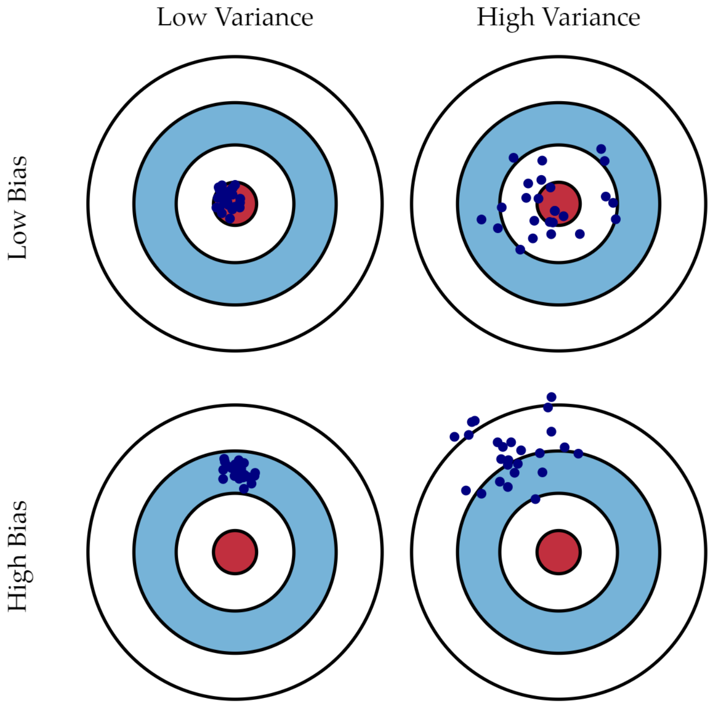
# Bias-Variance-Tradeoff: Crossvalidation & Learning Curves



$$\mathrm{Bias}\big[\hat{f}(x)\big] = \mathrm{E}\big[\hat{f}(x) - f(x)\big]$$

$$\mathrm{Var}\big[\hat{f}(x)\big] = \mathrm{E}\Big[\big(\hat{f}(x) - \mathrm{E}[\hat{f}(x)]\big)^2\Big]$$

**Modul:** Machine Learning
**ML06** **Bias-Variance-Tradeoff**
**Dozent:** Prof. Dr. Christoph Würsch | ICE

NTB
Interstaatliche Hochschule für Technik Buchs
FHO Fachhochschule Ostschweiz

Machine Learning

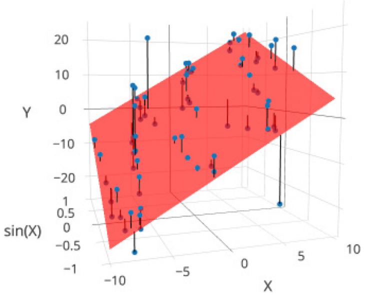
# Educational Objectives

- Know about the *«**folk knowledge** needed to **advance** machine learning **applications**»*
  (Quoting the abstract of Domingos' paper on *«a few useful things to know about machine learning»*)
- Explain the concept of **regularization** with at least **3** distinct **examples**
- Know how to **proceed** (**debug**) when machine learning **algorithms fail** in the first place
- Apply **learning curves** in your daily ML tasks

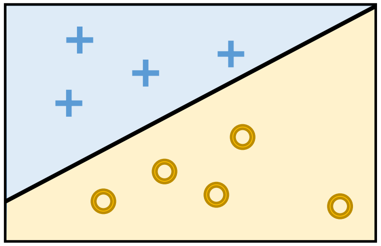
# Taxonomy of Machine Learning

## Supervised Learning

**(labeled data)**



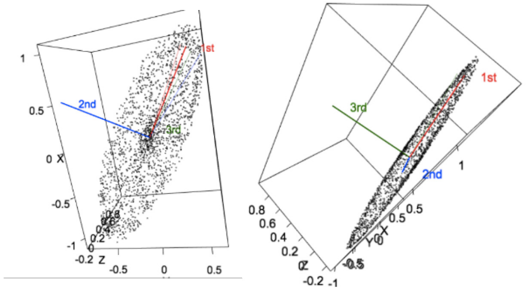
**Regression**



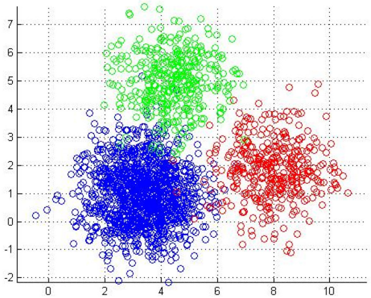
**Classification**

## Unsupervised Learining

**(unlabeled data)**

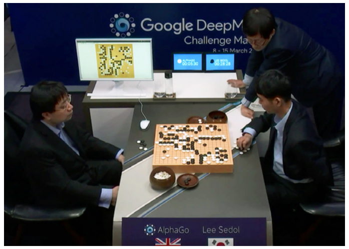


**Dimensionality reduction**



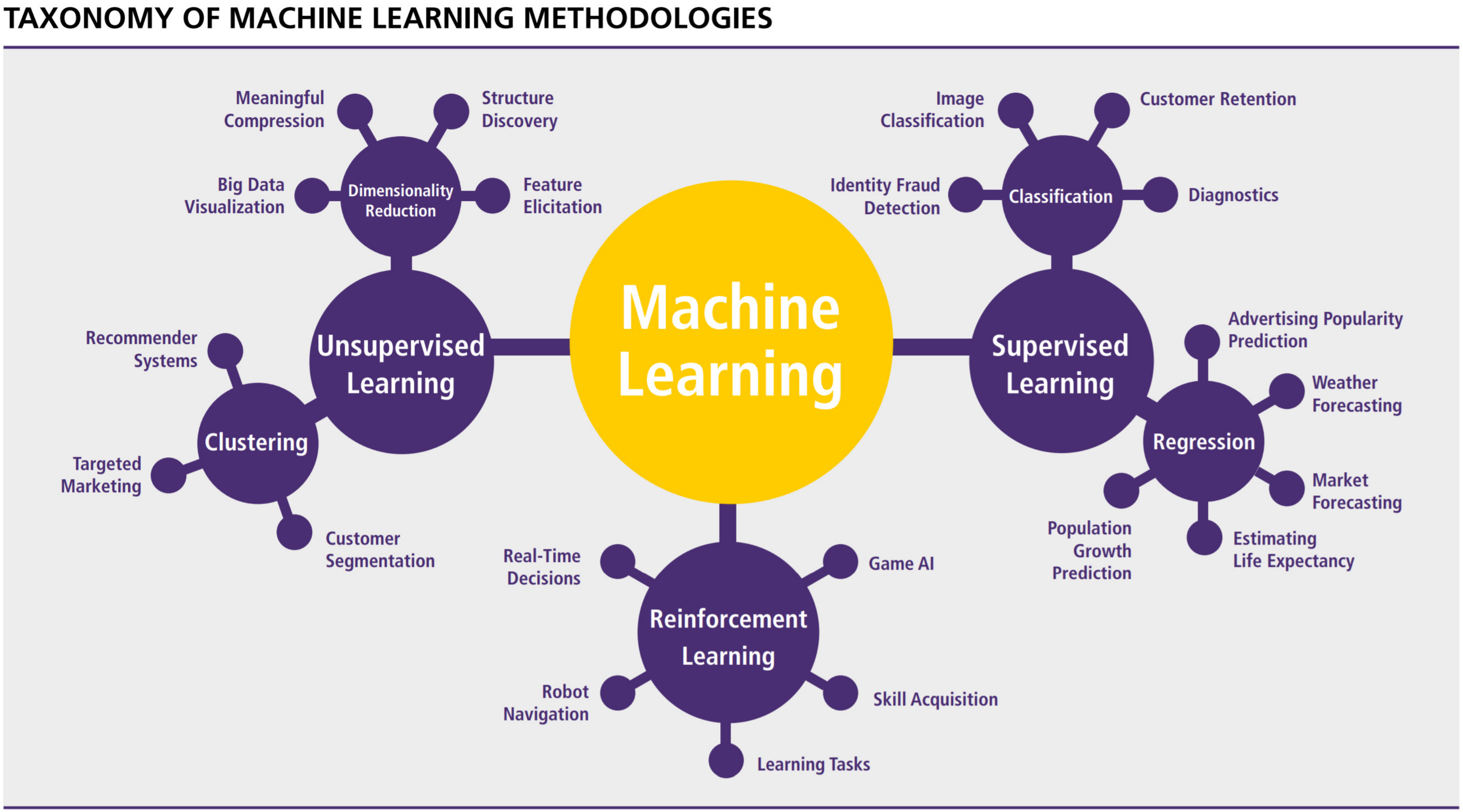
**Clustering**

## Reinforcement Learning



Machine Learning

# Applications

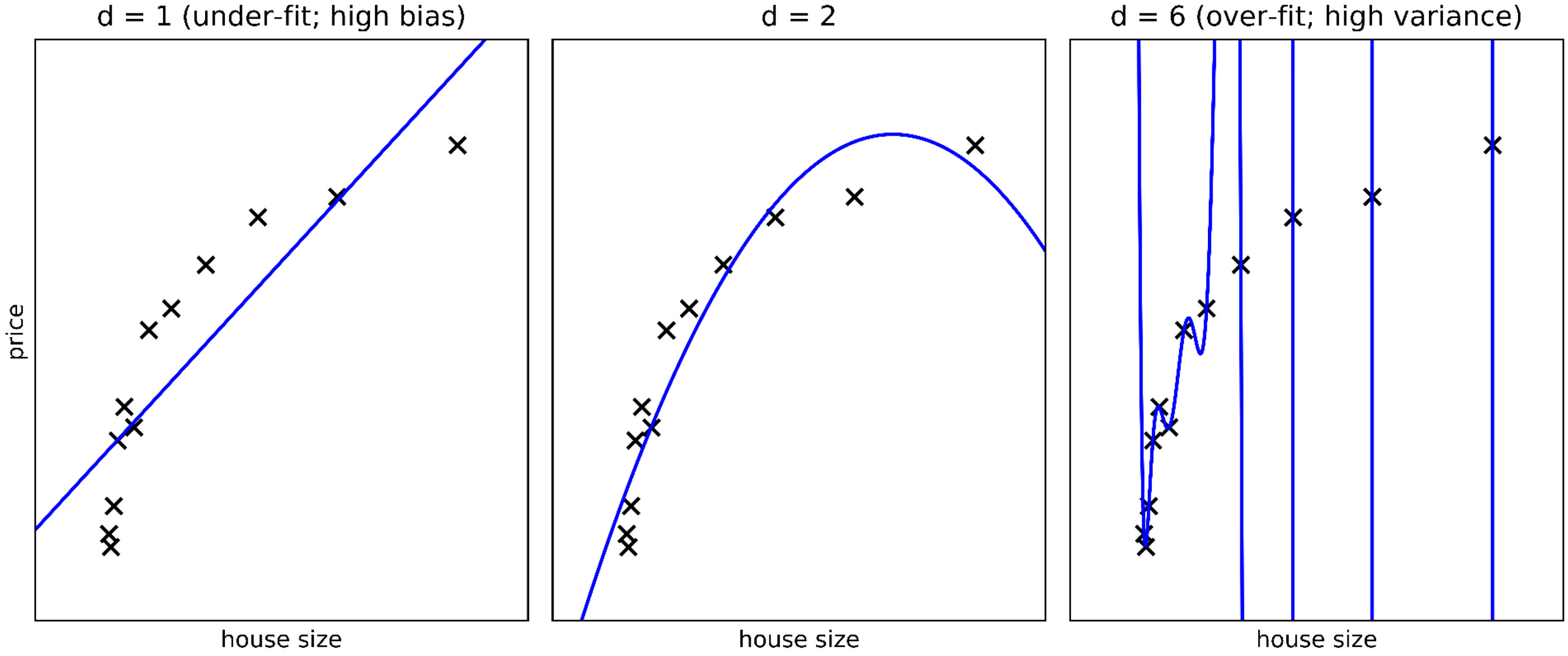


**Source: Jha, V.** An overview of machine learning techniques.
https://www.techleer.com/articles/203-machine-learning-algorithm-backbone-of-emerging-technologies/

# The Bias-Variance-Tradeoff

- **Bias-Variance-Tradeoff**
  - A little bit of PAC learning theroy
  - polynomial regression using pipelines
- **Cross-Validation (cv)**
  - Train / Test / Validation datasets
  - K-fold, stratified k-fold crossvalidation
- **Hyperparameter Tuning** (cross validated)
  - Single parameter: validation curve
  - Many parameters: grid search
- **Optimum Bias-Variance-Tradeoff**
  - **Learning Curves**
  - Reduction of model complexity
  - **Regularization** of linear models: **Ridge** und **Lasso**

# Which Polynomial fits the data best?



plot_bias_variance.ipynb

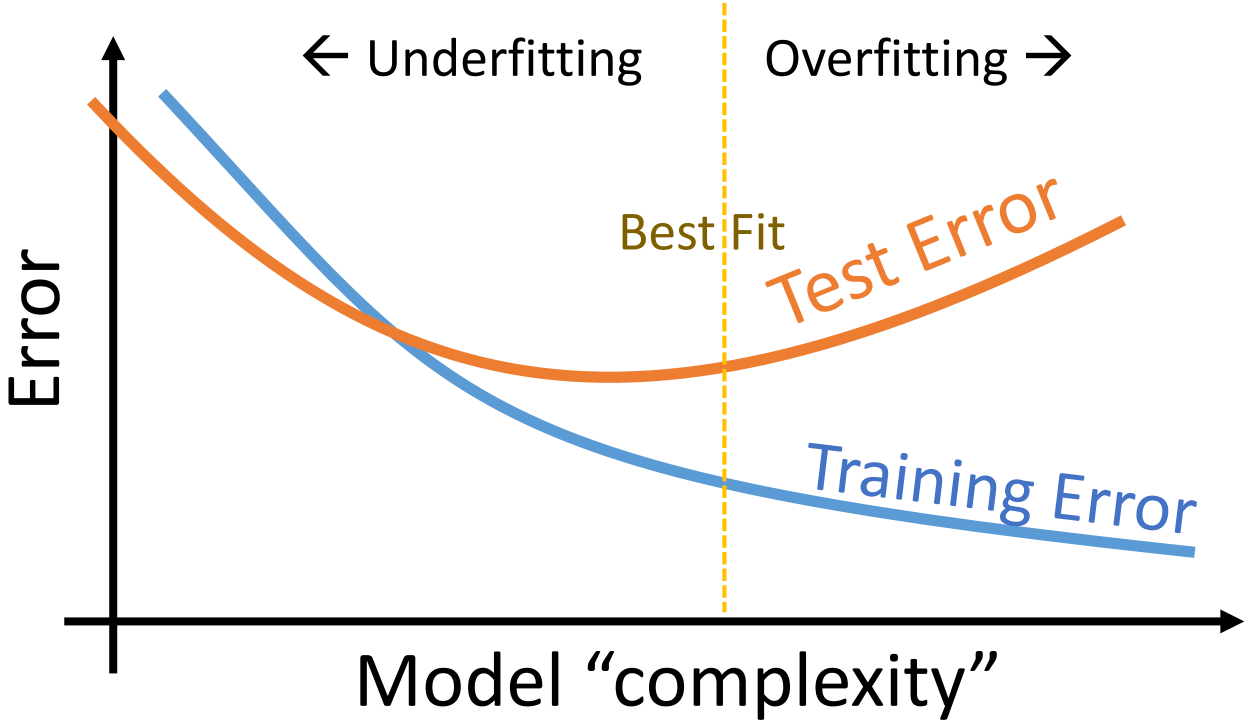
# Polynomial Regression

- Using a pipeline → a fantastic API

```python
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

model = make_pipeline(PolynomialFeatures(degree),
LinearRegression())
model.fit(x[:, np.newaxis], y)

ax.plot(x_test, model.predict(x_test[:, np.newaxis]), '-b')
```

