Classic ML

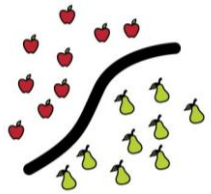
Deep Learning and Image Processing

MACHINE LEARNING

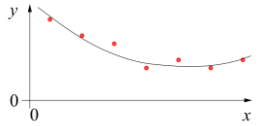
training target available (labeled or past data)

SUPERVISED

CLASSIFICATION



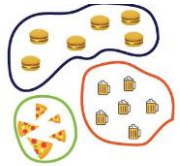
REGRESSION



data not labeled in any way

UNSUPERVISED

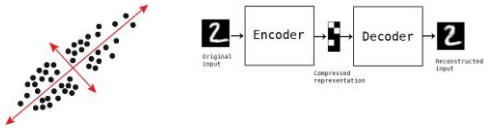
CLUSTERING



SELF-SUPERVISED

ASSOCIATION

DIMENSIONALITY REDUCTION



no supervision, but goal-based interaction with environment

REINFORCEMENT LEARNING

LEARN STATE OR ACTION VALUES

LEARN POLICY DIRECTLY

learning by teacher (high-dimensional curve fitting)

learning by observation (pattern recognition)

learning by trial-and-error (sequential decision making)

unsupervised and reinforcement learning can both be cast as supervised-learning setup

Supervised Learning

Target Quantity

- **known in training**: labeled samples or observations from past
- to be **predicted** for unknown cases (e.g., future values)

Features

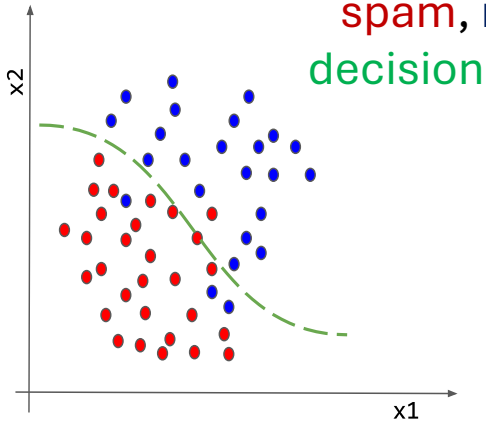
(prepared) input information that is
- correlated to target quantity
- known at prediction time

Example: Spam Filtering

classify emails as spam or no spam

*use accordingly **labeled emails as training set***

use information like occurrence of specific words or email length as **features**



features x1 and x2

spam, no spam

decision boundary

Optimization

General Recipe of Statistical Learning

ML algorithm by combining:

- **model** (e.g., linear function & Gaussian distribution)
- **objective function** (e.g., squared residuals)
- **optimization algorithm** (e.g., gradient descent)
- **regularization** (e.g., L2, dropout)

Supervised Learning in Mathematical Terms

map input to output: $y = f(\boldsymbol{x})$ (estimated: $\hat{f}(\boldsymbol{x})$)

random variables Y and $\boldsymbol{X} = (X_1, X_2, \cdots, X_p)$ ← usually high-dimensional

training (curve fitting / parameter estimation):

fit data set of $(y_i, \boldsymbol{x}_i)$ pairs → minimize deviations between y and $\hat{f}(\boldsymbol{x})$

