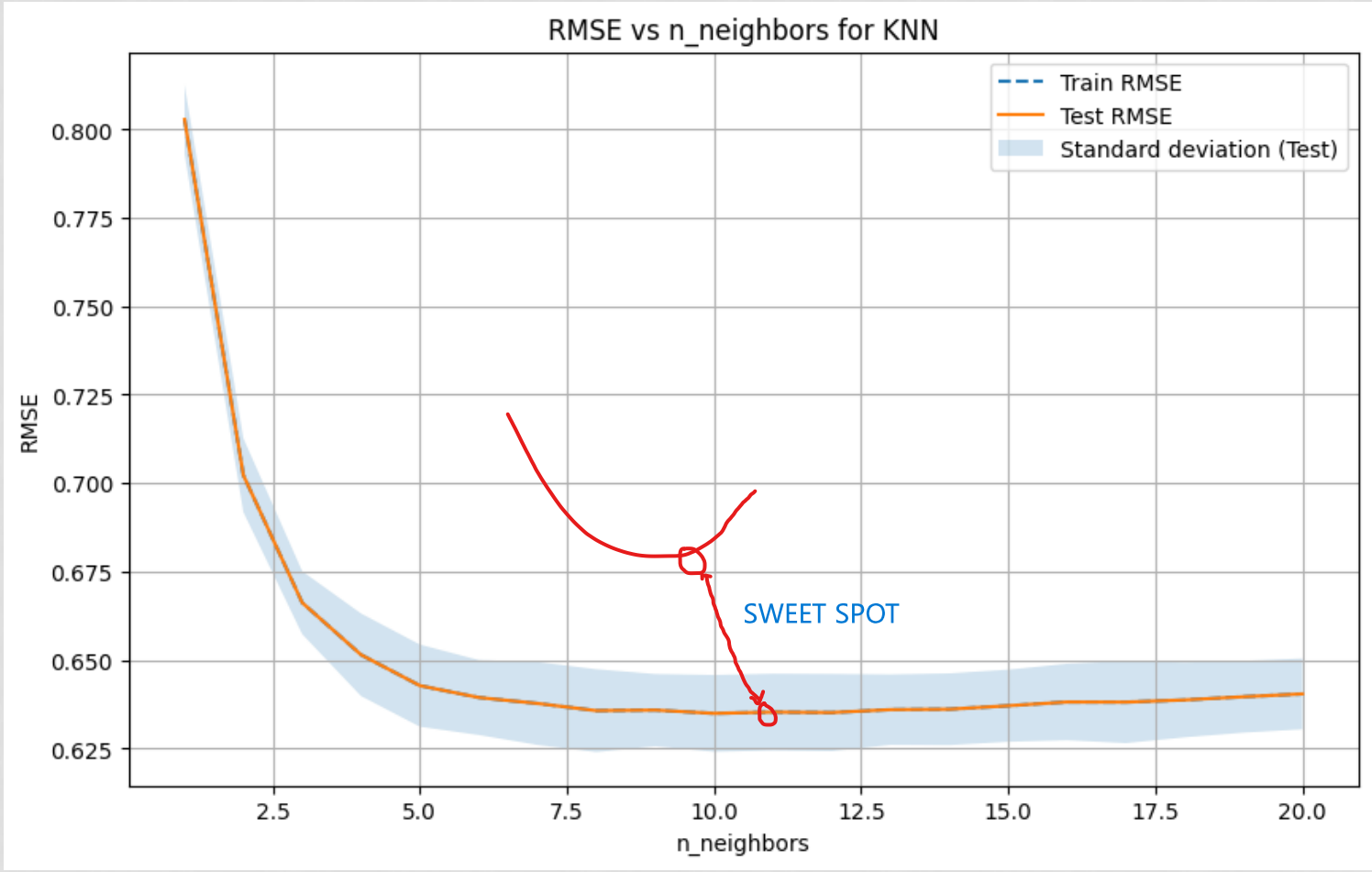
Our goal is to find the sweet spot



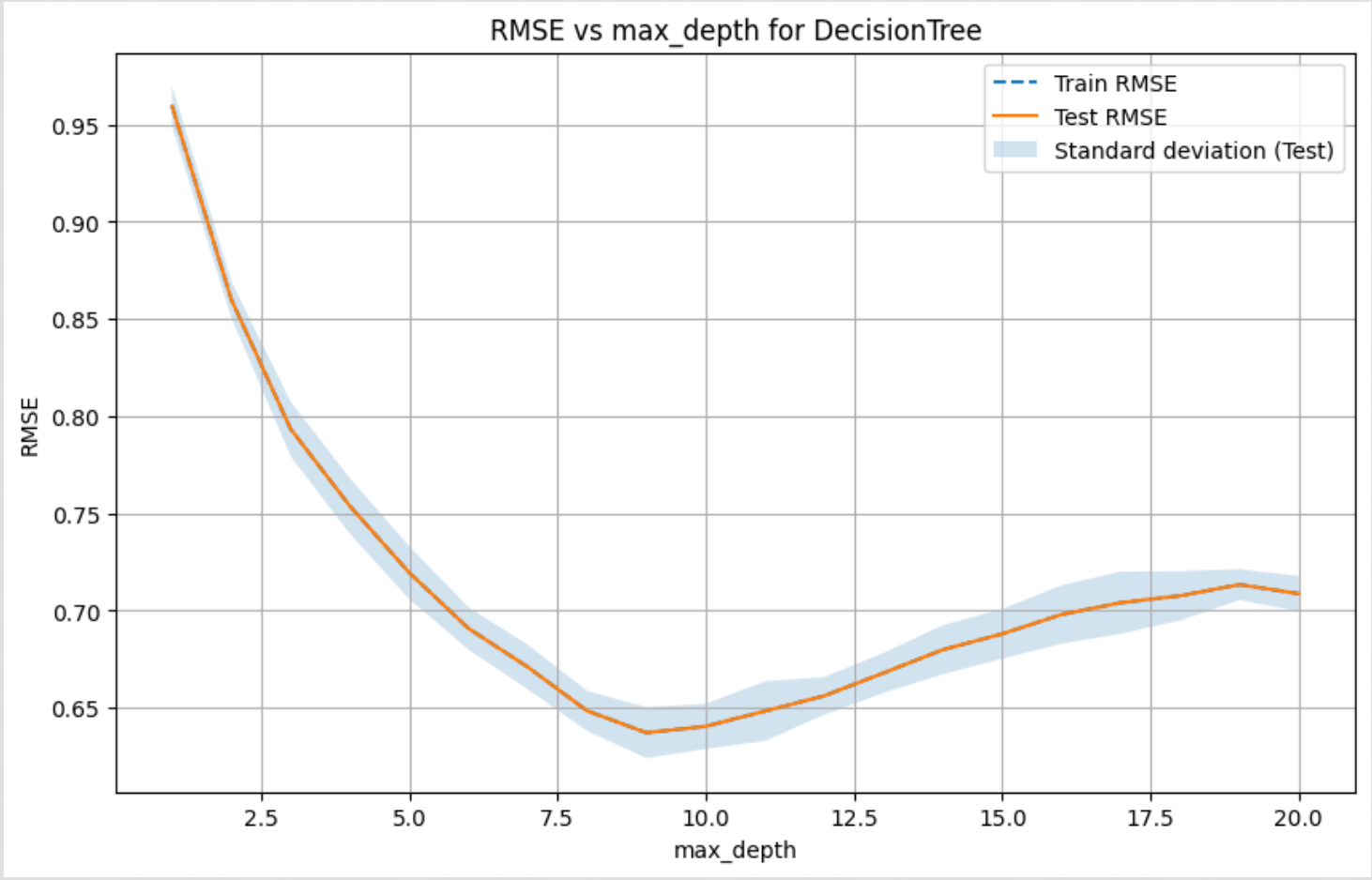
TREES: MAX_DEPTH VS. TEST ACCURACY



HYPER-PARAMETERS FOR A REGRESSION PROBLEM. KNN



HYPER-PARAMETERS FOR A REGRESSION PROBLEM. TREES



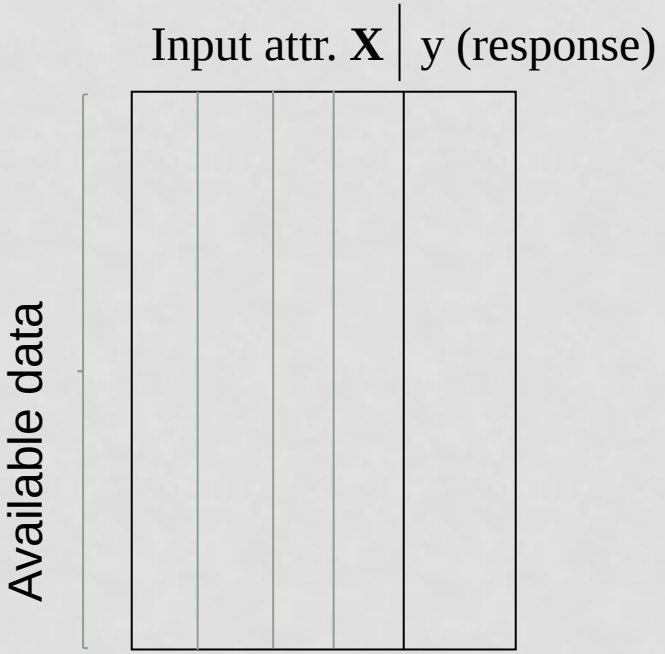
HYPER-PARAMETER OPTIMIZATION (**HPO**) INDEX

- Motivation for HPO:
  - Finding out the h.p. that result in models with the right complexity
  - HPO is "improved training": instead of using the default h.p. values, optimized h.p. will be found out and used.
- **HPO and estimation of future performance wih train/validation/test**
- HPO and estimation of future performance wih inner = crossvalidation and outer = test (and inner/outer with crossvalidation)
- Standard methods for HPO:
  - Grid-search
  - Random-search
- Improved methods for HPO:
  - Sequential Model Based Optimization / Bayes Optimization
  - Fixed vs. Non-fixed hyper-parameters search space: Optuna (define-by-run)
  - Successive Halving
- The CASH problem (Combined Algorithm Selection and Hyper-parameter tuning)

HYPER-PARAMETER OPTIMIZATION (**HPO**)

- HPO (Hyper-Parameter Optimization) can be automated by using a validation (or development) partition.
- Idea: train-and-evaluate: train models with different hyper-parameter values, evaluate these different models with a validation partition and select the best hyper-parameter value.
- However, let us remember that we also want to have an estimation of future performance, therefore we also need a test partition.

HPO WITH A VALIDATION PARTITION



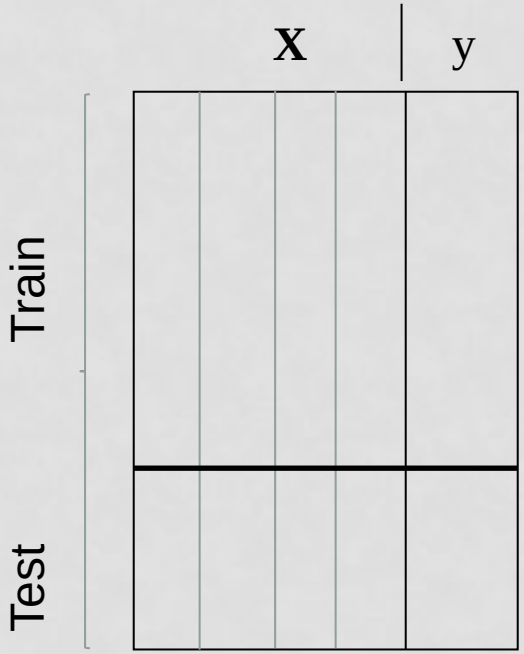
Machine Learning

Different tasks

Estimation of future performance from the final model (evaluation by train/test and training again with all data)

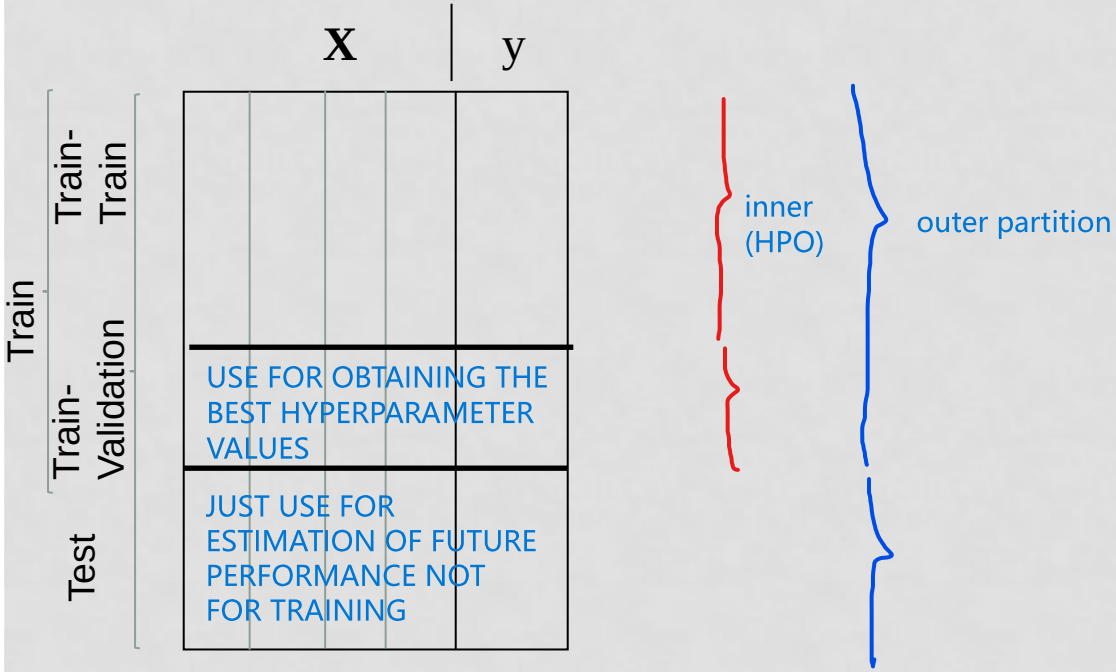
- This is my available data.