# Validation Curves: Training vs .Test Error

- **Used to tune the model complexity (hyper parameters)**



(e.g. number of features, parameters, neighbors, order of the polynomial)

# Bias-Variance-Decomposition

- We can write the relationship between predictor variables X and the response Y as variables:

$$Y = f(X) + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$$

- Then the expected value of the quadratic error can be written as :

$$\mathrm{SE}(x) = E\left[(Y - \hat{f}(x))^2\right]$$

- After some transformations you get:

$$\mathrm{SE}(x) = \underbrace{\left(E[\hat{f}(x)] - f(x)\right)^2}_{\text{Bias}^2} + \underbrace{E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right]}_{\text{Variance}} + \underbrace{\sigma_e^2}_{\text{irreducibile error}}$$

(optional)

# Proof:

- Expected value of the square of a random variable X:

$$\mathrm{E}[X^2] = \mathrm{Var}[X] + \mathrm{E}[X]^2$$

$$\mathrm{E}[y] = \mathrm{E}[f + \epsilon] = \mathrm{E}[f] = f$$

- With this:

$$\begin{aligned}
\mathrm{E}\big[(y - \hat{f})^2\big] &= \mathrm{E}[y^2 + \hat{f}^2 - 2y\hat{f}] = \mathrm{E}[y^2] + \mathrm{E}[\hat{f}^2] - \mathrm{E}[2y\hat{f}] \\
&= \mathrm{Var}[y] + \mathrm{E}[y]^2 + \mathrm{Var}[\hat{f}] + \mathrm{E}[\hat{f}]^2 - 2f\mathrm{E}[\hat{f}] \\
&= \mathrm{Var}[y] + \mathrm{Var}[\hat{f}] + (f - \mathrm{E}[\hat{f}])^2 \\
&= \mathrm{Var}[y] + \mathrm{Var}[\hat{f}] + \mathrm{E}[f - \hat{f}]^2 \\
&= \sigma^2 + \mathrm{Var}[\hat{f}] + \mathrm{Bias}[\hat{f}]^2.
\end{aligned}$$

# Empirical Risk Minimization (ERM)

- The definition of **Probably Approximately Correct** learnability contains two approximation parameters. The **accuracy parameter** $\varepsilon$ determines how far the output classifier can be from the optimal one (this corresponds to the "*approximately correct*"), and a *confidence parameter* $\delta$ indicating how likely the classifier is to meet that accuracy requirement (corresponds to the "*probably*" part of PAC).

**True Error**
(Generalization error, always unknown)

$$L_{\mathcal{D},f}(h)$$

$f$: true, unknown labelling function
$\mathcal{D}$: full distribution over data space

**Empirical risk minimization**

$$\mathrm{ERM}_{\mathcal{H}}(S) = \operatorname*{argmin}_{h\in\mathcal{H}} L_S(h)$$

Selected Hypothesis Space (our set of learners)

our training samples

**Training Error**
(empirical risk)

Shai Shalev-Shwartz, Shai Ben-David: UNDERSTANDING MACHINE LEARNING
From Theory to Algorithms, Cambridge University Press

# The No-Free-Lunch  theorem (PAC theory)

- **There is no universal learner**: no learner can succeed on all learning tasks, as formalized in the following theorem:

**Theorem: (No-Free-Lunch)** *Let $A$ be any learning algorithm for the task of binary classication with respect to the $0-1$ loss over a domain $\mathcal{X}$. Let $m$ be any number smaller than $|\mathcal{X}|/2$, representing a training set size. Then, there exists a distribution $\mathcal{D}$ over $\mathcal{X}$ such that:*

1. *There exists a function $f : \mathcal{X} \longrightarrow \{0, 1\}$ with with $L_{\mathcal{D}}(f) = 0$.*
2. *With probability of at least $1/7$ over the choice of $S \sim \mathcal{D}^m$ we have that $L_{\mathcal{D}}(A(S)) \geq 1/8$ .*

**This theorem states that for every learner, there exists a task on which it fails, even though that task can be successfully learned by another learner.**

# Lower Bounds for the numeber of samples $m$

- For a sufficiently large $m$, the ERM rule over a finite hypothesis class $\mathcal{H}$ will be **probably** (with condfidence 1-$\delta$) **approximately** (up to an error of $\varepsilon$ ) **correct**.

**Theorem: (PAC)** *Let $\mathcal{H}$ be a finite hypothesis class. Let $\delta \in (0,1)$ and $\varepsilon > 0$, and let $m$ be an integer that satisfies*

$$m \geq \frac{\log(|\mathcal{H}|/\delta)}{\varepsilon}$$

*Then, for any labeling function $f$ and for any distribution $\mathcal{D}$ for which the realizability assumption holds (that is, for some $h \in \mathcal{H}, L_{(D,f)}(h) = 0$), with probability of at least $1 - \delta$ over the choice of an i.i.d. sample $S$ of size $m$, we have that for every* ERM *hypothesis $h(S)$ it holds that*

$$L_{(D,f)}(h_S) \leq \varepsilon$$

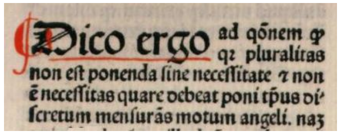
# Ockham’s Rasiermesser

## Lex Parsimoniae

Occam's razor (also Ockham's razor or Ocham's razor; Latin: lex parsimoniae "law of simlicity") is the problem-solving principle that the simplest solution tends to be the correct one.

When presented with competing hypotheses to solve a problem, one should select the solution with the fewest assumptions.

**Ockham's razor can be interpreted in Bayesian terms: in the choice of the prior probabilities of hypotheses, using scientific experience to judge that simpler hypotheses are more likely to be correct; as a consequence of the fact that a hypothesis with fewer adjustable parameters will automatically have an enhanced posterior probability, due to the fact that the predictions it makes are sharp; and in the choice of parsimonious empirical models.**



Dico ergo ad q̃onem ꝙ qꝛ pluralitas
non eſt ponenda ſine neceſſitate ⁊ non
ē neceſſitas quare debeat poni tp̃us di-
ſcretum menſurās motum angeli. naȝ

William of Ockham
(1288–1347)

[1] https://de.wikipedia.org/wiki/Ockhams_Rasiermesser
[2] Information Theory, Inference, and Learning Algorithms“. Archived (PDF) from the original on 15 September 2012.
[3] Jefferys, William H.; Berger, James O. (1991). "Ockham's Razor and Bayesian Statistics« . *American Scientist*. **80** (1): 64–72. JSTOR 29774559.

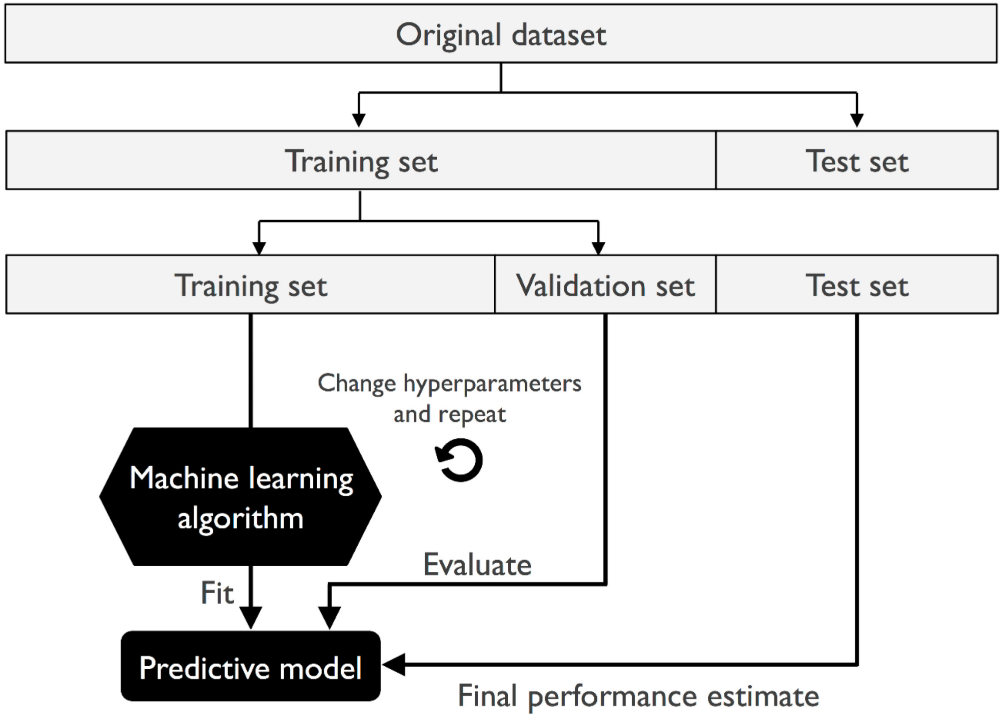
# Estimation of the accuracy on unknown data

- **Generalization** is decisive for a model: How well does the model work on unknown data?
  - A model that is too simple can lead to an underadjustment (large bias).
  - If the model is too complex for the underlying training data, over-fitting (high variance) may occur.
- In order to find an acceptable **bias-variance compromise**, we need to carefully evaluate the model.
- In this section, you will learn about two useful methods for reliable estimations of the generalization error, which provide information on how well the model works with unknown data:
- **2-fold and k-fold cross-validation**

# 1. Cross-Validation

## The outer and the inner loop

- Hyperparameter tuning
- Validation Curves
- Final Performance Evaluation



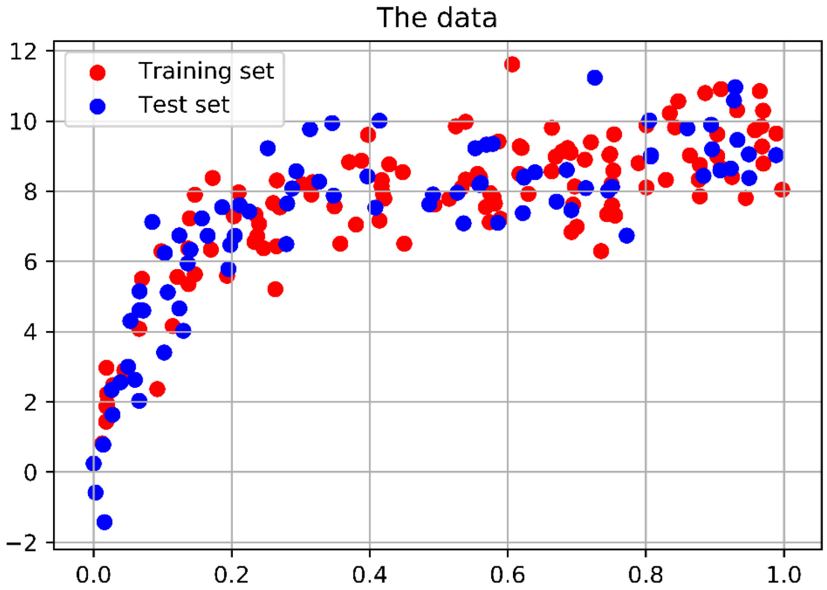
# Split Training and Test Data !

- **Learning the parameters of a prediction function and testing it on the same data is a methodological mistake:**
- a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data (huge generalization error).
- This situation is called **overfitting**.
- To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a test set `X_test, y_test`.

# train_test_split

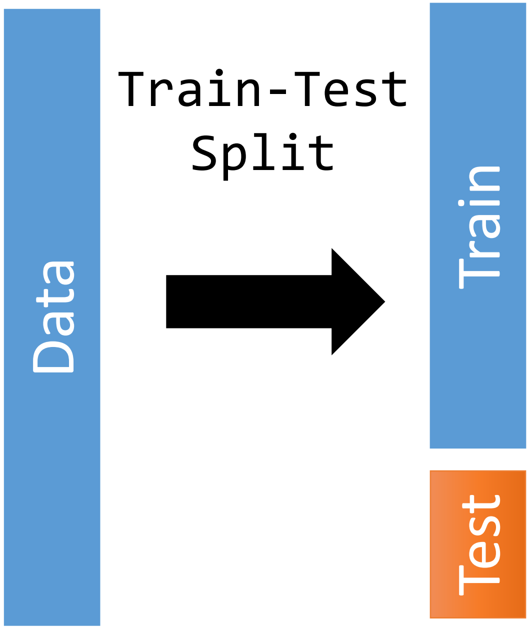
- **Absolute must**: Separate into training and test data:

```
from sklearn.model_selection import train_test_split

# split into training, validation, and testing sets.
test_size = 0.4 x_train, x_test, y_train, y_test =
     train_test_split(x, y, test_size=test_size)
```



# 2-fold crossvalidation = *Hold-out-method*

- Data is divided into a **training data set** and a **test data set**. The former is used to train the model, the latter to evaluate performance.
- **Training Data**: used for fitting the model.
- **Test Data**: Checking the generalization error.
- How to split?
  - temporal, random, geographical,...
  - Depends on the application (usually random)
- Which size for the test data quantity? (90%-10%)
  - Larger training data suitable for more complex models
  - Larger test data:  Better estimation of the generalization error possible
  - Typically between 75%-25% and 90%-10%



You can only use the test dataset once after deciding on the model.