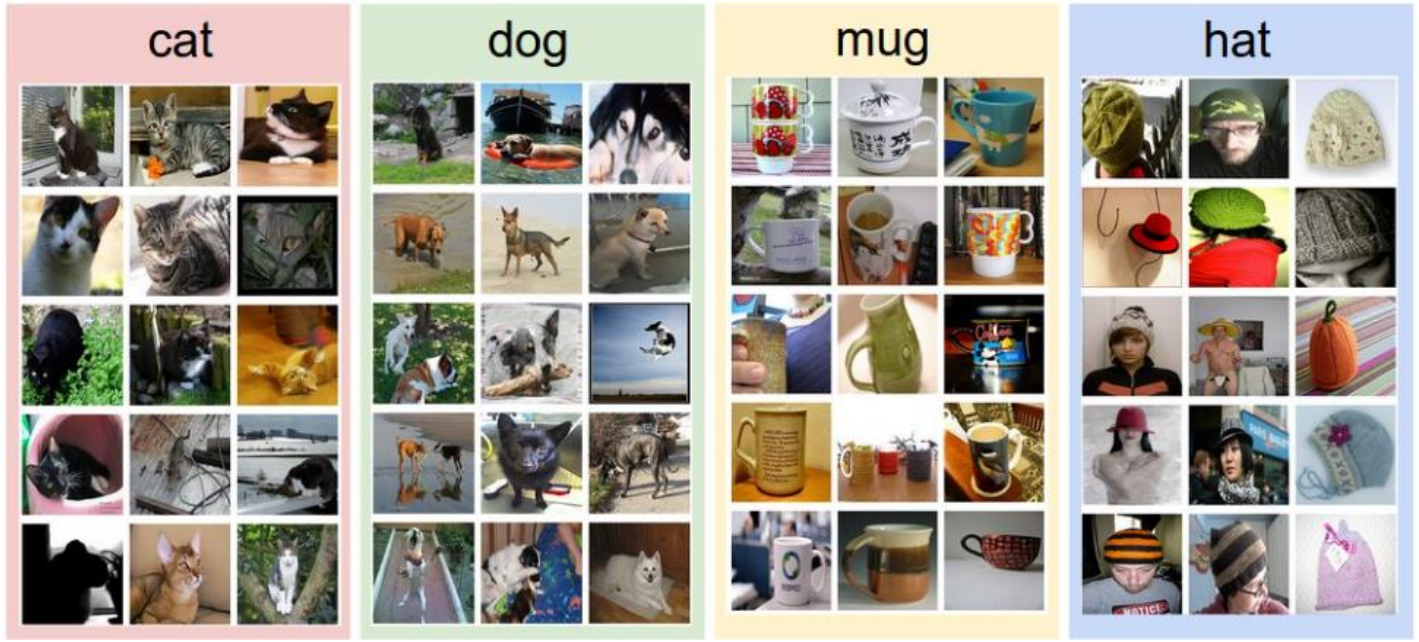
discriminative models: predict conditional density function $p(y|\boldsymbol{x})$

as opposed to generative models: predict $p(y, \boldsymbol{x})$ (or just $p(\boldsymbol{x})$) → $\boldsymbol{x}$ not given, more difficult

Example: Image Classification

training data set:

labels:



test data with learned classifier:

$f($  $) =$ “Cat”

$f($  $) =$ “Dog”

Linear Regression

fit:

$$y_i = \hat{b} + \sum_{j=1}^{p} \hat{w}_j \, x_{ij} + \varepsilon_i \qquad \hat{f}(\boldsymbol{x}_i)$$

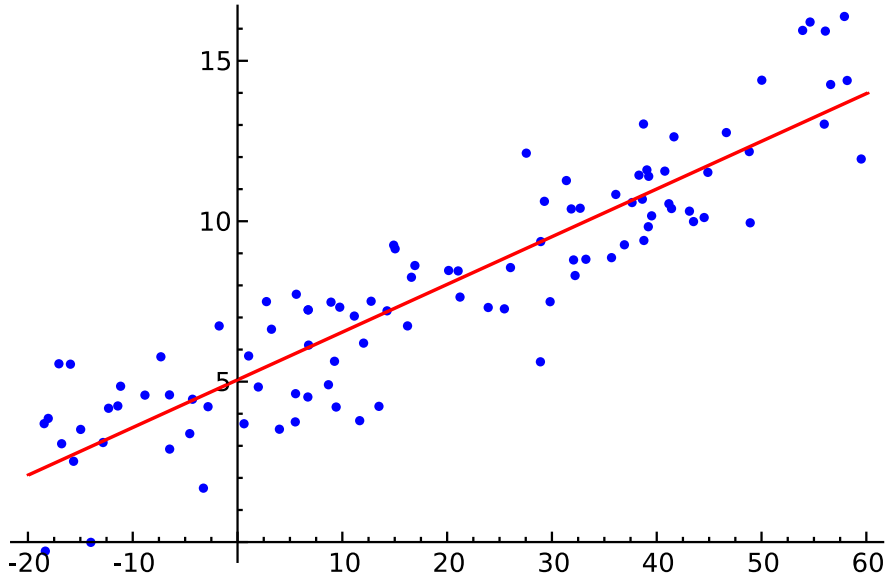
predict:

$$\hat{y}_i = E[Y|\boldsymbol{X} = \boldsymbol{x}_i] = \hat{f}(\boldsymbol{x}_i) = \hat{b} + \sum_{j=1}^{p} \hat{w}_j \, x_{ij}$$

