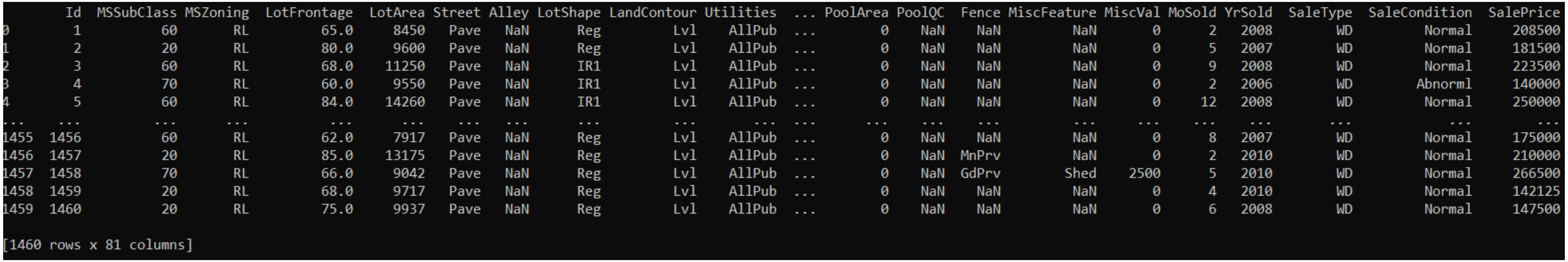
Machine Learning for Tabular Data

Fundamentals of Artificial Intelligence



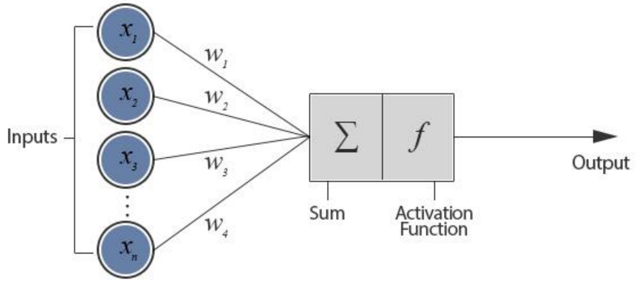
```
        Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape LandContour Utilities  ... PoolArea PoolQC  Fence MiscFeature  MiscVal MoSold YrSold  SaleType  SaleCondition  SalePrice
0        1          60       RL         65.0     8450   Pave   NaN      Reg         Lvl    AllPub  ...        0    NaN    NaN         NaN        0      2   2008        WD         Normal     208500
1        2          20       RL         80.0     9600   Pave   NaN      Reg         Lvl    AllPub  ...        0    NaN    NaN         NaN        0      5   2007        WD         Normal     181500
2        3          60       RL         68.0    11250   Pave   NaN      IR1         Lvl    AllPub  ...        0    NaN    NaN         NaN        0      9   2008        WD         Normal     223500
3        4          70       RL         60.0     9550   Pave   NaN      IR1         Lvl    AllPub  ...        0    NaN    NaN         NaN        0      2   2006        WD        Abnorml     140000
4        5          60       RL         84.0    14260   Pave   NaN      IR1         Lvl    AllPub  ...        0    NaN    NaN         NaN        0     12   2008        WD         Normal     250000
...    ...         ...      ...          ...      ...    ...   ...      ...         ...       ...  ...      ...    ...    ...         ...      ...    ...    ...       ...            ...        ...
1455  1456          60       RL         62.0     7917   Pave   NaN      Reg         Lvl    AllPub  ...        0    NaN    NaN         NaN        0      8   2007        WD         Normal     175000
1456  1457          20       RL         85.0    13175   Pave   NaN      Reg         Lvl    AllPub  ...        0    NaN  MnPrv         NaN        0      2   2010        WD         Normal     210000
1457  1458          70       RL         66.0     9042   Pave   NaN      Reg         Lvl    AllPub  ...        0    NaN  GdPrv        Shed     2500      5   2010        WD         Normal     266500
1458  1459          20       RL         68.0     9717   Pave   NaN      Reg         Lvl    AllPub  ...        0    NaN    NaN         NaN        0      4   2010        WD         Normal     142125
1459  1460          20       RL         75.0     9937   Pave   NaN      Reg         Lvl    AllPub  ...        0    NaN    NaN         NaN        0      6   2008        WD         Normal     147500

[1460 rows x 81 columns]
```

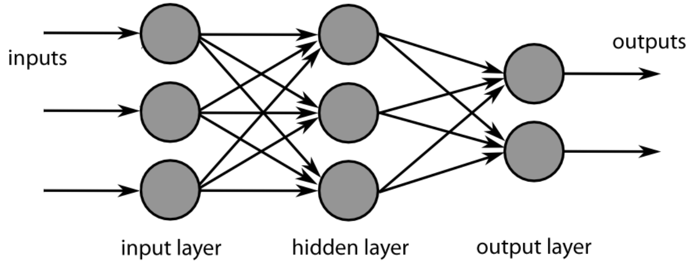
Algorithmic Families of Supervised Learning