HPO WITH A VALIDATION PARTITION (AND MODEL EVALUATION WITH TEST)



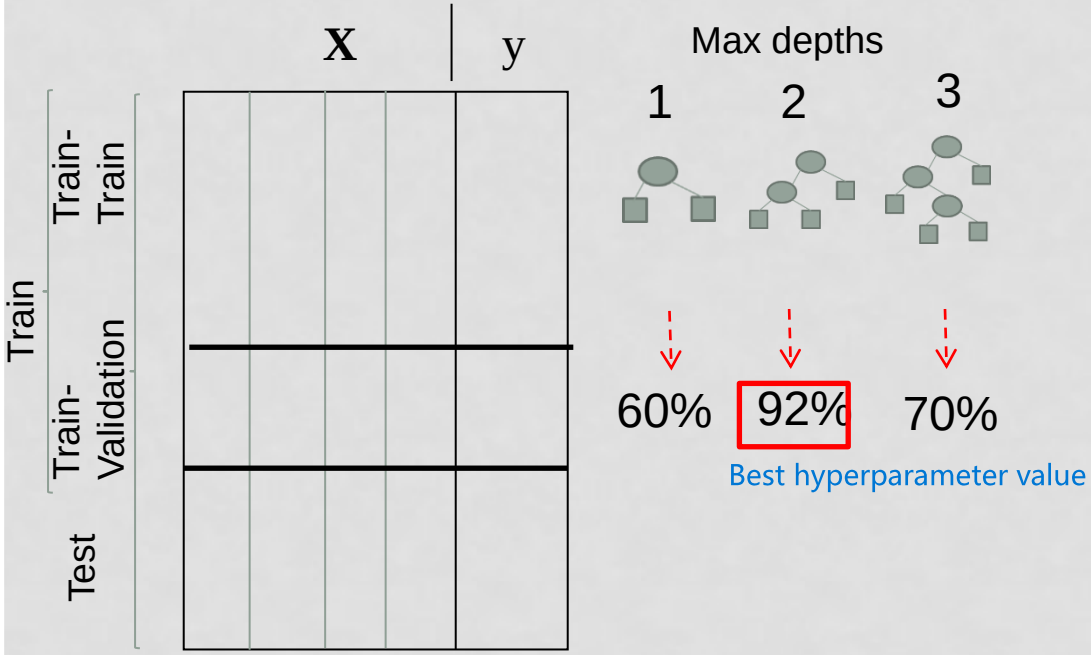
- A test partition is held out for estimating future performance (model evaluation)

HPO WITH A VALIDATION PARTITION (AND MODEL EVALUATION WITH TEST)



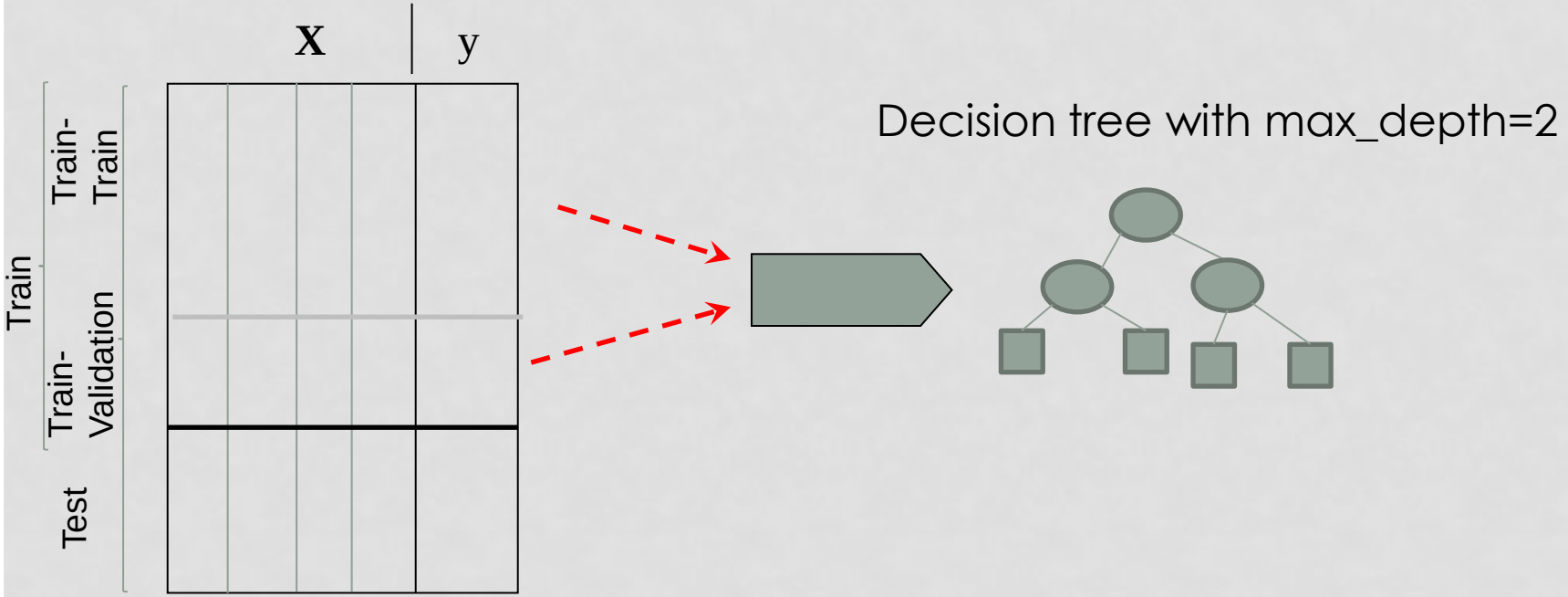
- The train partition is subdivided again into train and validation (or dev set)
  - Both train and validation are subsets used for training, so we can call them train-train and train-validation to make this more explicit
- Train is for building models with different hyper-parameters
- Validation is for evaluating these models, and selecting the model with the best hyper-parameter value

HPO WITH A VALIDATION PARTITION (AND MODEL EVALUATION WITH TEST)



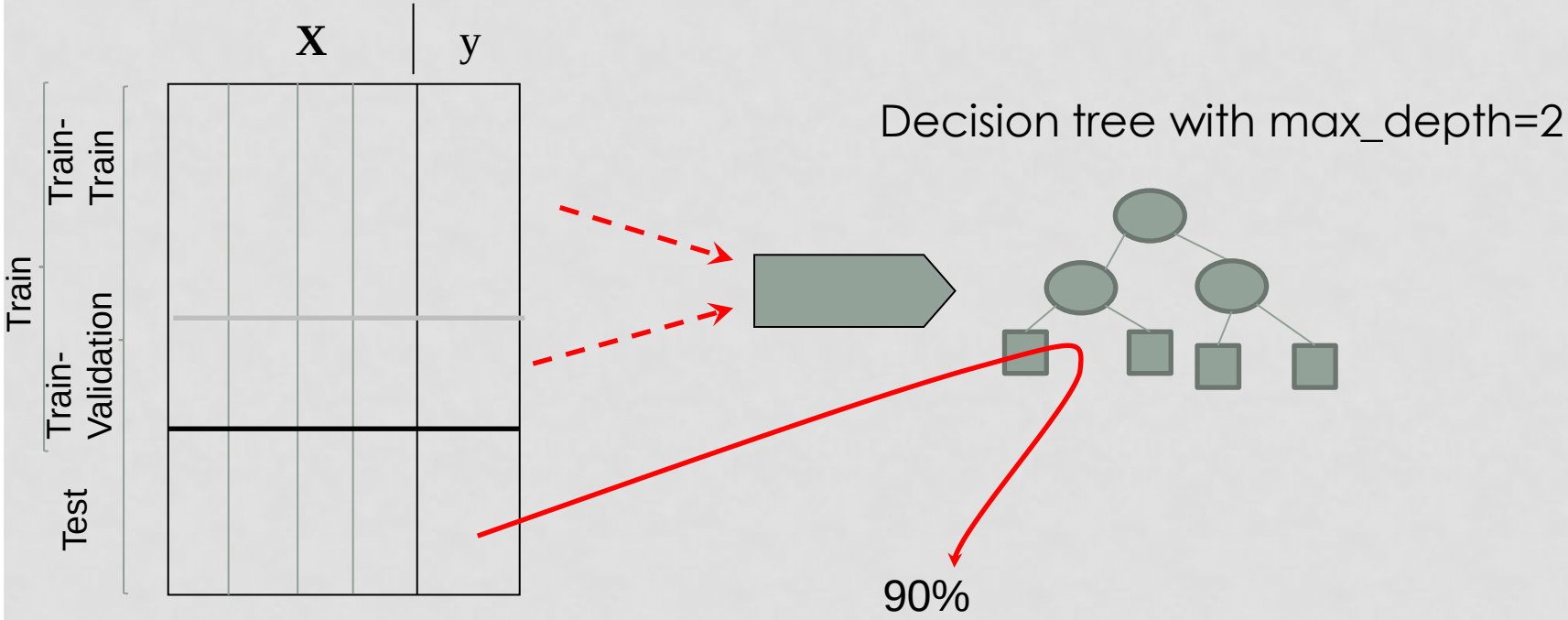
- The best max_depth is 2

HPO WITH A VALIDATION PARTITION (AND MODEL EVALUATION WITH TEST)



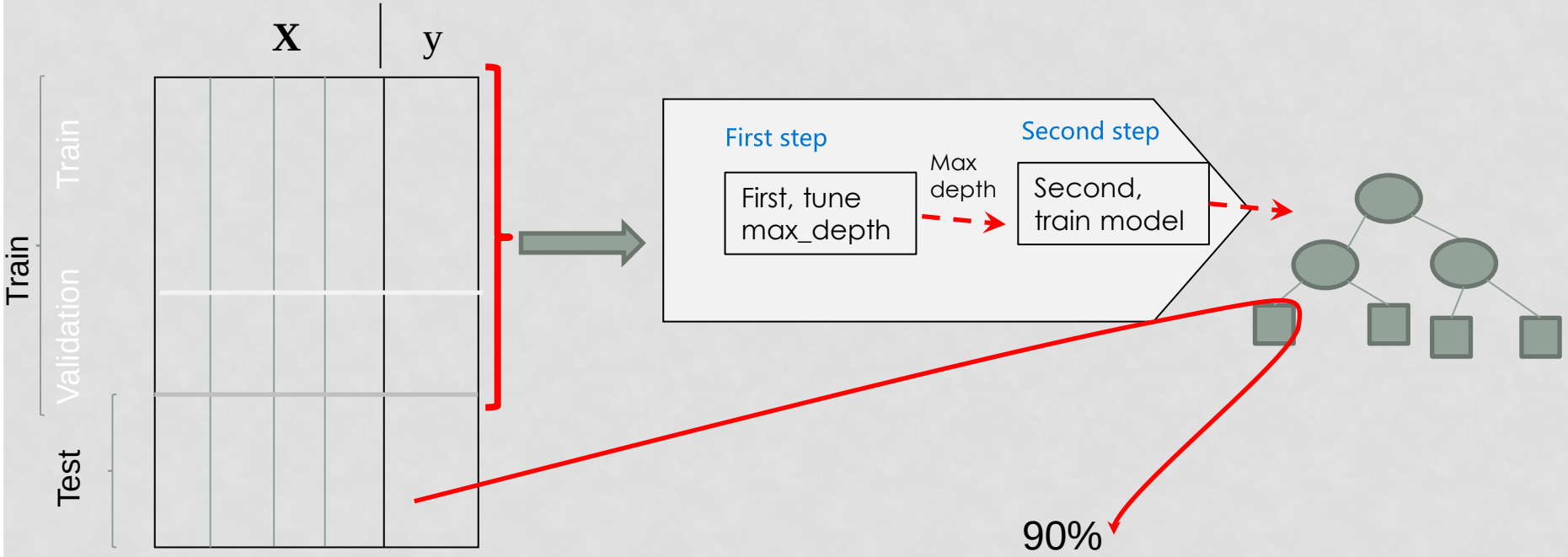
- Max_depth = 2 is selected and a new model with max_depth = 2 is trained with the complete train set (train-train + train-validation).