Gaussian

$$p(y|\boldsymbol{x}_i) = \mathcal{N}(y; \hat{y}_i, \hat{\sigma}^2)$$

Gaussian mean variance



parameter estimation:
e.g., least squares

parameters to be estimated: $\hat{b}, \hat{\boldsymbol{w}}$

$\rightarrow \hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}(\boldsymbol{x}_i)\right)^2$

(approximating assumed true $b, \boldsymbol{w}, \sigma$)

Brief Notation

data: vector with p different feature values for this example i

$$f(\boldsymbol{x}_i) = \boldsymbol{w} \cdot \boldsymbol{x}_i + b$$

vector with slope parameters (one for each of the p features, but identical for all examples), aka weights

single intercept parameter, aka bias

for simplicity of notation, dropping the hats of estimated parameters here

Loss Function

loss function L: expressing deviation between prediction and target

$L\left(y_i, \hat{f}(\boldsymbol{x}_i); \widehat{\boldsymbol{\theta}}\right)$
with $\widehat{\boldsymbol{\theta}}$ corresponding to parameters of model $\hat{f}(\boldsymbol{x})$
e.g., $\hat{b}, \widehat{\boldsymbol{w}}$ in linear regression

prime example: squared residuals for regression problems
$L\left(y_i, \hat{f}(\boldsymbol{x}_i); \widehat{\boldsymbol{\theta}}\right) = \left(y_i - \hat{f}\left(\boldsymbol{x}_i; \widehat{\boldsymbol{\theta}}\right)\right)^2$

Cost Function

averaging losses over (empirical) training data set:

$$J(\widehat{\boldsymbol{\theta}}) = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, \hat{f}(\boldsymbol{x}_i); \widehat{\boldsymbol{\theta}}\right)$$

cost function to be minimized according to model parameters $\widehat{\boldsymbol{\theta}}$

→ objective function

Cost Minimization

minimize training costs $J(\widehat{\boldsymbol{\theta}})$ according to model parameters $\widehat{\boldsymbol{\theta}}$:

$$\nabla_{\widehat{\boldsymbol{\theta}}} J(\widehat{\boldsymbol{\theta}}) = 0$$

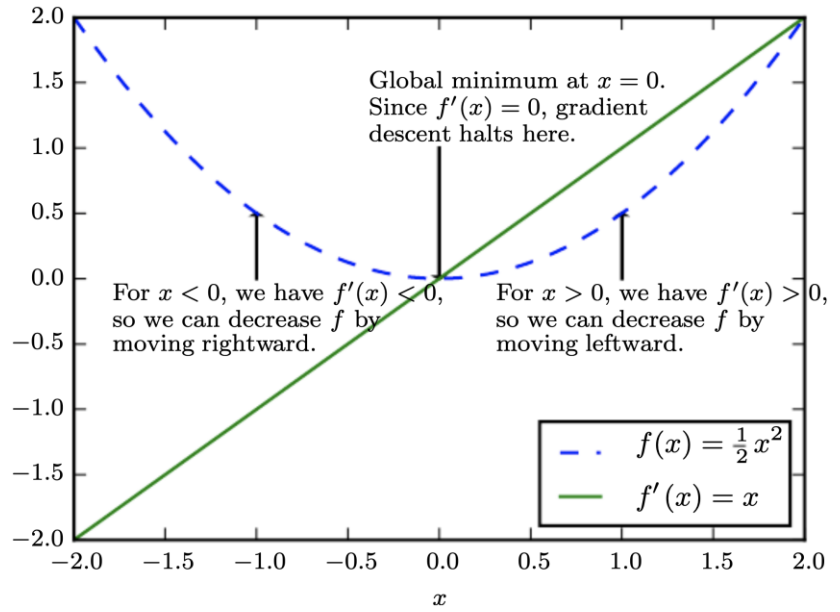
for mean squared error (aka least squares method):

$$\nabla_{\widehat{\boldsymbol{\theta}}} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{f}(\boldsymbol{x}_i; \widehat{\boldsymbol{\theta}}) \right)^2 = 0$$