# Model Selection: training-validation-test

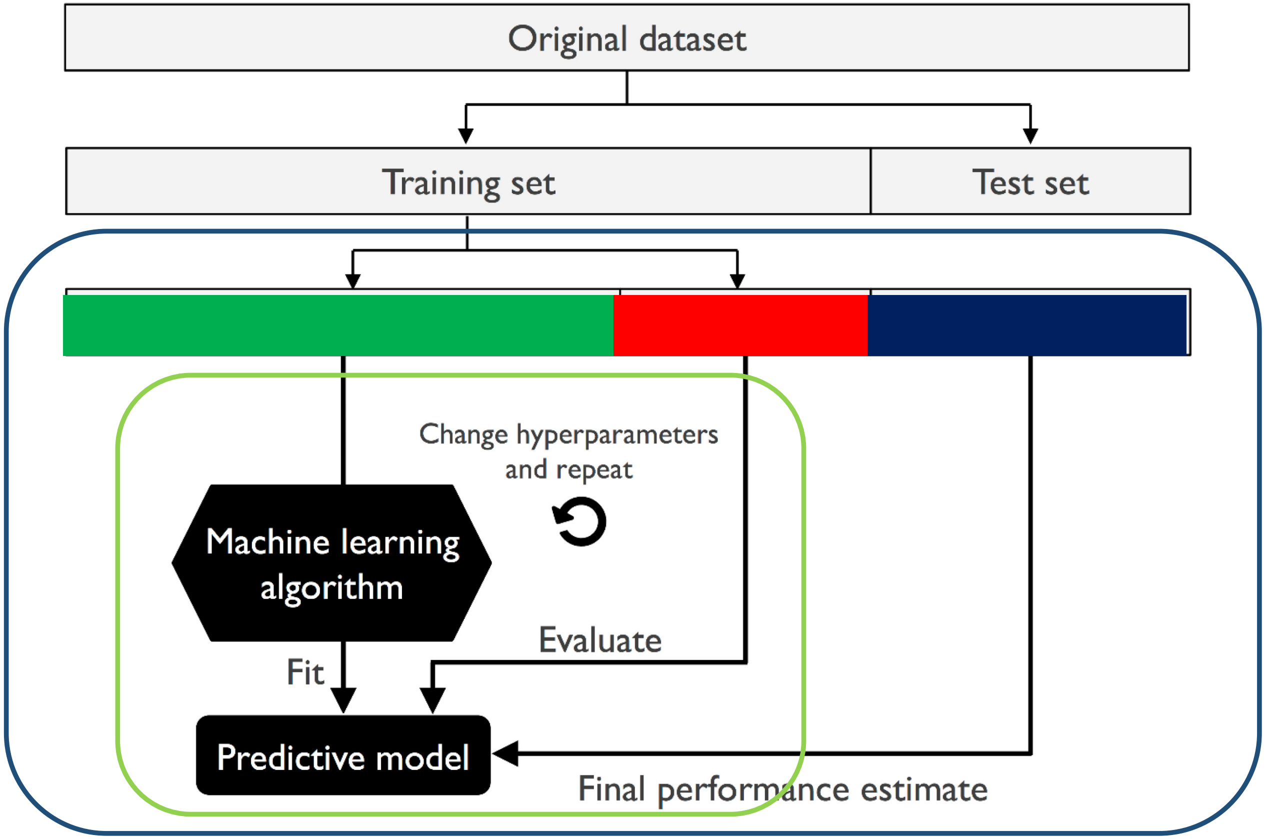
- **Model selection** = Adjustment and comparison of various parameters to improve the predictive power of unknown data, e.g. selection of optimal values for the **hyperparameters** (k for kNN) of a classification task.
- However, if we use the same test data over and over again when selecting the model, they become part of the training data → overadaptation of the model
- A better way to use dual cross-validation for model selection is to divide the data into **three parts**: a **training data set**, a **validation data set**, and a **test data** set.
- The **training data set** is used to *fit the different models* **`(.fit)`** - and the performance of a model on the **validation data** is then used for model selection (e.g. hyperparameter tuning using **`GridSearch`**).

# Validation: Hyperparameters in `sklearn`

- Hyper-parameters are parameters that are not directly learnt within estimators. In **`scikit-learn`** they are passed as arguments to the constructor of the estimator classes.
- Typical examples include **`C, kernel`** and **`gamma`** for Support Vector Classifier, **`alpha`** for Lasso, etc.
- It is possible and recommended to search the hyper-parameter space for the best cross validation score.
- Any parameter provided when constructing an estimator may be optimized in this manner. Specifically, to find the names and current values for all parameters for a given estimator, use

```
estimator.get_params()
```

# Model Selection: training-validation-test

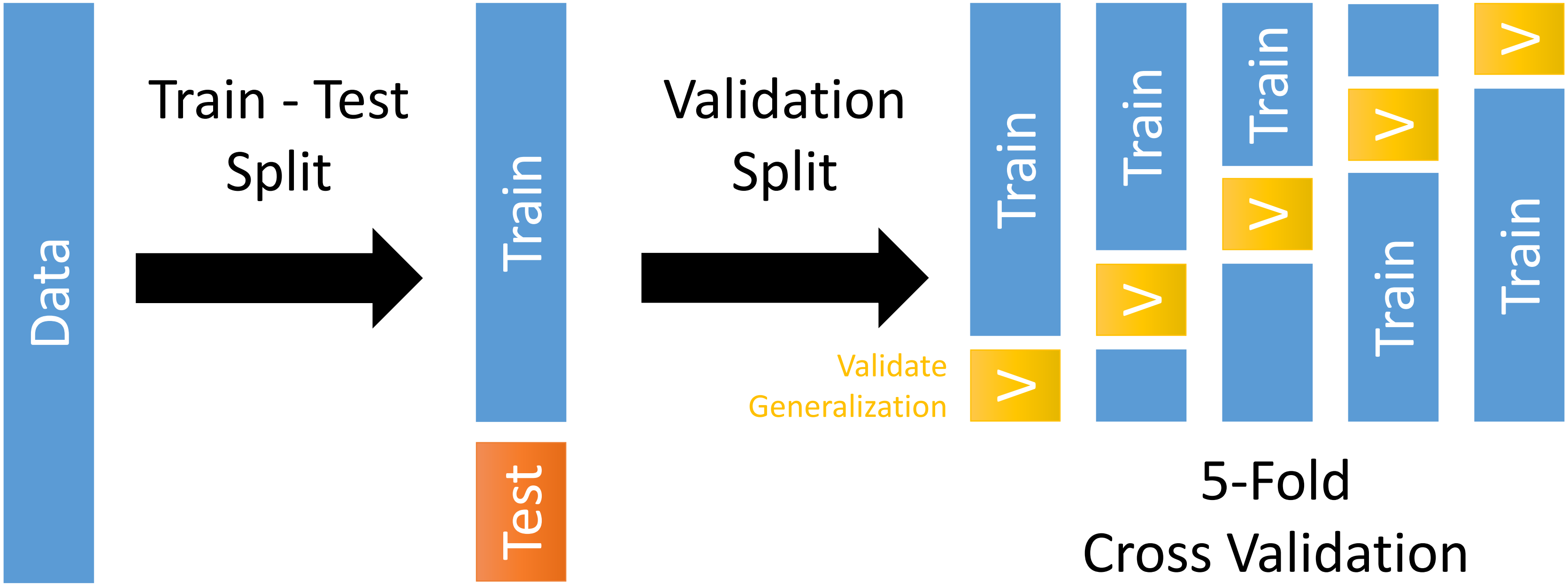

Original dataset
Training set
Test set
Change hyperparameters and repeat
Machine learning algorithm
Evaluate
Fit
Predictive model
Final performance estimate

- **Disadvantage**: the assessment of the performance can strongly depend on how the training data is divided into the subsets **training data** and **validation data**.

# k-fold cross-validated model adjustment

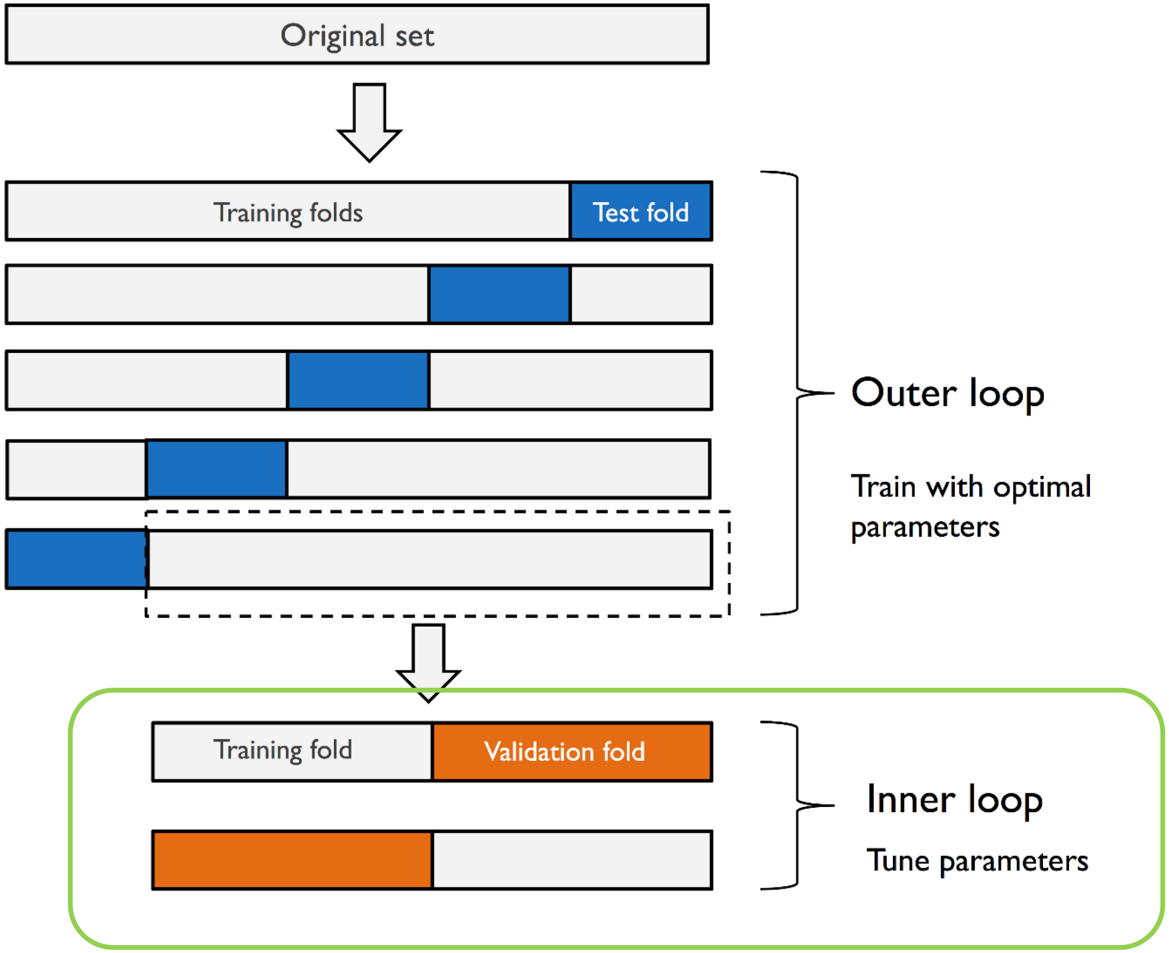
- In k-fold cross validation, the training data is divided into **k subsets** (without substitution). Then k-1 subsets are used to train the model and a subset is used for testing.
- This procedure is **repeated** k times so that we get k models and k estimates of performance.
- The k-fold cross-validation is used in model tuning to find the optimal values of the hyper-parameters that provide an acceptable generalization capability.
- Once we have found acceptable values for the **hyper-parameters**, we can re-train the model with the **entire training data set**.
- This provides more training data to the learning algorithm and generally leads to a more accurate and stable model.

# k-fold cross-validated model adjustment



*Cross validation simulates multiple train test-splits on the training data.*

# Inner and outer loop



# 10-fold crossvalidation

- The performance measure reported by *k*-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but does not waste too much data.
- The simplest way to use cross-validation is to call the **`cross_val_score`** helper function on the estimator and the dataset.
- **Ron Kohavi has shown with practical data that 10-fold cross-validation is the best compromise between bias and variance in most cases [1].**

```
from sklearn.model_selection import cross_val_score
from sklearn import metrics

clf = svm.SVC(kernel='linear', C=1)
scores = cross_val_score(clf, iris.data, iris.target,
         scoring= f1_macro', cv=5)
scores array([0.96..., 1.  ..., 0.96..., 0.96..., 1.  ])
```

[1] R. Kohavi et al., *A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection*, International Joint Conference on Artificial Intelligence (IJCAI), 14(12), Seiten 1137–1145, 1995).
[2] https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

# Leave-One out for very small data volumes

- The so-called Leave-**One-Out cross validation (LOO)** is a special case of k-fold cross validation.
- **In LOO, the number of subsets is equal to the number of training data sets (k=n).**
- Only one training object is used for testing in each run.
- This approach is recommended if you are dealing with very small amounts of data.

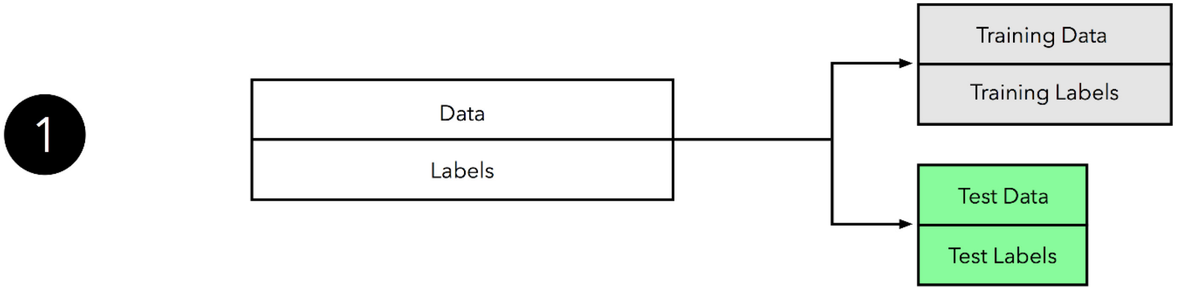
# stratified k-fold cv for unequal classes

- in cases where classes are **unevenly distributed**, **stratified k-fold cross-validation** reduces bias and variance of the estimation
- In stratified cross-validation, care is taken to ensure that the classes are distributed approximately equally in the subsets.
- This ensures that the individual subsets are representative of the class distribution in the training data set.
- This can be illustrated with the **`StratifiedKFold`** generator from **`scikit-learn`**:

# General methodology

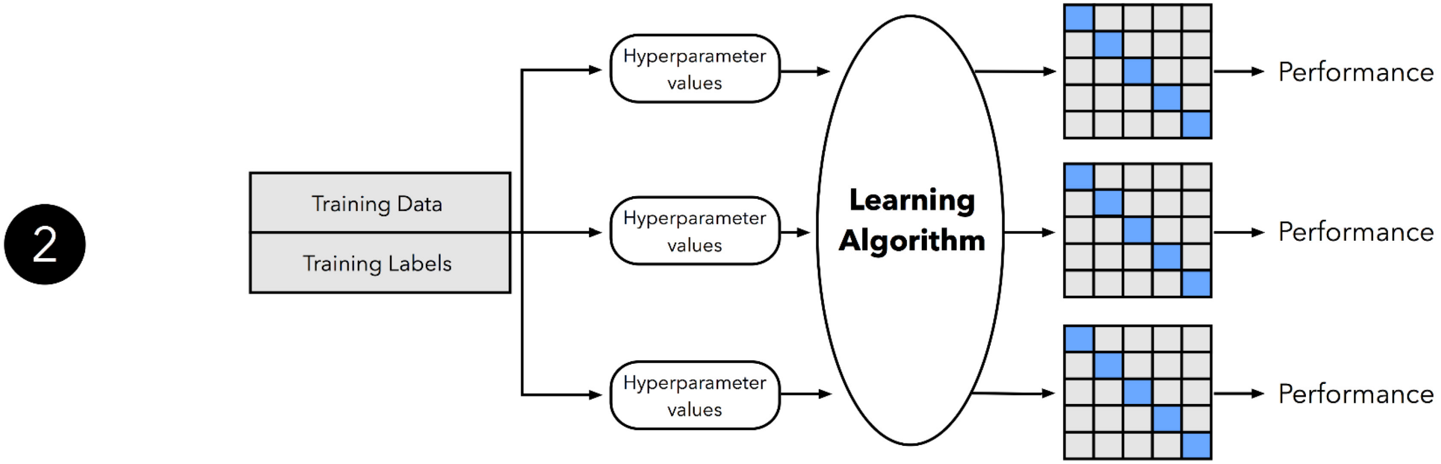
## 1. Step: Split Data in Training and Test

```
sklearn.model_selection.train_test_split
```



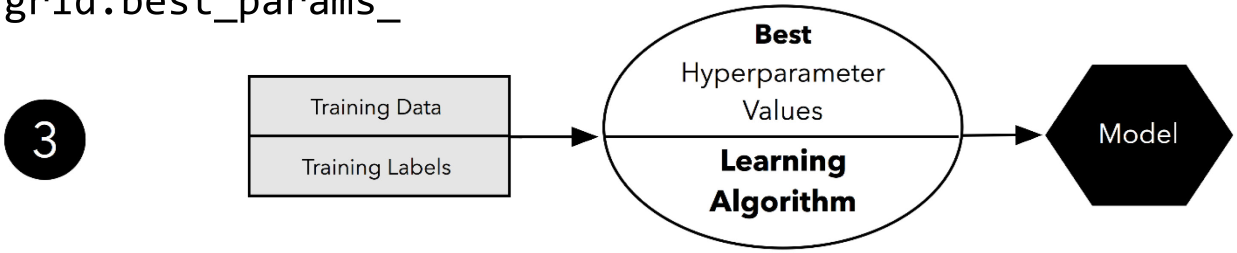
## 2. Step: Hyperparameter Tuning (GridSearch, CV, outer loop)

```
grid = GridSearchCV(pipe_knn, param_grid, cv=10)
sklearn.model_selection.cross_val_score
```

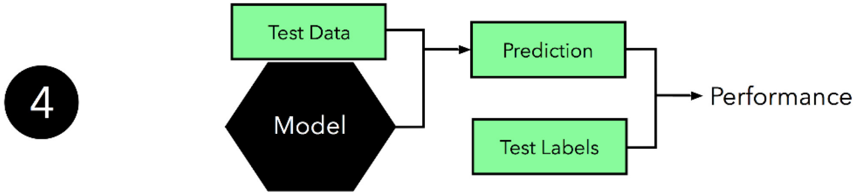


## 3. Step: Selection of the optimum parameter

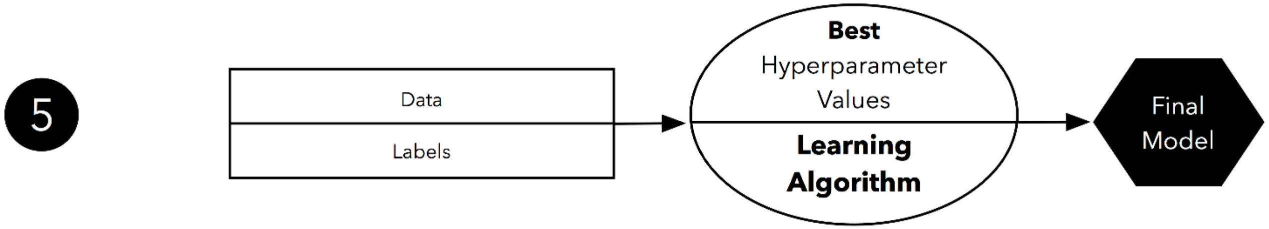
```
grid.fit(X, y);
grid.best_params_
```



## 4. Step: Performance Evaluation on Test Data



## 5. Step: Deployment



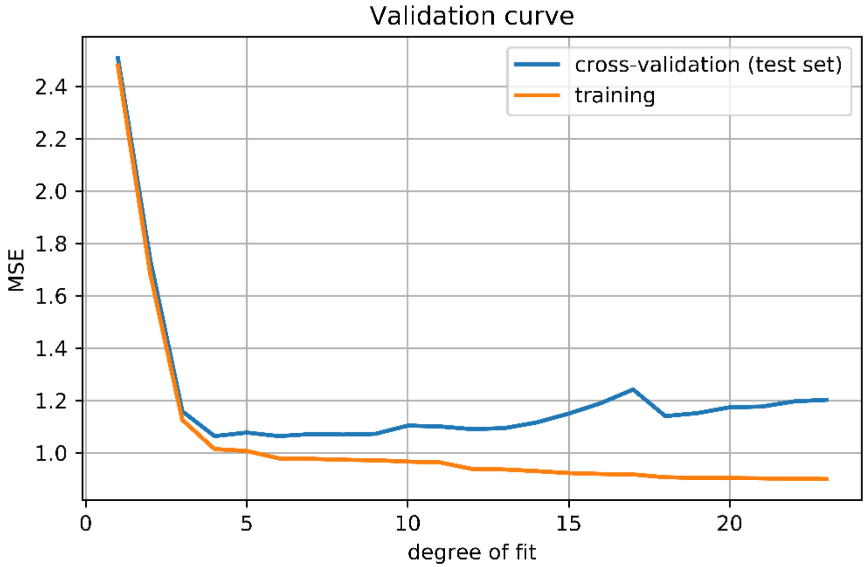
# Single parameter tuning using a validation curve

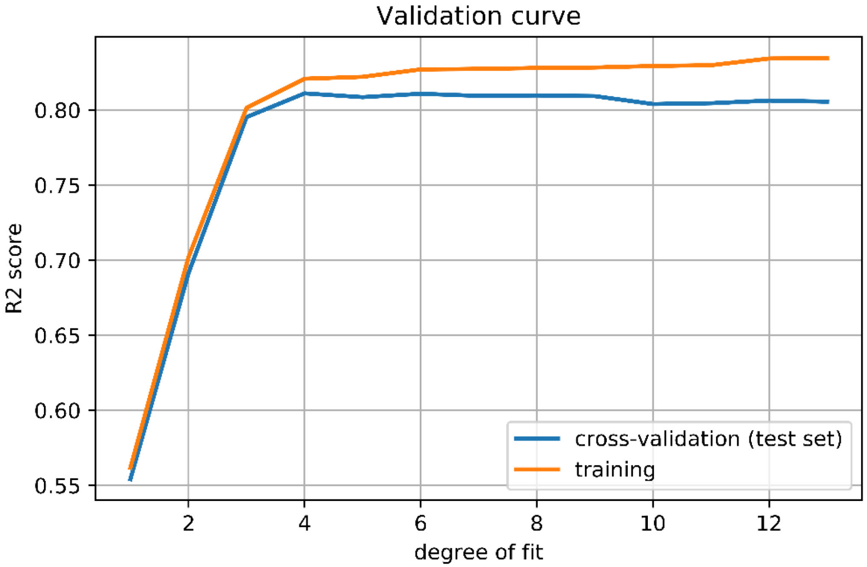
- Determine training and test scores for varying parameter values.
- Compute scores for an estimator with different values of a specified parameter. **This is similar to grid search but with one single parameter**. However, this will also compute training scores and is merely a utility for plotting the results.

```
from sklearn.model_selection import validation_curve

degrees = np.arange(1, 14)
model = make_pipeline(PolynomialFeatures(), LinearRegression())
# The parameter to vary is the "degrees" on the pipeline step #
"polynomialfeatures"

train_scores, validation_scores =
    validation_curve( model, x[:, np.newaxis], y,
    param_name='polynomialfeatures__degree',
    param_range=degrees)
```

# In our example (live)

- hyper parameter: order of the polynomial





Machine Learning

# Scoring Functions

- In scikit-learn:

```
scoringFunctions= ['accuracy', 'adjusted_mutual_info_score',
'adjusted_rand_score', 'average_precision', 'completeness_score',
'explained_variance', 'f1', 'f1_macro', 'f1_micro', 'f1_samples',
'f1_weighted', 'fowlkes_mallows_score', 'homogeneity_score',
'mutual_info_score', 'neg_log_loss', 'neg_mean_absolute_error',
'neg_mean_squared_error', 'neg_mean_squared_log_error',
'neg_median_absolute_error', 'normalized_mutual_info_score',
'precision', 'precision_macro', 'precision_micro',
'precision_samples', 'precision_weighted', 'r2', 'recall',
'recall_macro', 'recall_micro', 'recall_samples',
'recall_weighted', 'roc_auc', 'v_measure_score']
```

See also: https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

## A cross-validated grid search consists of:

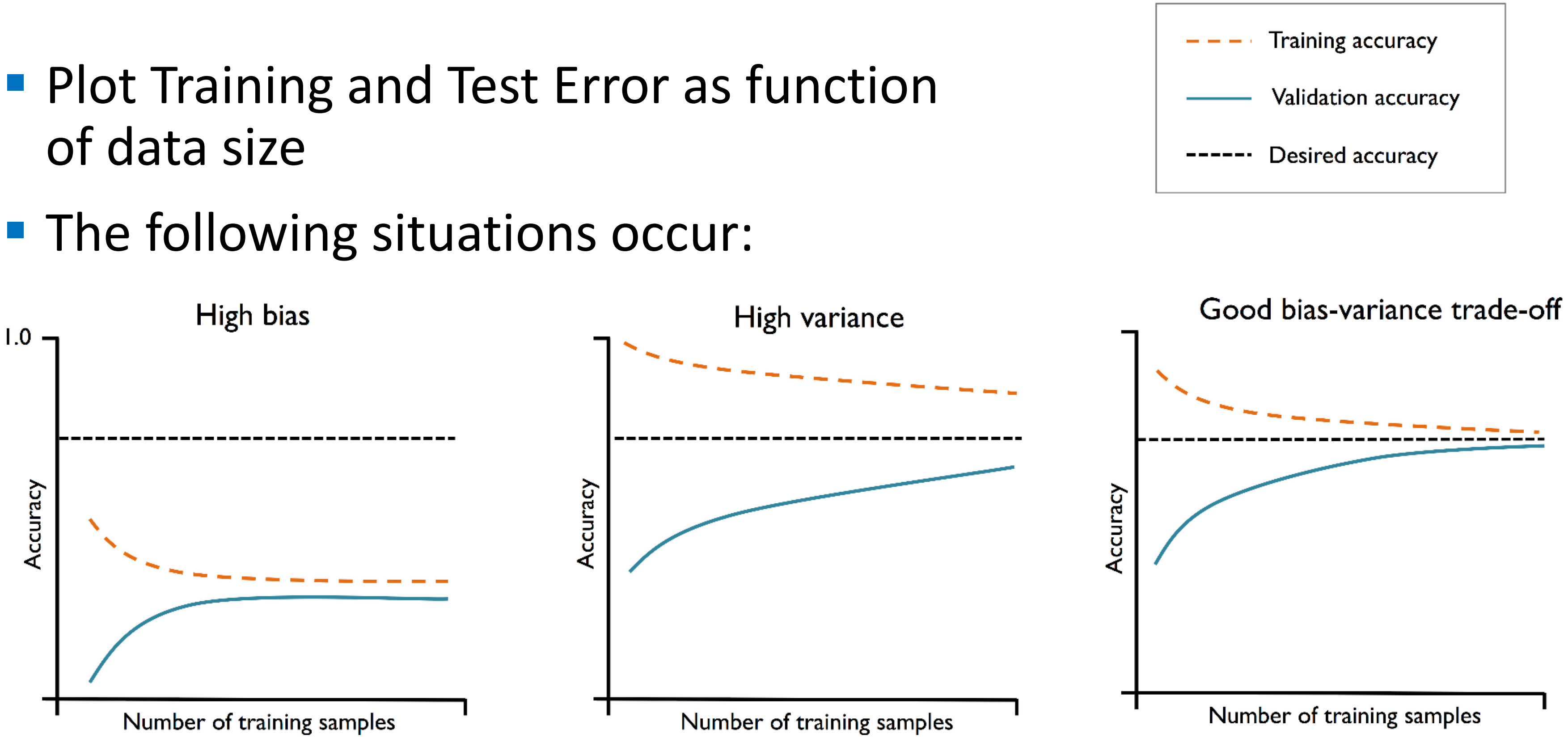
- an **estimator** (regressor or classifier such as `sklearn.svm.SVC()`);
- a **parameter space**;
- a method for searching or **sampling** candidates;
- a **cross-validation scheme**; and
- a **score** function.

```python
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
iris = datasets.load_iris()
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
svc = svm.SVC(gamma="scale")
clf = GridSearchCV(svc, parameters, cv=5)
clf.fit(iris.data, iris.target)
sorted(clf.cv_results_.keys())
```

# 2. Learning Curves

## Checking the Bias-Variance-Tradeoff

- Plot Training and Test Error as function of data size
- The following situations occur:



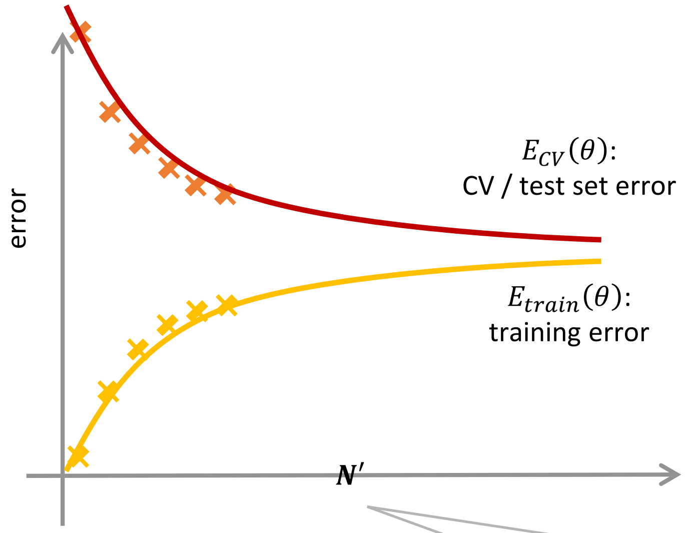
# Diagnosing bias-variance regardless of $\lambda$