HPO WITH A VALIDATION PARTITION (AND MODEL EVALUATION WITH TEST)



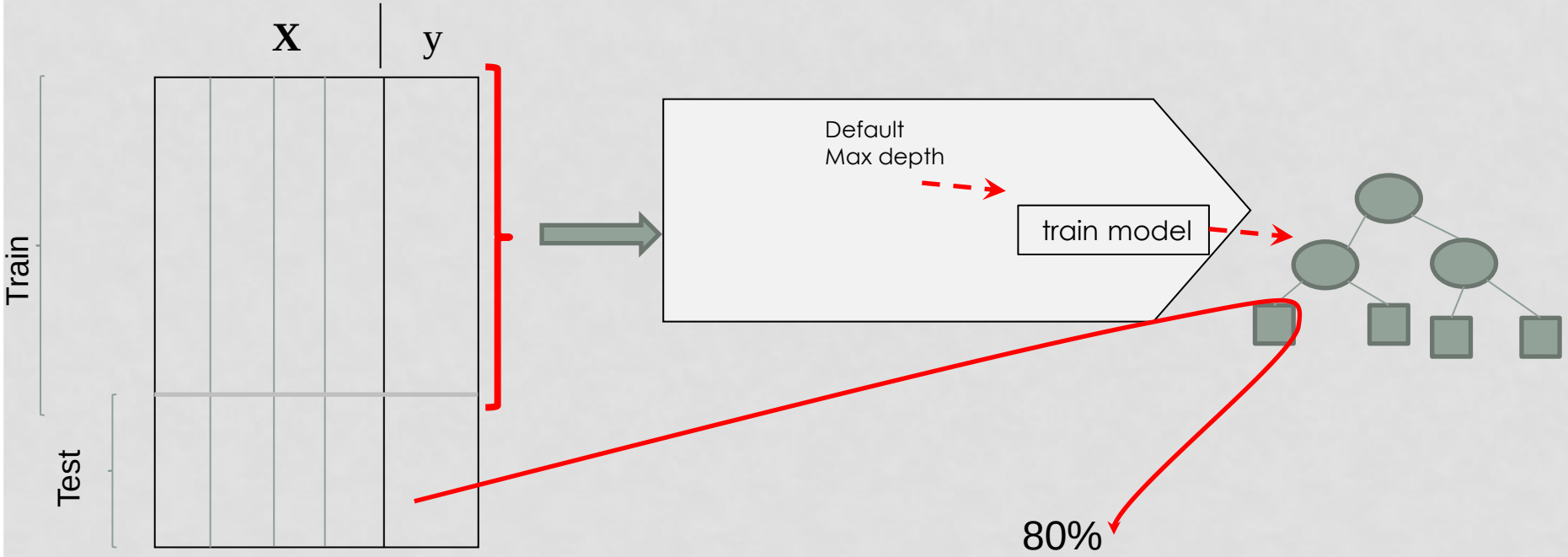
- Finally, the model is evaluated with the test partition.

HPO WITH A VALIDATION PARTITION (AND MODEL EVALUATION WITH TEST)



- All this is equivalent to having a two-step method that takes data as input and returns a model as output.
  - First, the best max_depth hyper-parameter was selected
  - Second, a decision tree was trained with the best max_depth

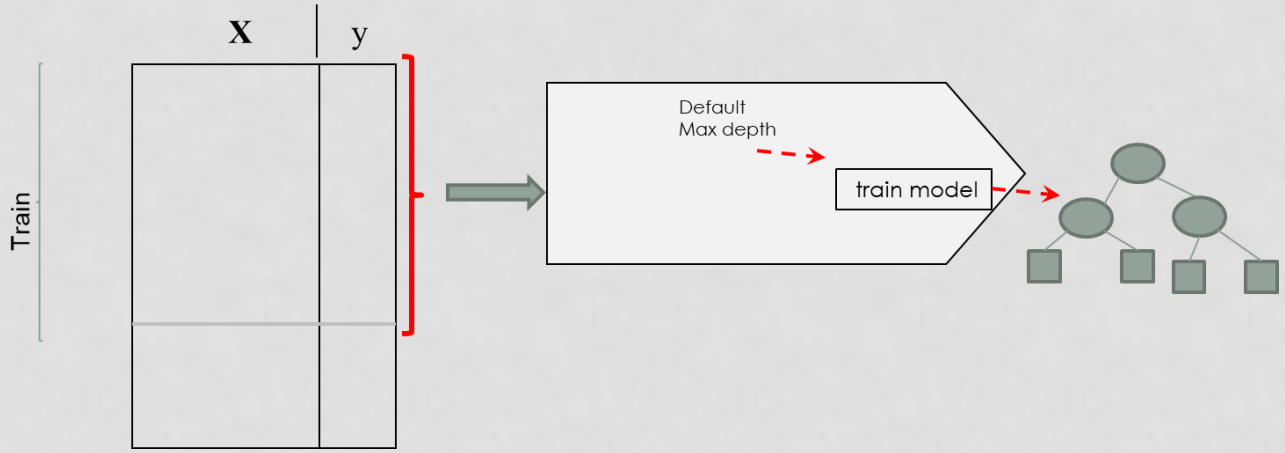
HPO WITH A VALIDATION PARTITION (AND MODEL EVALUATION WITH TEST)



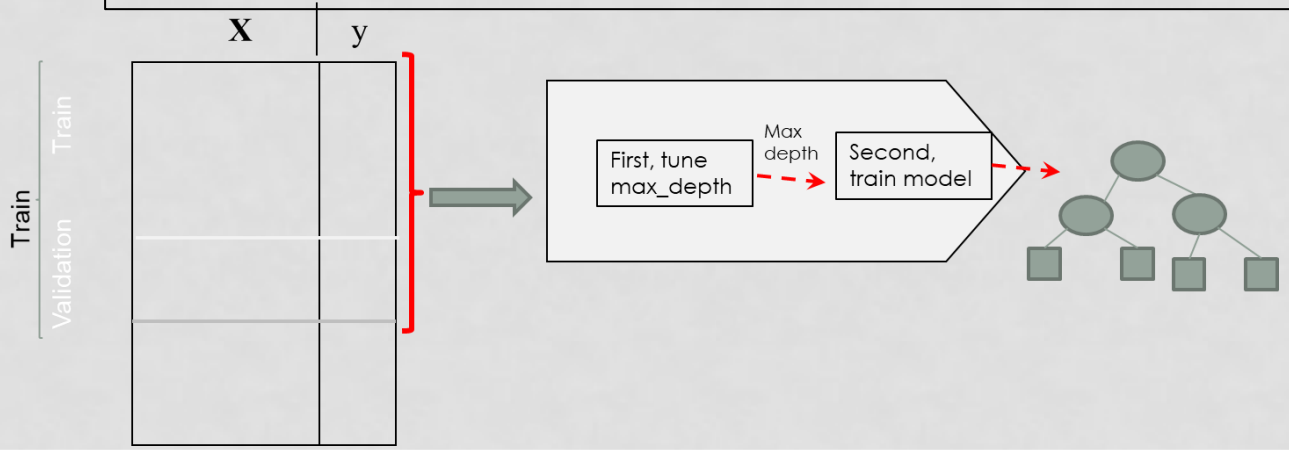
- Compare the two-step process (with HPO, previous slide) with using default hyper-parameter values (this slide).

NOTE: HPO IS "IMPROVED TRAINING"

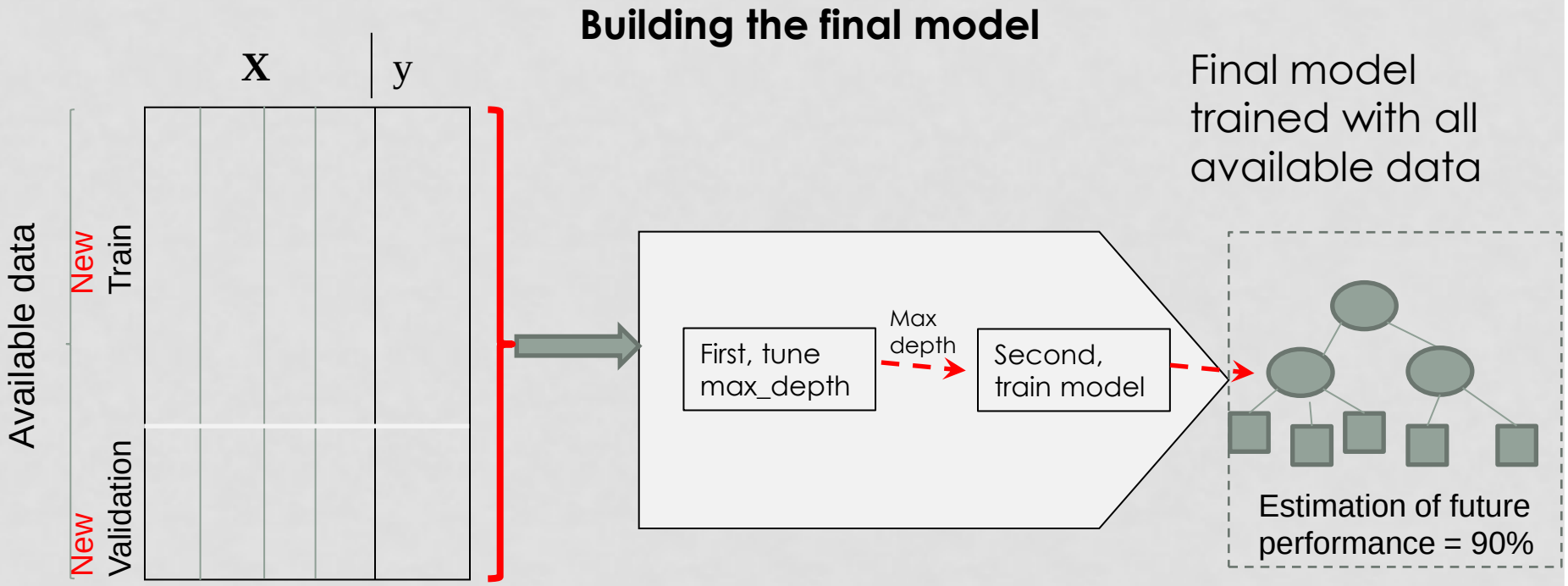
Training a model with default hyper-parameter values



"Improved training" of a model (training with HPO)



HPO WITH A VALIDATION PARTITION (AND MODEL EVALUATION WITH TEST)

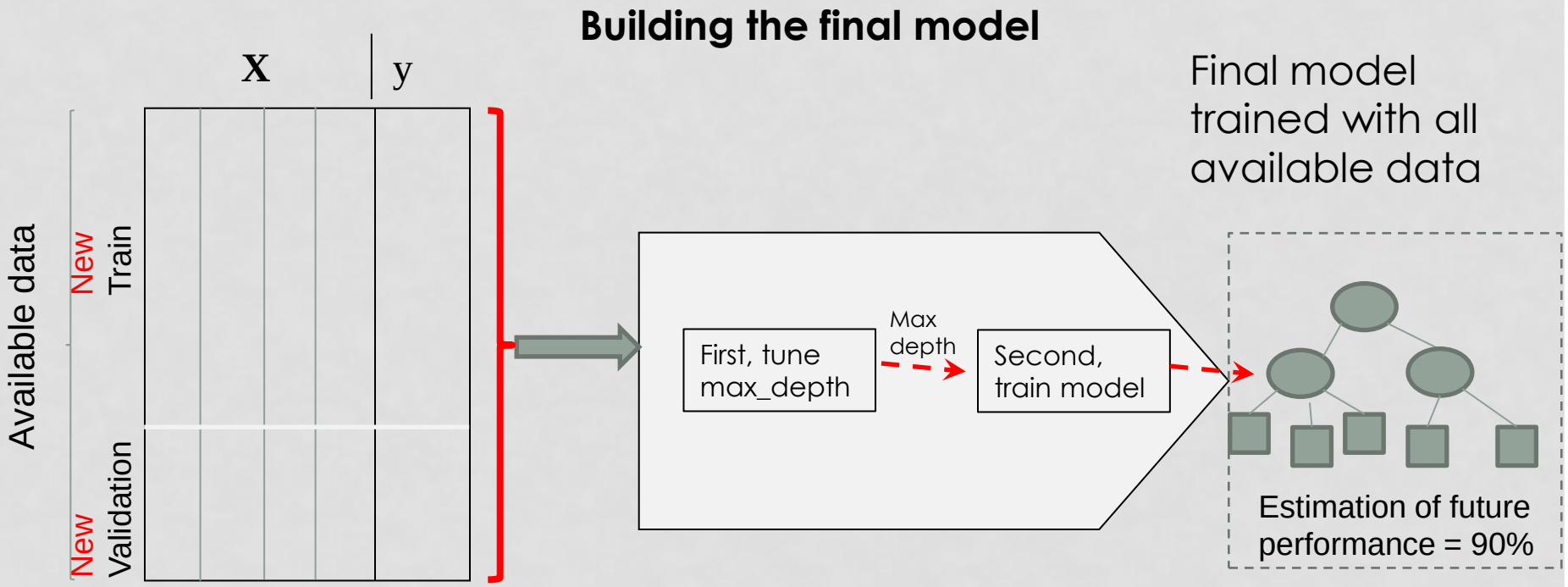


- This two-step method is the one we will use if we want to create a final model that will actually be used by our company (or sent to a competition) :
  - First, the best max_depth hyper-parameter was selected
  - Second, a decision tree was trained with the best max_depth
- This two-step process will be applied to the entire available data (train+validation+test)
- The estimation of future performance is kept (we know it is a pessimistic estimation)

OPTIONS FOR TRAINING THE FINAL MODEL (WITH HPO)