Gradient Descent

typically no closed-form solution to minimization of cost function: $\nabla_{\hat{\boldsymbol{\theta}}} J(\hat{\boldsymbol{\theta}}) = 0$

→ need for numerical methods



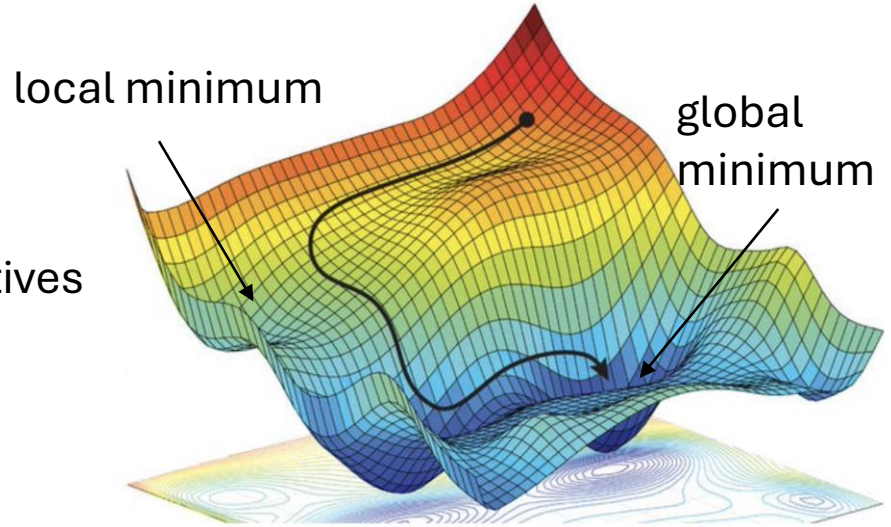
source

popular choice: gradient descent

iteratively moving in direction of steepest descent (negative gradient): $\hat{\boldsymbol{\theta}} \leftarrow \hat{\boldsymbol{\theta}} - \eta \nabla_{\hat{\boldsymbol{\theta}}} J(\hat{\boldsymbol{\theta}})$

step size (learning rate)

vector containing all partial derivatives



L^2 Regularization (Ridge Regression)

add **penalty term** on parameter L^2 norm to cost function

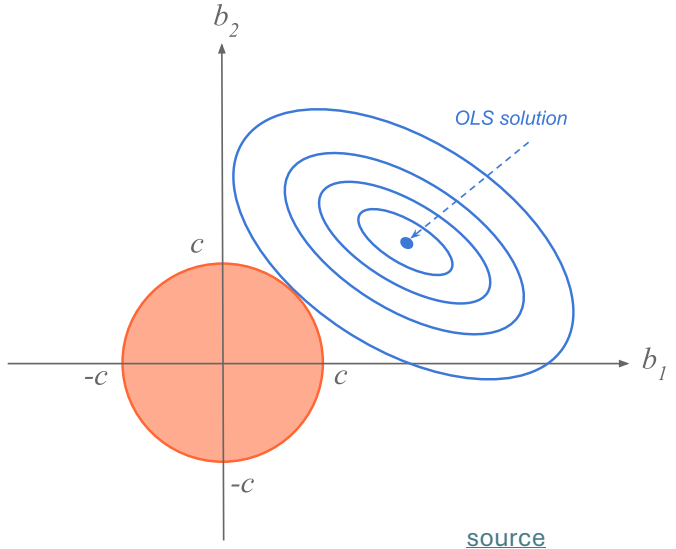
$$\tilde{J}(\hat{\boldsymbol{\theta}}) = J(\hat{\boldsymbol{\theta}}) + \lambda \cdot \hat{\boldsymbol{\theta}}^{\mathrm{T}} \hat{\boldsymbol{\theta}}$$

hyperparameter controlling strength of regularization

aka **weight decay** in neural networks

corresponds to imposing constraint on parameters to lie in L^2 region (size controlled by λ)

goal: reduce overfitting



source

Classification: Logistic Regression

binary classification: $y = 1$ or $y = 0$

→ predict probabilities p_i for $y = 1$

- logit (log-odds) as link function
- Y following Bernoulli distribution

$$\text{logit}(E[Y|\boldsymbol{X} = \boldsymbol{x}_i]) = \ln\left(\frac{p_i}{1 - p_i}\right) = \boldsymbol{w} \cdot \boldsymbol{x}_i + b$$