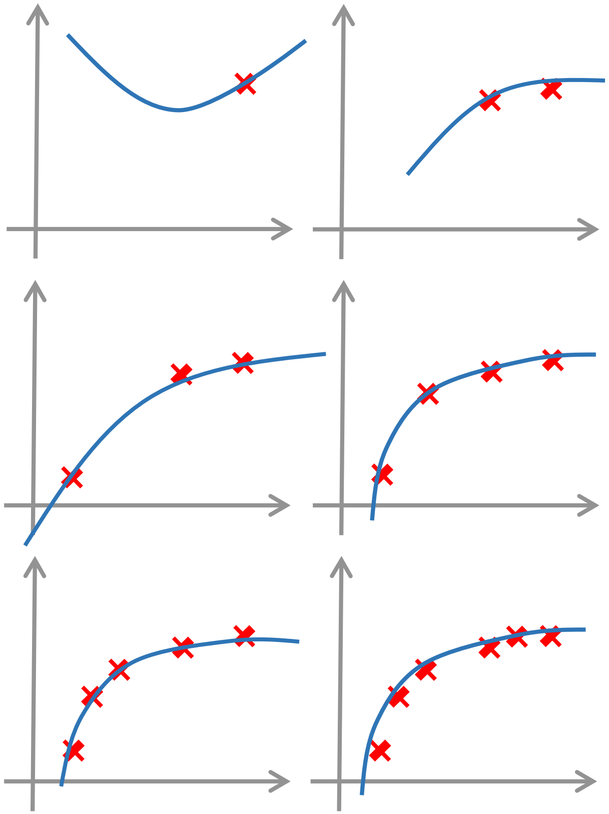
**Example**

- Fitting a quadratic regression function to data: $h(x,\theta) = \theta_0 + \theta_1 x + \theta_2 x^2$

**Building the learning curve**

- Plot training and test (CV) **errors vs. training set size** $N'$
- Vary $N' = 1..N$ (test/validation set stays full size)





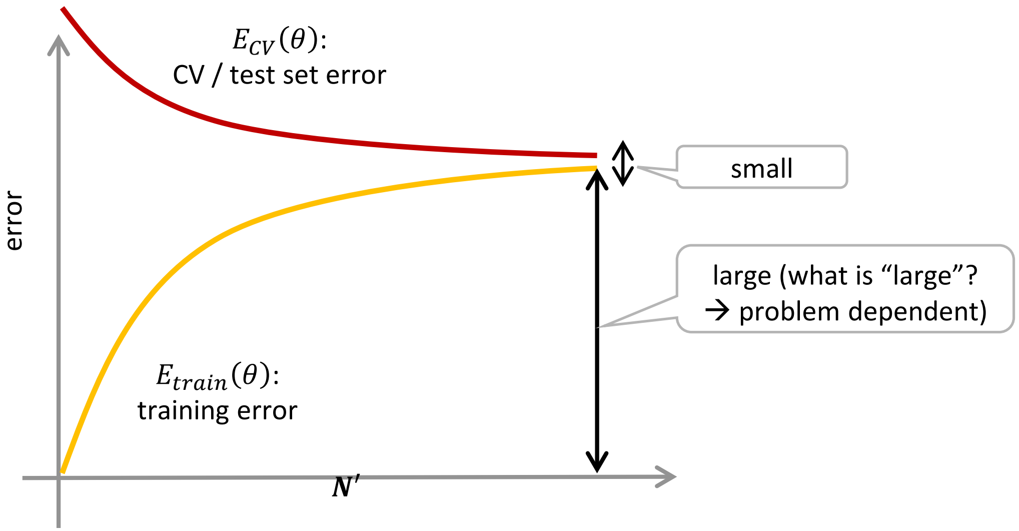
Mind: neither the same as a plot of error/loss per training step in e.g. NN; nor like last slide's plot.

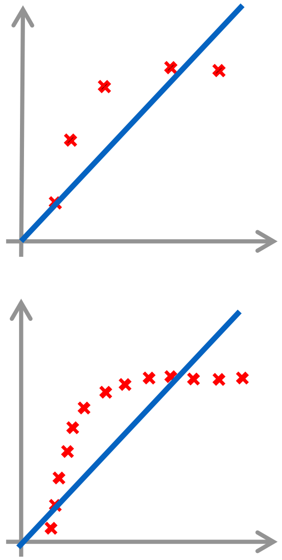
# Learning curves with high bias

I.e., systematic error, underfit

## High bias

- Getting **more** training **data** will **not** (by itself) **help** much
- Learning curve is characterized by **high** training and test **errors**:
  ➔ $E_{train} \approx E_{CV}$ is high even for large amounts of data





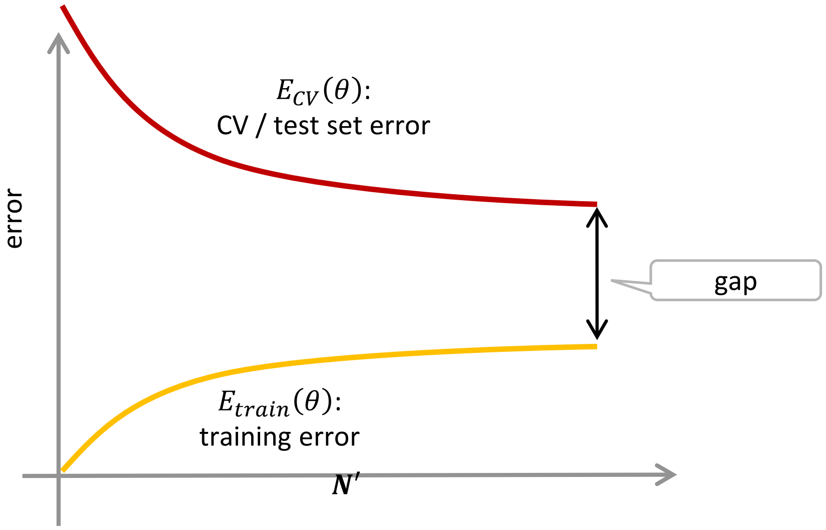
Bias example: If the model is linear, a quadratic function for example can not be learned even with more data.

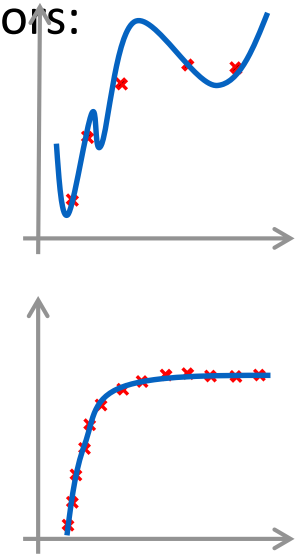
# Learning curves with high variance

I.e., instability, **overfit**

## High variance

- Getting more training **data** is likely to **help**
- Learning curve is **characterized by gap** between the two errors:
  ➔ $E_{train} \ll E_{CV}$ even for the full training set





Variance example: A high variance model like e.g. a polynomial of degree $d = 100$ profits from more data (or regularization).

# Debugging a learning algorithm – revisited

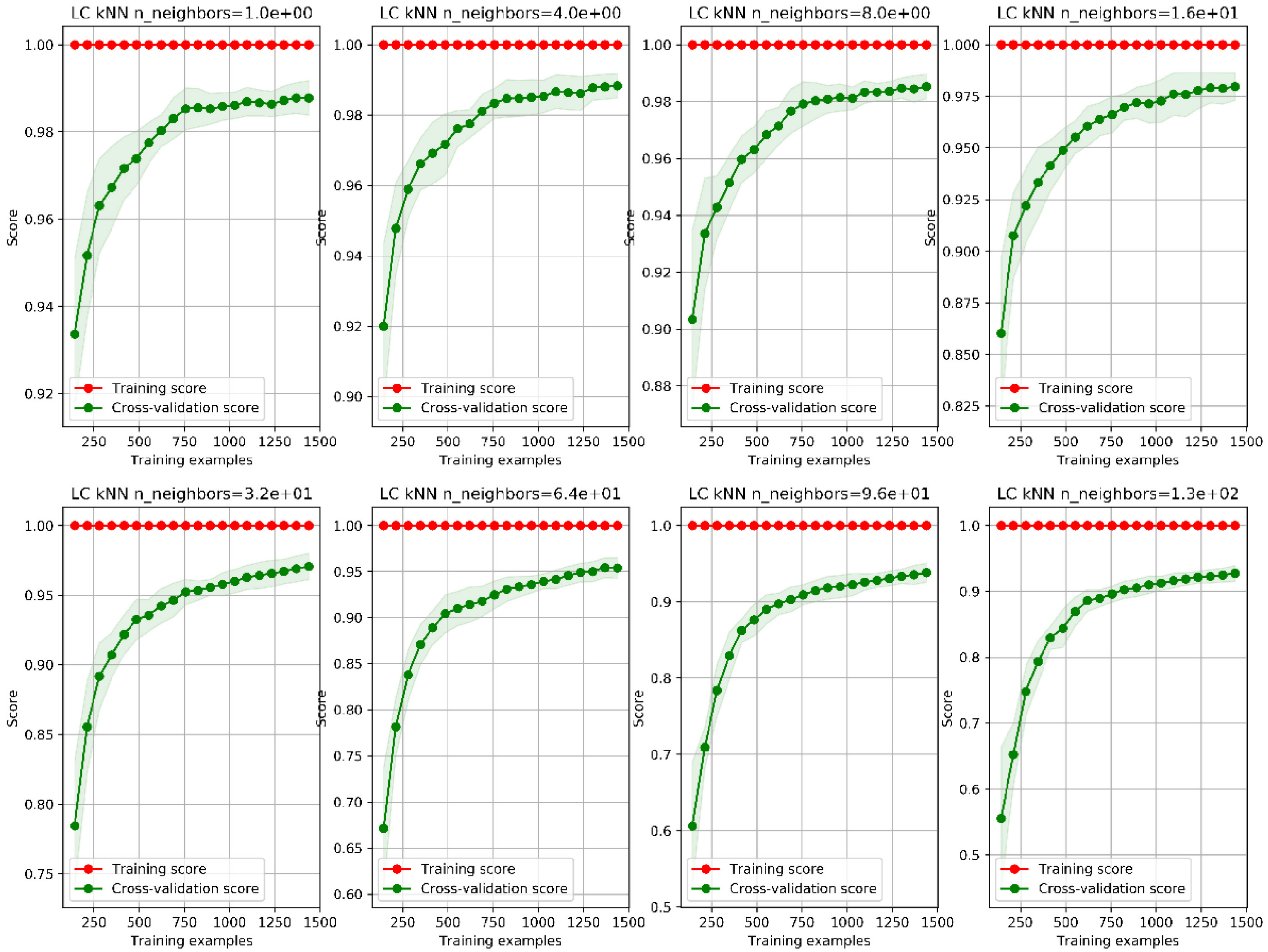
Suppose you are using some algorithm on a given training set

- However, it makes **unacceptably large errors** in its predictions on unseen data

- What should be tried next?
  - Get **more training** examples
    → fixes high variance/overfit
- Try **smaller sets** of **features**
  → fixes high variance/overfit
- Try getting **additional features**
  → fixes high bias/underfit
- Try **adding polynomial** features $x_1, x_2, {x_1}^2, {x_2}^2, \ldots$
  → fixes high bias/underfit
- Try **decreasing** $\lambda$ (i.e., loosing regularization)
  → fixes high bias/underfit
- Try **increasing** $\lambda$ (i.e., intensifying regularization)
  → fixes high variance/overfit

Put also on the list: Building ensembles.
→ fixes high variance, utilizes limited data best.

# Learning Curves



# Bias-Variance-Decomposition

- We can write the relationship between predictor variables X and the response Y as variables:

$$Y = f(X) + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$$

- Then we can write the expected value of the quadratic error as :

$$\mathrm{SE}(x) = E\left[(Y - \hat{f}(x))^2\right]$$

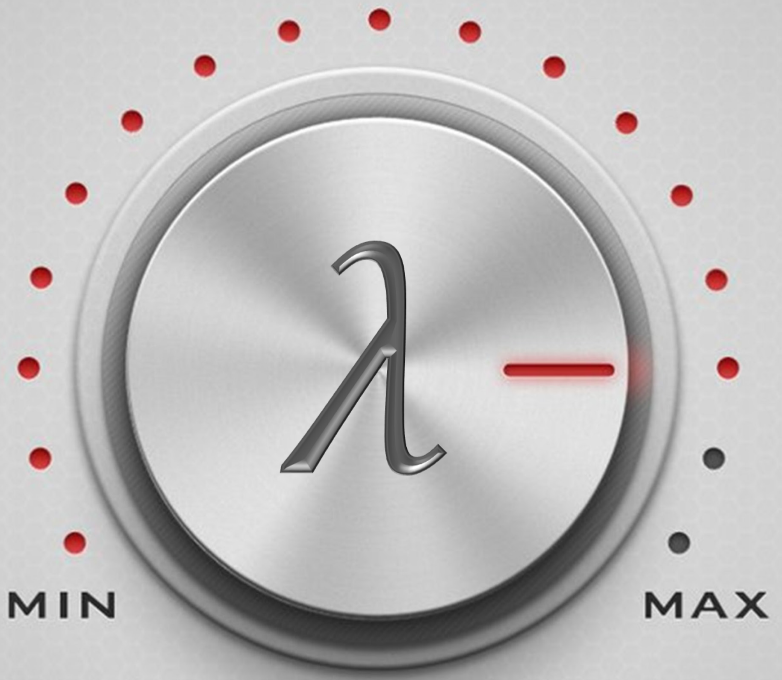
- After some transformations one obtains:

$$\mathrm{SE}(x) = \underbrace{\left(E[\hat{f}(x)] - f(x)\right)^2}_{\text{Bias}^2} + \underbrace{E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right]}_{\text{Variance}} + \underbrace{\sigma_e^2}_{\substack{\text{irreduzierbarer} \\ \text{Fehler}}}$$

# 3.Regularization

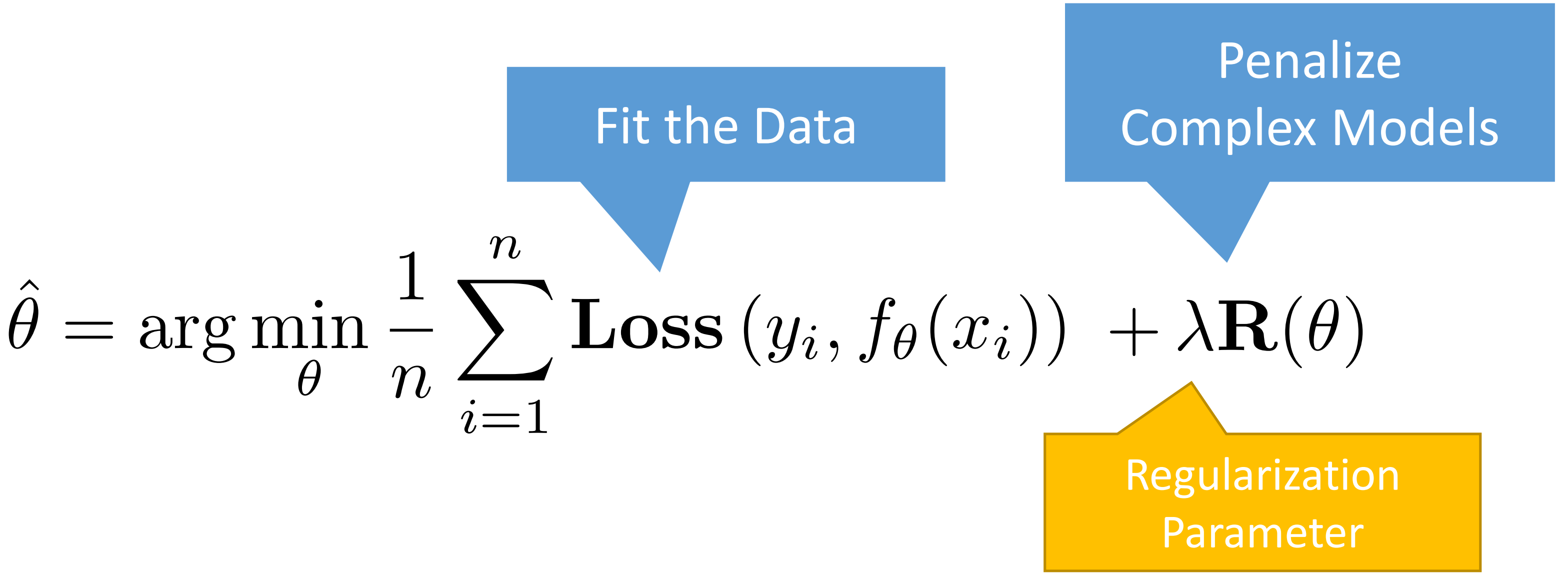
## Parametrically Controlling the *Model Complexity*

- Tradeoff:
  - **Increase bias**
  - **Decrease variance**



# Basic Idea of Regularization

- How should we define **R(θ)**?
- How do we determine $\lambda$?



