What is Learning?

- memorizing something
- learning facts through observation and exploration
- improving motor and/or cognitive skills through practice
- organizing new knowledge into general, effective representations

"Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the task or tasks drawn from the same population more efficiently and more effectively the next time." -- Herb Simon

CS 2750 Machine Learning

Lecture 1

Machine Learning

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square, x4-8845

http://www.cs.pitt.edu/~milos/courses/cs2750/

Machine Learning

- The field of **machine learning** studies the design of computer programs (agents) capable of learning from past experience or adapting to changes in the environment

- The need for building agents capable of learning is everywhere
  - predictions in medicine,
  - text and web page classification,
  - speech recognition,
  - image/text retrieval,
  - commercial software

Learning

Learning process:

Learner (a computer program) processes data ***D*** representing past experiences and tries to either develop an appropriate response to future data, or describe in some meaningful way the data seen

Example:

Learner sees a set of patient cases (patient records) with corresponding diagnoses. It can either try:

- to predict the presence of a disease for future patients
- describe the dependencies between diseases, symptoms

Types of learning

- **Supervised learning**
  - Learning mapping between input **x** and desired output **y**
  - Teacher gives me y's for the learning purposes
- **Unsupervised learning**
  - Learning relations between data components
  - No specific outputs given by a teacher
- **Reinforcement learning**
  - Learning mapping between input **x** and desired output y
  - Critic does not give me y's but instead a signal (reinforcement) of how good my answer was
- **Other types of learning:**
  - **Concept learning, Active learning**

Supervised learning

Data: $D = \{d_1, d_2, .., d_n\}$ **a set of *n* examples**

$d_i = < \mathbf{x}_i, y_i >$

$\mathbf{x}_i$ is input vector, and y is desired output (given by a teacher)

Objective: learn the mapping $f : X \to Y$

s.t. $y_i \approx f(x_i)$ for all $i = 1, .., n$

Two types of problems:

- **Regression:** X discrete or continuous → Y is **continuous**
- **Classification:** X discrete or continuous → Y is **discrete**

Supervised learning examples

- **Regression:** Y is **continuous**

Debt/equity
Earnings
Future product orders

→ company stock price

- **Classification:** Y is **discrete**

→ Label "3"

Handwritten digit (array of 0,1s)

Unsupervised learning

- **Data:** $D = \{d_1, d_2, .., d_n\}$

  $d_i = \mathbf{x}_i$ vector of values

  No target value (output) y

- **Objective:**
  - learn relations between samples, components of samples

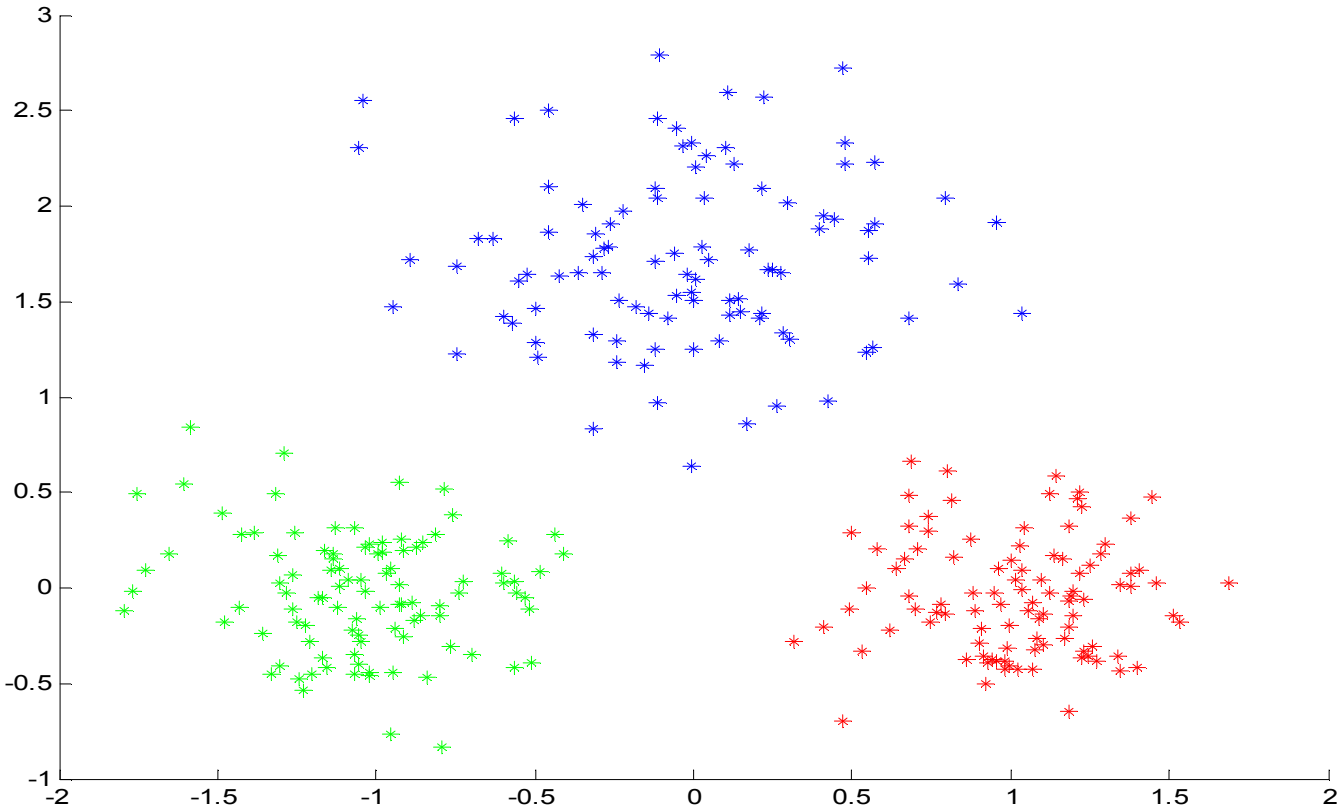
Types of problems:

- **Clustering**

  Group together "similar" examples, e.g. patient cases
- **Density estimation**
  - Model probabilistically the population of samples

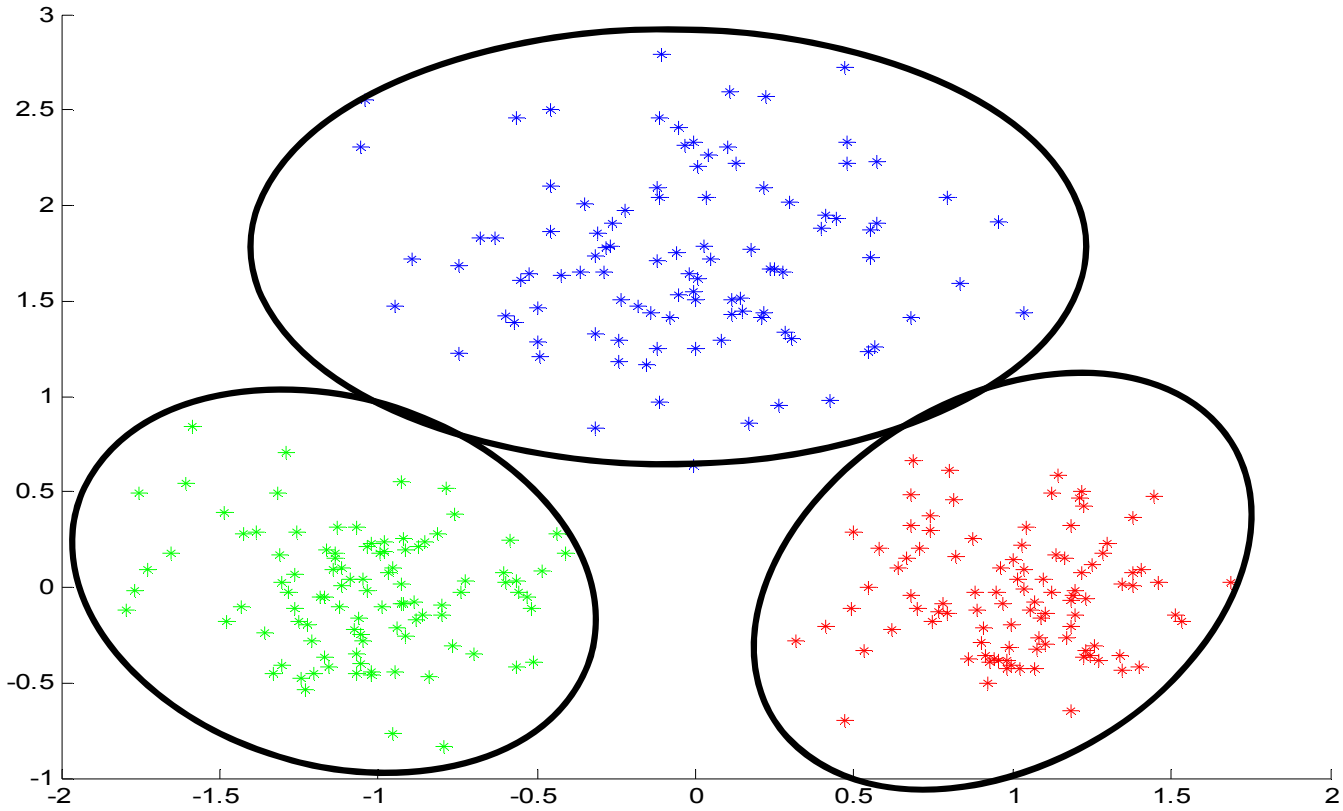
Unsupervised learning example

- **Clustering.** Group together similar examples $d_i = \mathbf{x}_i$



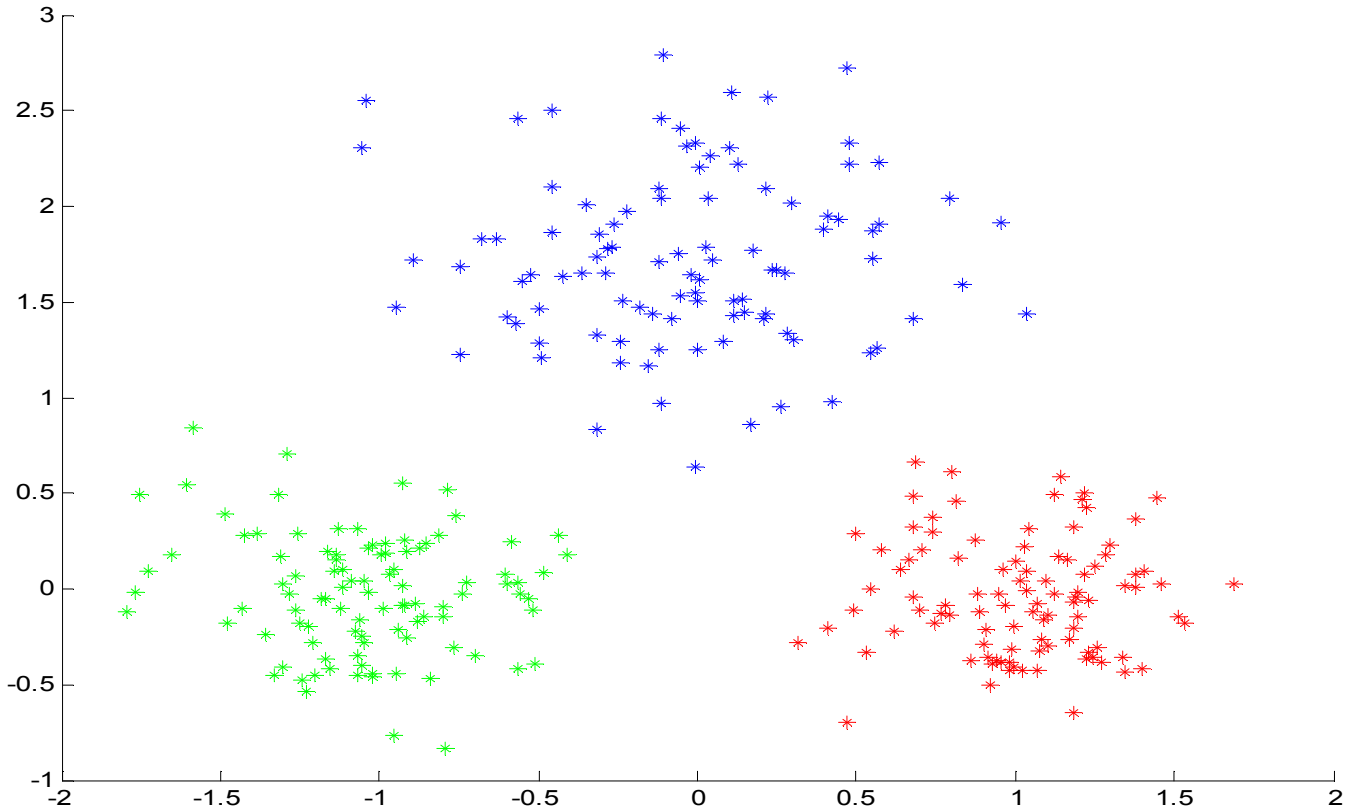
Unsupervised learning example

- **Clustering.** Group together similar examples $d_i = \mathbf{x}_i$



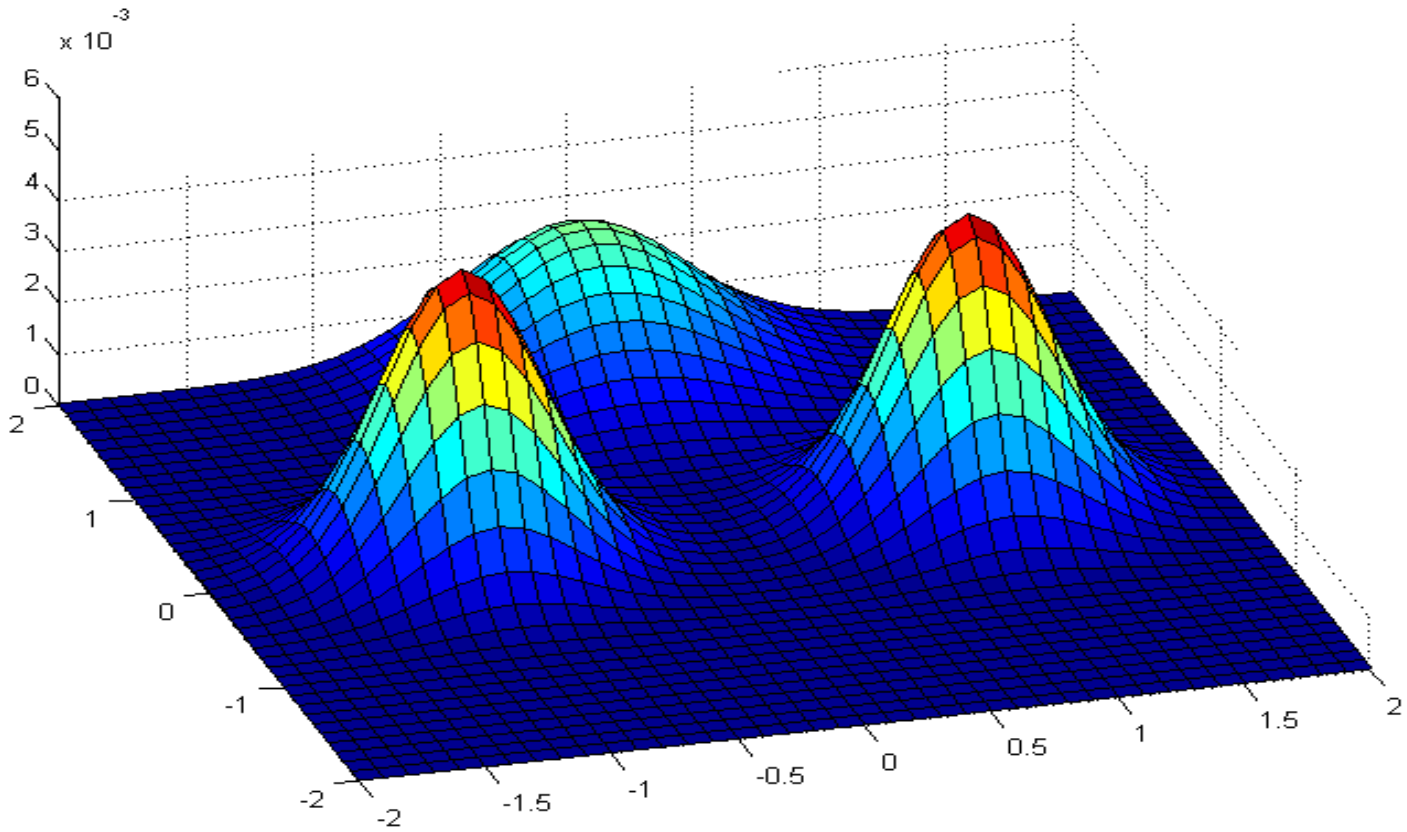
Unsupervised learning example

- **Density estimation.** We want to build the probability model P(**x**) of a population from which we draw examples $d_i = \mathbf{x}_i$