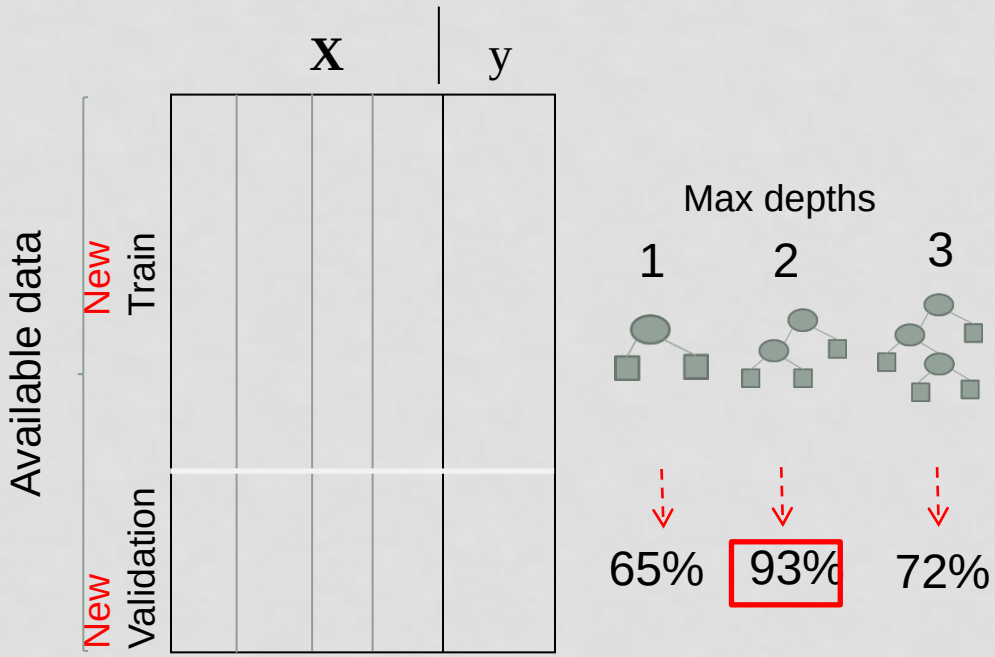
1. **<u>Preferred option</u>: train the final model with the all available data. Do HPO again (using all available data).**
2. Less time consuming option: train the final model with the complete dataset, but keep the hyper-parameters obtained during model evaluation.
3. Even less time consuming: keep the model obtained during model evaluation (do not retrain the model on the complete dataset)

THE FINAL MODEL: **OPTION 1**



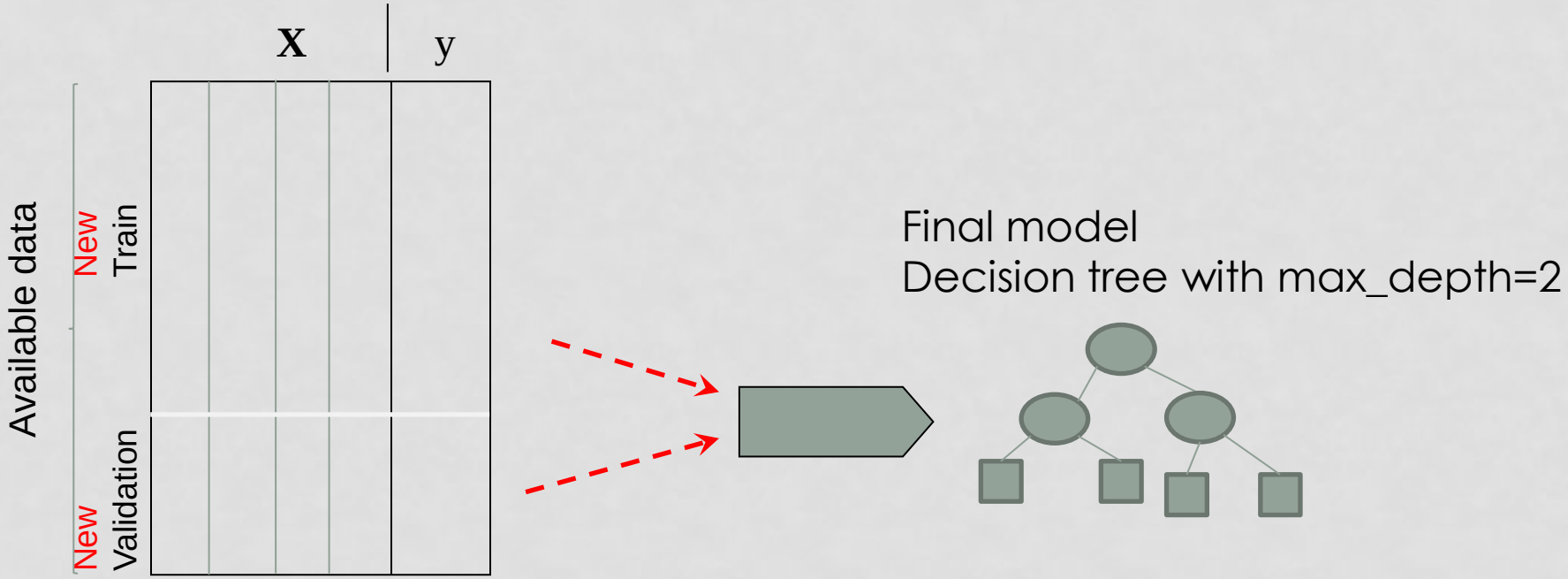
THE FINAL MODEL: **OPTION 1**

Preferred option: train the final model with the complete dataset. Do HPO again (using the complete dataset).



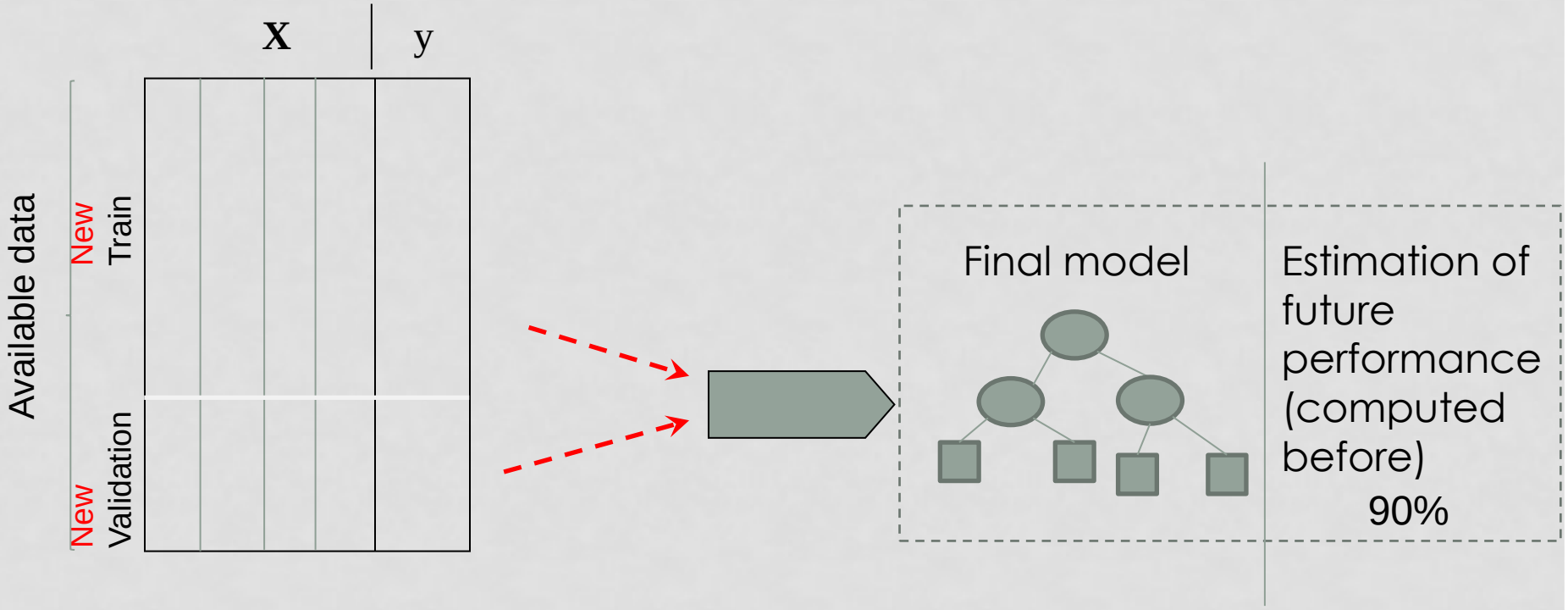
THE FINAL MODEL: **OPTION 1**

- Preferred option: train the final model using all available data. Do hyper-parameter tuning **again** (using using all available data).
- Given that training now involves HPO, we do hyper-parameter tuning again by splitting the data into new training and validation partitions.



THE FINAL MODEL: **OPTION 1**

- … and we keep the estimation of future performance computed **before**

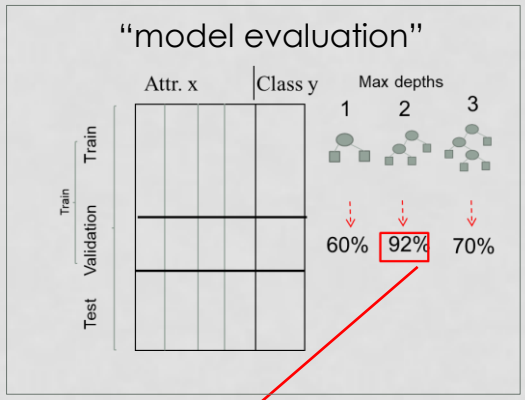


OPTIONS FOR TRAINING THE FINAL MODEL (WITH HPO)

1. Preferred option: train the final model with the complete dataset. Do hyper-parameter tuning again (using the complete dataset).
2. **Less time consuming: train the final model with the complete dataset, but keep the hyper-parameters obtained during model evaluation.**
3. Even less time consuming. Keep the model obtained during model evaluation (do not retrain on the complete dataset)

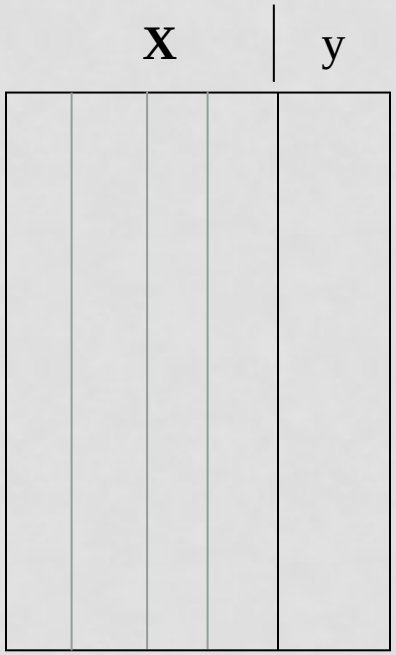
THE FINAL MODEL: **OPTION 2**

- Second option (less time consuming): train the final model with the complete dataset, but keep the hyper-parameters obtained during model evaluation

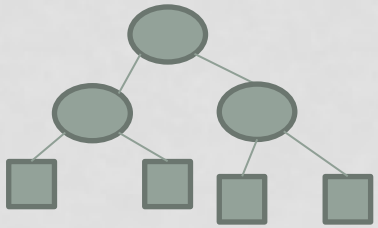


Max_depth=2 comes from "model evaluation"

Available data



Final model
Decision tree with max_depth=2

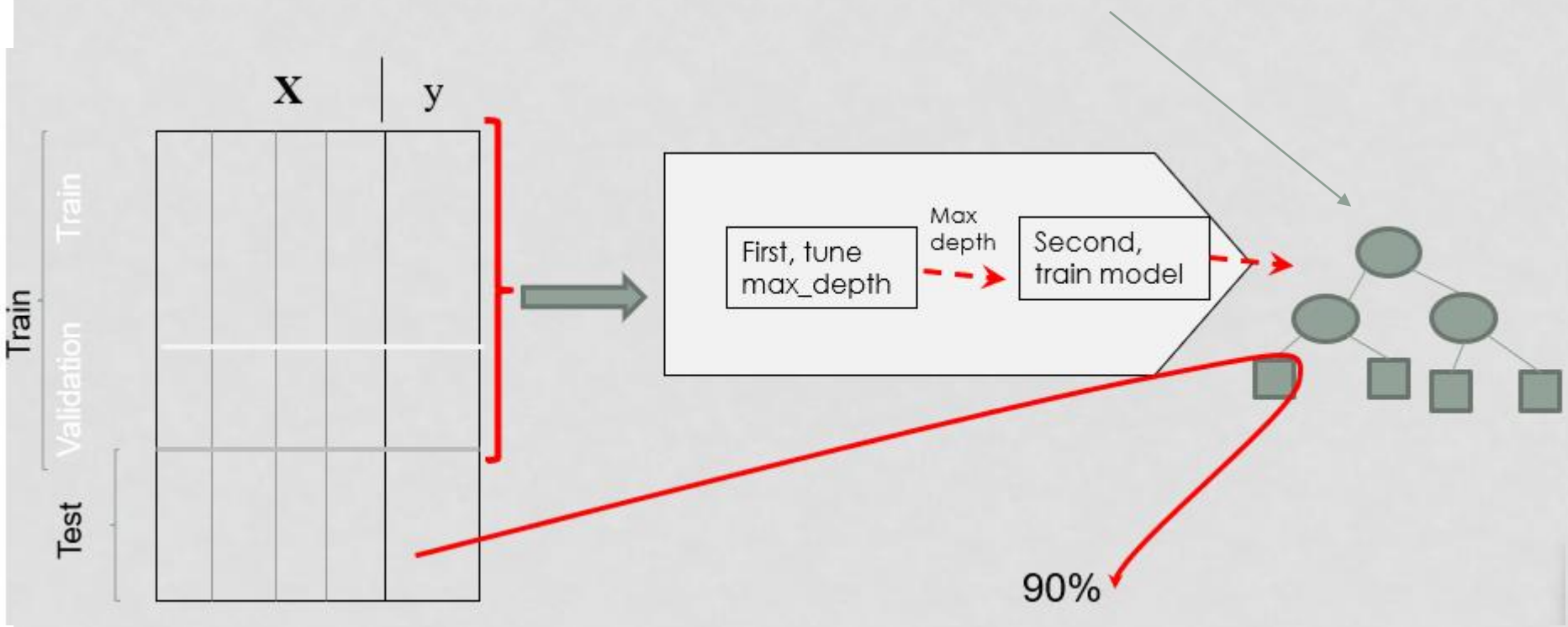


OPTIONS FOR TRAINING THE FINAL MODEL (WITH HPO)