Multi-Classification: Softmax

for classification in K categories:

- use “score” function with K outputs

$$f_k(\boldsymbol{x}_i) = \boldsymbol{w}_k \cdot \boldsymbol{x}_i + b_k$$

- and convert into probabilities by means of softmax function

$$\frac{\exp\left(f_k(\boldsymbol{x}_i)\right)}{\sum_{l=1}^{K} \exp\left(f_l(\boldsymbol{x}_i)\right)}$$

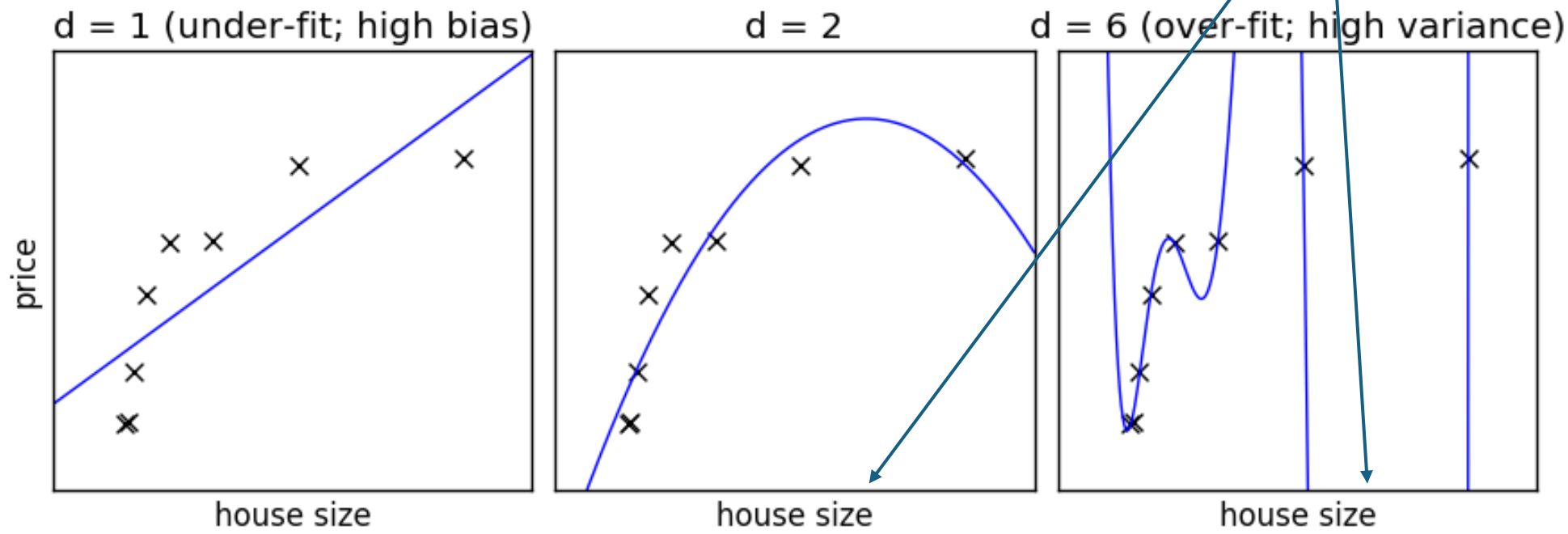
Need for Generalization

Under- and Over-Fitting

example: polynomial fit

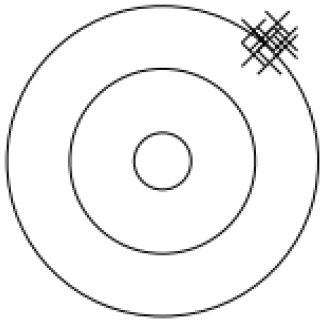
need to predict new samples (not in training data set): interpolation