parametric models

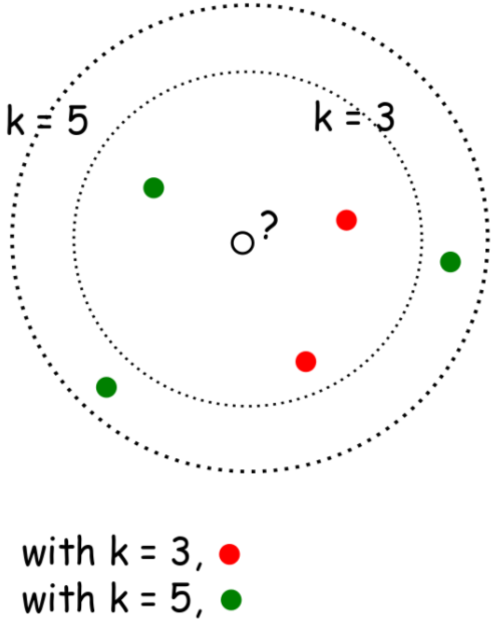
linear regression

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



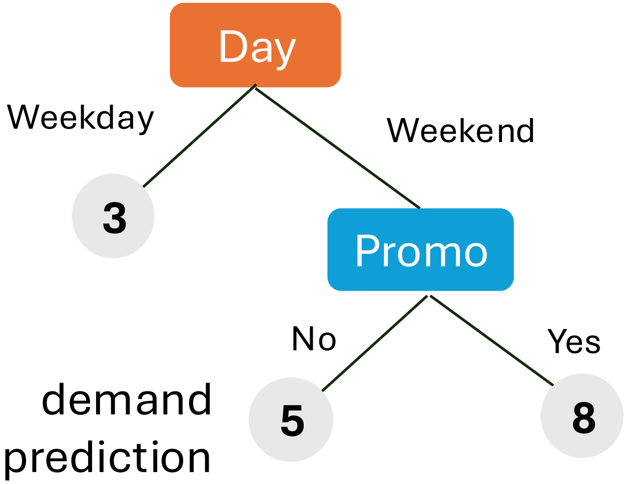


nearest neighbors (local methods, instance-based learning): non-parametric models



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

decision trees: rule learning



demand prediction

often used in ensemble methods

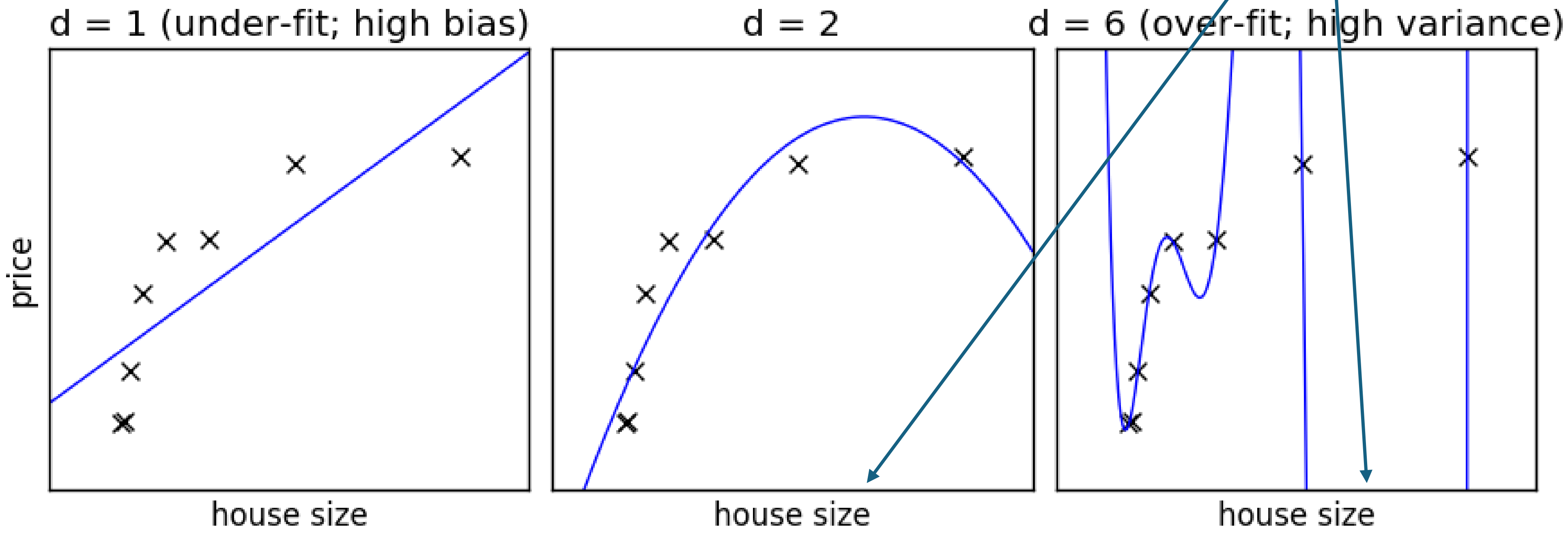
- bagging: random forests
- boosting: gradient boosting

Under- and Over-Fitting

example: polynomial fit

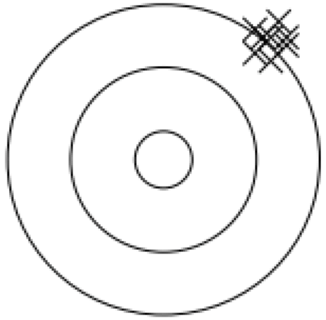
need to predict new samples (not in training data set): interpolation

degree of fitted polynomial



Bias, Variance, Irreducible Error

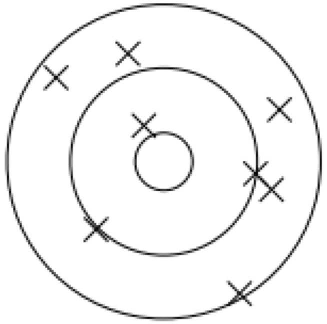
think of fitting ML algorithms as repeatable processes with different data sets