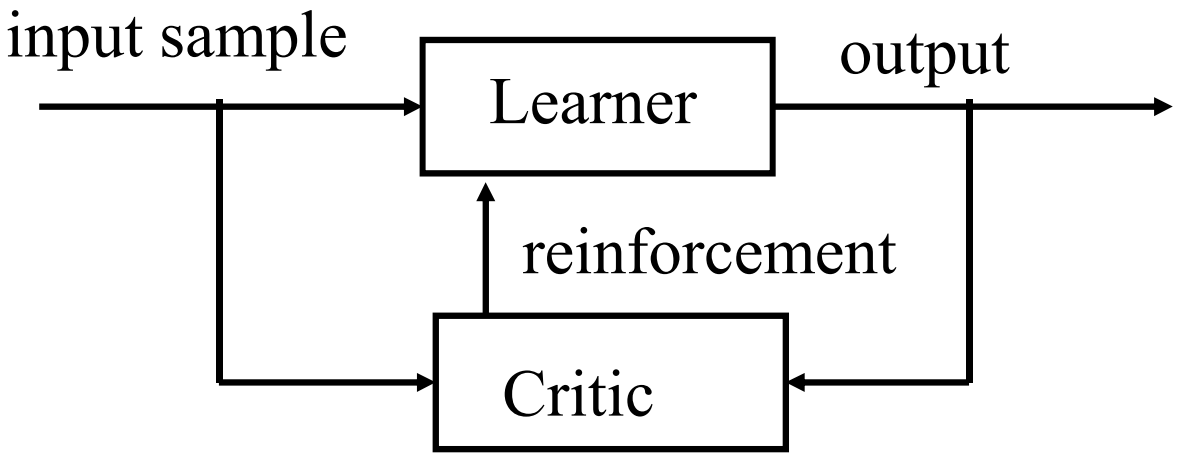
Unsupervised learning. Density estimation

- A probability density of a point in the two dimensional space
  - Model used here: **Mixture of Gaussians**



Reinforcement learning

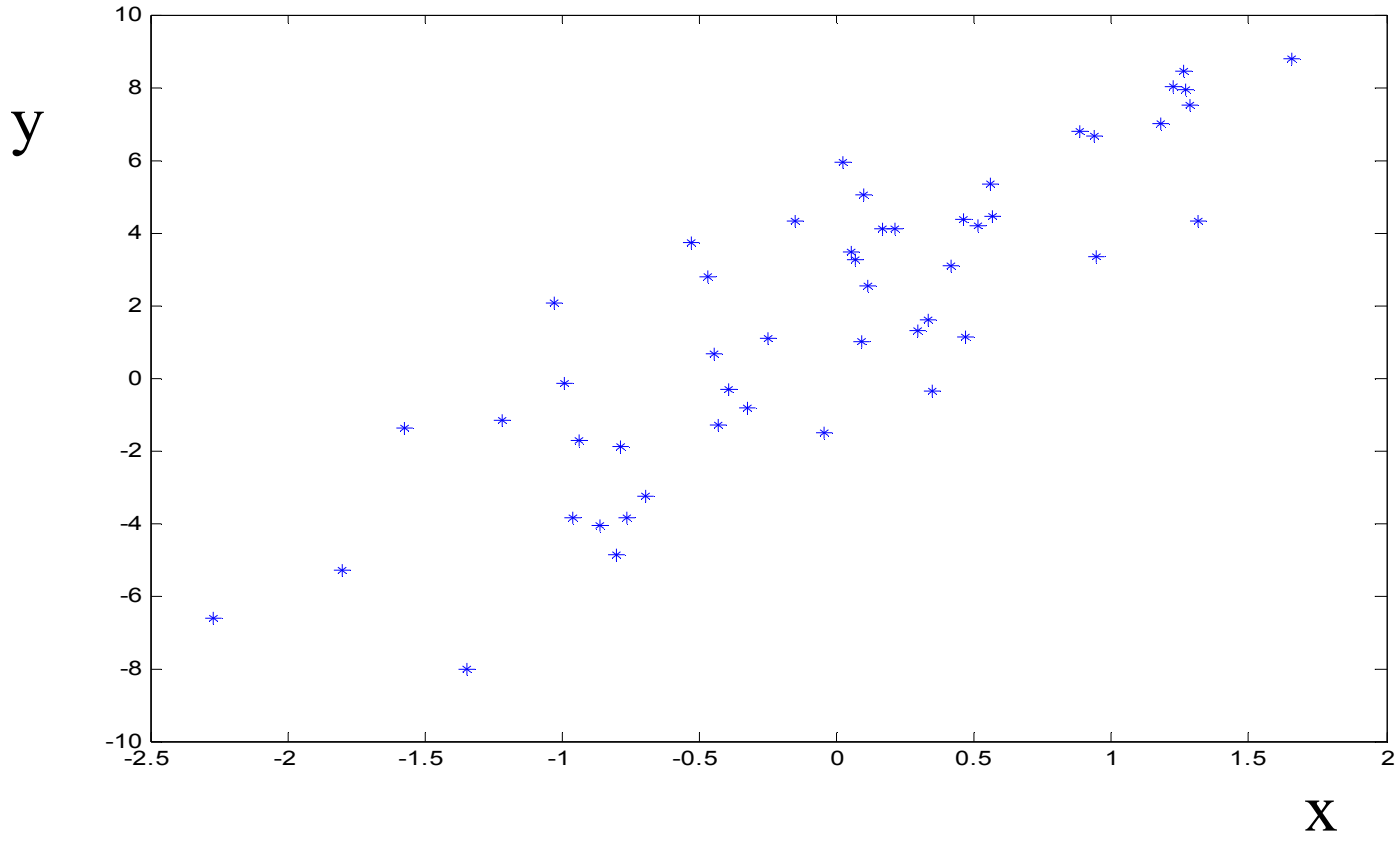
- We want to learn: $f : X \rightarrow Y$
- We see samples of **x** but not y
- Instead of y we get a feedback (reinforcement) from a **critic** about how good our output was



- The goal is to select outputs that lead to the best reinforcement

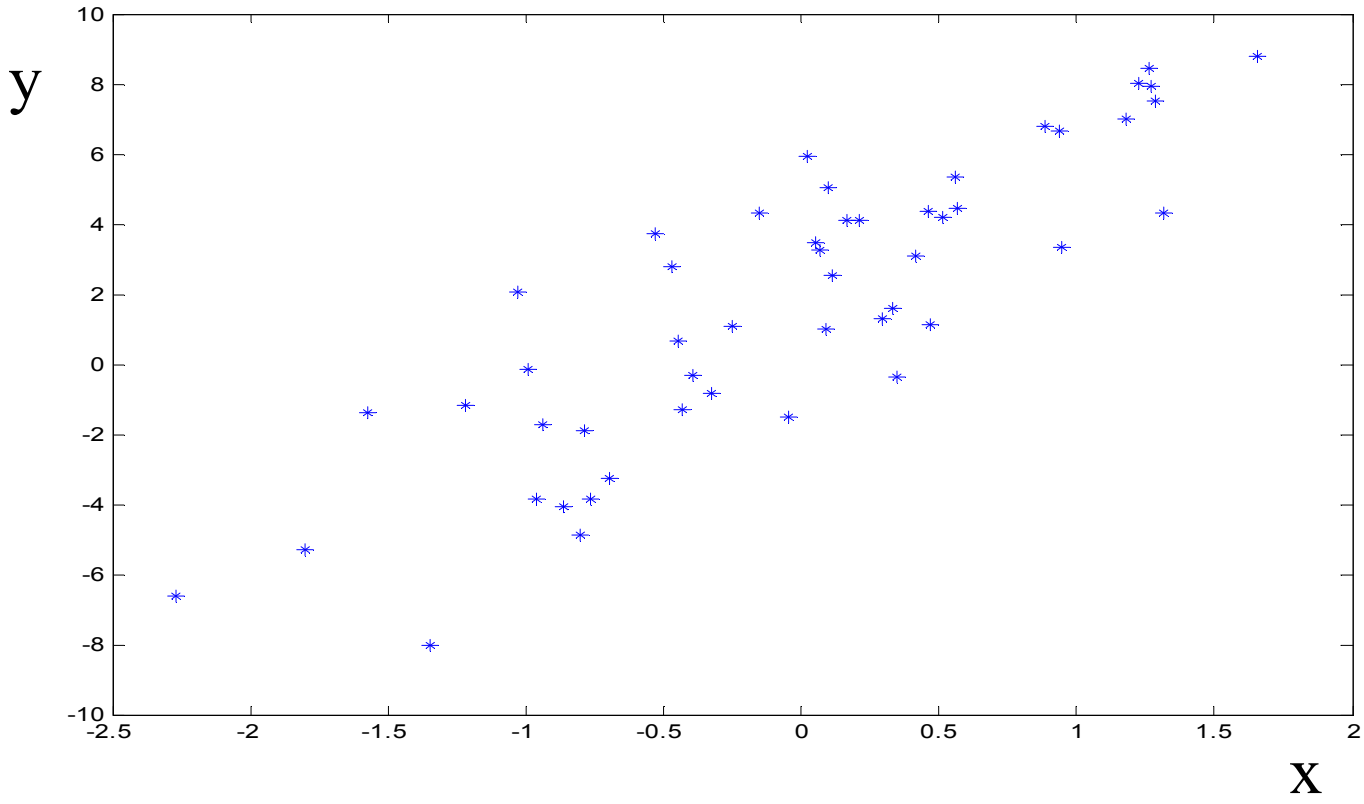
Learning: first look

- Assume we see examples of pairs (**x** , y) in D and we want to learn the mapping $f : X \rightarrow Y$ to predict y for some future **x**
- We get the data D - what should we do?



Learning: first look

- **Problem:** many possible functions $f : X \rightarrow Y$ exists for representing the mapping between **x** and y
- Which one to choose? Many examples still unseen!