degree of fitted polynomial

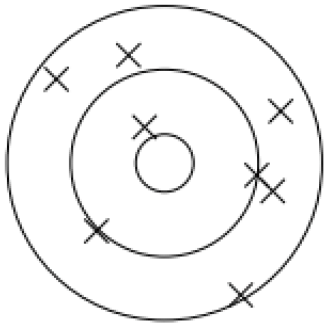


Bias, Variance, Irreducible Error

think of fitting ML algorithms as repeatable processes with different data sets



bias
due to too simplistic model
(same for all training data sets)
“underfitting”



variance
due to sensitivity to specifics (noise)
of different training data sets
“overfitting”

irreducible error (aka Bayes error):

inherent randomness (target generated from random variable)

→limiting accuracy of ideal model

Bias-Variance Tradeoff

models of higher complexity have lower bias but higher variance

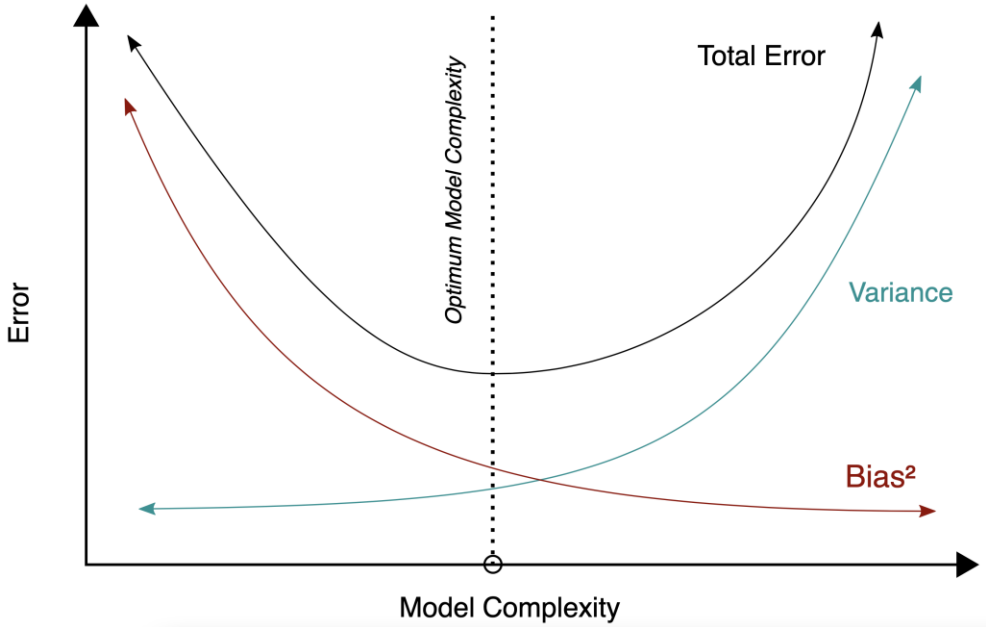
(given the same number of training examples)

generalization error follows U-shaped curve: overfitting once model complexity (number of parameters) passes certain threshold