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

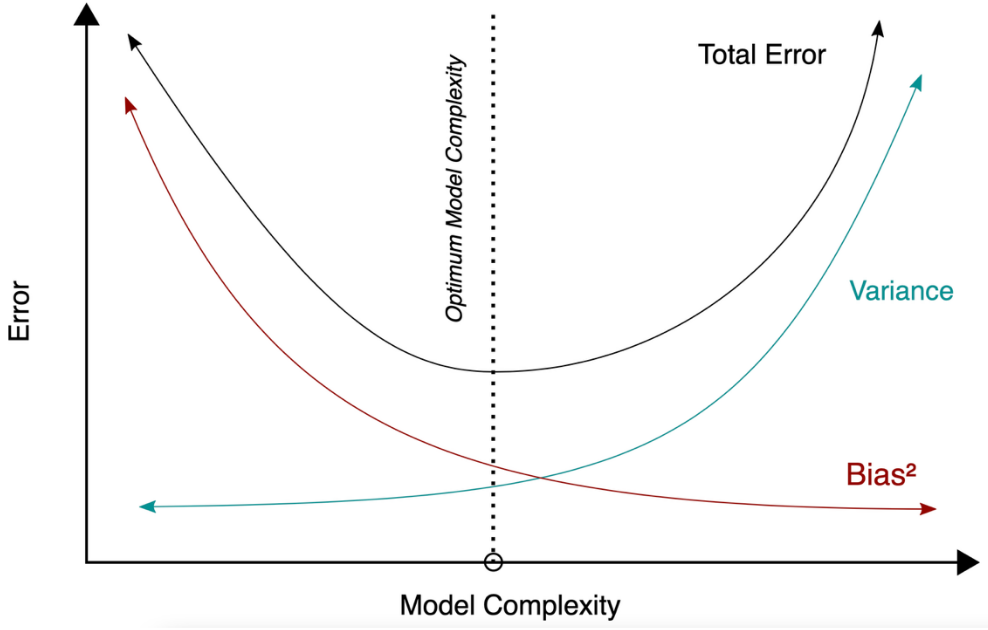
(given the same number of training examples)

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 ...