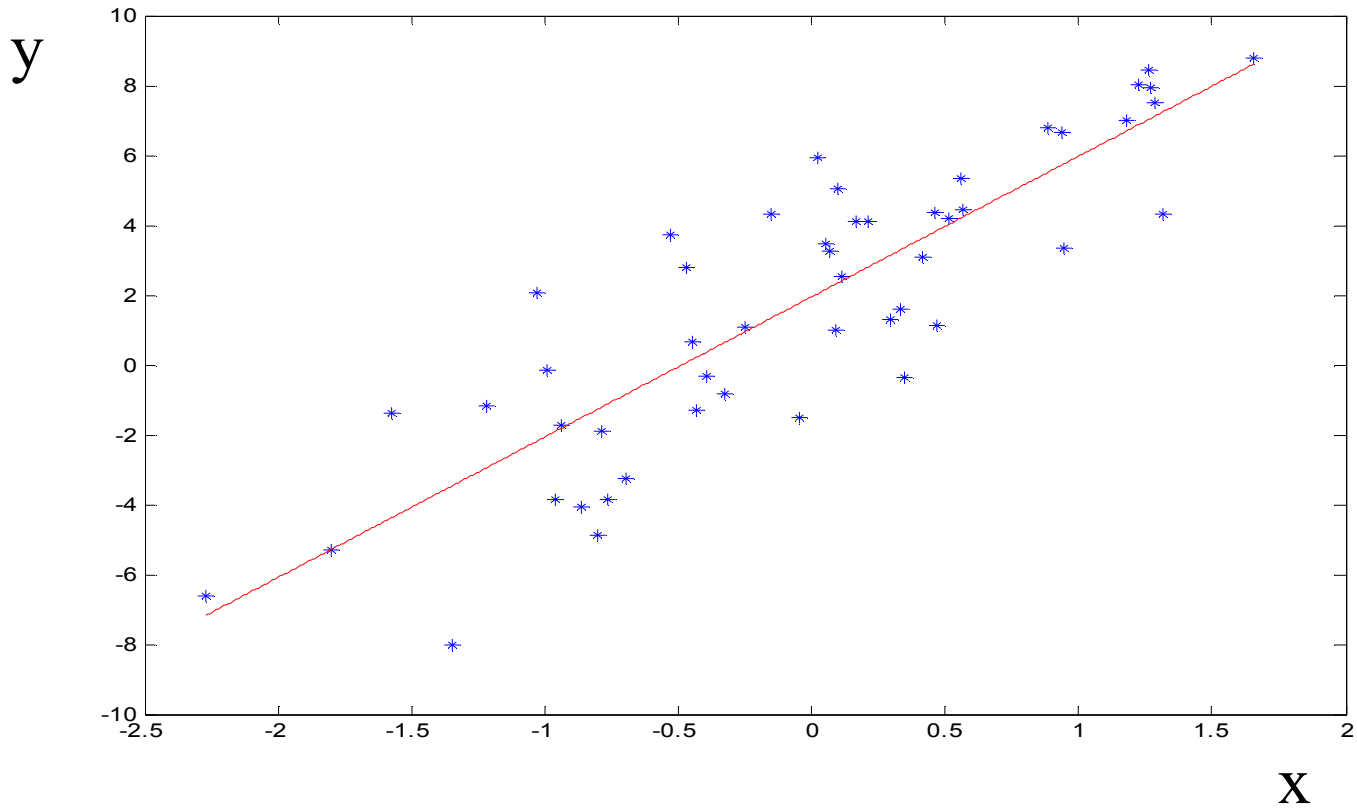
Learning: first look

- **Solution:** make an assumption about the model, say,

$$f(x) = ax + b + \varepsilon$$

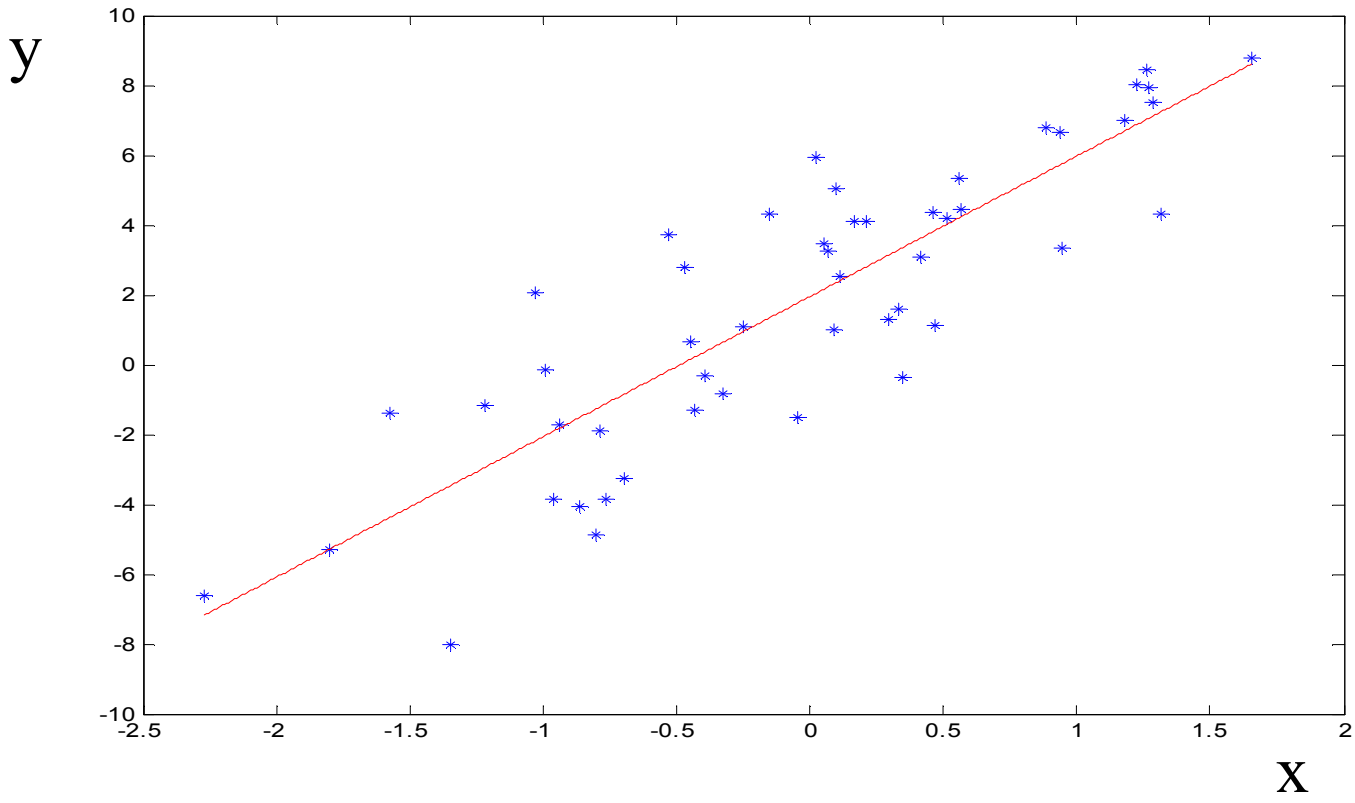
$\varepsilon = N(0, \sigma)$ - random (normally distributed) noise

- Restriction to a linear model is an example of learning bias



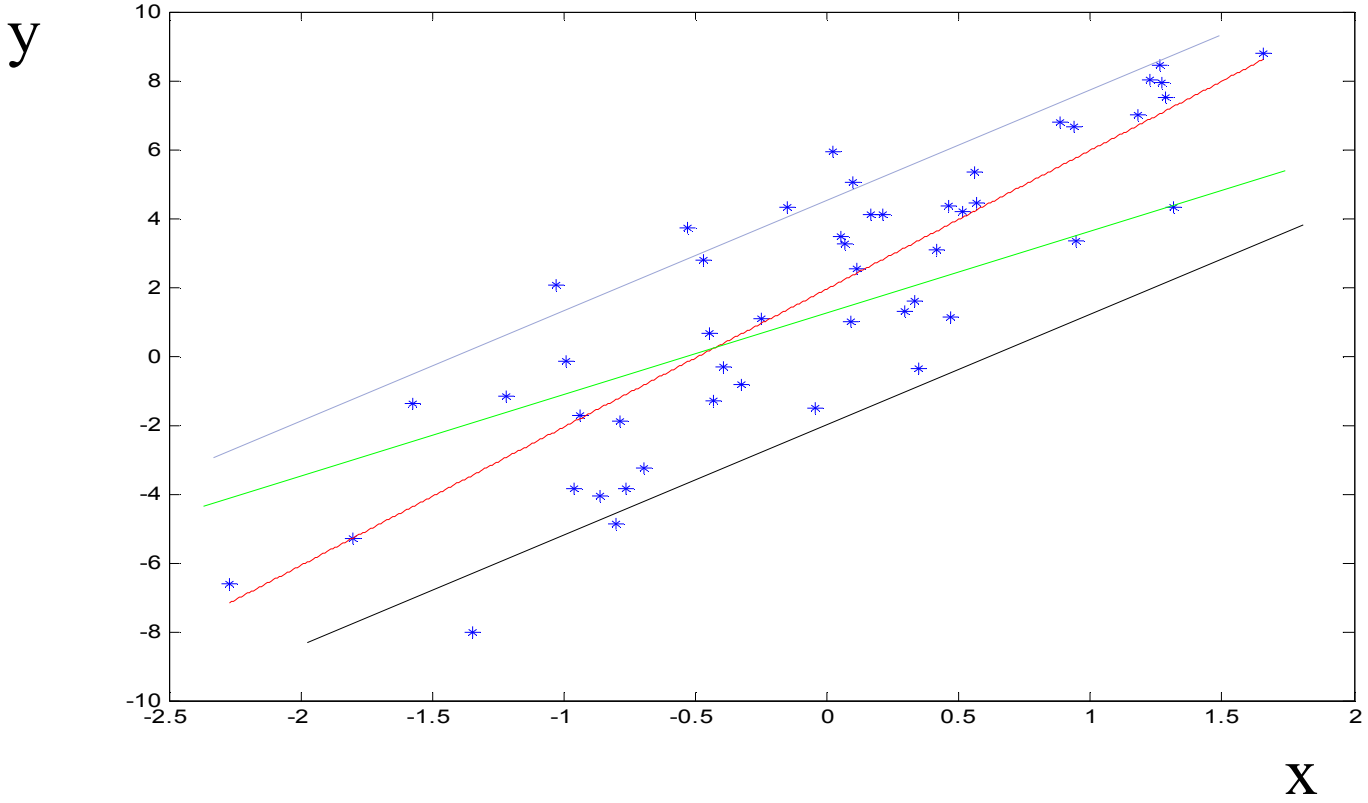
Learning: first look

- **Bias** provides the learner with some basis for choosing among possible representations of the function.
- **Forms of bias:** constraints, restrictions, model preferences
- **Important: There is no learning without a bias!**



Learning: first look

- Choosing a parametric model or a set of models is not enough
  Still too many functions $f(x) = ax + b + \varepsilon \quad \varepsilon = N(0, \sigma)$
  - One for every pair of parameters *a, b*



Fitting the data to the model

- We want the **best set** of model parameters

Objective: Find parameters that:

- reduce the misfit between the model **M** and observed data ***D***
- Or, (in other words) explain the data the best

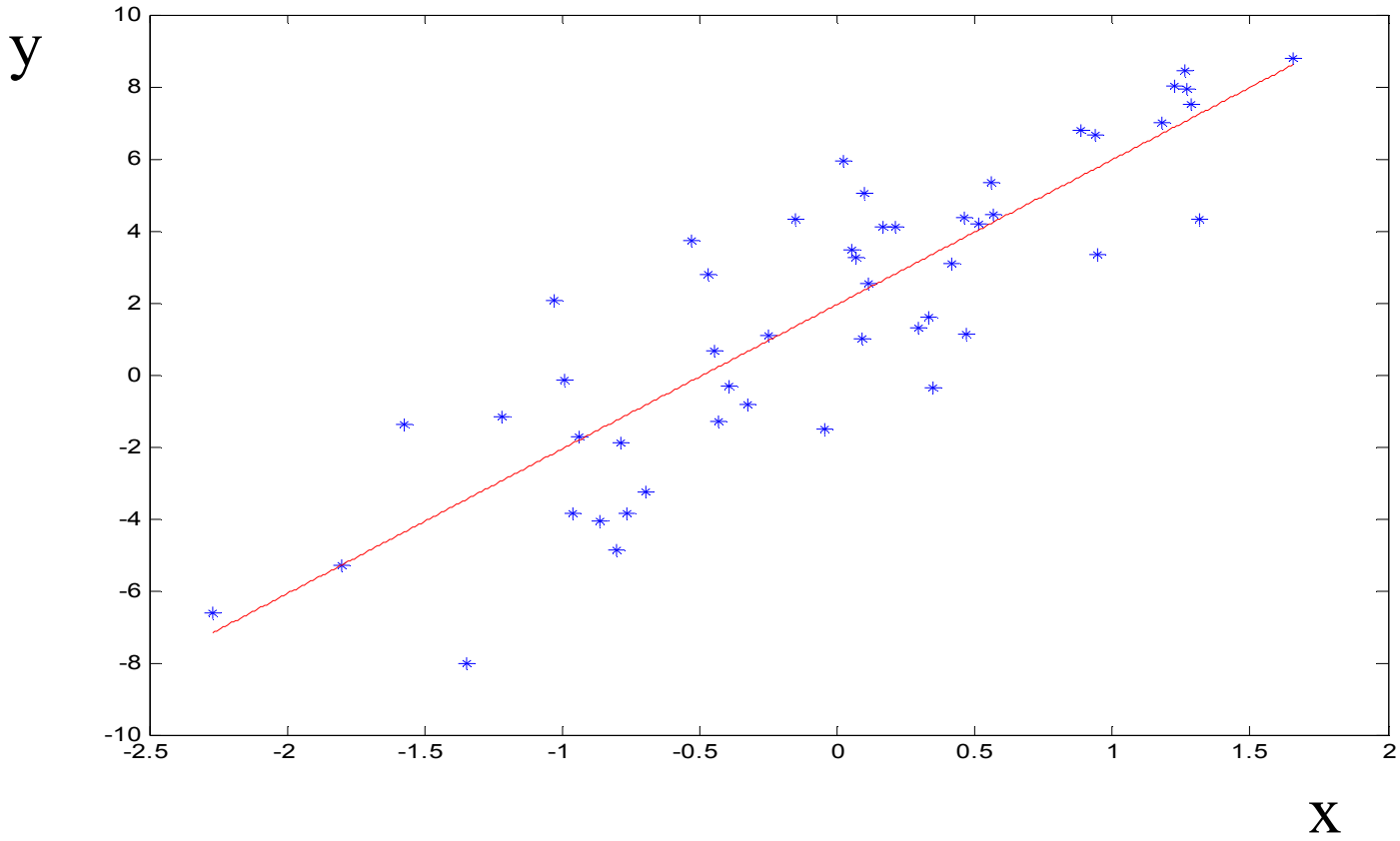
Objective function:

- **Error function: Measures the misfit between *D* and M**
- **Examples of error functions:**
  - Average Square Error $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$
  - Average misclassification error $\frac{1}{n}\sum_{i=1}^{n} 1_{y_i \neq f(x_i)}$

Average # of misclassified cases

Fitting the data to the model

- **Linear regression problem**
  - Minimizes the squared error function for the linear model
  - minimizes $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$



Learning: summary

Three basic steps:

- **Select a model** or a set of models (with parameters)

  E.g. $y = ax + b$

- **Select the error function** to be optimized

  E.g. $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$

- **Find the set of parameters optimizing the error function**
  - The model and parameters with the smallest error represent the best fit of the model to the data

But there are problems one must be careful about …

Learning

Problem

- We fit the model based on past examples observed in ***D***
- But ultimately we are interested in learning the mapping that performs well on the whole population of examples

Training data: **Data used to fit the parameters of the model**

Training error:

$$Error(D, f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

True (generalization) error (over the whole population):

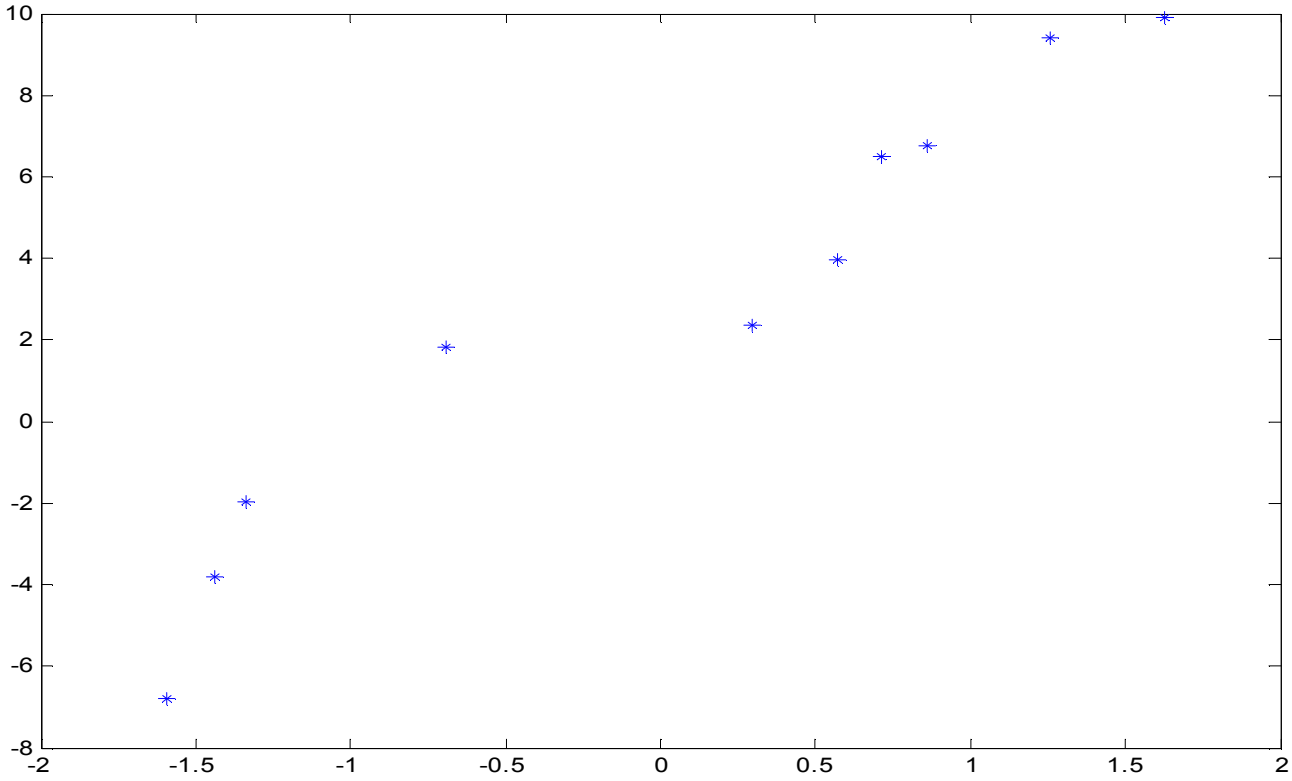
$$E_{(x,y)}[(y - f(x))^2] \qquad \text{Mean squared error}$$

Training error tries to approximate the true error !!!!