$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \mathbf{Loss}\left(y_i, f_\theta(x_i)\right) + \lambda \mathbf{R}(\theta)$$

Fit the Data

Penalize Complex Models

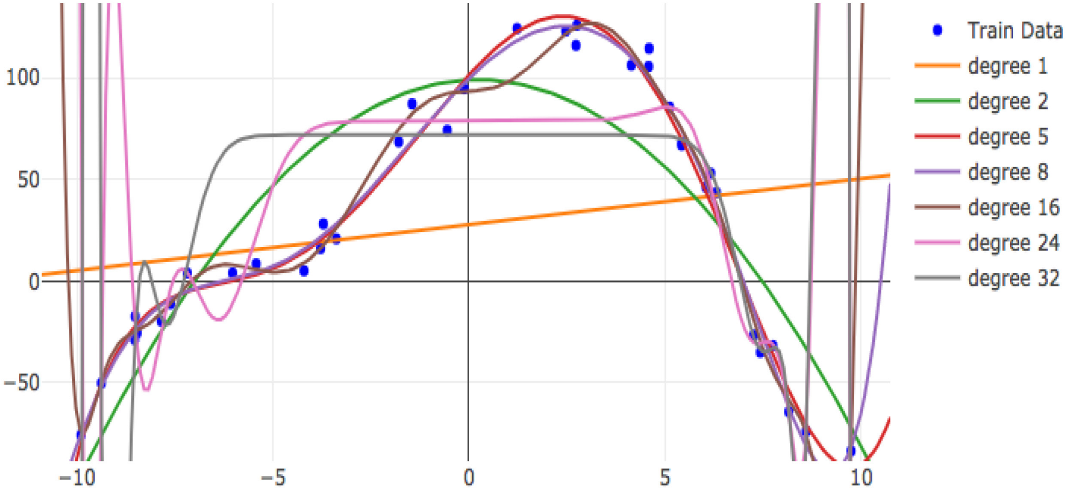
Regularization Parameter

# The Regularization Function R(θ)

**Goal:** *Penalize model complexity*

Recall earlier: $\phi(x) = [x, x^2, x^3, \ldots, x^p]$

- More features → overfitting …
- How can we control overfitting through **θ**
- **Proposal:** set weights = 0 to remove features

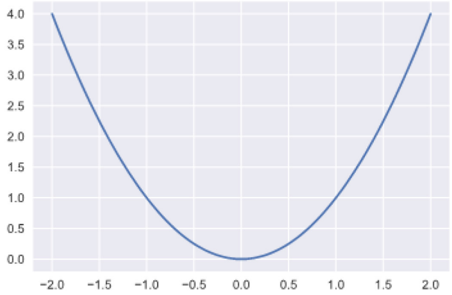


# Common Regularization Functions

- Distributes weight across related features (robust)
- Analytic solution (easy to compute)
- Does not encourage sparsity → small but non-zero weights.

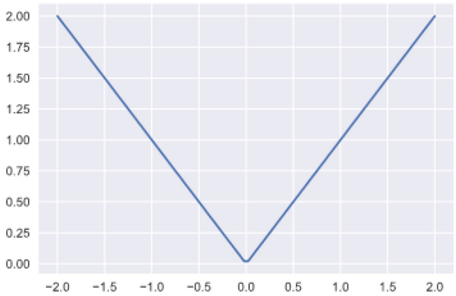
Ridge Regression (L2-Reg)

$$R_{\text{Ridge}}(\theta) = \sum_{i=1}^{d} \theta_i^2$$



LASSO (L1-Reg)

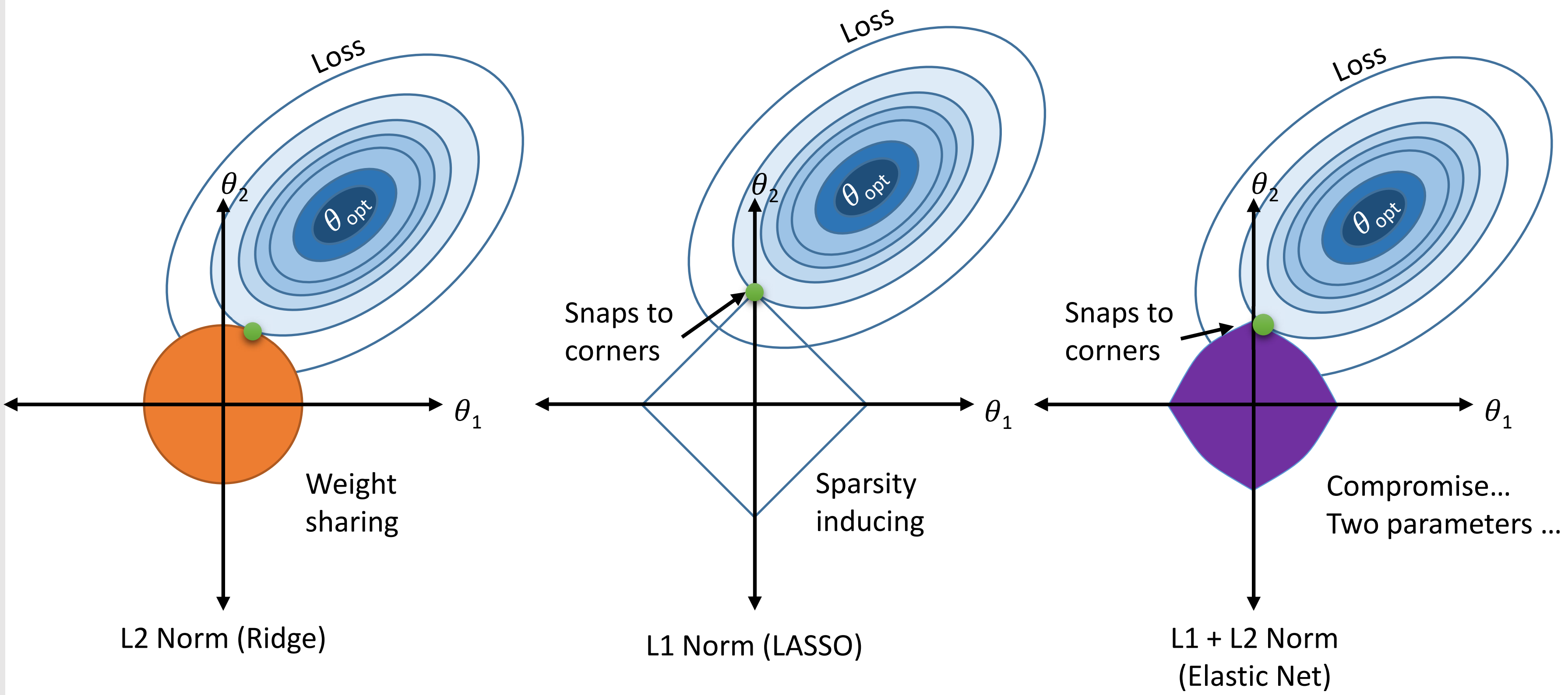
$$R_{\text{Lasso}}(\theta) = \sum_{i=1}^{d} |\theta_i|$$



- ➢ **Encourages sparsity** by setting weights = 0
  - ➢ Used to select informative features
- ➢ Does not have an analytic solution → numerical methods

# Regularization and Norm Balls

- Constrained Optimization:



Weight sharing

L2 Norm (Ridge)

Snaps to corners

Sparsity inducing

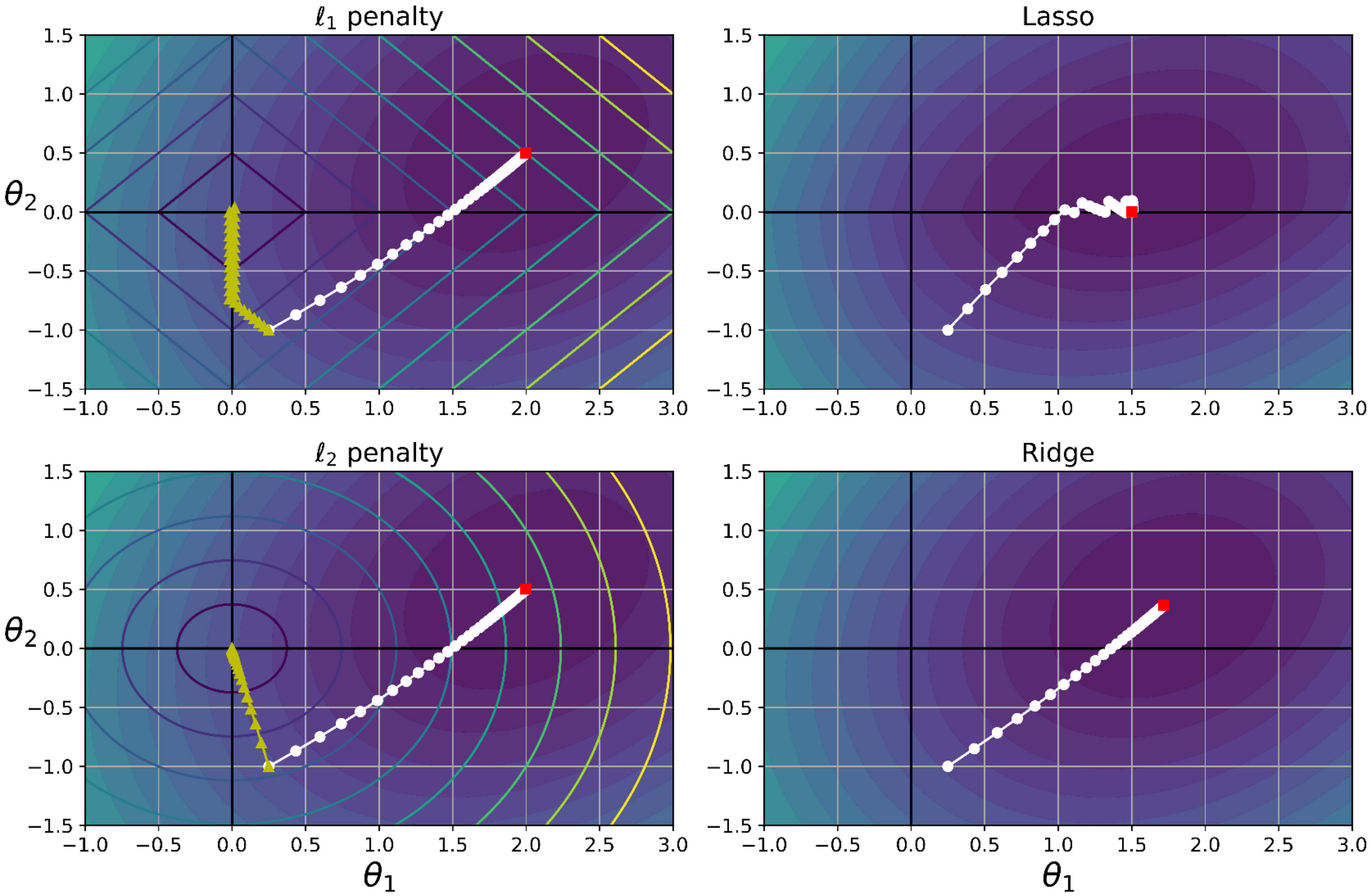
L1 Norm (LASSO)

Snaps to corners

Compromise… Two parameters …

L1 + L2 Norm (Elastic Net)

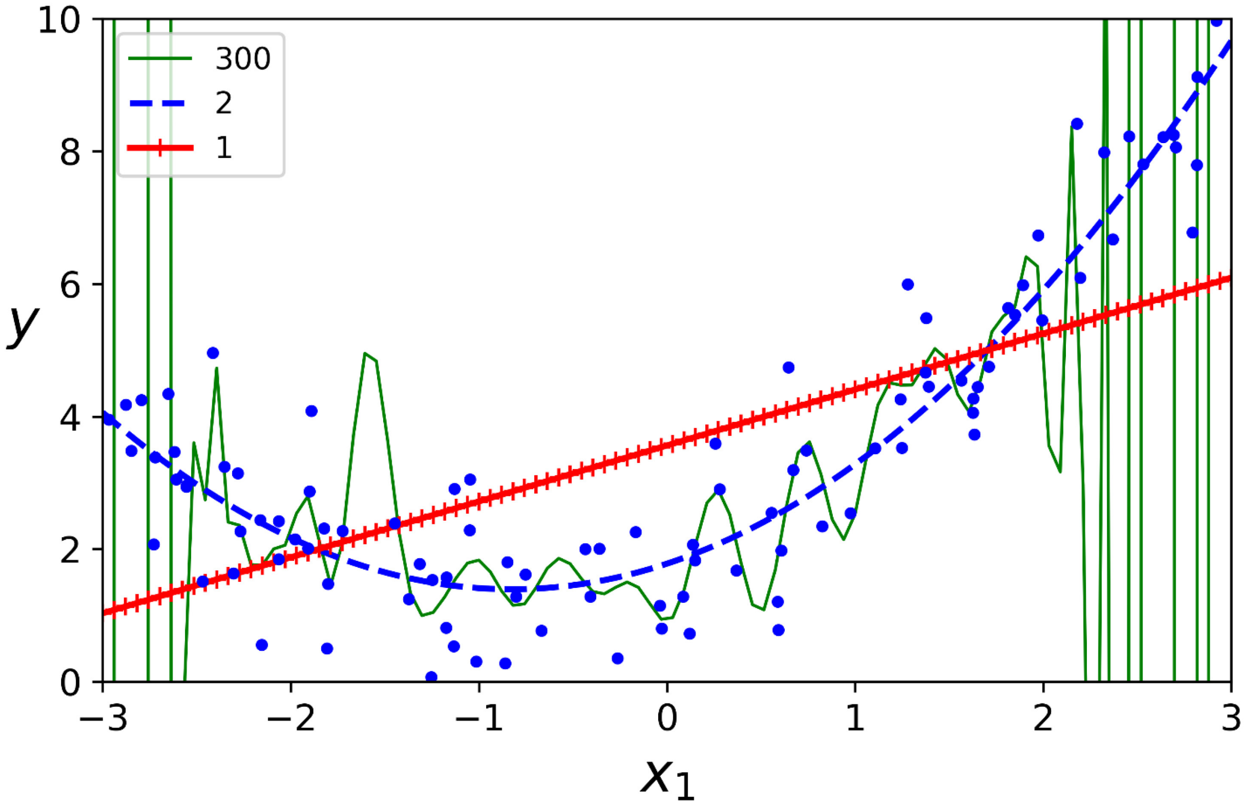
# Difference between Lasso und Ridge-Regression



# Reduction of the model complexity

- Increasing the complexity of a model usually increases its variance and lowers its bias.
- Conversely, a lower complexity of the model increases its bias and lowers its variance. Therefore, this is called equilibrium.
- **Regularizing the model (restricting it) is one way to avoid overfitting:**
- The less freedom the model has, the more difficult it becomes to overfit the data. For example, a polynomial model can easily be regularized by reducing the degree of the polynomial.