1. Preferred option: train the final model with the complete dataset. Do hyper-parameter tuning again (using the complete dataset).
2. Less time consuming: train the final model with the complete dataset, but keep the hyper-parameters obtained during model evaluation.
3. **Even less time consuming. Keep the model obtained during model evaluation (do not retrain the model on the complete dataset)**

THE FINAL MODEL: **OPTION 3**

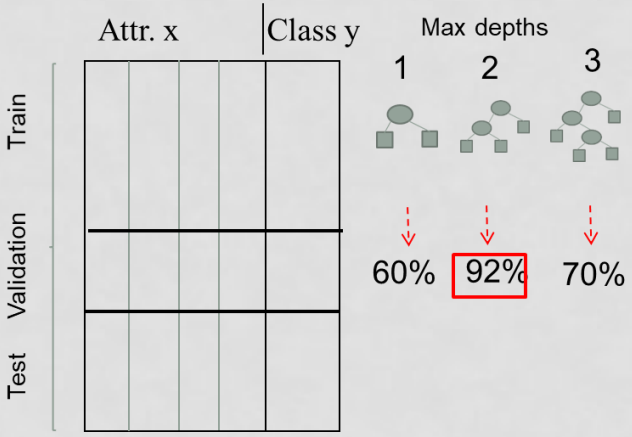
This is the model for getting the estimation of future performance (90%), but it can also be kept as the "final model"



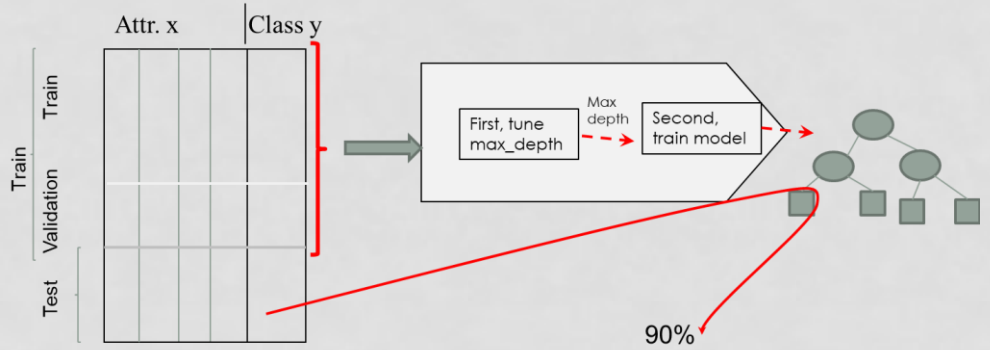
NOTE

- Estimation of future performance is also called "**outer evaluation**"
  - It happens outside the training process (outside HPO)
- Evaluation for comparing different hyper-parameter values is also called "**inner evaluation**"
  - It happens inside the training process (inside HPO)

Inner evaluation



Outer evaluation



- Let's illustrate HPO with code:
  - Outer = train/test
  - Inner = train_train / train_validation

Preparatory code

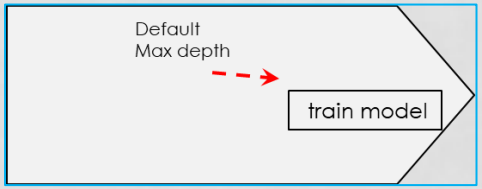
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.dummy import DummyRegressor
from sklearn.metrics import mean_squared_error
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import GridSearchCV, KFold, PredefinedSplit

# Fetch the California housing dataset
california = fetch_california_housing()

# Features and target
X = california.data
y = california.target

rs = 42
```

Estimation of future performance, outer = train/test

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=rs)
```

Regression tree with default hyper-parameters

regr_default =




```
# Define a regression tree estimator
regr_default = DecisionTreeRegressor(random_state=rs)
```

```
# Train estimator with default hp
regr_default.fit(X_train, y_train)
```

outer = train/test

```
# Make predictions with trained model
y_test_pred = regr_default.predict(X_test)

print("Regression tree RMSE with default hp:",
      np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

the default value of *max_depth* is None, which means "build the tree as deep as possible":

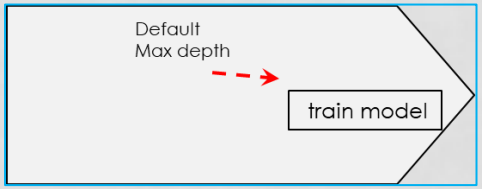
https://scikit-learn.org/dev/modules/generated/sklearn.tree.DecisionTreeRegressor.html

DecisionTreeRegressor

```
class sklearn.tree.DecisionTreeRegressor(*, criterion='squared_error',
splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features=None, random_state=None,
max_leaf_nodes=None, min_impurity_decrease=0.0, ccp_alpha=0.0, monotonic_cst=None)
```

Regression tree with default hyper-parameters

regr_default =




```
# Define a regression tree estimator
regr_default = DecisionTreeRegressor(random_state=rs)
```

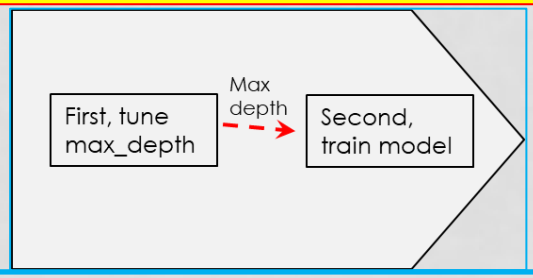
```
# Train estimator with default hp
regr_default.fit(X_train, y_train)
```

outer = train/test

```
# Make predictions with trained model
y_test_pred = regr_default.predict(X_test)

print("Regression tree RMSE with default hp:",
      np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

Regression tree with HPO

regr_hpo =




```
from sklearn.model_selection import ShuffleSplit
# Define estimator with HPO:
# search space + inner evaluation + estimator + scoring

# Search space
param_grid = {
    'max_depth': range(2, 16, 2)
}

# Inner evaluation: train/validation
inner = ShuffleSplit(n_splits=1,test_size=1/3,random_state=rs)
# Estimator
regr_default = DecisionTreeRegressor(random_state=rs)

# Definition of the 2-step process for HPO
regr_hpo = GridSearchCV(regr_default,
                        param_grid,
                        scoring='neg_mean_squared_error',
                        cv=inner,
                        n_jobs=1, verbose=1)
```

Inner = train/validation

regr_hpo.best_params_
regr_hpo.best_score_

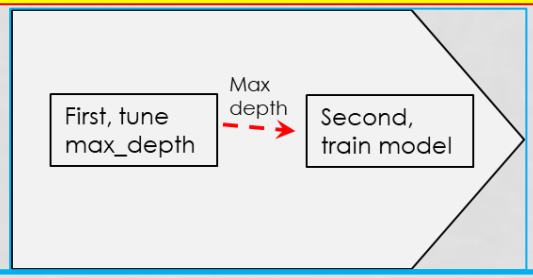
```
# Train estimator
regr_hpo.fit(X_train, y_train)
```

```
# Make predictions with trained model
y_test_pred = regr_hpo.predict(X_test)

# outer = train/test
print("Regression tree RMSE with HPO:",
      np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

outer = train/test

Regression tree with HPO

regr_hpo =

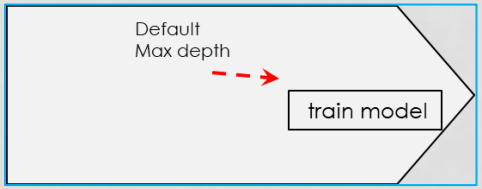



```
from sklearn.model_selection import ShuffleSplit
# Define estimator with HPO:
# search space + inner evaluation + estimator + scoring

# Search space
param_grid = {
    'max_depth': range(2, 16, 2)
}

# Inner evaluation: train/validation
inner = ShuffleSplit(n_splits=1,test_size=1/3,random_state=rs)
# Estimator
regr_default = DecisionTreeRegressor(random_state=rs)

# Definition of the 2-step process for HPO
regr_hpo = GridSearchCV(regr_default,
                        param_grid,
                        scoring='neg_mean_squared_error',
                        cv=inner,
                        n_jobs=1, verbose=1)
```

Inner = train/validation

Regression tree with default hyper-parameters

regr_default =




```
# Define a regression tree estimator
regr_default = DecisionTreeRegressor(random_state=rs)
```

```
# Train estimator with default hp
regr_default.fit(X_train, y_train)
```

```
# Make predictions with trained model
y_test_pred = regr_default.predict(X_test)

print("Regression tree RMSE with default hp:",
      np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

outer = train/test

```
# Train estimator
regr_hpo.fit(X_train, y_train)
```

```
# Make predictions with trained model
y_test_pred = regr_hpo.predict(X_test)

# outer = train/test
print("Regression tree RMSE with HPO:",
      np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

outer = train/test

Training the final model

With default hyper-parameters

```
regr_final = regr_default.fit(X, y)
```

With entire dataset (X,y) and HPO (option 1, preferred)

```
regr_final = regr_hpo.fit(X, y)
```

---

best_params_ contains the best hp

With entire dataset but keeping best hp (option 2)

```
regr_with_optimal_hp = DecisionTreeRegressor(**regr_hpo.best_params_,
                                             random_state=rs)
regr_final = regr_with_optimal_hp.fit(X, y)

print(regr_hpo.best_params_)
```

```
{'max_depth': 10                          }
```

With model used for estimation of performance (trained with just (X_train, y_train) (option 3)

```
regr_final = regr_hpo
```

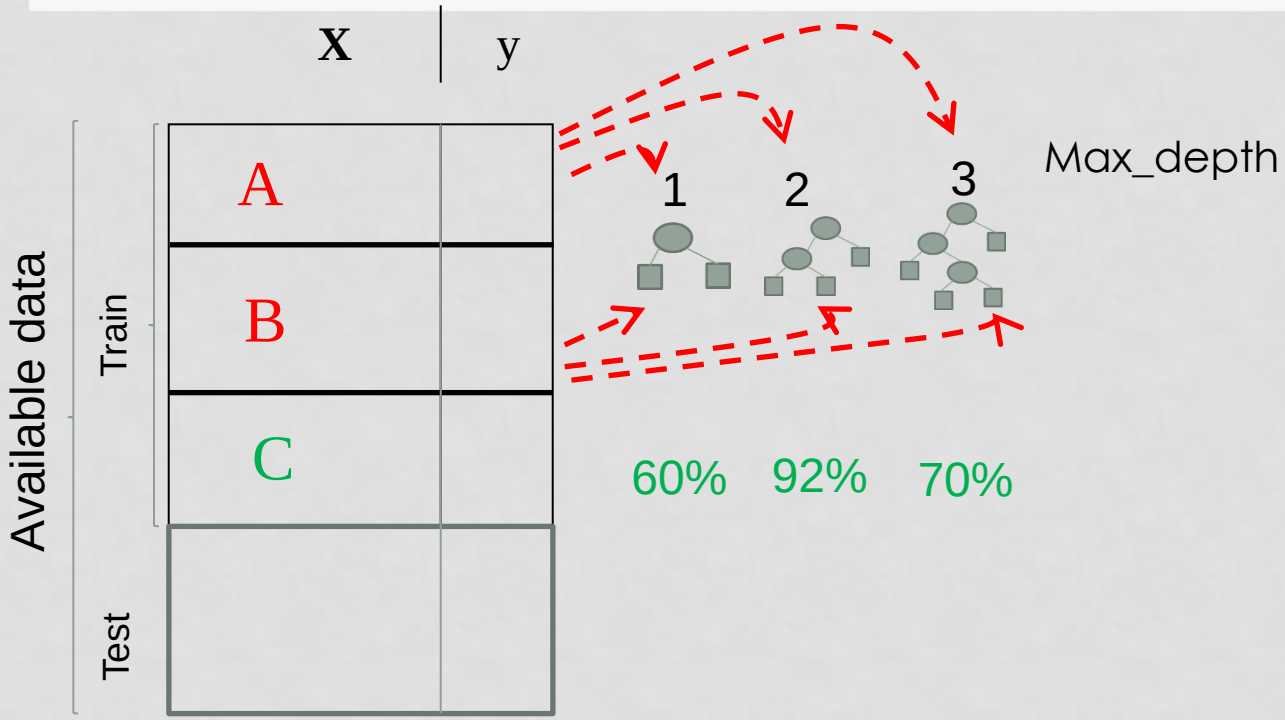
2 equivalent ways: best_estimator_ contains the model with best hp