Does a good training error imply a good generalization error ?

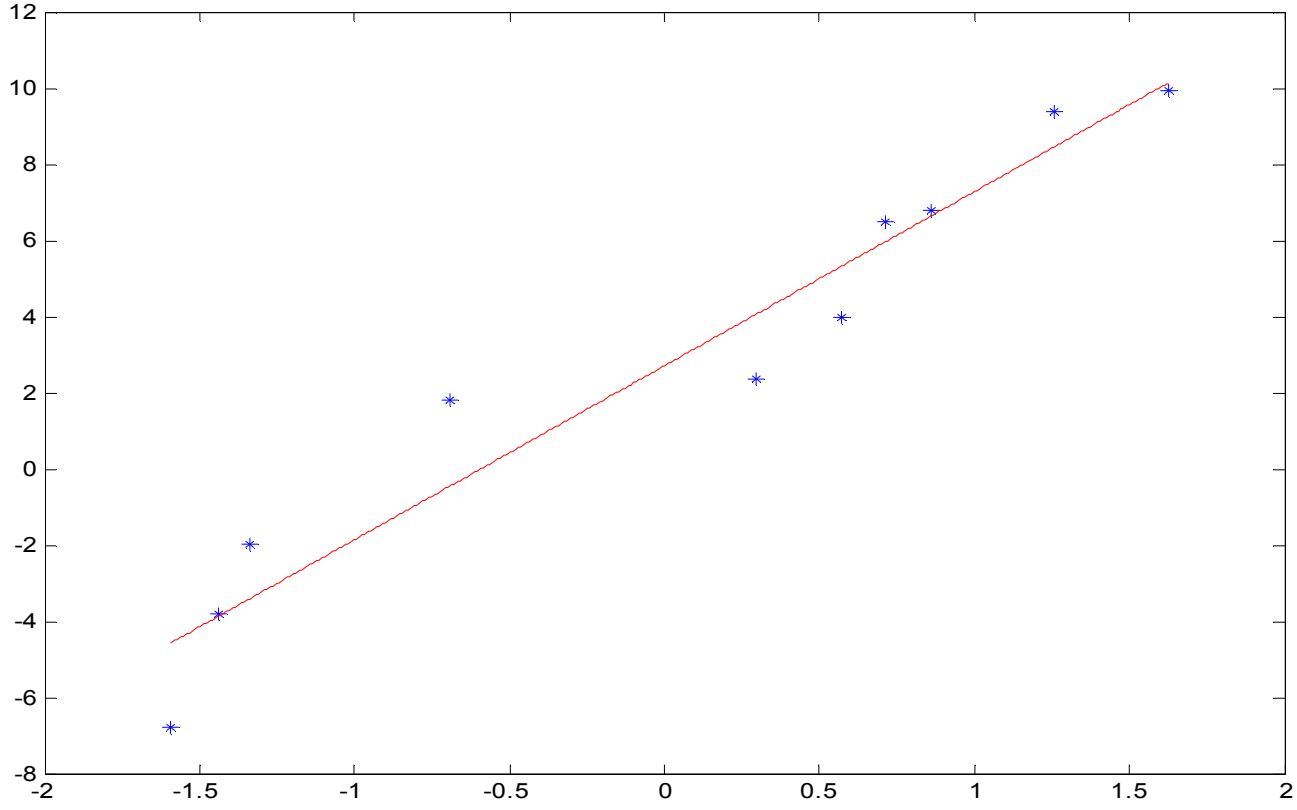
Overfitting

- Assume we have a set of 10 points and we consider polynomial functions as our possible models



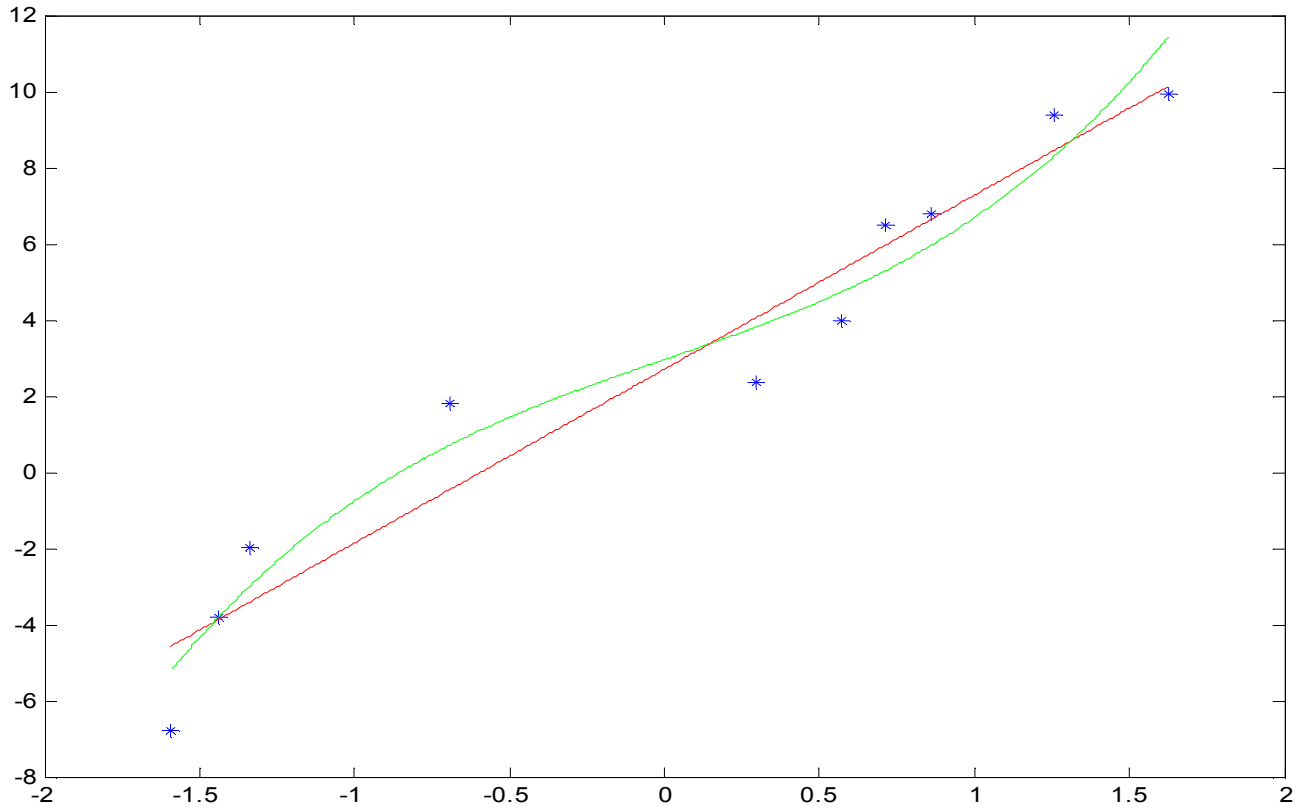
Overfitting

- Fitting a linear function with the square error
- Error is nonzero



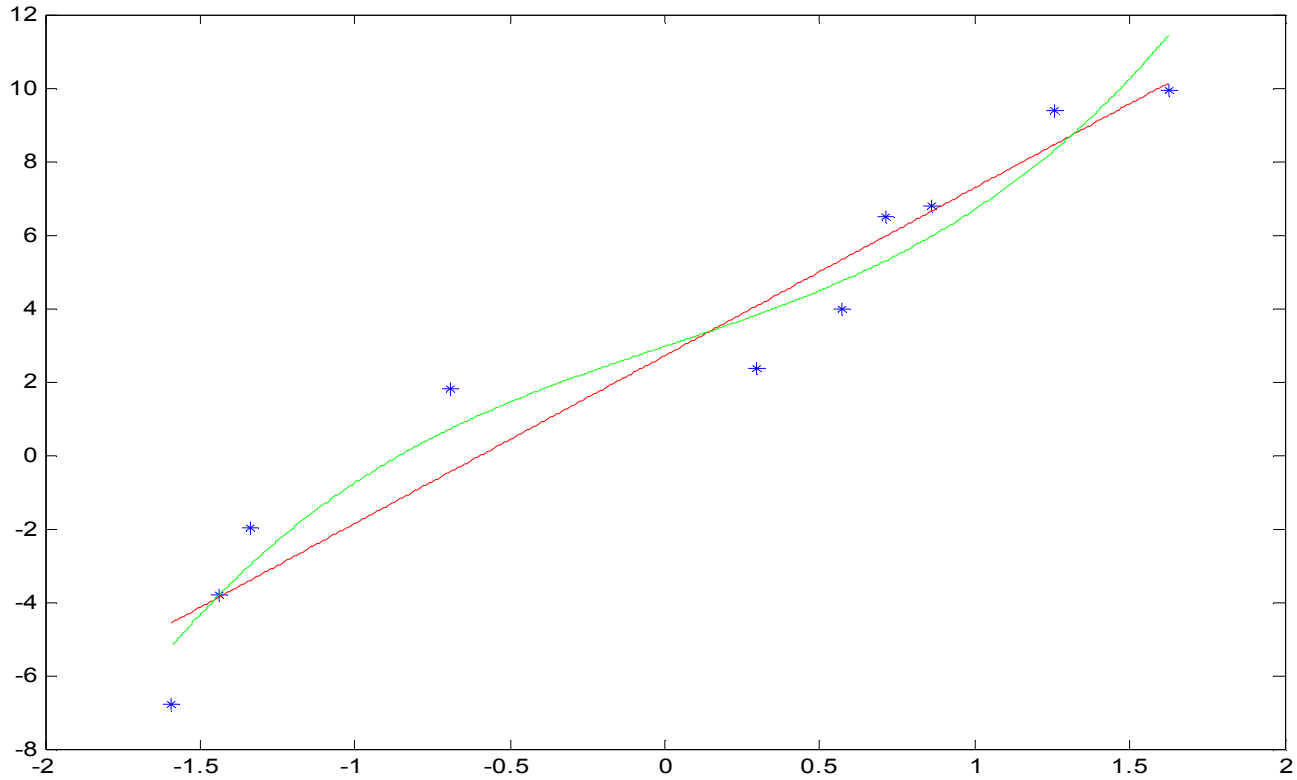
Overfitting

- Linear vs. cubic polynomial
- Higher order polynomial leads to a better fit, smaller error



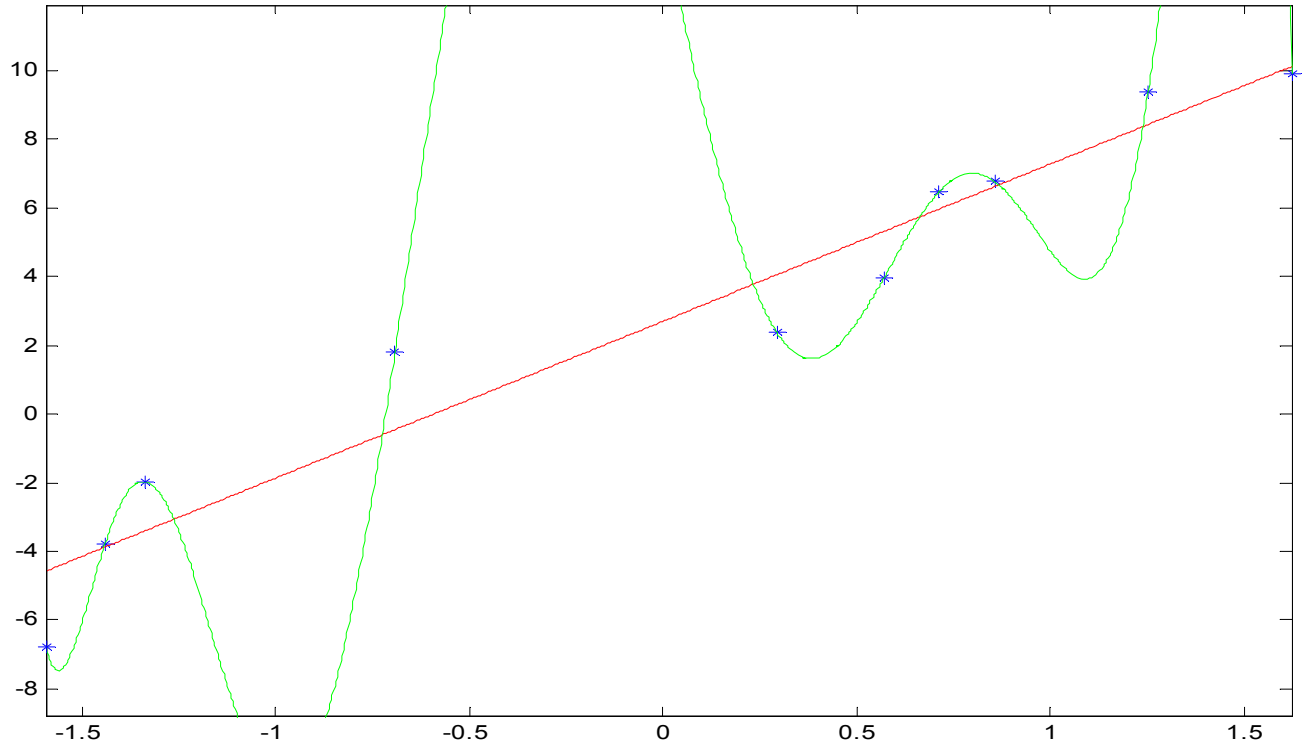
Overfitting

- Is it always good to minimize the error of the observed data?



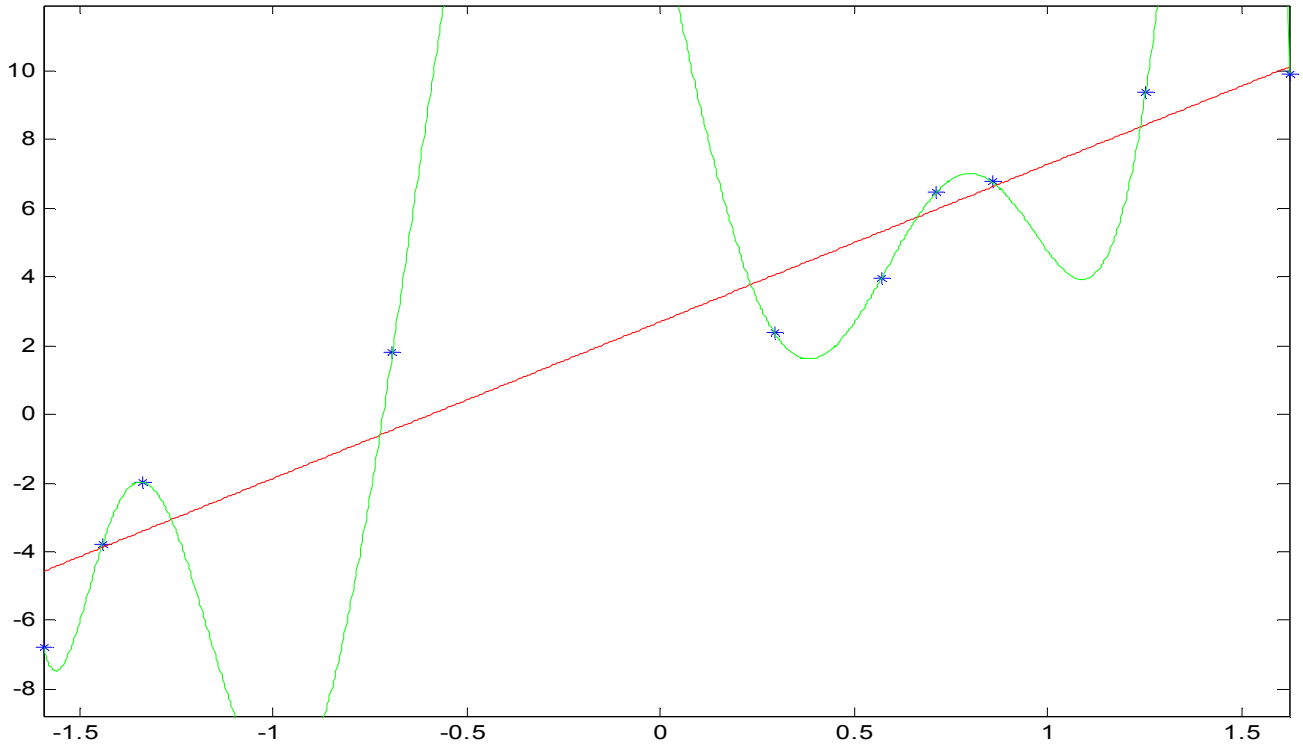
Overfitting

- For 10 data points, the degree 9 polynomial gives a perfect fit (Lagrange interpolation). Error is zero.
- Is it always good to minimize the training error?