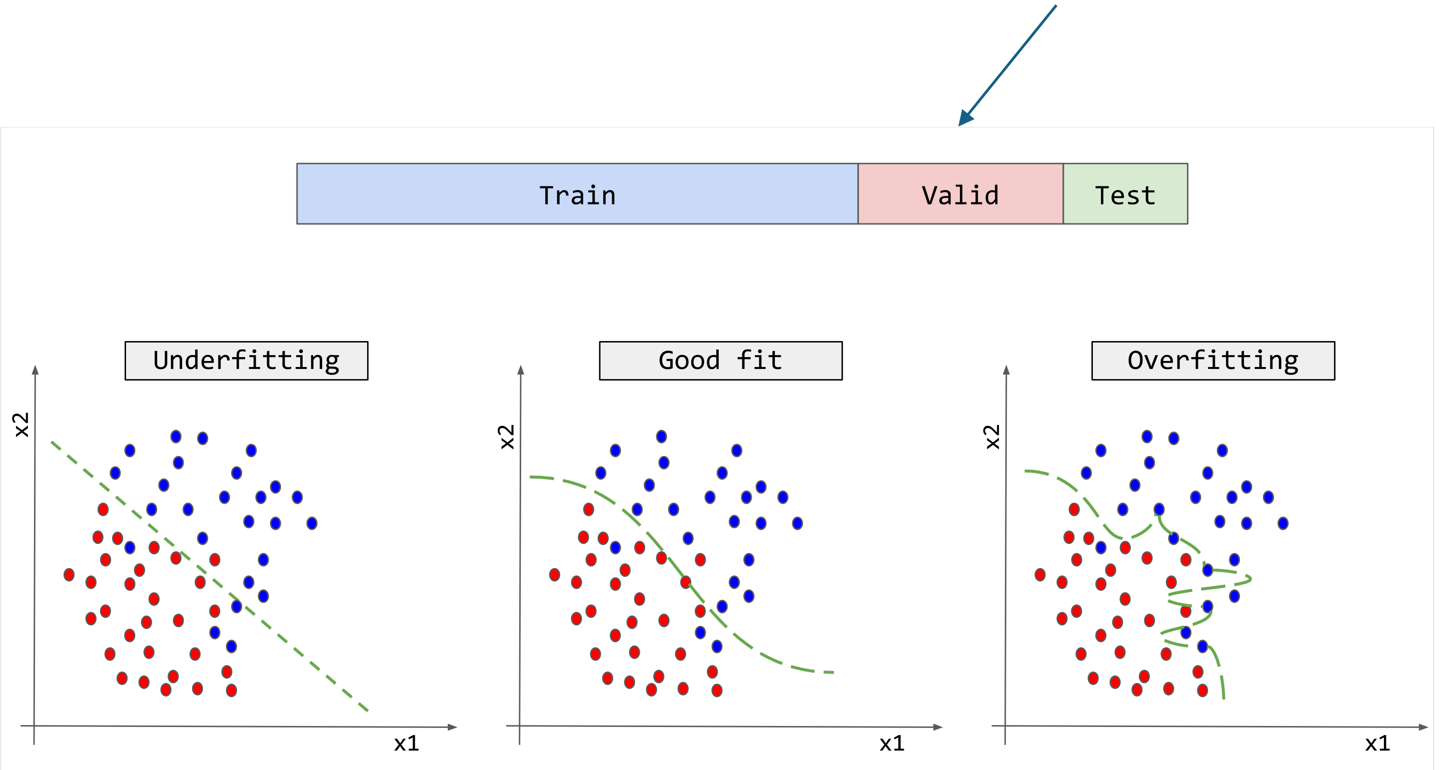
Hyperparameters

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

hyperparameter tuning



Train-Dev-Test vs. Model fitting

Generalization

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

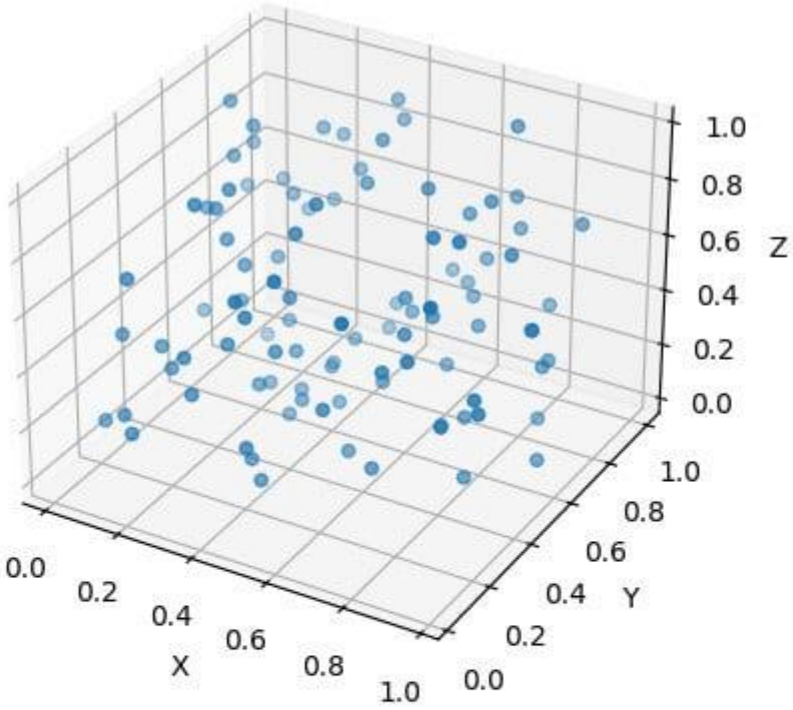
generalization gap: difference between test and training error

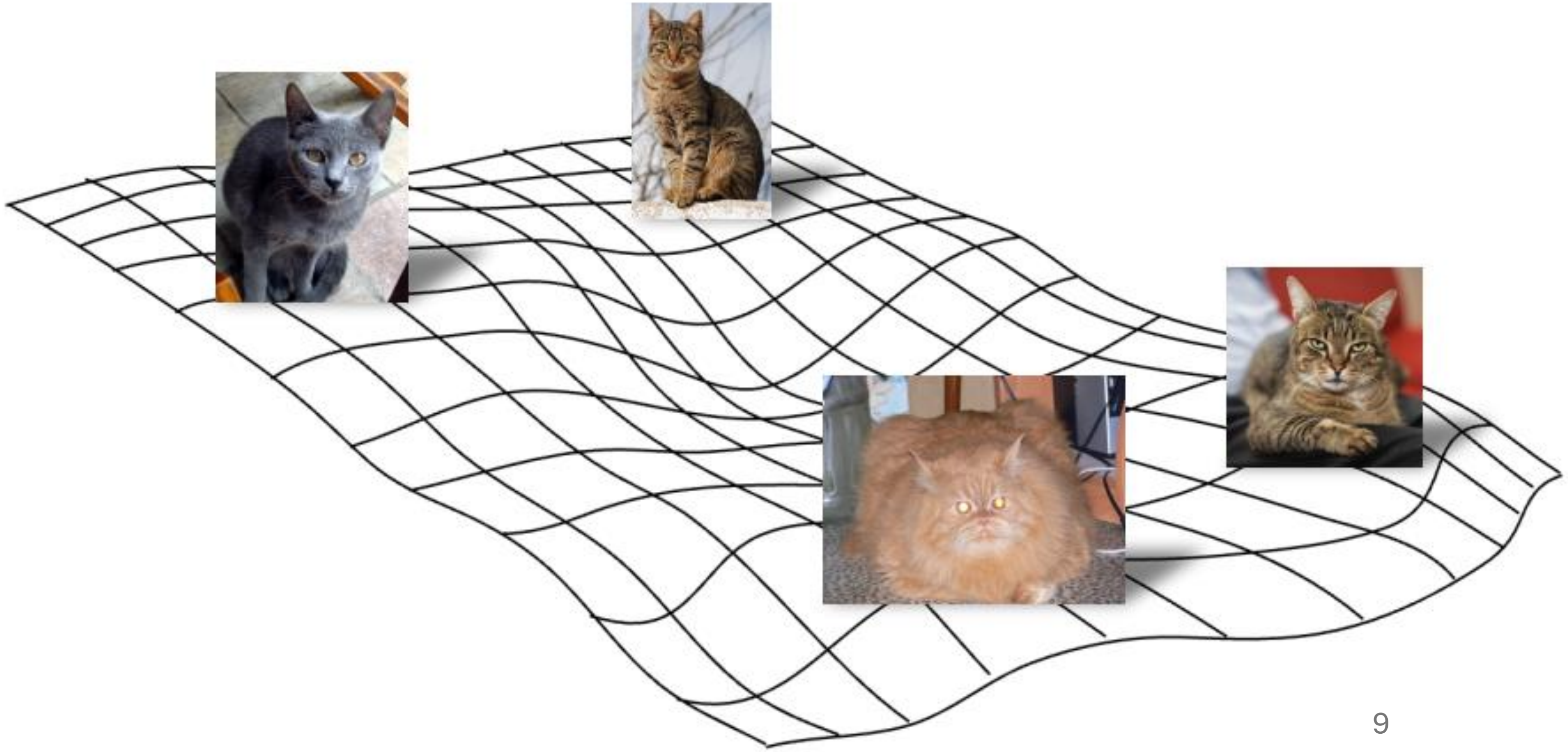
curse of dimensionality: typically, many features (dimensions)