```
regr_final = regr_hpo.best_estimator_
```

HYPER-PARAMETER OPTIMIZATION (**HPO**) INDEX

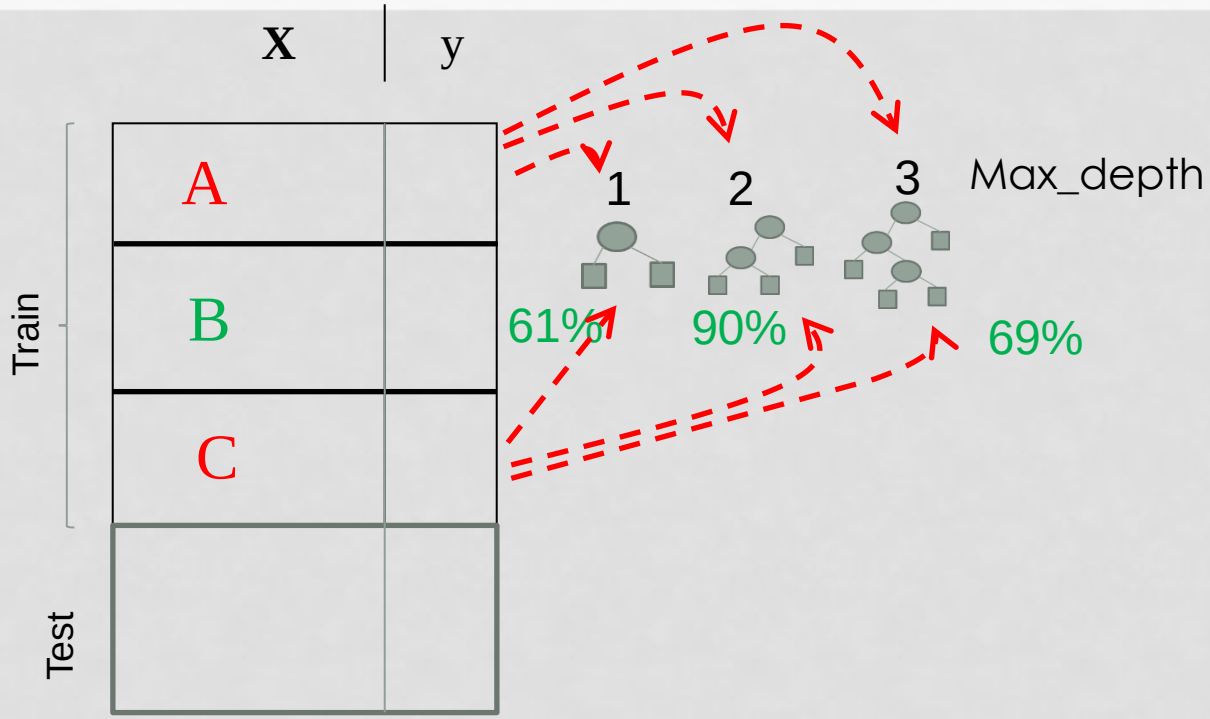
- Motivation for HPO
- HPO and estimation of future performance wih train/validation/test
- **HPO and estimation of future performance wih inner = crossvalidation and outer = test (and inner/outer with crossvalidation)**
- Standard methods for HPO:
  - Grid-search
  - Random-search
- Improved methods for HPO:
  - Sequential Model Based Optimization / Bayes Optimization
  - Fixed vs. Non-fixed hyper-parameters search space: Optuna (define-by-run)
  - Successive Halving
- The CASH problem (Combined Algorithm Selection and Hyper-parameter tuning)

HPO WITH CROSSVALIDATION (INNER)



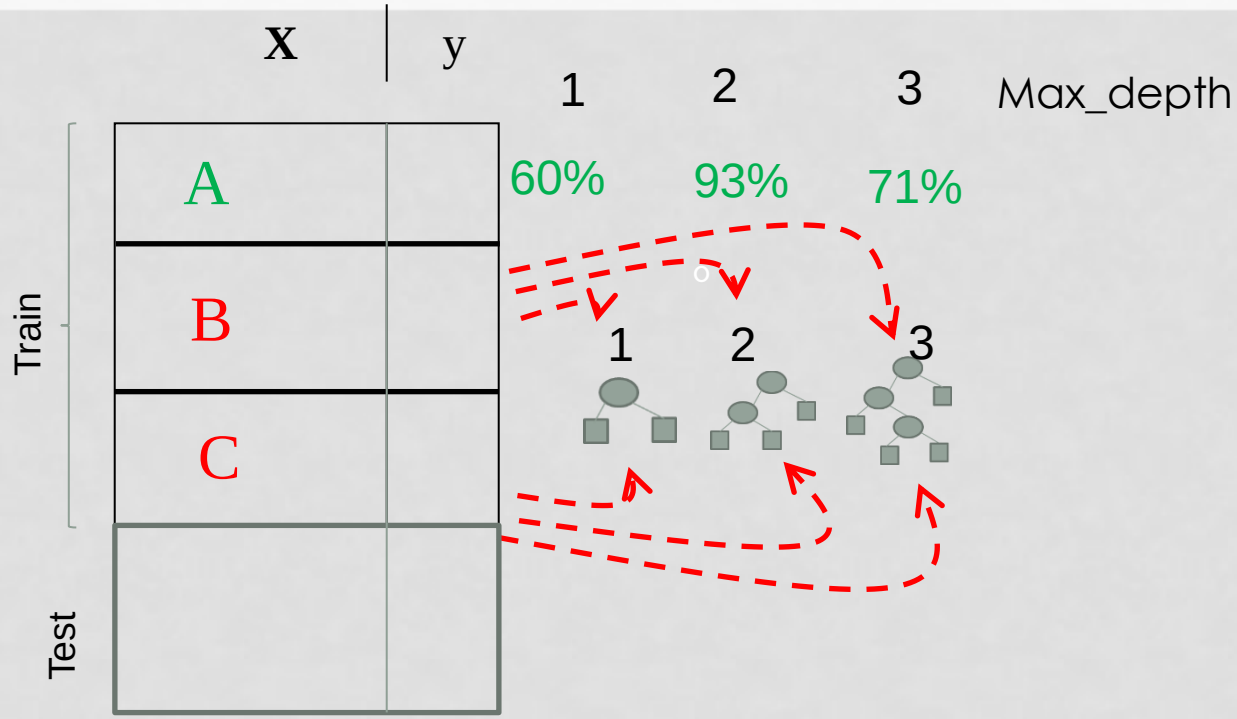
- Now, we are going to use 3-fold crossvalidation for hyper-parameter tuning, but train/test (holdout) for model evaluation (a.k.a. estimation of future performance)
- First, we train with A and B (train_train), and validate with C (train_validation)

HPO WITH CROSSVALIDATION



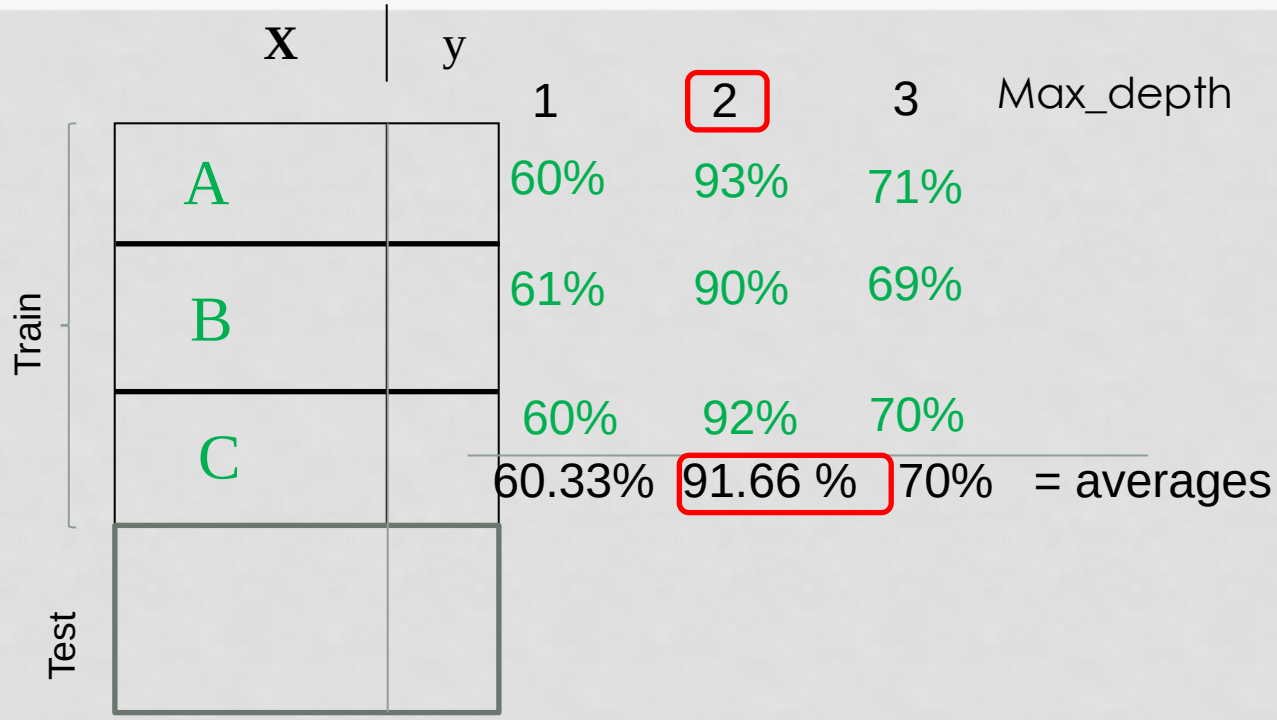
- Then, we train with A and C, and validate with B

HPO WITH CROSSVALIDATION



- Finally, we train with B and C, and validate with A

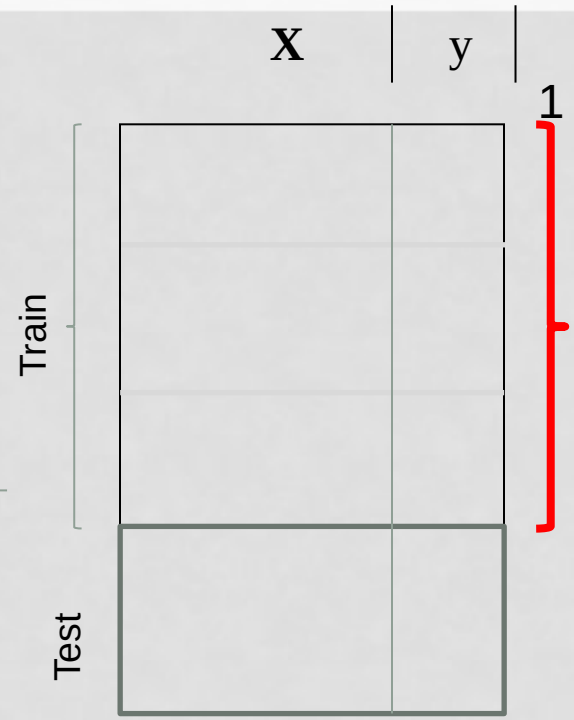
HPO WITH CROSSVALIDATION



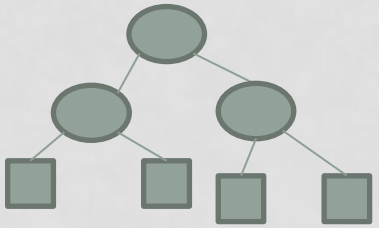
- Finally, each hyper-parameter value is evaluated by computing the average of the three folds.
- Max depth = 2 is the best.

HPO WITH CROSSVALIDATION

Max_depth		
1	2	3
60%	93%	71%
61%	90%	69%
60%	92%	70%
60.33%	91.66 %	70%



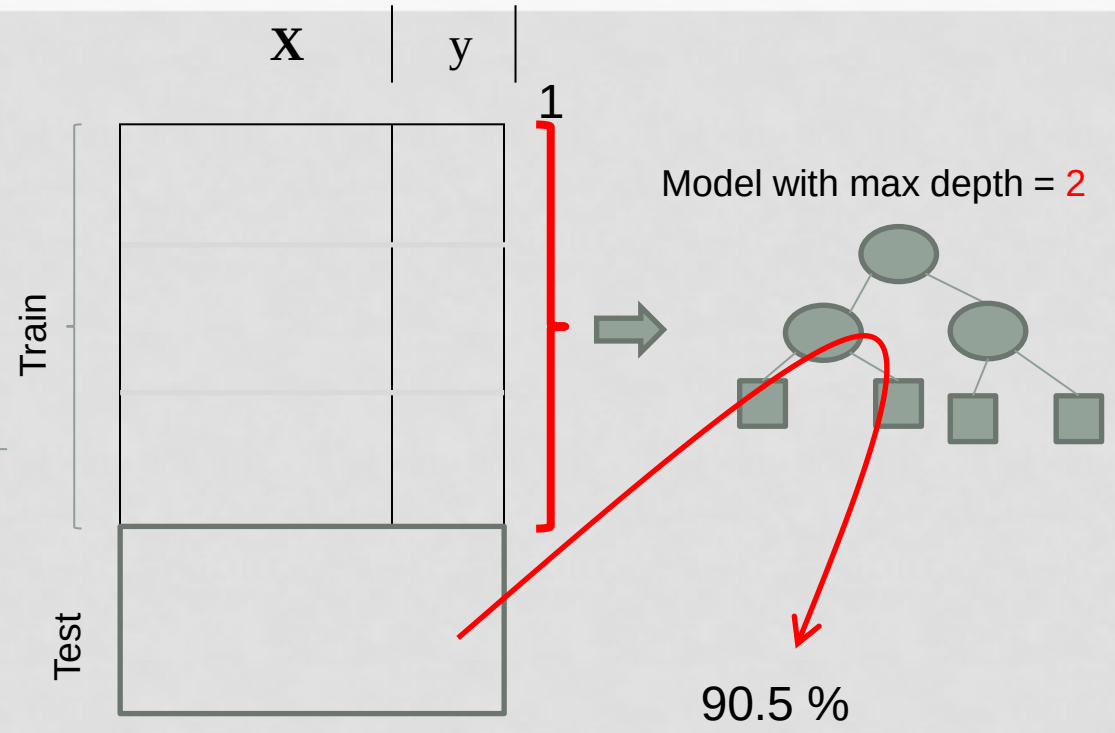
Model with max depth = 2



- A model is trained with the whole train partition, with the best max depth.

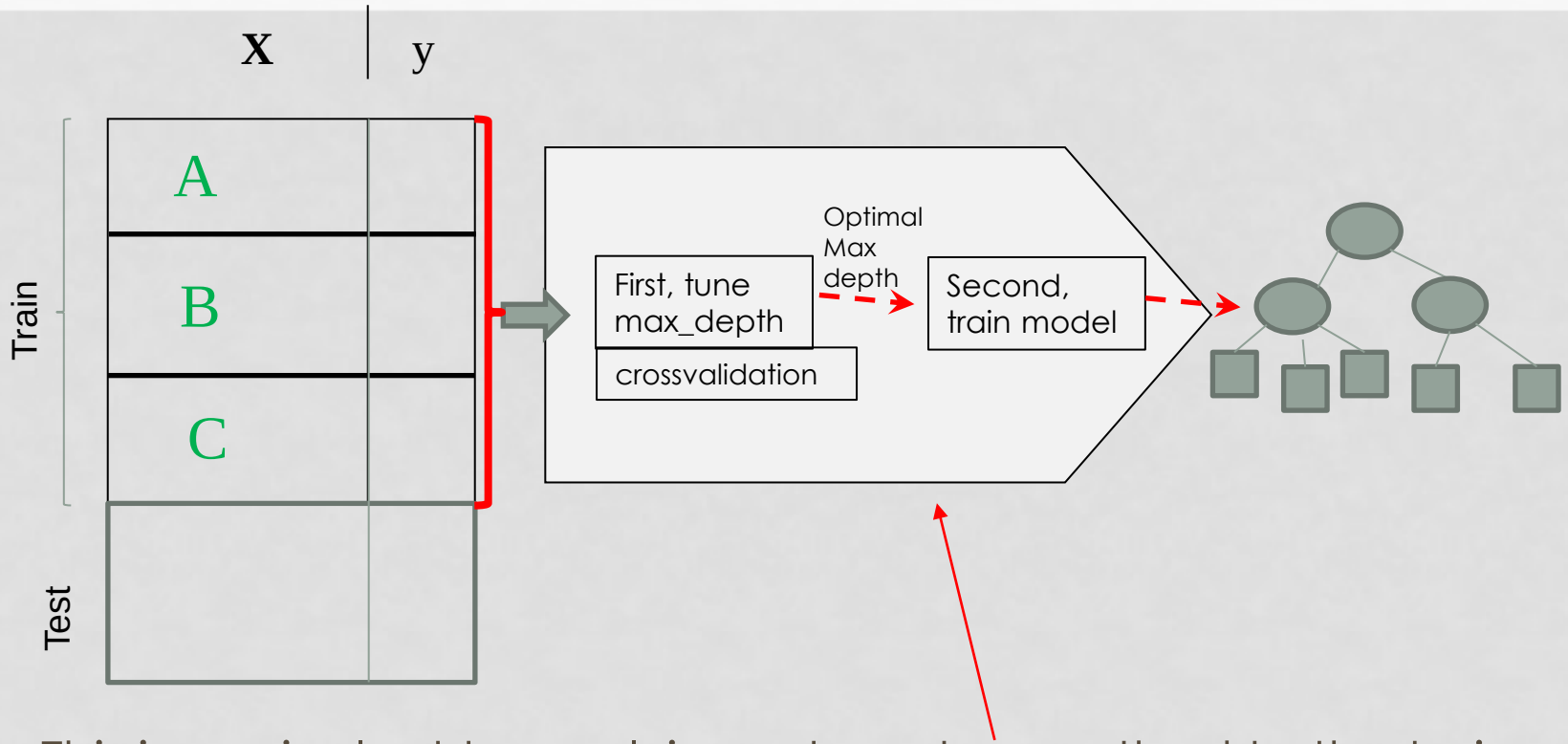
HPO WITH CROSSVALIDATION

Max_depth		
1	2	3
60%	93%	71%
61%	90%	69%