Overfitting

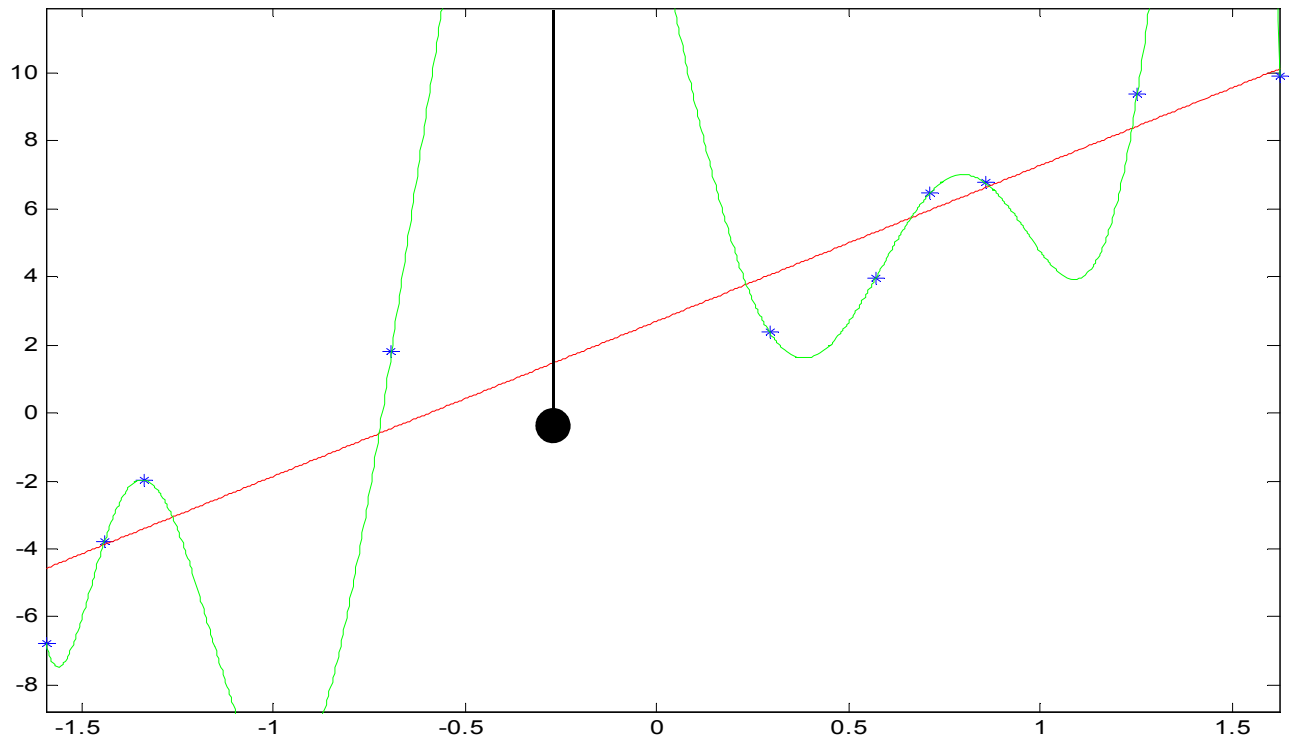
- For 10 data points, degree 9 polynomial gives a perfect fit (Lagrange interpolation). Error is zero.
- Is it always good to minimize the training error? NO !!
- **More important:** How do we perform on the unseen data?



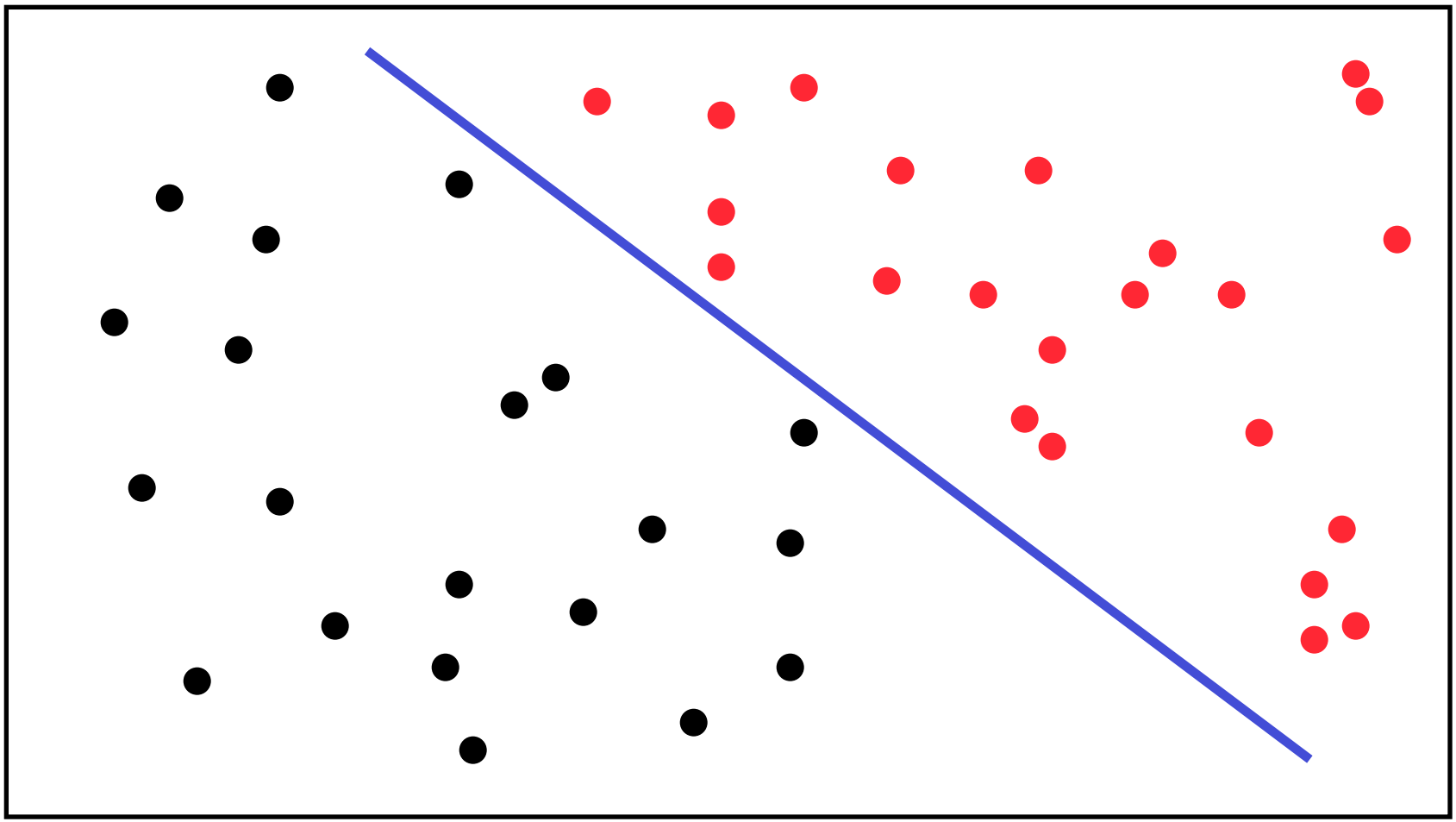
Overfitting

Situation when the training error is low and the generalization error is high. Causes of the phenomenon:

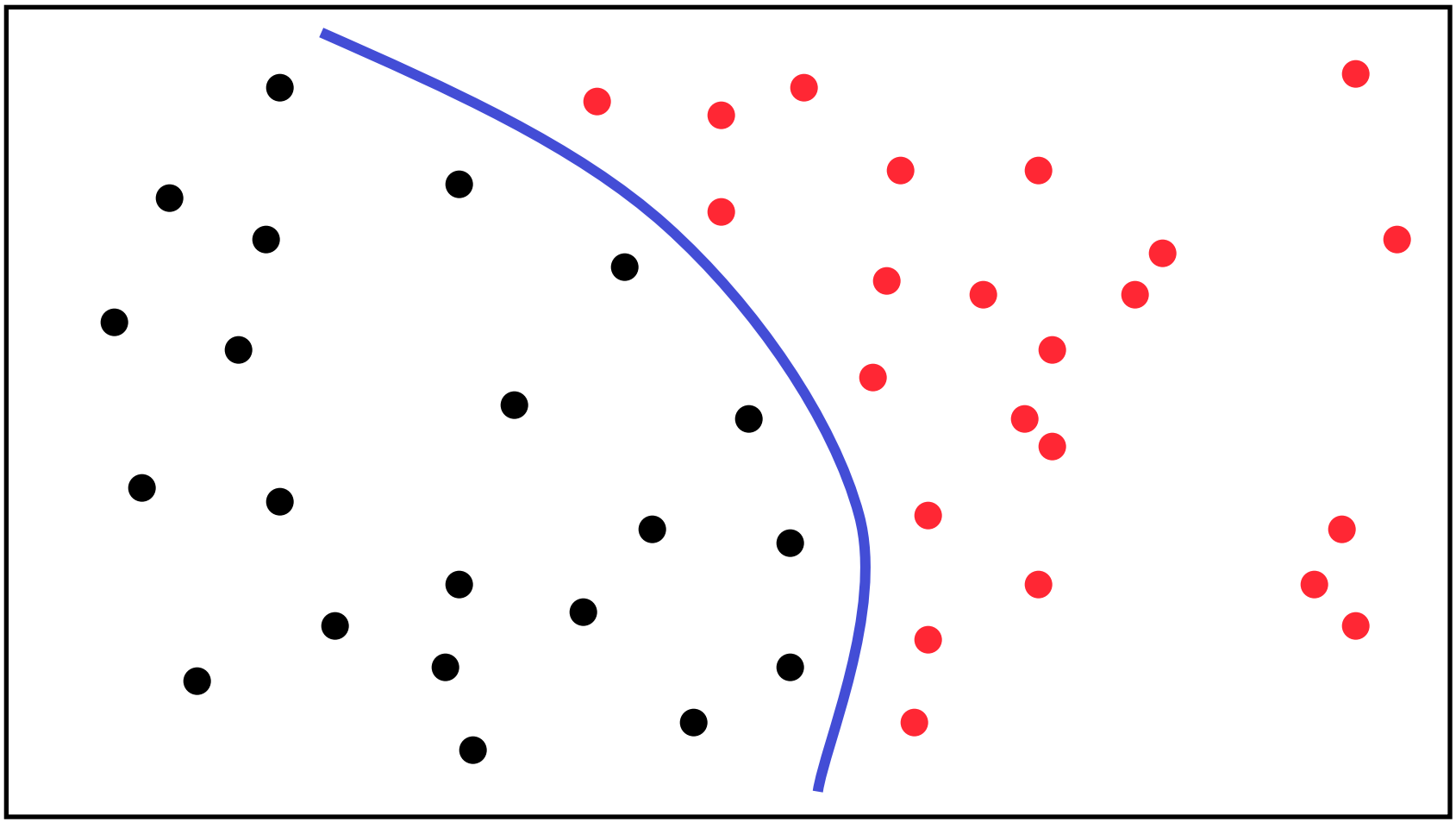
- Model with a large number of parameters (degrees of freedom)
- Small data size (as compared to the complexity of the model)



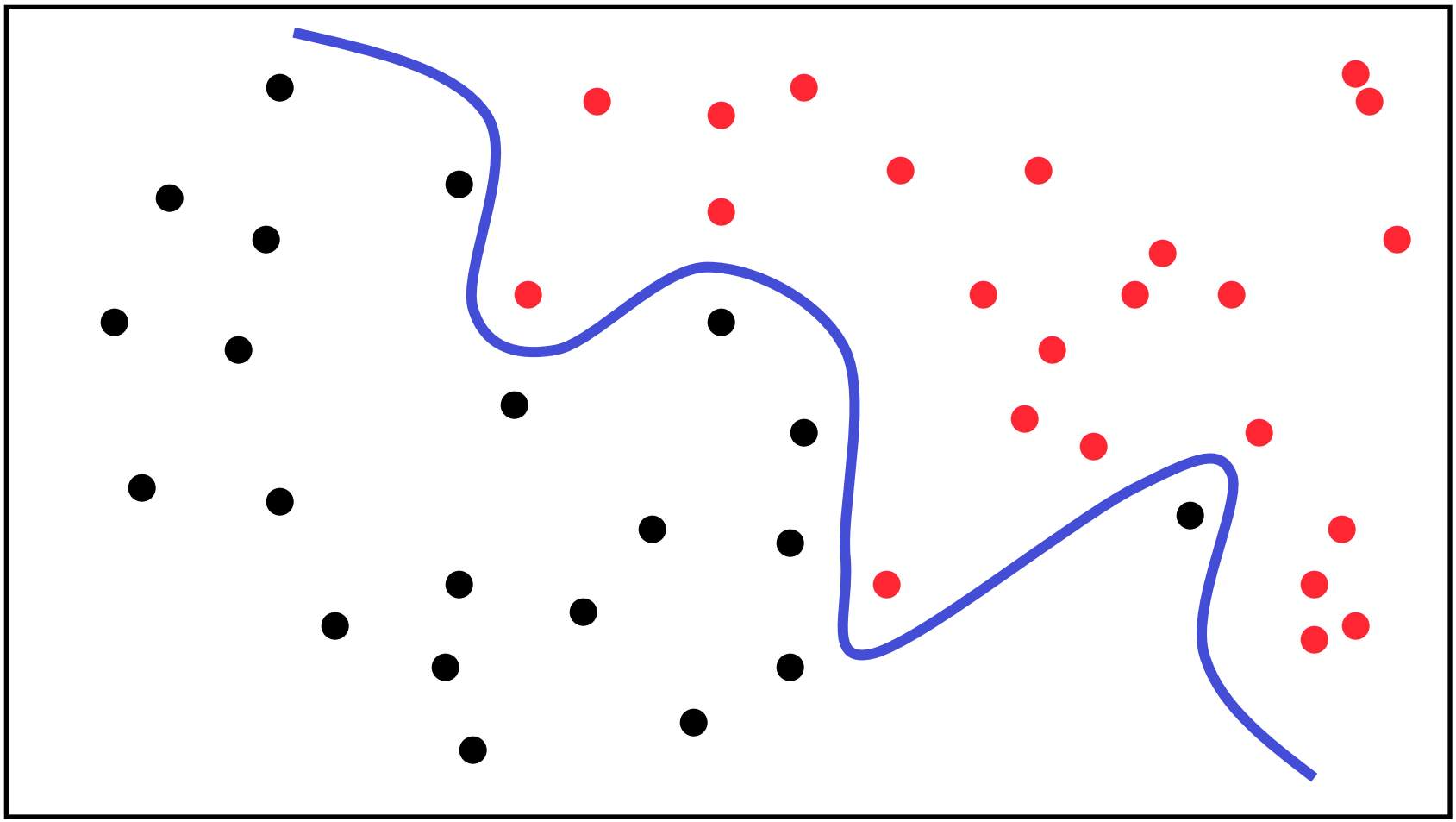
What's the right hypothesis?