→ 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





k-Nearest Neighbors (kNN)

local method, instance-based learning

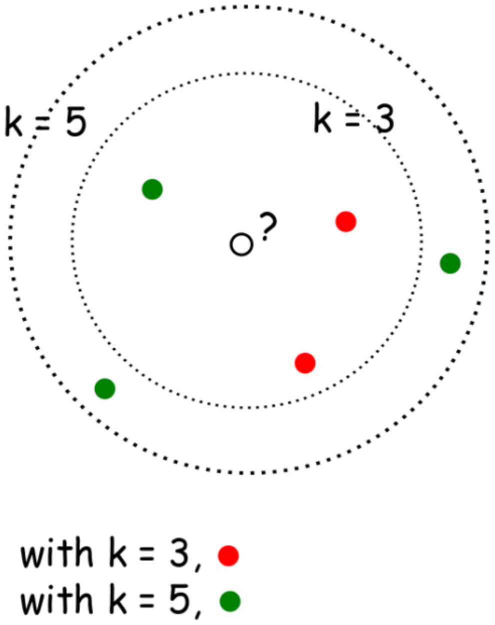
non-parametric

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 $\boldsymbol{x}_0$



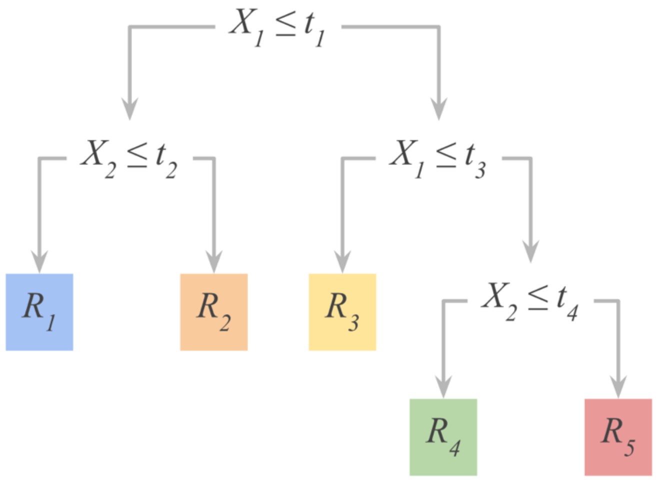
low k: low bias but high variance

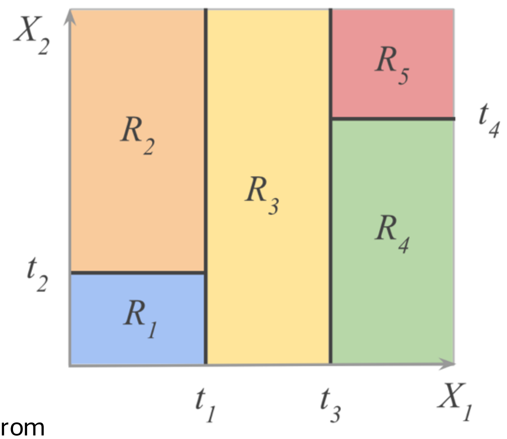
high k: low variance but high bias

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

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

Decision Trees

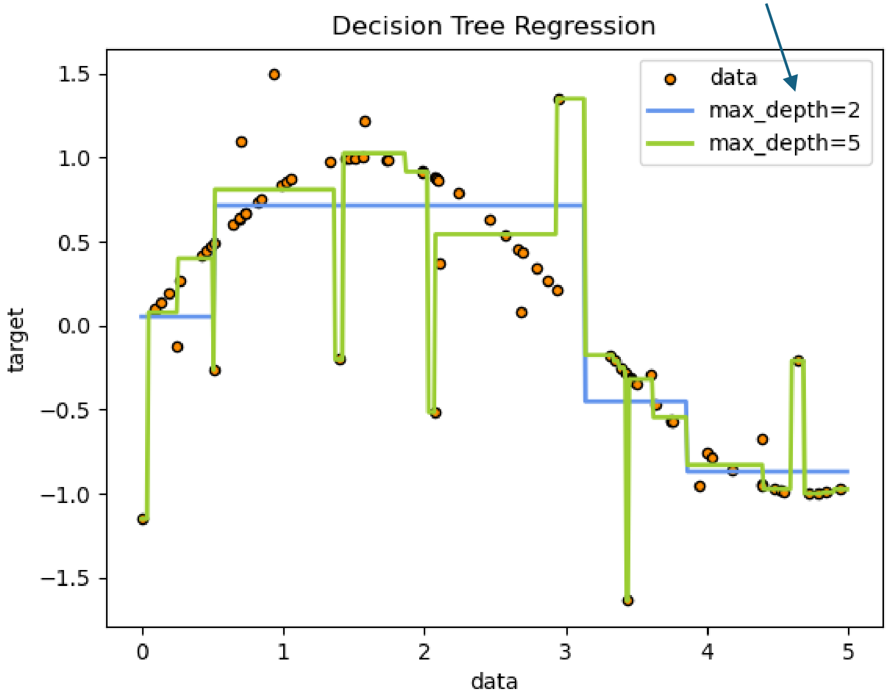




from
Hastie

- non-parametric learning of simple decision rules
- usually, binary trees
- axis-parallel decision boundaries (box-shaped regions in feature space)
- fit constant $\hat{c}$ in each box
  - classification: majority class of target
  - regression: average of target
- fully explainable models

hyperparameter



scikit-learn

Ensemble Learning

idea: combine several individual models (often of the same type) to form an ensemble model with better predictive performance than each of its constituents