# Which order of the polynom?



plot_bias_variance.ipynb

# Ridge – Lasso and Elastic Net

- In a linear model, regularization is usually implemented in the form of **constraints on the weights** of the model.

We will now consider three different types of constraints:

a) **Ridge** regression (L2 regularization, Tikhonov)

b) **Lasso** regression (L1 regularization)

c) **Elastic Net** (combination of a and b)

# Minimization of the loss function with a penalty

- **Ridge-Regression**: regularization using Euklidian norm (L2)

$$L(\theta) = \sum_{i=1}^{N} (y_i - \hat{y}_i(\theta))^2 + \lambda \cdot \|\theta\|_2^2$$

$$\|\theta\|_2^2 = \sum_{k=1}^{n} (\theta_k)^2$$

- **Lasso-Regression**: regularization using L1 norm

$$L(\theta) = \sum_{i=1}^{N} (y_i - \hat{y}_i(\theta))^2 + \lambda \cdot \|\theta\|_1$$

$$\|\theta\|_1 = \sum_{k=1}^{n} |\theta_k|$$

$\lambda$: Regularization Parameter (Hyper-parameter)

# regression strength

- By assigning larger values to the **hyper parameter** λ, we increase the regularization strength and reduce the weights of our model.
- The axis segment term $\theta_0$ is not regularized.
- **LASSO** is an approach that **can lead to sparse models**. Depending on the regularization strength, certain weights can become zero, which means that LASSO can also function as a **monitored feature selection procedure**.
- The regularization strength (parameter λ) is freely selectable and must be optimized, for example, by k-fold cross validation.

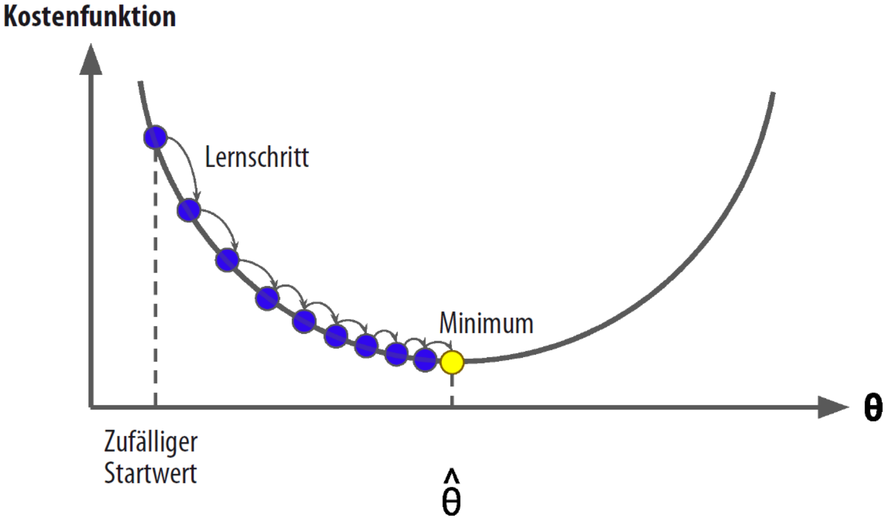
# Idea of the gradient descent

$$L(\theta) = \sum_{i=1}^{N} (y_i - \hat{y}_i(\theta))^2 + \lambda \cdot \|\theta\|_2^2$$

$$\theta^{(n+1)} = \theta^{(n)} - \eta \cdot \nabla_\theta L(\theta)$$

$\eta$: Learning Rate

**Gradient descent**



# Gradient of the MSE

## Linear Regression

- In the case of **linear regression**, the gradient can be calculated **analytically**:

$$L(\theta) = \mathrm{MSE}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}(\theta)\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\theta^T \cdot x^{(i)} - y^{(i)}\right)^2$$

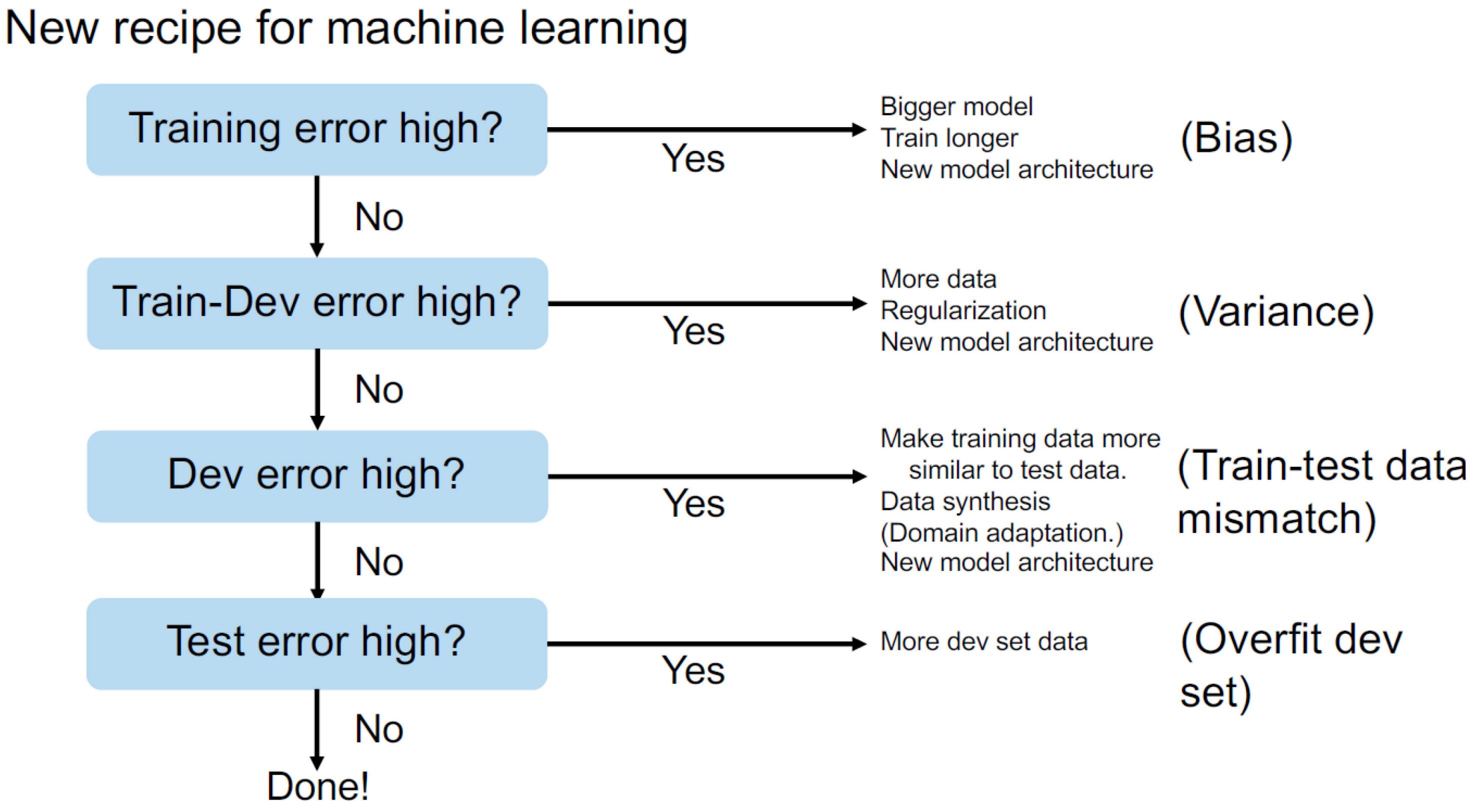
$$\nabla_{\theta_j} L(\theta) = \nabla_{\theta_j}\mathrm{MSE}(\theta) = \frac{2}{n}\sum_{i=1}^{n}\left(\theta^T \cdot x^{(i)} - y^{(i)}\right)\cdot\theta_j$$

$$\nabla_{\theta} L(\theta) = \nabla_{\theta}\mathrm{MSE}(\theta) = \frac{2}{n}\mathbf{X}^T\left(\mathbf{X}\cdot\theta - \mathbf{y}\right)$$

# Summary

- **Take** a **low bias algorithm** and **feed** it **tons of data** (ensures low variance) ➔ small test error
- **Try simpler** algorithms **first** (e.g., naïve Bayes before logistic regression, kNN before SVM); **try different** algorithms.
- **Regularization** combats overfitting by **penalizing** (but still allowing) high **flexibility**
- **Learning curves** ($E_{\text{train}}$ and $E_{\text{CV}}$ vs. increasing $N$) help **diagnosing** problems in terms of **bias** and **variance** ➔ decide what to do next
- If **more data** is needed: Can be manually **labeled**, artificially **created** (*data augmentation*) or **bought**
- Assess **covariate shift** through **2 distinct development sets**: one resembling training data, one resembling real data

# Recipe for Machine Learning



New recipe for machine learning

Training error high? — Yes → Bigger model, Train longer, New model architecture (Bias)

No ↓

Train-Dev error high? — Yes → More data, Regularization, New model architecture (Variance)

No ↓

Dev error high? — Yes → Make training data more similar to test data. Data synthesis (Domain adaptation.) New model architecture (Train-test data mismatch)

No ↓

Test error high? — Yes → More dev set data (Overfit dev set)

No ↓

Done!

Source: Andrew Ng, Nuts and Bolts of Building Applications using Deep Learning
https://nips.cc/Conferences/2016/Schedule?showEvent=6203

# Further reading (can be found on Moodle)

| Author | Title | Year |
|---|---|---|
| Domingos | ***A Few Useful Things to Know about Machine Learning***<br>Learning = Representation + Evaluation + Optimization; It's Generalization that Counts; Data Alone is not Enough; Overfitting has many Faces; Intuition Fails in High Dimensions; Theoretical Guarantees are not what They Seem; Feature Engineering is the Key; More Data Beats Clever Algorithm; Learn Many Models, not just One; Simplicity does not Imply Accuracy; Representable does not Imply Learnable; Correlation does not Imply Causation | 2012 |
| Wu et al. | ***Top 10 algorithms in data mining***<br>C4.5; k-Means; SVM; Apriori; EM; PageRank; AdaBoost; kNN; Naive Bayes; CART | 2008 |
| Langford and Piatetsky-Shapiro | ***11 Clever Methods of Overfitting and how to avoid them***<br>Traditional overfitting; Parameter tweak overfitting; Brittle measure; Bad statistics; Choice of measure; Incomplete prediction; Human-loop overfitting; Data set selection; Reprobleming; Old datasets; Overfitting by review | 2015 |
| Kotsiantis et al. | ***Data Preprocessing for Supervised Leaning***<br>Instance Selection & Outliers Detection; Missing Feature Values; Discretization; Data Normalization; Feature Selection; Feature Construction | 2006 |
| Chu | ***Machine Learning Done Wrong***<br>Take default loss function for granted; Use plain linear models for non-linear interaction; Forget about outliers; Use high variance model when $n << p$; L1/L2/… regularization without standardization; Use linear models without considering multi-collinear predictors; Interpreting absolute value of coefficients from linear or logistic regression as feature importance | 2015 |
| Hinman | ***The Do's and Don'ts of Data Mining***<br>Do: plan for data to be messy; create a clearly-defined & measurable objective for every project; ask questions; simplify the solution; cross-check data coming out of the ETL process; use more than one technique/algorithm; be informed;<br>Don't: ever (!) underestimate the power of good data preparation; use the default model accuracy metric; forget to document all modeling steps and underlying data; overfit; just collect a pile of data and "toss it into the big data mining engine"; wrongly think "it's all about the algorithms"; underestimate the power of a simpler-to-understand solution that is slightly less accurate; blindly trust assumptions made to satisfy frequency statistics, as well as p-values and AIC | 2014 |
| Raschka | ***What is wrong when my neural network's error increases?***<br>Debug using: Gradient checking, feature scaling and learning rate adaptation (we would add: check for proper weight initialization in conjunction with scaling [see batchnorm and http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization] and self-adapting training procedures like ADAM [see http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, and for their problems: Reddi, Kale & Kumar "On the Convergence of Adam and Beyond" @ ICLR'2018]) | 2016 |
| Zinkevich | ***Rules of Machine Learning: Best Practices for ML Engineering***<br>Google's best practices in machine learning: It presents a style for machine learning, similar to the Google C++ Style Guide and other popular guides to practical programming. | 2018 |

# Appendix

- Proof of the Bias-Variance Decomposition
- Gradient Descent
- Batch Gradient Descent
- Importance of Scaling
- Exact Solution for Ridge Regression

# Proof of the Bias-Variance-Decomposition

- Expected value of the square of a random variable X:

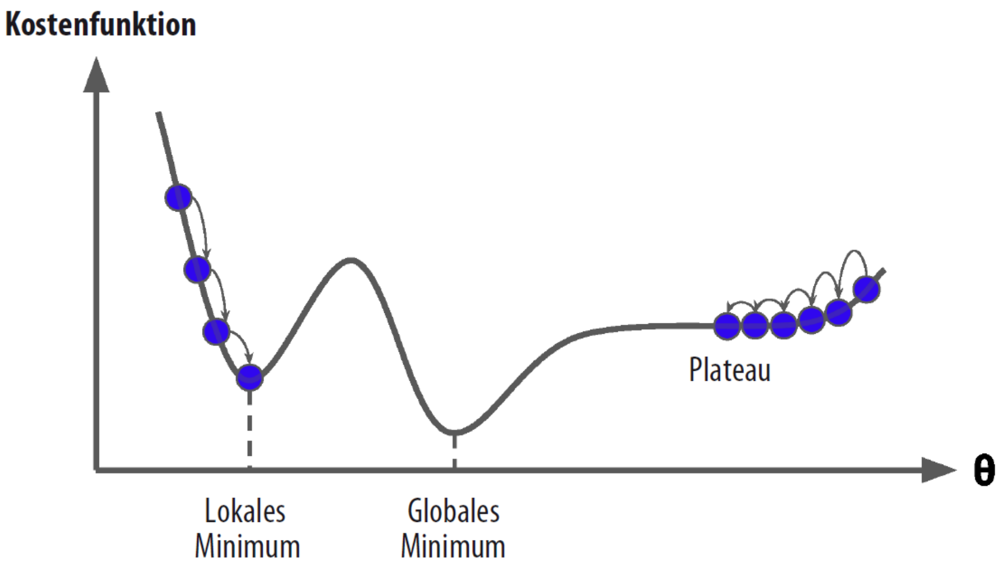
$$\mathrm{E}[X^2] = \mathrm{Var}[X] + \mathrm{E}[X]^2$$

$$\mathrm{E}[y] = \mathrm{E}[f + \epsilon] = \mathrm{E}[f] = f$$

- With this:

$$\begin{aligned}
\mathrm{E}\big[(y - \hat{f})^2\big] &= \mathrm{E}[y^2 + \hat{f}^2 - 2y\hat{f}] = \mathrm{E}[y^2] + \mathrm{E}[\hat{f}^2] - \mathrm{E}[2y\hat{f}] \\
&= \mathrm{Var}[y] + \mathrm{E}[y]^2 + \mathrm{Var}[\hat{f}] + \mathrm{E}[\hat{f}]^2 - 2f\mathrm{E}[\hat{f}] \\
&= \mathrm{Var}[y] + \mathrm{Var}[\hat{f}] + (f - \mathrm{E}[\hat{f}])^2 \\
&= \mathrm{Var}[y] + \mathrm{Var}[\hat{f}] + \mathrm{E}[f - \hat{f}]^2 \\
&= \sigma^2 + \mathrm{Var}[\hat{f}] + \mathrm{Bias}[\hat{f}]^2.
\end{aligned}$$

# convergence: local minima and saddle points



- MSE as cost function of a linear regression model is a **convex function**
  - no local minima, only a global minimum.
  - Derivation is also a continuous function with a slope that never changes abruptly (Lipschitz continuous).
- With the gradient method you can approach the global minimum arbitrarily close

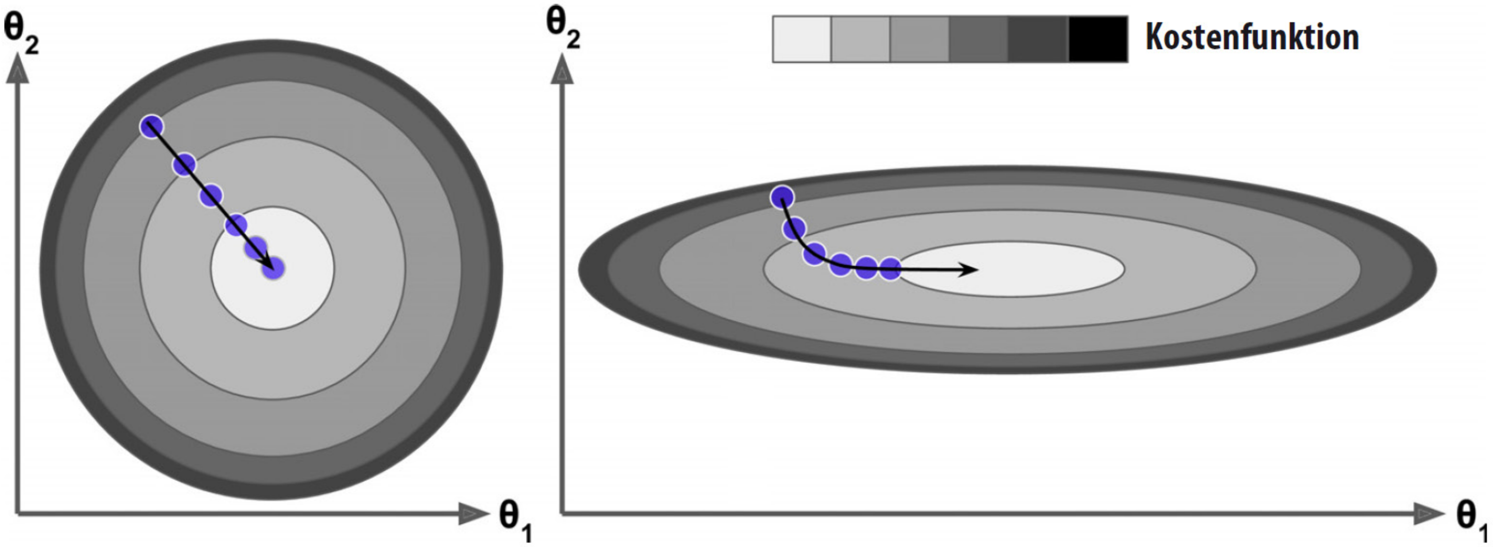
# Batch gradient method in Python

```
eta = 0.1 # Learning Rate
n_iterations = 1000
m = 100
theta = np.random.randn(2,1) # random initialization
for iteration in range(n_iterations):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
    theta = theta - eta * gradients
```

- **Batch gradient method**: This method uses the entire stack of training data for each step and is therefore **noticeably slow** for very large training data sets.
- The gradient method scales well with the number of features; training a linear regression model with hundreds of thousands of features is **much faster with the gradient method than with the normal equation**.

# important: scale the data

- Make sure that **all characteristics are similarly scaled** (e.g. using the **`StandardScaler`** class in Scikit-Learn).
- otherwise it will take significantly longer for the algorithm to converge.



# Exact Solutions

- Exact solution for the **Ridge regression**:

$$\hat{y} = (\mathbf{X}^T \cdot \mathbf{X} + \alpha \cdot \mathbb{A})^{-1} \mathbf{X}^T y = \hat{\theta}\, y$$

- There’s no exact solution for Lasso or elastic net. An **optimization algorithm** (such as gradient descent) has to be uesed.

# BIC and AIC

- Some models can offer an information-theoretic closed-form formula of the optimal estimate of the regularization parameter by computing a single regularization path (instead of several when using cross-validation).
- Here is the list of models benefiting from the **Akaike Information Criterion (AIC)** or the **Bayesian Information Criterion (BIC)** for automated model selection:

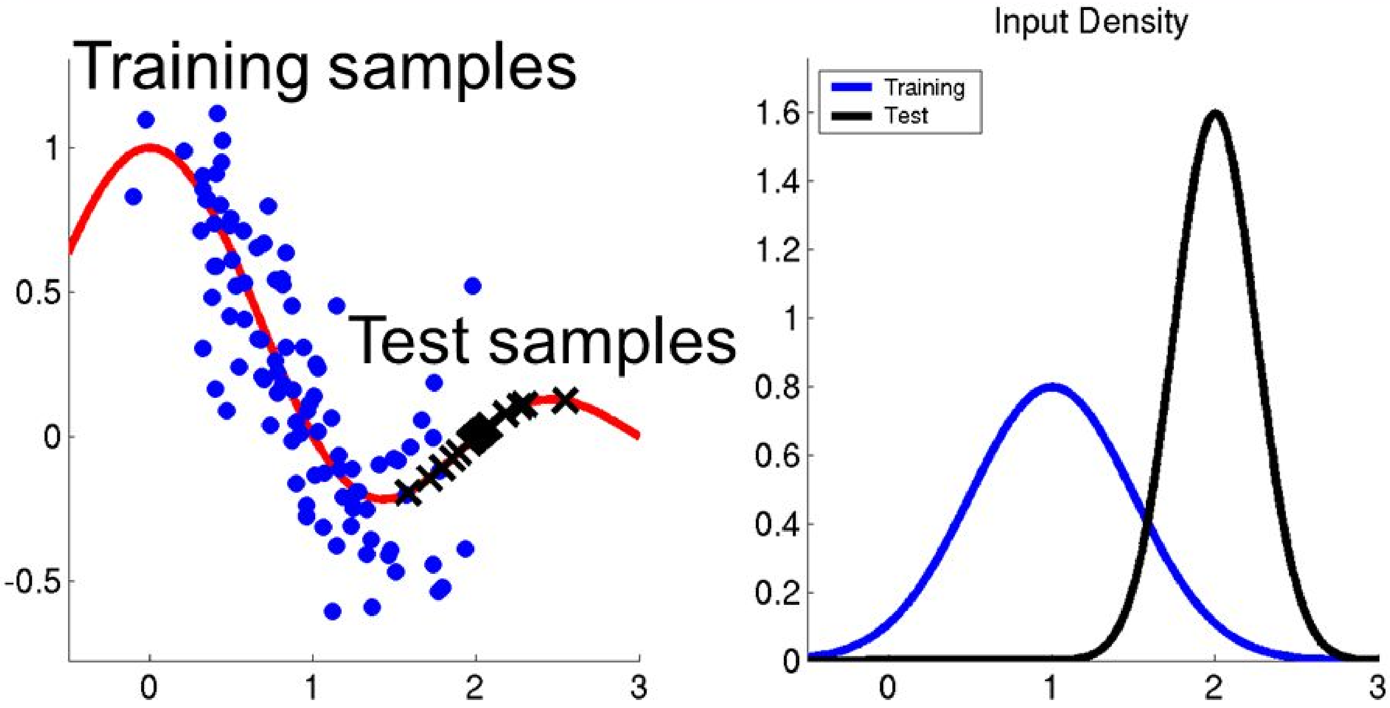
| | |
|---|---|
| `linear_model.LassoLarsIC([criterion, …])` | Lasso model fit with Lars using BIC or AIC for model selection |

# train-test data mismatch

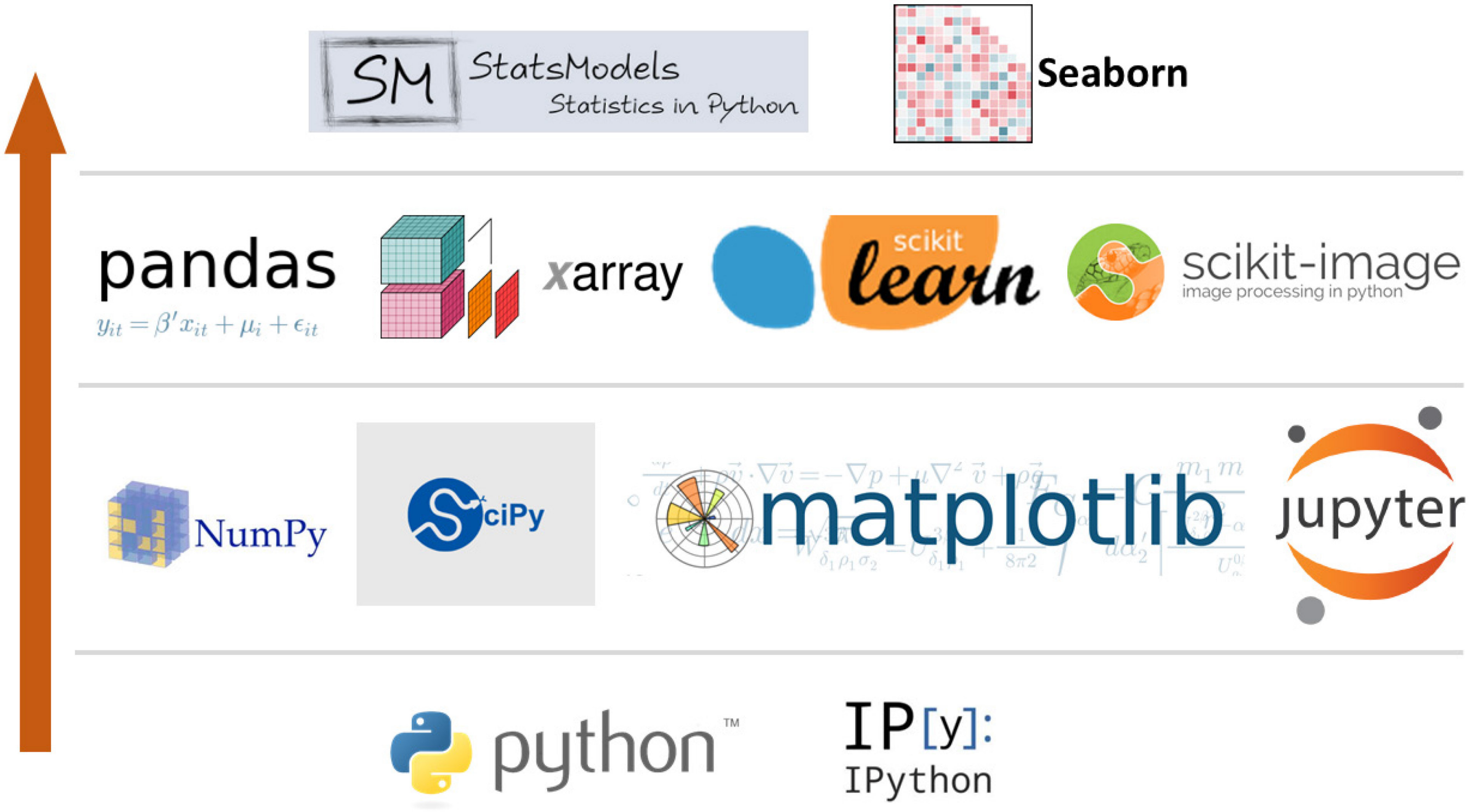
- The **covariate shift** is the change in the distribution of training data sets and the distribution of test data sets. The functional relationship is retained, but not, for example, the mean value and covariance.
- Normally one would expect them to come from the same multivariate distribution (**stationary**), but that almost never happens.
- **Latent variables:** We will never be able to observe all the factors that influence the distribution of a population. Some of them are latent (hidden, i.e. not observable per se). When this latent variable changes, the conditional distribution of observed data changes.
- Therefore, training data must be constantly updated or rebalanced with the latest test sets.
- **Covariate shift methods** reweight instances in the training data so that the distribution of training instances matches more closely with the distribution of instances in the predicted set. This is achieved by giving more weight to an instance in the training set that is similar to an instance in the forecast set during model building.

# weak extrapolation

- Test data and training data have the same functional form, but a different mean and variance.
- The test data lies outside (the convex envelope) of the training dataset
- Extrapolation, prediction outside the training dataset is only conditionally possible



# Overview: Python for data analysis


SM StatsModels Statistics in Python
Seaborn
pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$
xarray
scikit learn
scikit-image
image processing in python
NumPy
SciPy
matplotlib
jupyter
python™
IP[y]:
IPython