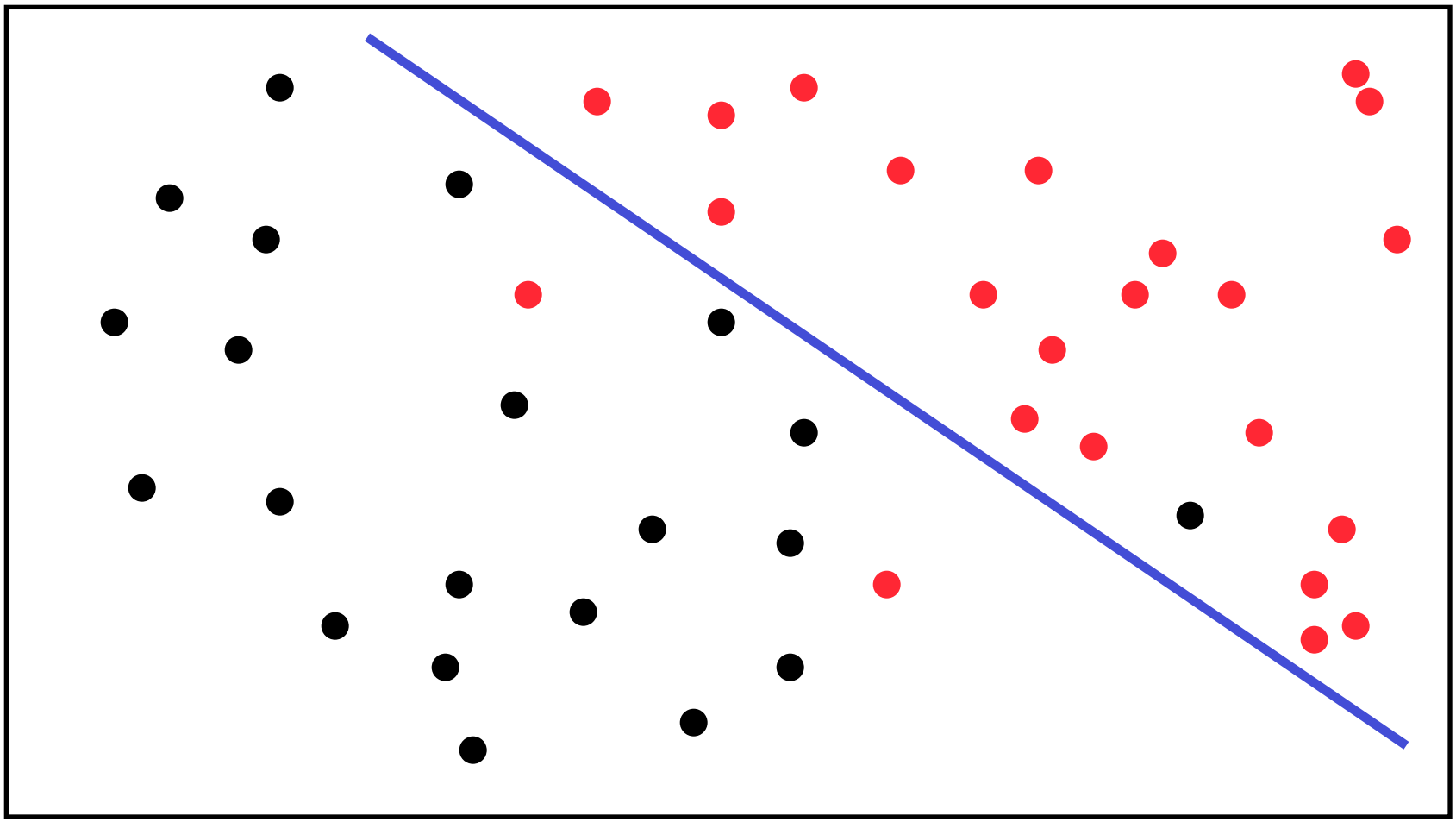
Now, what's the right hypothesis?