→ learning in several different ways from the training data

- the trainings of the individual constituent models are (kind of) independent
- outputs of individual models are combined (e.g., averaged)

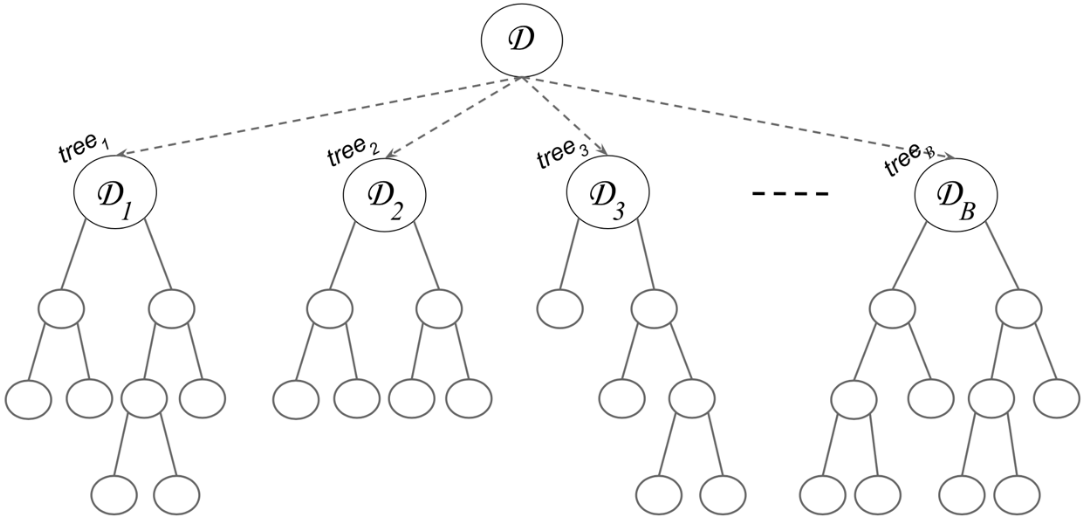
Bagging (Bootstrap Aggregating)

idea: train individual models of ensemble on random subsets of the training data (random sample with replacement)

- reduces variance of ensemble model compared to individual models
- but not bias → use low-bias base models

individual models combined by averaging (regression) or majority voting (classification) → committee of models

example with decision trees:



source

Random Subspace Method

aka feature bagging

idea: train different members (models) of an ensemble method on random subsets of all features (random sample with replacement)

→ reduce correlation between ensemble members

individual models combined by averaging (regression) or majority voting (classification) → committee of models

Random Forests

ensemble method using

- decision trees as individual models (usually, rather large, low-bias decision trees)
- combination of bagging and random subspace method (features sampled at each node in the trees)

compared to individual decision trees, random forests

+ reduce variance → improve accuracy

- lose explainability

one of the most popular off-the-shelf ML algorithms (often, good performance with little hyperparameter tuning and data preparation)

Boosting

idea: sequentially learn and combine several “weak” learners (such as small decision trees, but in principle any ML algorithm) to construct a “strong” one

→ gradually (in a greedy fashion) reducing bias of ensemble model

in simple terms:

building a model from the training data, then creating a second model that attempts to correct the errors from the first model, …

→ each subsequent weak learner is forced to concentrate on the examples that are missed by the previous ones in the sequence

not a committee of models:

typically, use simple, high-bias methods as individual models (e.g., small decision trees)