overfitting: variance term dominating test error

→ increasing model complexity increases test error

but beware: training error keeps decreasing with more complex model …

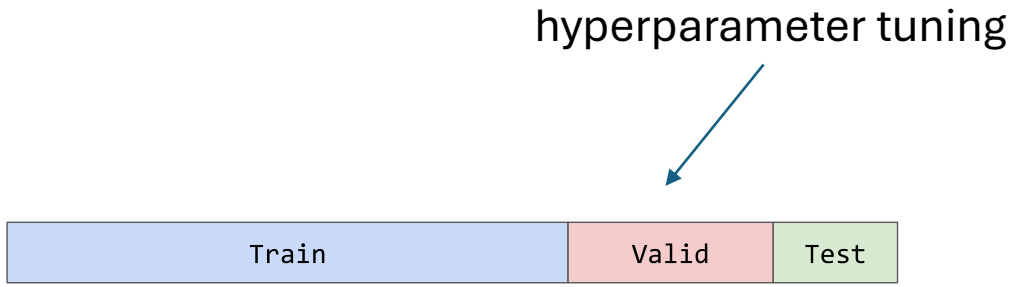


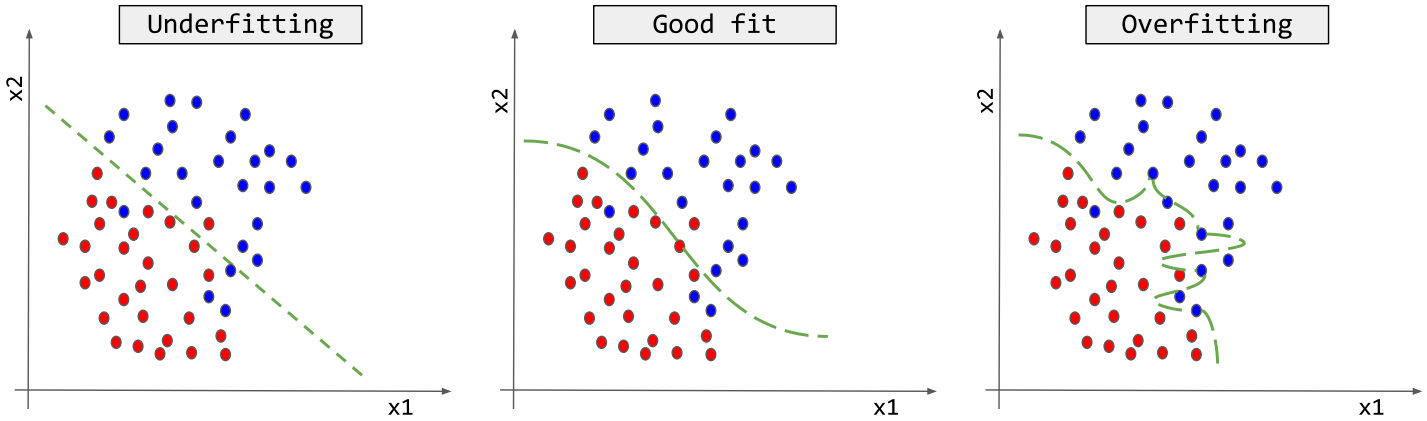
Hyperparameters

model complexity often controlled via hyperparameters (parameters not fitted but set in advance)

examples:

- degree of fitted polynomial d
- number of considered nearest neighbors k
- maximum depth of decision tree
- number of hidden nodes in neural network layer





Train-Dev-Test vs. Model fitting

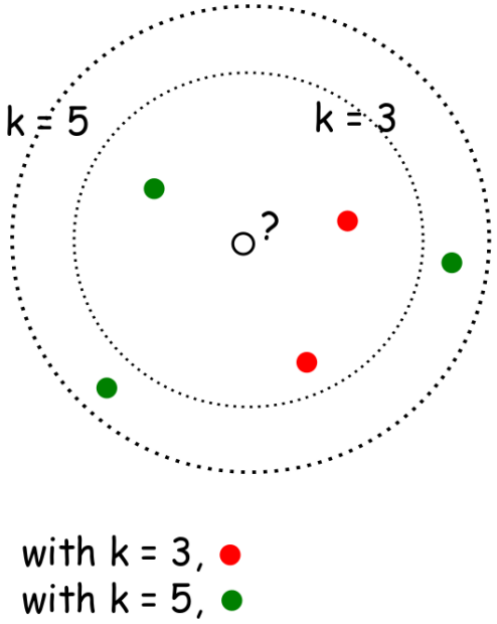
Another Example: k-Nearest Neighbors (kNN)

local method, instance-based learning
non-parametric
distance defined by metric on $\boldsymbol{x}$ (e.g., Euclidean)

regression:

$$\hat{f}(\boldsymbol{x}_0) = \frac{1}{k}\sum_{j=1}^{k} y_j$$

with j running over
k nearest neighbors of $\boldsymbol{x}_0$



low k: low bias but high variance
high k: low variance but high bias

$$bias = f(\boldsymbol{x}) - \frac{1}{k}\sum_{j=1}^{k} y_j$$

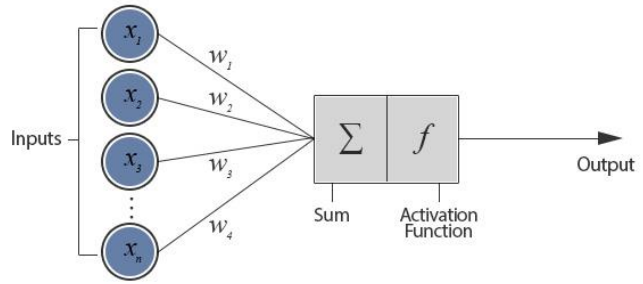
$$var = \frac{\sigma^2}{k}$$

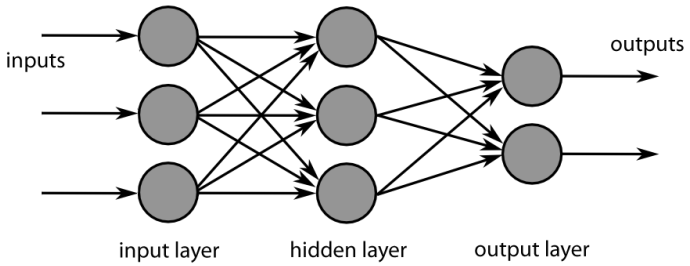
Algorithmic Families of Supervised Learning

parametric models

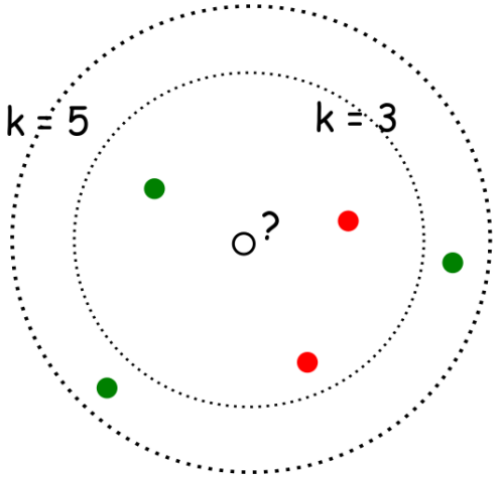
linear regression

neural networks: many linear models, non-linear by means of activation functions





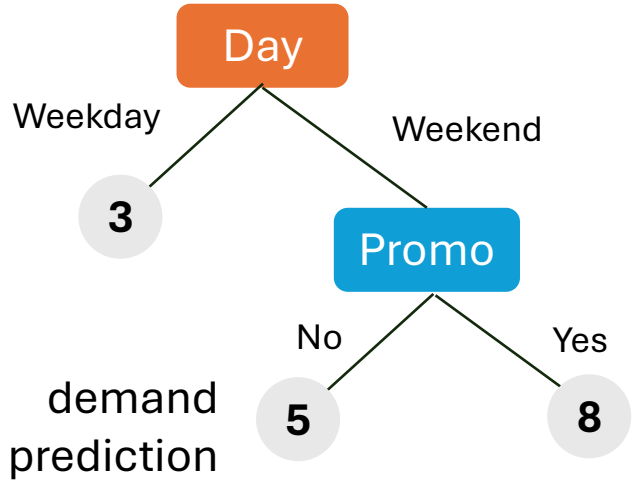
nearest neighbors (local methods, instance-based learning): non-parametric models



with k = 3,
with k = 5,

kernel/support-vector machines: linear model (maximum-margin hyperplane) with kernel trick

decision trees: rule learning



often used in ensemble methods

- bagging: random forests
- boosting: gradient boosting

Common Idea: Learning from Data

ML algorithm + data = explicit algorithm

→ much better generalizability than handcrafted algorithms

unfortunately, no general best ML algorithm:
need to pick right method for the task at hand

Generalization

core of ML: **empirical risk minimization** (training error) as proxy for minimizing unknown population risk (test error)

generalization gap: difference between test and training error

curse of dimensionality: typically, many features (dimensions)

→ lots of data needed to densely sample volume

Manifold Hypothesis

luckily, reality is friendly: most high-dimensional data sets reside on lower-dimensional manifolds → enabling effectiveness of ML