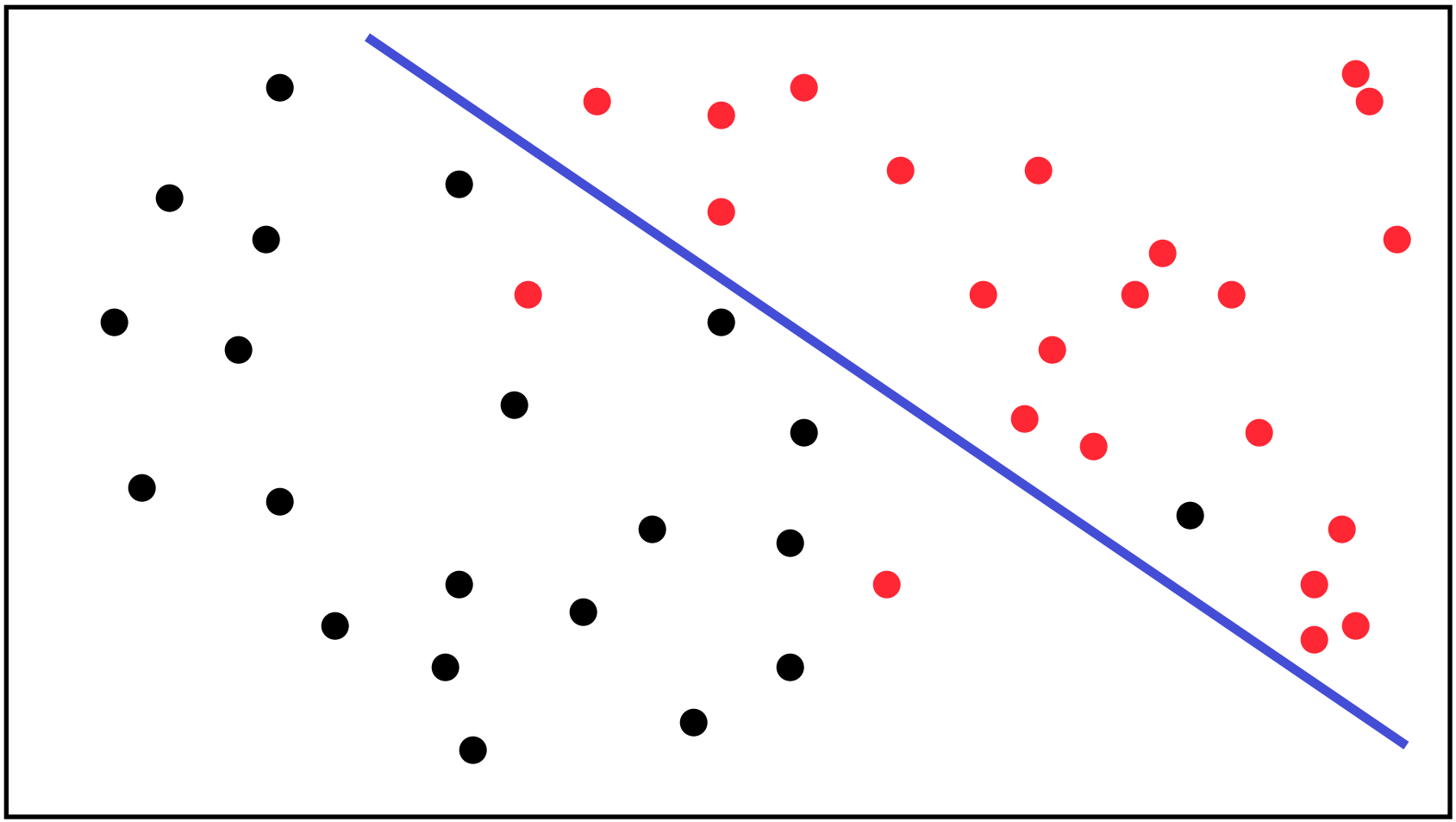
How about now? Answer 1



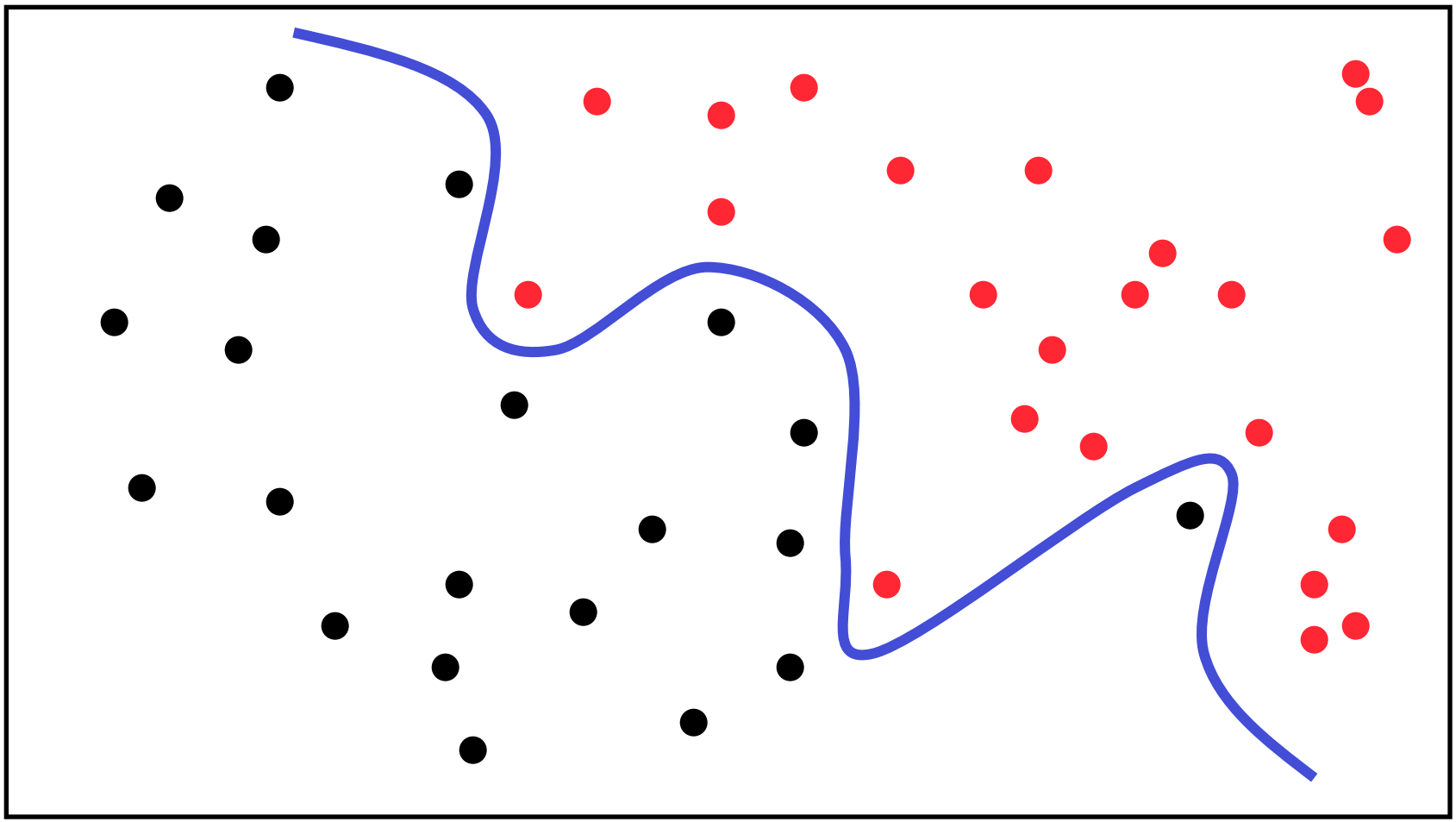
How about now? Answer 2



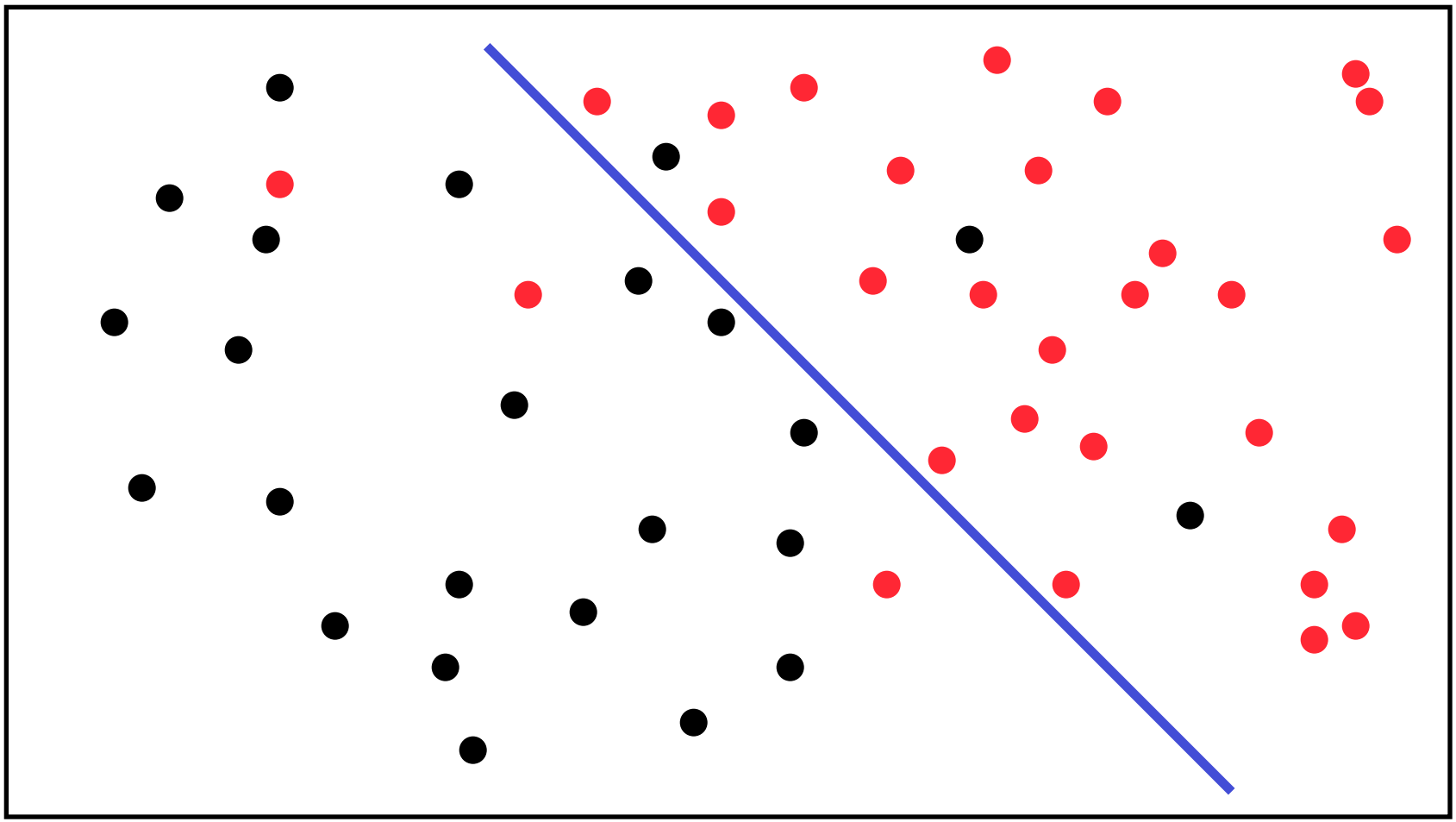
Bias vs Variance



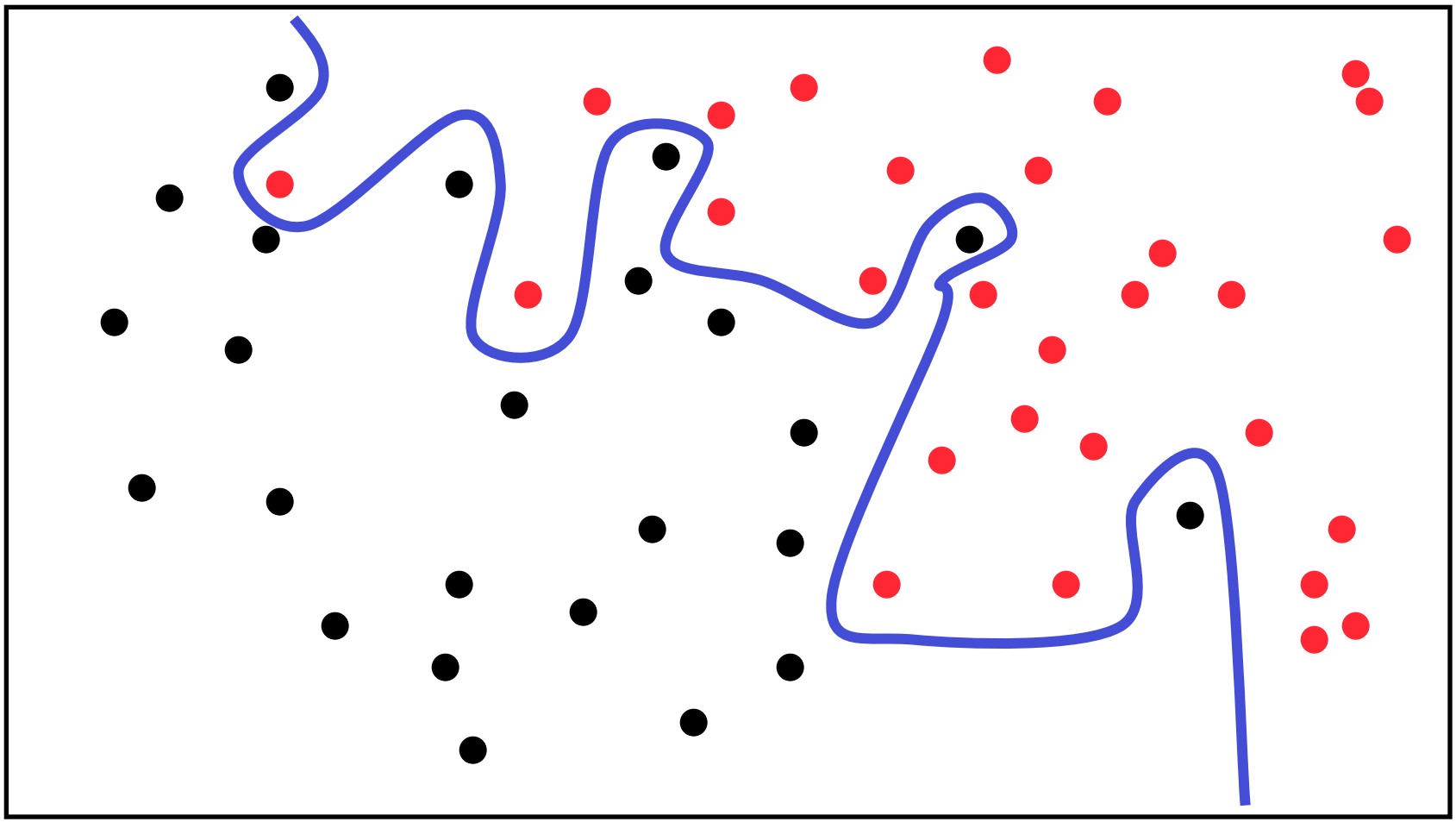
Bias vs Variance



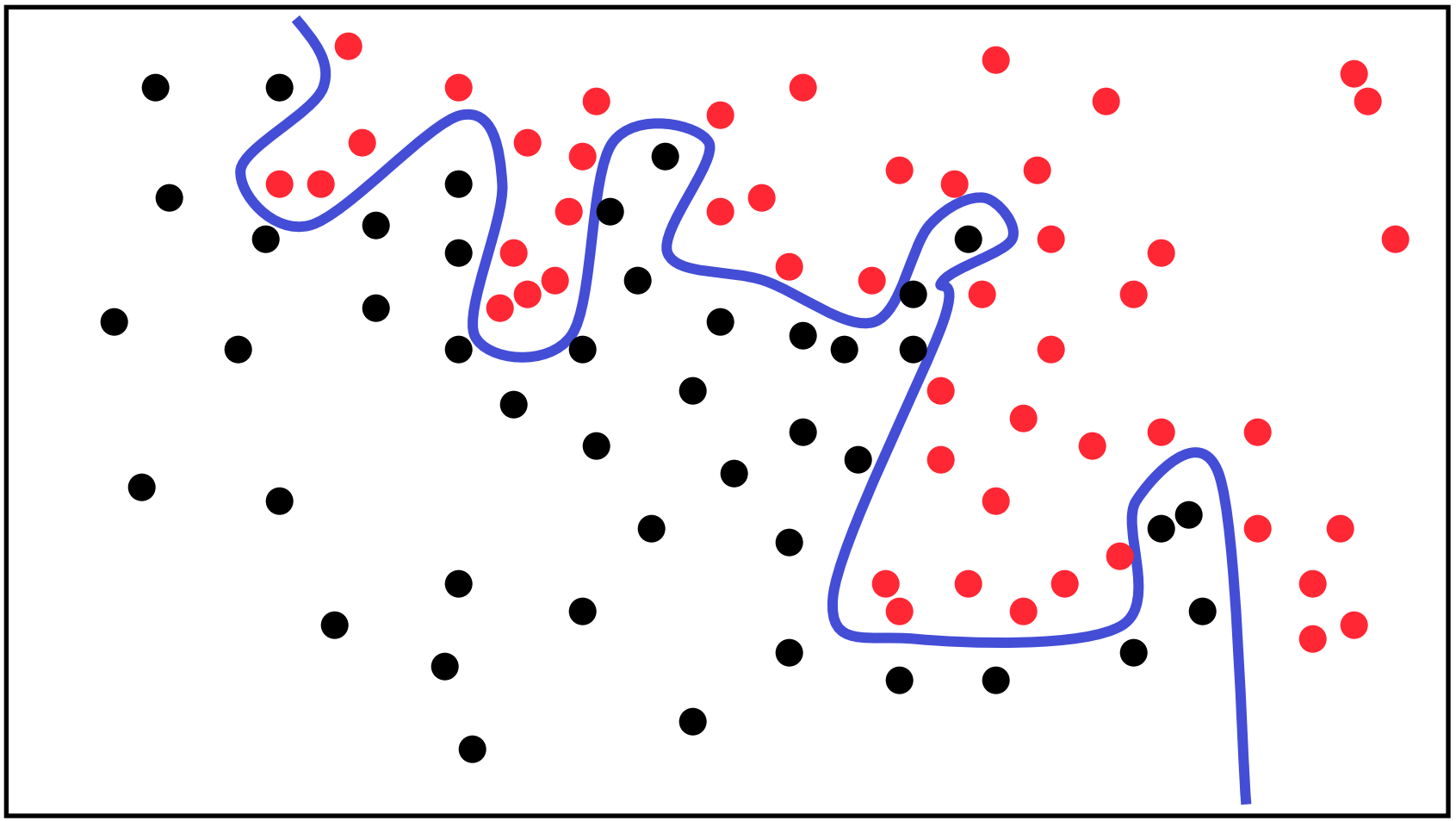
Bias vs Variance



Bias vs Variance



Bias vs Variance



How to evaluate the learner's performance?

- **Generalization error** is the true error for the population of examples we would like to optimize

$$E_{(x,y)}[(y - f(x))^2]$$

- But it cannot be computed exactly
- **Sample mean only approximates the true mean**

- **Optimizing (mean) training error can lead to the overfit, i.e.** training error may not reflect properly the generalization error

$$\frac{1}{n}\sum_{i=1,..n}(y_i - f(x_i))^2$$

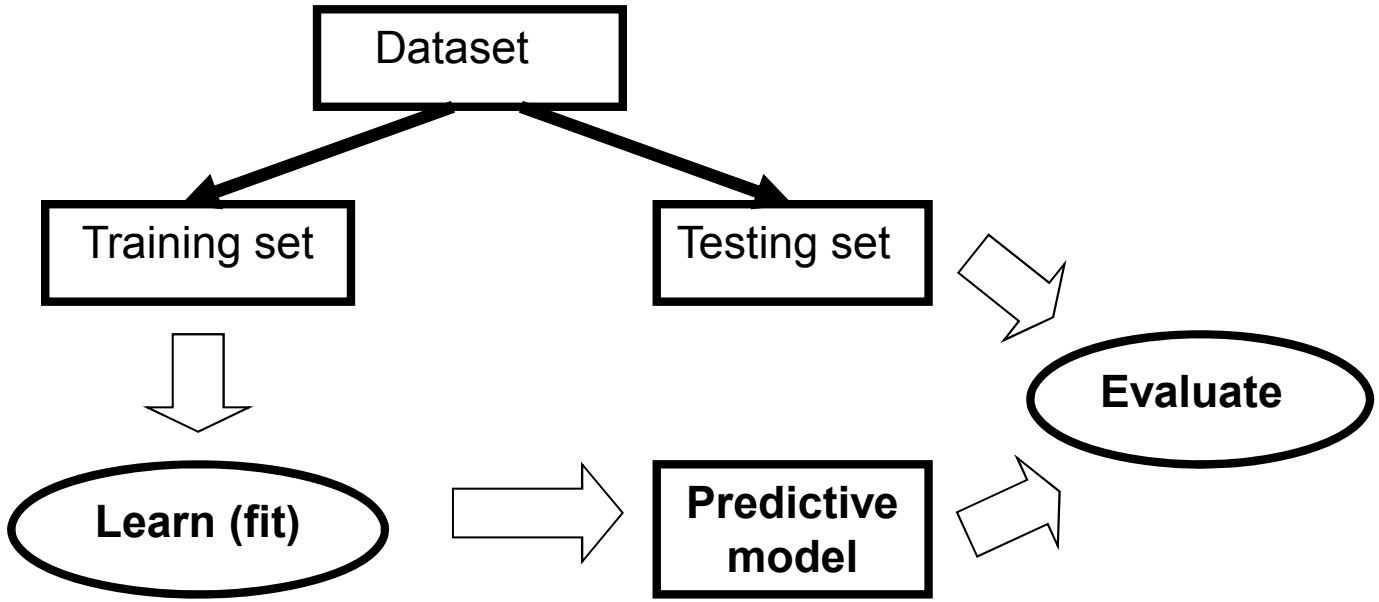
- So how to test the generalization error?

How to evaluate the learner's performance?

- **Generalization error** is the true error for the population of examples we would like to optimize
- **Sample mean only approximates it**
- **Two ways to assess the generalization error is:**
  - **Theoretical: Law of Large numbers**
    - statistical bounds on the difference between true and sample mean errors
  - **Practical:** Use a separate data set with m data samples to test the model
    - **(Mean) test error** $\frac{1}{m}\sum_{j=1,..m}(y_j - f(x_j))^2$

Testing of learning models

- **Simple holdout method**
  - Divide the data to the training and test data



  - Typically 2/3 training and 1/3 testing

Basic experimental setup to test the learner's performance

1. Take a dataset D and divide it into:

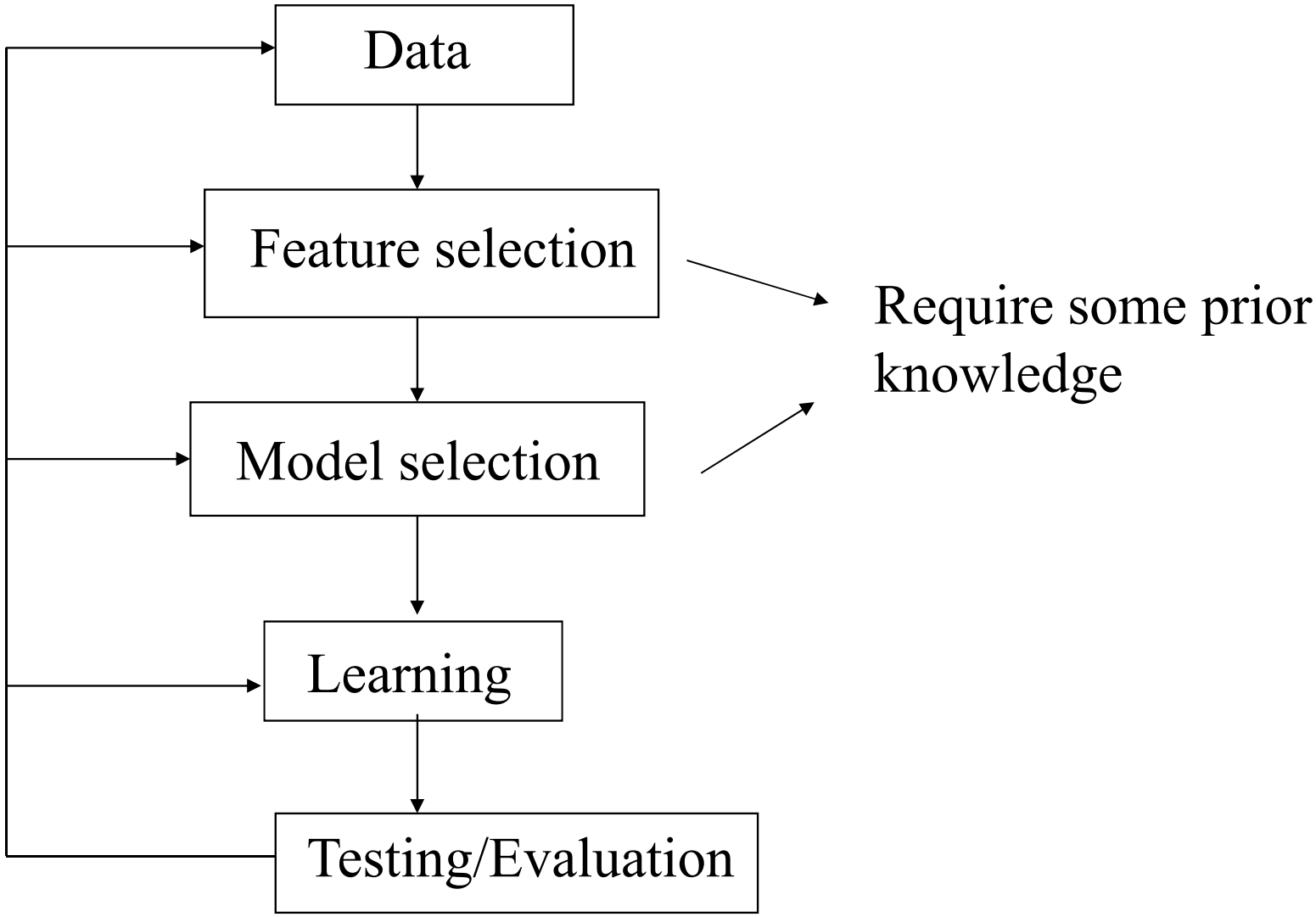
- **Training data set**
- **Testing data set**