most prominent: AdaBoost, Gradient Boosting

Forward Stagewise Additive Modeling

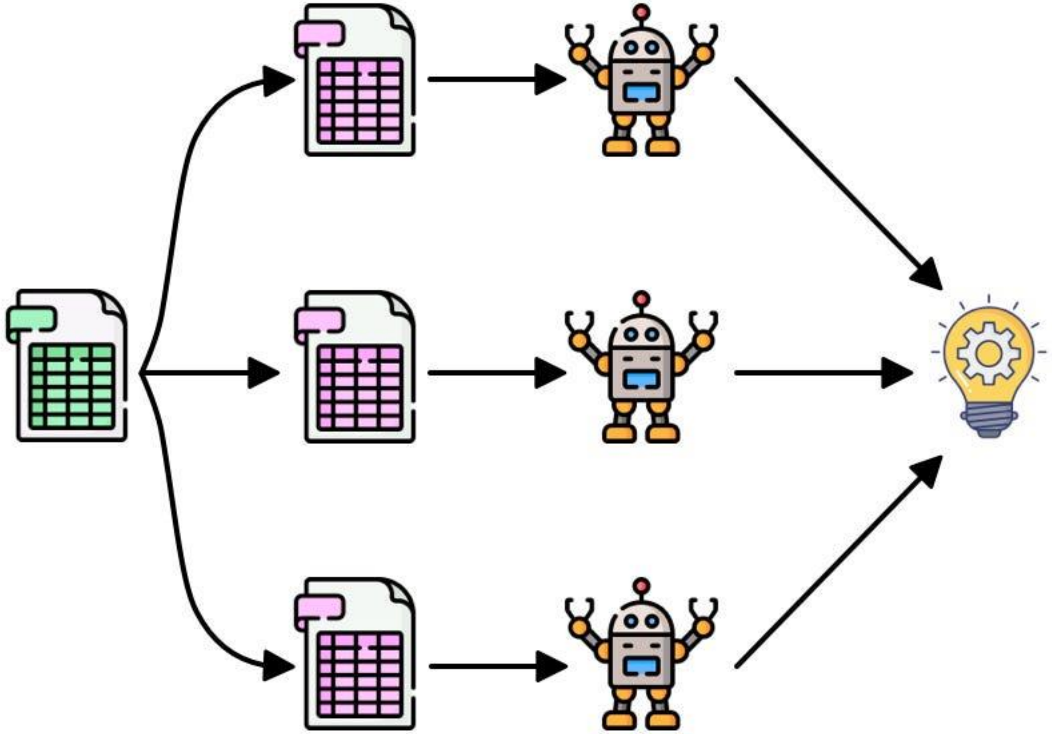
boosting can be understood as fitting an additive expansion in a set of basis functions $b(\boldsymbol{x}; \hat{\gamma}_m)$ (e.g., decision trees, with $\hat{\gamma}_m$ parametrizing splits): $\hat{f}(\boldsymbol{x}) = \sum_{m=1}^{M} \hat{\beta}_m \, b(\boldsymbol{x}; \hat{\gamma}_m)$

loss minimization of $\hat{f}(\boldsymbol{x})$ computationally expensive → boosting: sequentially add and fit individual basis functions without changing already fitted ones

boosting algorithm for sequential optimization using basis functions:

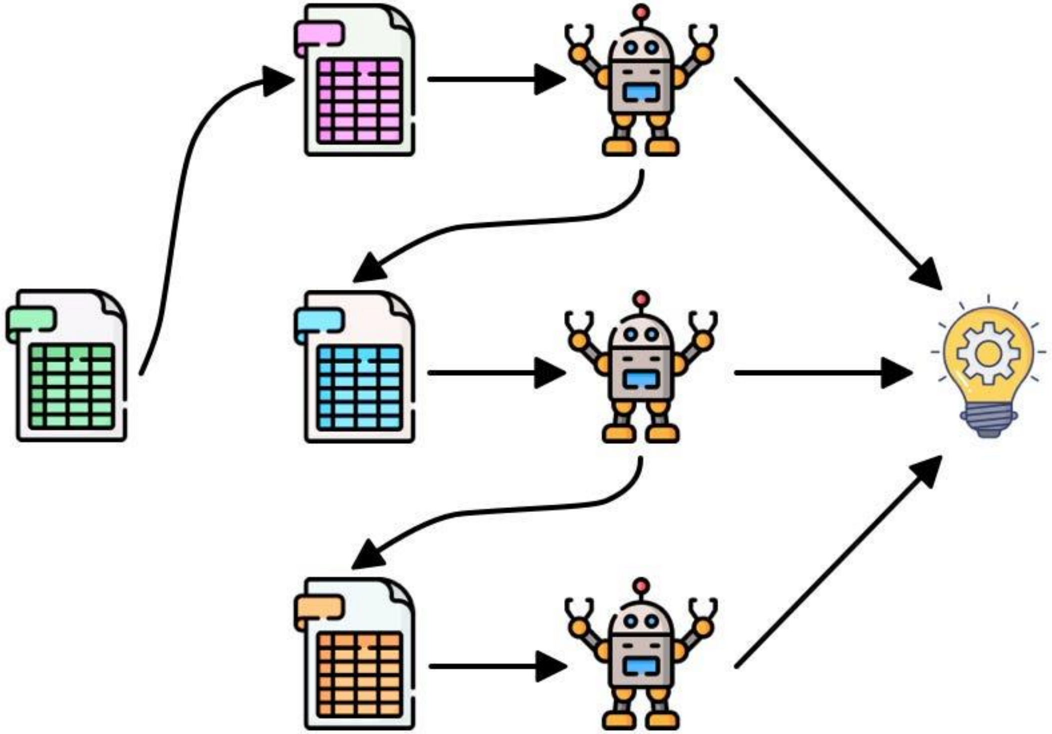
1. initialize $\hat{f}_0(\boldsymbol{x}) = 0$
2. for $m = 1$ to M
   a) compute $\underset{\hat{\beta}_m, \hat{\gamma}_m}{\operatorname{argmin}} \sum_{i=1}^{N} L\left(y_i, \hat{f}_{m-1}(\boldsymbol{x}_i) + \hat{\beta}_m \, b(\boldsymbol{x}_i; \hat{\gamma}_m)\right)$
   b) set $\hat{f}_m(\boldsymbol{x}) = \hat{f}_{m-1}(\boldsymbol{x}) + \hat{\beta}_m \, b(\boldsymbol{x}; \hat{\gamma}_m)$

Bagging



Parallel

Boosting



Sequential

AdaBoost (Adaptive Boosting)

in its simplest form: binary classification

forward stagewise additive modeling with individual classifiers $\hat{G}_m$ and loss function $L\left(y, \hat{f}(\boldsymbol{x})\right) = e^{-y\,\hat{f}(\boldsymbol{x})}$