60%	92%	70%
60.33%	91.66 %	70%



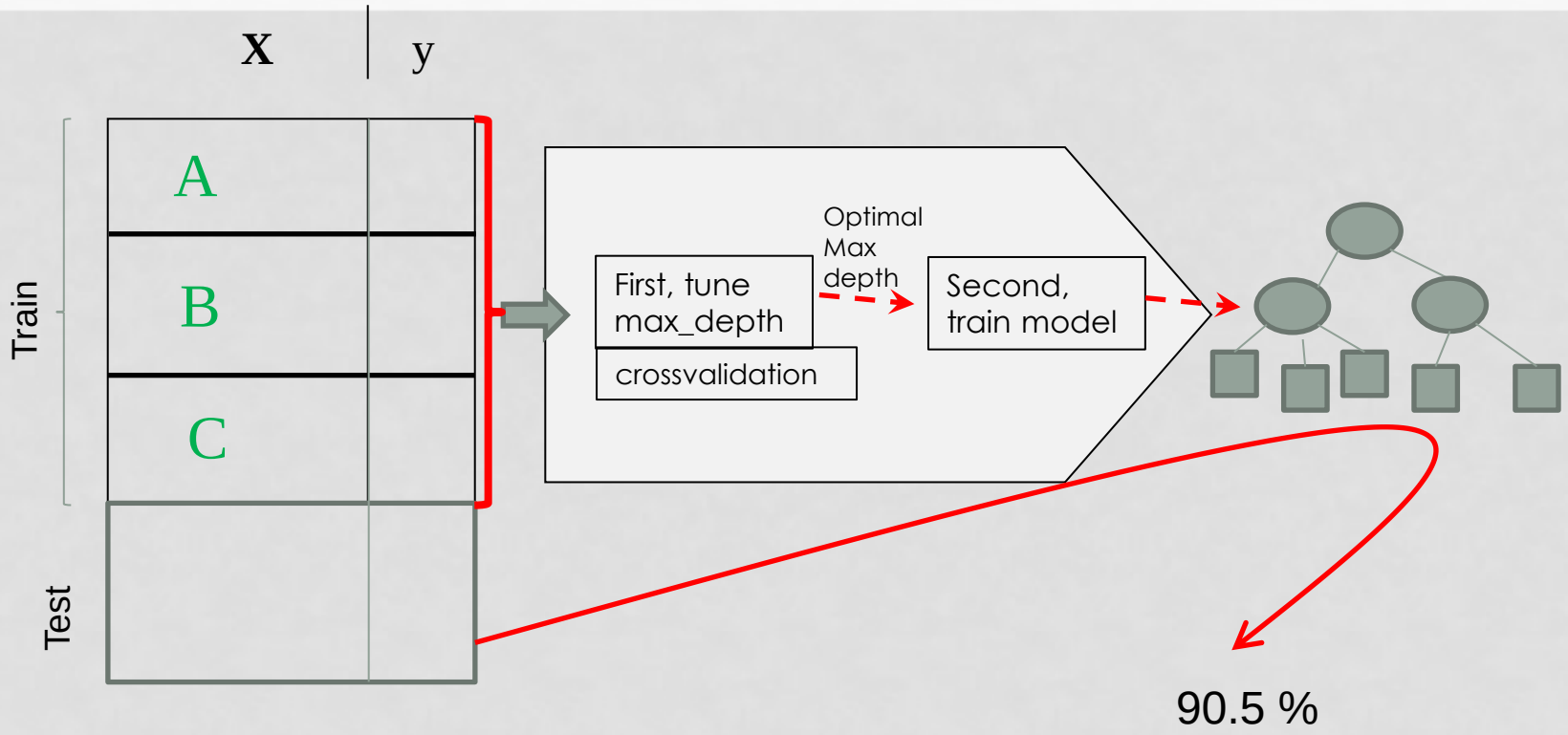
- And then it is evaluated with the test partition

HPO WITH CROSSVALIDATION

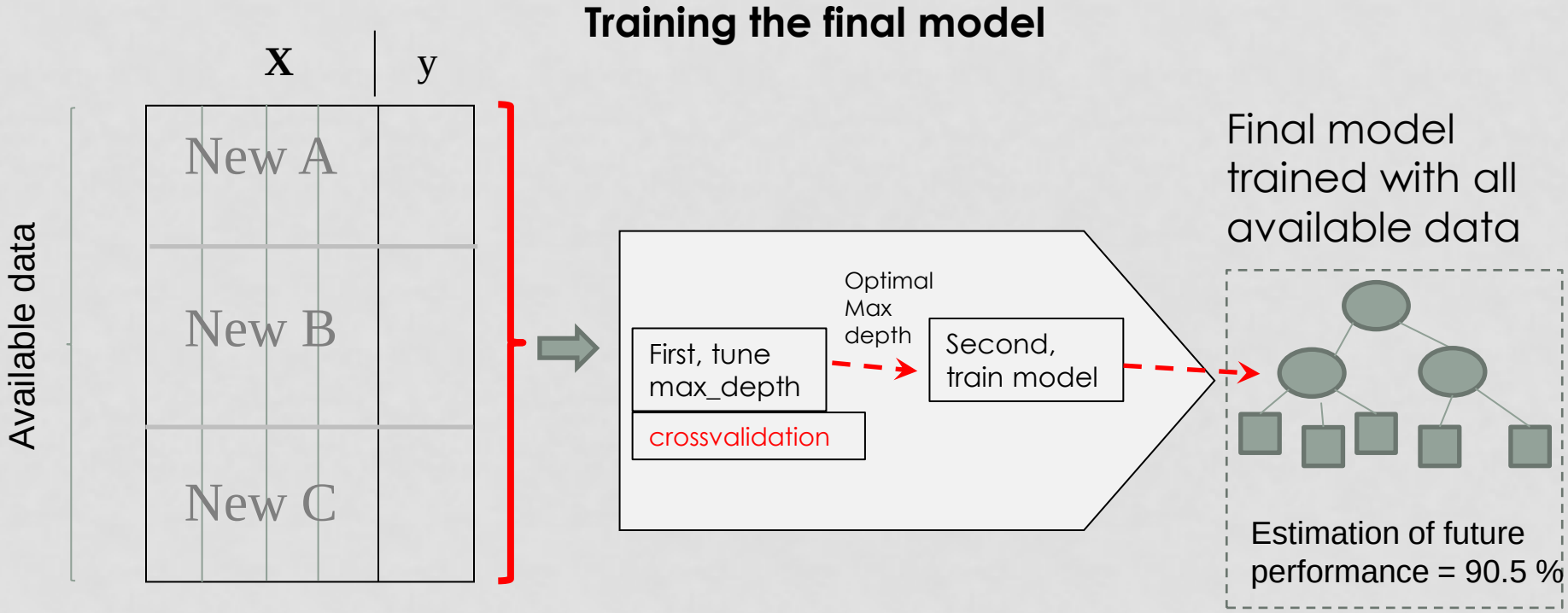


- This is equivalent to applying a two-step method to the train partition (A+B+C).

HPO WITH CROSSVALIDATION



HPO WITH A CROSSVALIDATION: TRAINING THE FINAL MODEL



- Finally, we apply the two-step method to the entire available data, in order to produce the final model.
- The main difference with "hyper-parameter tuning with a validation partition" is that now, the hyper-parameter tuning step uses crossvalidation instead of train_train/train_validation.
- The estimation of future performance computed previously is kept.

TRAINING THE FINAL MODEL

- Like previously, we could also use the second and third options, if we don't have too much compute time:

1. Preferred option: train the final model with the complete dataset. This involves doing hyper-parameter tuning again (using the complete dataset), as explained in the previous slide.
2. Less time consuming: train the final model with the complete dataset, but keep the hyper-parameters obtained during estimation of future performance.
3. Even less time consuming. Keep the model obtained during model evaluation (do not retrain the model on the complete dataset)

- Let's illustrate the different options with code

Preparatory code

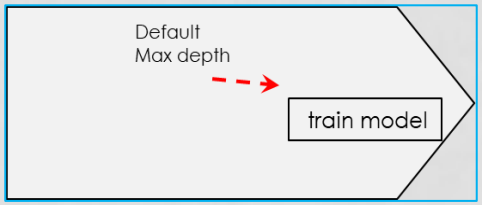
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.dummy import DummyRegressor
from sklearn.metrics import mean_squared_error
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import GridSearchCV, KFold, PredefinedSplit

# Fetch the California housing dataset
california = fetch_california_housing()

# Features and target
X = california.data
y = california.target

rs = 42
```

Estimation of future performance, **outer= train/test**

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=rs)
```

Regression tree with default hyper-parameters

regr_default =



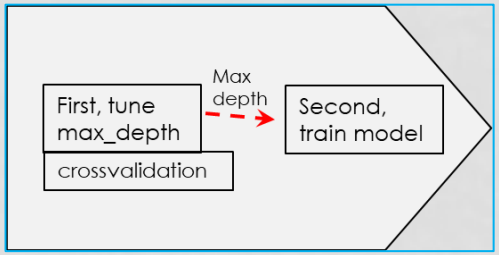

```
# Define a regression tree estimator
regr_default = DecisionTreeRegressor(random_state=rs)
```

```
# Train estimator with default hp
regr_default.fit(X_train, y_train)
```

outer = train/test

```
# Make predictions with trained model
y_test_pred = regr_default.predict(X_test)

print("Regression tree RMSE with default hp:",
      np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

regr_hpo =



Regression tree with HPO with **inner = crossvalidation**

```
# Define estimator with HPO:
# search space + inner evaluation + estimator + scoring

# Search space
param_grid = {
    'max_depth': list(range(2, 16, 2))

}

# Inner evaluation
inner = KFold(n_splits=3, shuffle=True, random_state=42)

# Estimator
regr_default = DecisionTreeRegressor(random_state=rs)

# Definition of the 2-step process for HPO
regr_hpo = GridSearchCV(regr_default,
                        param_grid,
                        scoring='neg_mean_squared_error',
                        cv=inner,
                        n_jobs=1, verbose=1)
```

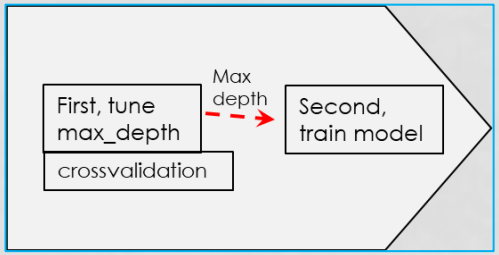
Inner = 3-fold xval

```
# Train estimator with HPO
regr_hpo.fit(X_train, y_train)
```

```
# Make predictions with trained model
y_test_pred = regr_hpo.predict(X_test)

print("Regression tree RMSE with HPO :",
      np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

outer = train/test

regr_hpo =



Regression tree with HPO with **inner = crossvalidation**

```
# Define estimator with HPO:
# search space + inner evaluation + estimator + scoring

# Search space
param_grid = {
    'max_depth': list(range(2, 16, 2))

}

# Inner evaluation
inner = KFold(n_splits=3, shuffle=True, random_state=42)

# Estimator
regr_default = DecisionTreeRegressor(random_state=rs)

# Definition of the 2-step process for HPO
regr_hpo = GridSearchCV(regr_default,
                        param_grid,
                        scoring='neg_mean_squared_error',
                        cv=inner,
                        n_jobs=1, verbose=1)
```

Inner = 3-fold xval

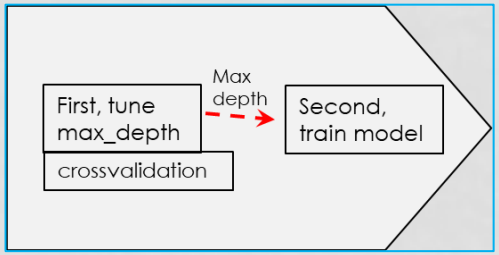
regr_hpo.best_params_
regr_hpo.best_score_

```
# Train estimator with HPO
regr_hpo.fit(X_train, y_train)
```

```
# Make predictions with trained model
y_test_pred = regr_hpo.predict(X_test)

print("Regression tree RMSE with HPO :",
      np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

outer = train/test

regr_hpo =



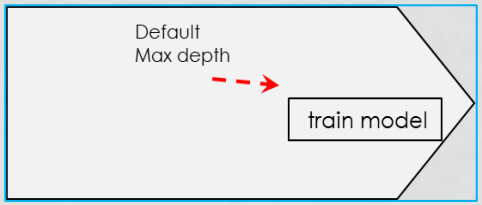
Regression tree with HPO with **inner = crossvalidation**

```
# Define estimator with HPO:
# search space + inner evaluation + estimator + scoring

# Search space
param_grid = {
    'max_depth': list(range(2, 16, 2))

}

# Inner evaluation
inner = KFold(n_splits=3, shuffle=True, random_state=42)

# Estimator
regr_default = DecisionTreeRegressor(random_state=rs)

# Definition of the 2-step process for HPO
regr_hpo = GridSearchCV(regr_default,
                        param_grid,
                        scoring='neg_mean_squared_error',
                        cv=inner,
                        n_jobs=1, verbose=1)
```

Inner = 3-fold xval

Regression tree with default hyper-parameters

regr_default =




```
# Define a regression tree estimator
regr_default = DecisionTreeRegressor(random_state=rs)
```

```
# Train estimator with default hp
regr_default.fit(X_train, y_train)
```

```
# Train estimator with HPO
regr_hpo.fit(X_train, y_train)
```

outer = train/test

```
# Make predictions with trained model
y_test_pred = regr_default.predict(X_test)

print("Regression tree RMSE with default hp:",
      np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

outer = train/test

```
# Make predictions with trained model
y_test_pred = regr_hpo.predict(X_test)

print("Regression tree RMSE with HPO :",
      np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

Regression tree with HPO with **inner = train/validation**

```
from sklearn.model_selection import ShuffleSplit
# Define estimator with HPO:
# search space + inner evaluation + estimator + scoring

# Search space
param_grid = {
    'max_depth': range(2, 16, 2)
}

# Inner evaluation: train/validation
inner = ShuffleSplit(n_splits=1,test_size=1/3,random_state=rs)

# Estimator
regr_default = DecisionTreeRegressor(random_state=rs)

# Definition of the 2-step process for HPO
regr_hpo = GridSearchCV(regr_default,
                        param_grid,
                        scoring='neg_mean_squared_error',
                        cv=inner,
                        n_jobs=1, verbose=1)

# Train estimator
regr_hpo.fit(X_train, y_train)

# Make predictions with trained model
y_test_pred = regr_hpo.predict(X_test)

# outer = train/test
print("Regression tree RMSE with HPO:",
      np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

Inner = train/validation

Regression tree with HPO with **inner = crossvalidation**

```
# Define estimator with HPO:
# search space + inner evaluation + estimator + scoring

# Search space
param_grid = {
    'max_depth': list(range(2, 16, 2)
}

# Inner evaluation
inner = KFold(n_splits=3, shuffle=True, random_state=42)

# Estimator
regr_default = DecisionTreeRegressor(random_state=rs)

# Definition of the 2-step process for HPO
regr_hpo = GridSearchCV(regr_default,
                        param_grid,
                        scoring='neg_mean_squared_error',
                        cv=inner,
                        n_jobs=1, verbose=1)

# Train estimator with HPO
regr_hpo.fit(X_train, y_train)

# Make predictions with trained model
y_test_pred = regr_hpo.predict(X_test)

print("Regression tree RMSE with  HPO      :",
      np.sqrt(mean_squared_error(y_test, y_test_pred)))
```

Inner = 3-fold xval

Training the final model

With entire dataset (X,y) and HPO (option 1, preferred)

```
regr_final = regr_hpo.fit(X, y)
```

best_params_ contains the best hp

With entire dataset but keeping best h.p. (option 2)

```
regr_with_optimal_hp = DecisionTreeRegressor(**regr_hpo.best_params_,
                                             random_state=rs)
regr_final = regr_with_optimal_hp.fit(X, y)

print(regr_hpo.best_params_)
```

```
{'max_depth': 10}
```

With model used for estimation of performance (trained with just (X_train, y_train) (option 3)

```
regr_final = regr_hpo
```

2 equivalent ways: best_estimator_ contains the model with best hp

```
regr_final = regr_hpo.best_estimator_
```

Training the final model

With default hyper-parameters

```
regr_final = regr_default.fit(X, y)
```

With entire dataset (X,y) and HPO (option 1, preferred)

```
regr_final = regr_hpo.fit(X, y)
```

best_params_ contains the best hp

With entire dataset but keeping best h.p. (option 2)

```
regr_with_optimal_hp = DecisionTreeRegressor(**regr_hpo.best_params_,
                                             random_state=rs)
regr_final = regr_with_optimal_hp.fit(X, y)

print(regr_hpo.best_params_)
```

```
{'max_depth': 10                          }
```

With model used for estimation of performance (trained with just (X_train, y_train) (option 3)

```
regr_final = regr_hpo
```

2 equivalent ways: best_estimator_ contains the model with best hp

```
regr_final = regr_hpo.best_estimator_
```

NESTED CROSSVALIDATION

- We can also use crossvalidation for both HPO and estimation of future performance :
  - Hyper-parameter tuning (inner) with **3-fold** crossvalidation
  - Estimation of future performance (outer) with **5-fold** crossvalidation (instead of train/test)
  - This structure is called nested crossvalidation

```
# Define estimator with HPO:
# Search space + inner evaluation + estimator + scoring

# Search space
param_grid = {
    'max_depth': list(range(2, 16, 2)

}

# Inner evaluation
inner = KFold( n_splits=3 , shuffle=True, random_state=rs)

# Estimator
regr_default = DecisionTreeRegressor(random_state=rs)

# Definition of the 2-step process for HPO
regr_hpo = GridSearchCV(regr_default,
                        param_grid,
                        scoring='neg_mean_squared_error',
                        cv=inner,
                        n_jobs=-1, verbose=1)

# Outer evaluation: 5-fold cross-validation
outer = KFold(n_splits=5, shuffle=True, random_state=rs)

# Perform cross-validation and calculate RMSE scores
mse_scores = cross_val_score(regr_hpo, X, y, scoring='neg_mean_squared_error',
                             cv=outer)
rmse_scores = np.sqrt(-mse_scores)

# Average RMSE
avg_rmse = np.mean(rmse_scores)

print("Average Regression Tree RMSE with HPO:", avg_rmse)
```

Inner = 3-fold xval

outer = 5-fold xval

```python
rs = 42
outer = KFold(n_splits=5, shuffle=True, random_state=rs)
inner = KFold(n_splits=3, shuffle=True, random_state=rs)

# Outer loop: 5-fold cross-validation
for train_index, test_index in outer.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    best_rmse = float('inf')
    best_params = None

    for max_depth in param_grid['max_depth']:
          # Initialize the model with current hyperparameters
          model = DecisionTreeRegressor(**{"max_depth": max_depth}, random_state=rs)

          # Perform 3-fold cross-validation for the current hyperparameters
          inner_rmse_scores = []
          for inner_train_index, inner_val_index in inner.split(X_train):
              X_train_train, X_train_val = X_train[inner_train_index], X_train[inner_val_index]
              y_train_train, y_train_val = y_train[inner_train_index], y_train[inner_val_index]

              # Train and evaluate the model
              model.fit(X_train_train, y_train_train)
              y_val_pred = model.predict(X_train_val)
              inner_rmse = np.sqrt(mean_squared_error(y_train_val, y_val_pred))
              inner_rmse_scores.append(inner_rmse)

          # Get the average RMSE for the current hyperparameters
          avg_inner_rmse = np.mean(inner_rmse_scores)

          # Update best parameters if current ones are better
          if avg_inner_rmse < best_rmse:
              best_rmse = avg_inner_rmse
              best_params = {"max_depth": max_depth}

    # Train the best model on the full training set
    best_model = DecisionTreeRegressor(** best_params, random_state=rs)
    best_model.fit(X_train, y_train)

    # Evaluate on the outer test set
    y_test_pred = best_model.predict(X_test)
    outer_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
    rmse_scores.append(outer_rmse)

# Average RMSE across all outer folds
print("Average Regression Tree RMSE with manual HPO:", np.mean(rmse_scores))
```

The previous code is equivalent to this one. The code in this slide is provided for improving your understanding, but it is more efficient to use the code with GridSearchCV and cross_val_score of the previous slide.

"ESTIMATION OF FUTURE PERFORMANCE" AND "TRAINING THE FINAL MODEL"

- "Estimation of Future Performance" and "Training the final model" are different tasks.
- Depending on the use case, we may need one, the other or both.
- Examples where "training the final model" is enough:
  - if our company only asks for the best possible model, we only need to "train the final model". The estimation of future performance would not be required.
  - Or, if we want to send to the competition the best possible model and we don't need an unbiased estimation of future performance, training the final model should be enough.
  - In those two cases, we wouldn't have an unbiased estimation of future performance. We would still have the results of the inner evaluation, which would be a (optimistically) biased estimation of future performance.
- Example where computing "the estimation of future performance" is enough:
  - When doing research and writing papers, "estimation of future performance" is enough, as typically the main purpose of research is to report our results to other researchers, but the final model is most usually, not put into production.