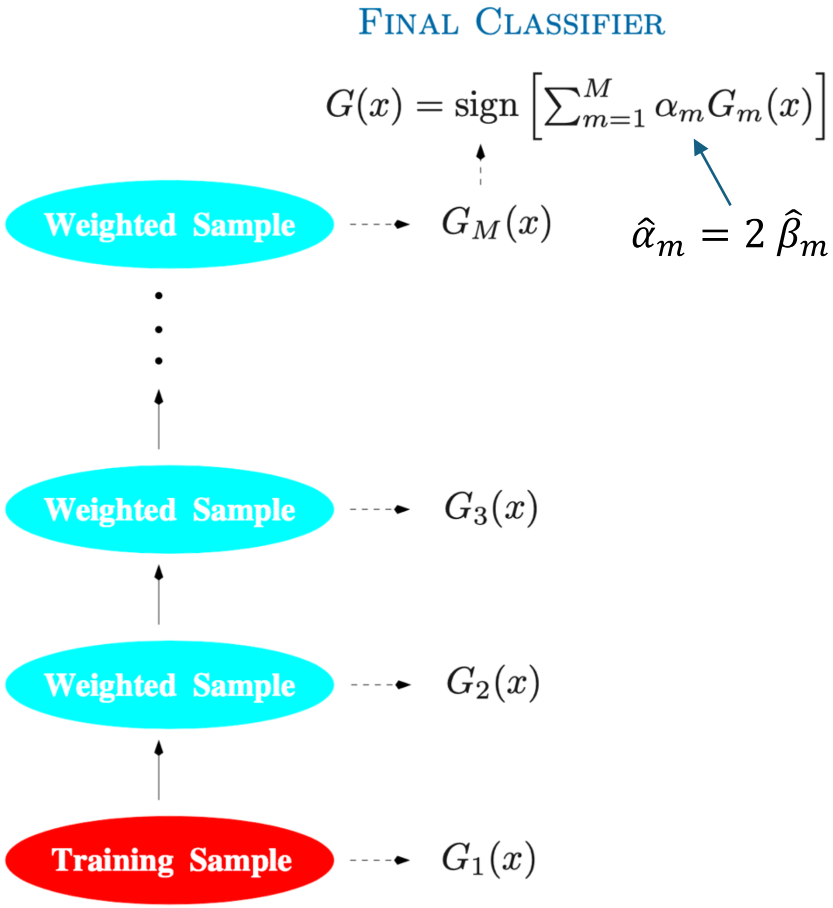
$$\left(\hat{\beta}_m, \hat{G}_m\right) = \underset{\hat{\beta},\hat{G}}{\operatorname{argmin}} \sum_{i=1}^{N} \underbrace{e^{-y_i\,\hat{f}_{m-1}(\boldsymbol{x}_i)}}_{w_i((\boldsymbol{x}_i,\, y_i))} e^{-\hat{\beta}\,y_i\,\hat{G}(\boldsymbol{x}_i)}$$

two steps:
(1): weighted error rate minimized by $\hat{G}_m$:

$$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)}\, I\left(y_i \neq \hat{G}_m(\boldsymbol{x}_i)\right)}{\sum_{i=1}^{N} w_i^{(m)}}$$

→ (2): $\hat{\beta}_m = \frac{1}{2}\log\frac{1-\mathrm{err}_m}{\mathrm{err}_m}$

corresponds to sample weights w_i (in the trainings for $\hat{G}_m$), reflecting accuracy of current ensemble at each successive iteration (weights increased for incorrect predictions, decreased for correct ones)



Gradient Boosting

approximation of forward stagewise additive modeling (solution can be hard for general L):

$\hat{f}_m(\boldsymbol{x}) = \hat{f}_{m-1}(\boldsymbol{x}) + \hat{h}_m(\boldsymbol{x})$ by successively solving $\underset{\hat{h}_m}{\operatorname{argmin}} \sum_{i=1}^{N} L\left(y_i, \hat{f}_m(\boldsymbol{x}_i)\right)$

$$\tilde{L}\left(y_i, \hat{f}_m(\boldsymbol{x}_i)\right) \approx L\left(y_i, \hat{f}_{m-1}(\boldsymbol{x}_i)\right) + \hat{h}_m(\boldsymbol{x}_i) \underbrace{\left[\frac{\partial L\left(y_i, \hat{f}(\boldsymbol{x}_i)\right)}{\partial \hat{f}(\boldsymbol{x}_i)}\right]_{\hat{f}=\hat{f}_{m-1}}}_{g_{im}}$$

→ $\hat{h}_m(\boldsymbol{x}) \approx \underset{\hat{h}_m}{\operatorname{argmin}} \sum_{i=1}^{N} \hat{h}_m(\boldsymbol{x}_i)\, g_{im}$

$\hat{h}_m$ fitted to predict negative gradients of samples at each iteration (regression model), corresponding to gradient descent in functional space

- $\hat{h}_m$: usually, decision tree regressors of fixed size (gradient-boosted decision trees)
- works like this for both regression and classification (just different loss functions)

Decision Tree Sizes for Boosting

consider degree to which features interact with each other in given data set (see ANOVA expansion)

- decision trees with single split (aka decision stumps): covering no interaction effects, just main effects of individual features
- decision trees with two splits: covering second-order interactions
- decision trees with three splits: covering third-order interactions

(usually, interaction order not much higher than ∼5 in real-world data sets)

why boosting often works better than single large, low-bias model: uncorrelated learners, each focusing on a specific aspect of the data