2. Use the training set and your favorite ML algorithm to train the learner

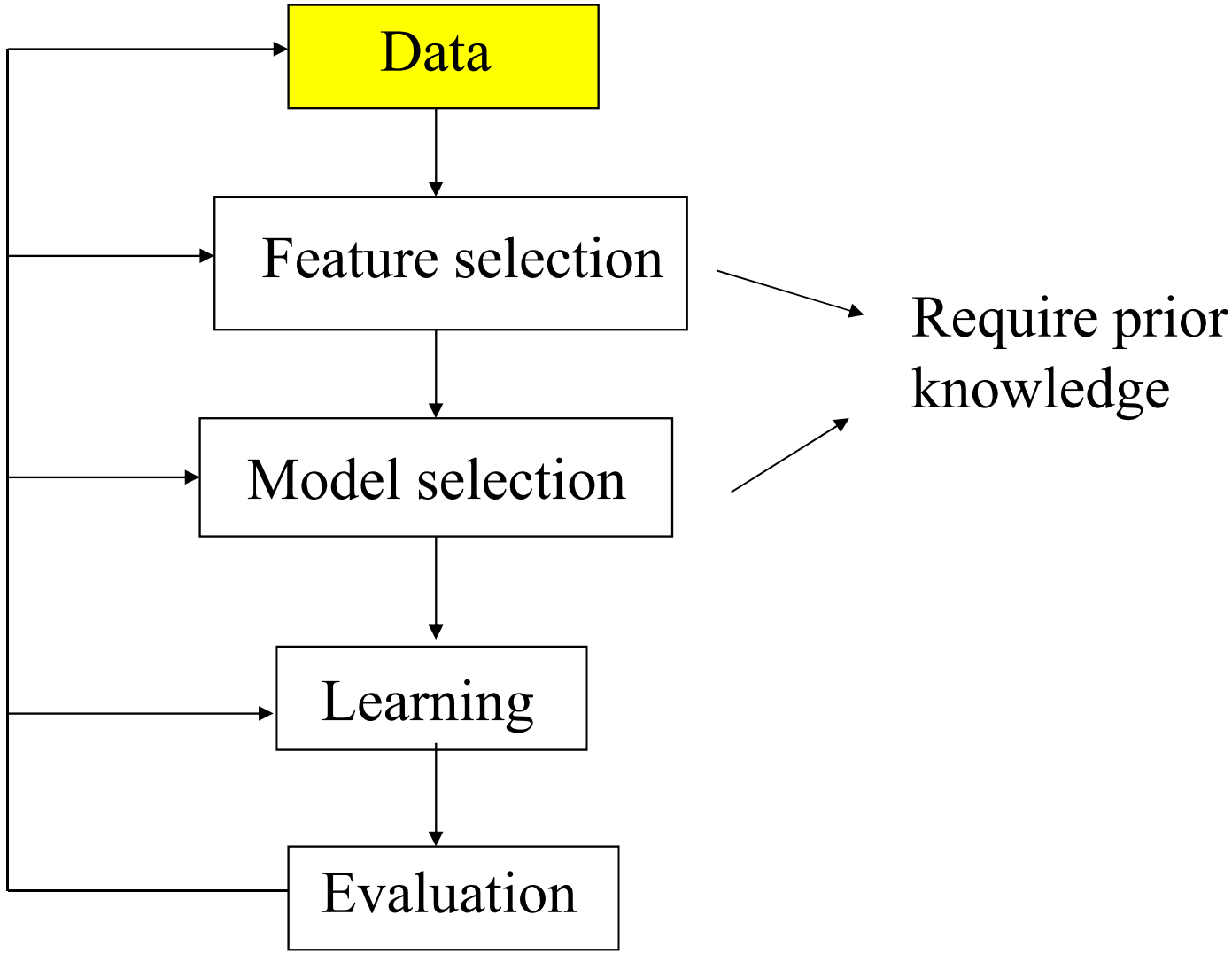
3. Test (evaluate) the learner on the testing data set

- The results on the testing set can be used to compare different learners powered with different models and learning algorithms

Design cycle



Design cycle



Data

Data may need a lot of:

- **Cleaning**
- **Preprocessing (conversions)**

Cleaning:

- **Get rid of errors, noise,**
- **Removal of redundancies**

Preprocessing:

- **Renaming**
- **Rescaling (normalization)**
- **Discretization**
- **Abstraction**
- **Aggregation**
- **New attributes**

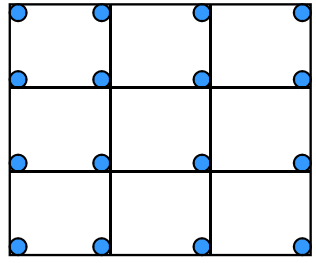
Data preprocessing

- **Renaming** (relabeling) categorical values to numbers
  - dangerous in conjunction with some learning methods
  - numbers will impose an order that is not warranted

High → 2
Normal → 1
Low → 0

True → 2
False → 1
Unknown → 0

- **Rescaling (normalization):** continuous values transformed to some range, typically [-1, 1] or [0,1].
- **Discretizations (binning):** continuous values to a finite set of discrete values



Data preprocessing

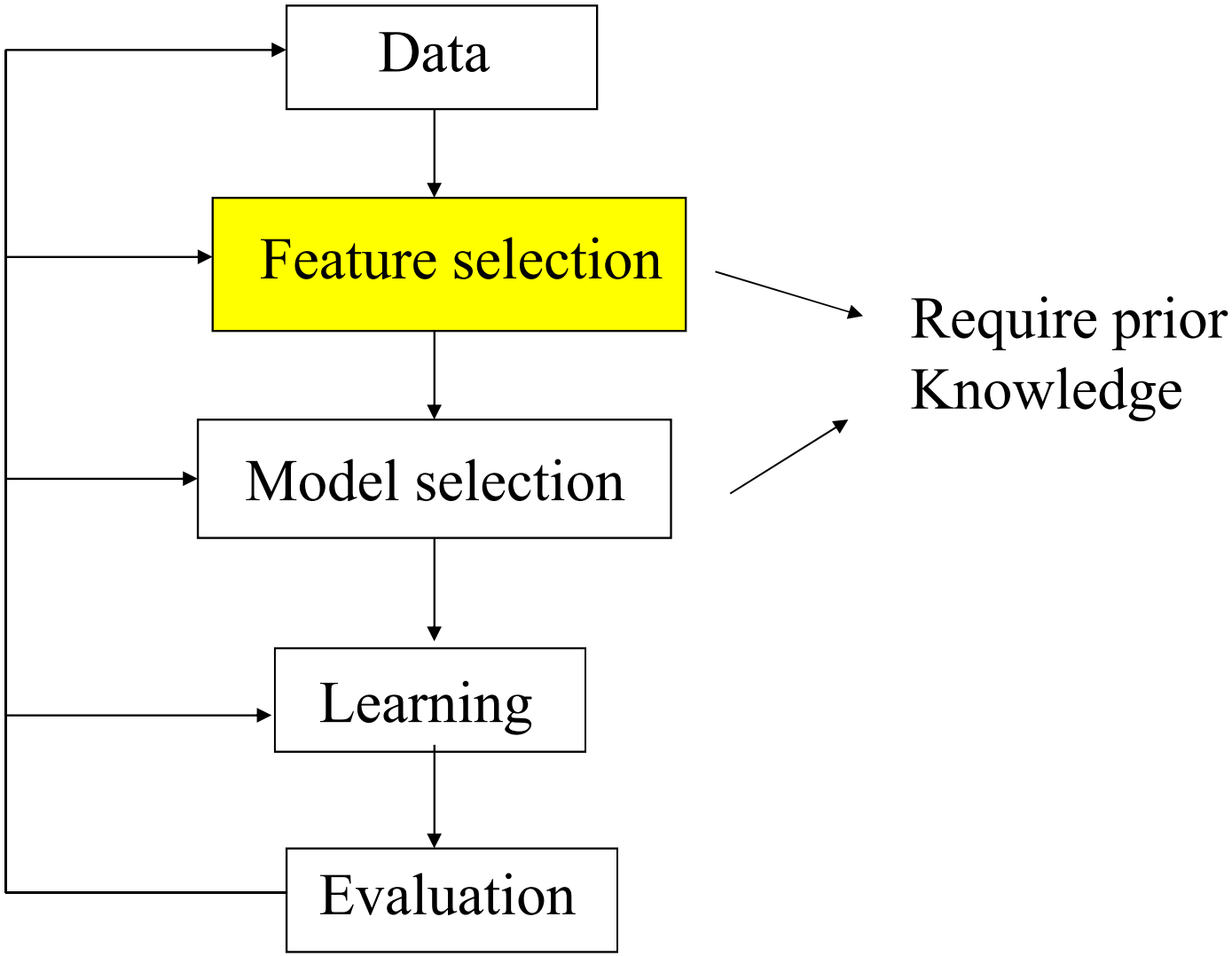
- **Abstraction:** merge together categorical values
- **Aggregation:** summary or aggregation operations, such minimum value, maximum value, average etc.
- **New attributes:**
  - example: obesity-factor = weight/height

Data biases

- **Watch out for data biases:**
  - Try to understand the data source
  - Make sure the data we make conclusions on are the same as data we used in the analysis
  - It is very easy to derive "unexpected" results when data used for analysis and learning are biased (pre-selected)

- **Results (conclusions) derived for biased data do not hold in general !!!**

Design cycle



Feature selection

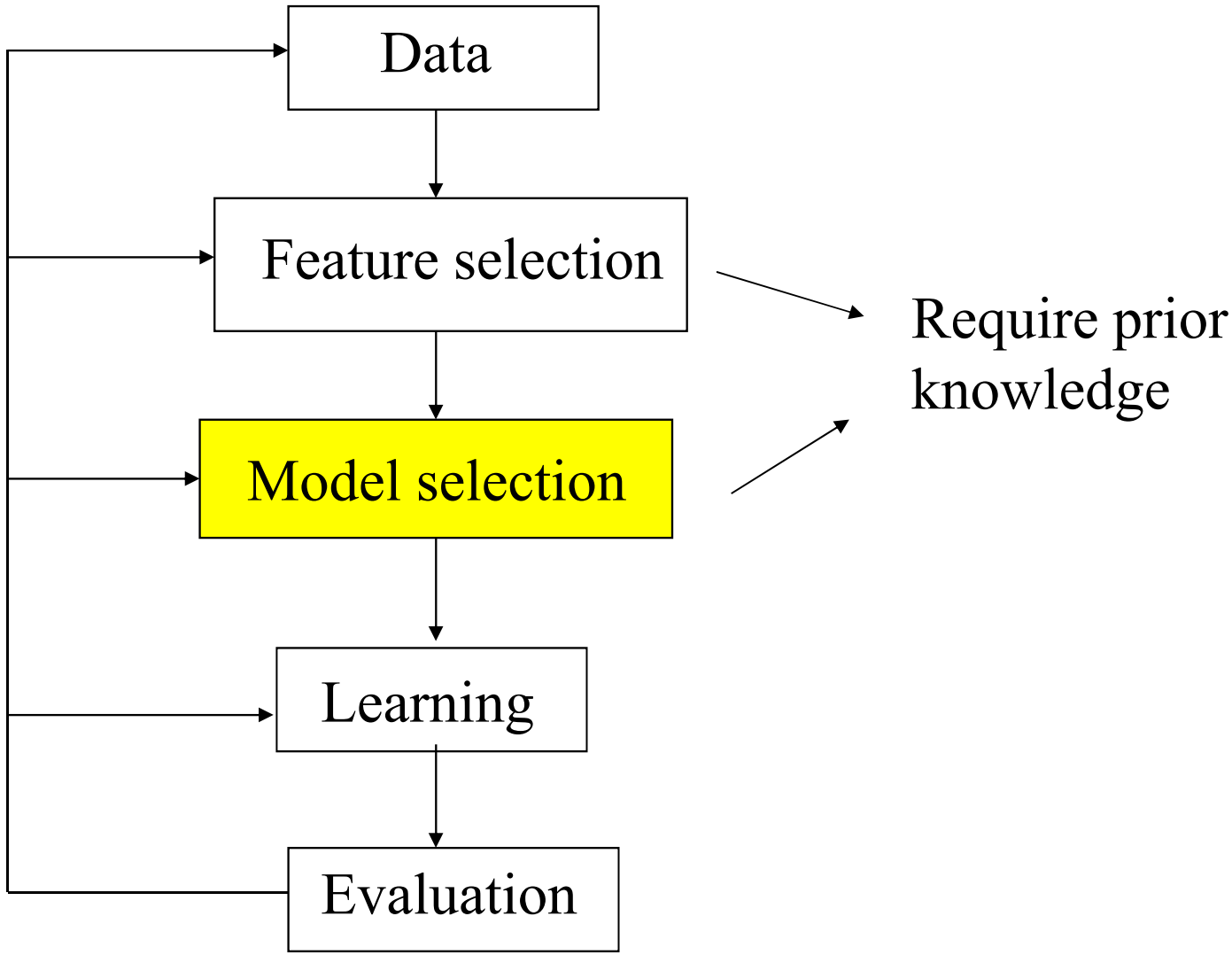
- **The size (dimensionality) of a sample** can be enormous

  $x_i = (x_i^1, x_i^2, .., x_i^d)$ d - very large

- **Example: document classification**
  - **thousands of documents**
  - 10,000 different words
  - **Features/Inputs:** counts of occurrences of different words
  - Overfit threat - too many parameters to learn, not enough samples to justify the estimates the parameters of the model

- **Feature selection: reduces the feature sets**
  - **Methods for removing input features**

Design cycle



Model selection

- **What is the right model to learn?**
  - A prior knowledge helps a lot, but still a lot of guessing
  - Initial data analysis and visualization
    - We can make a good guess about the form of the distribution, shape of the function
  - Independences and correlations
- **Overfitting problem**
  - Take into account the **bias and variance** of error estimates
  - Simpler (more biased) model – parameters can be estimated more reliably (smaller variance of estimates)
  - Complex model with many parameters – parameter estimates are less reliable (large variance of the estimate)

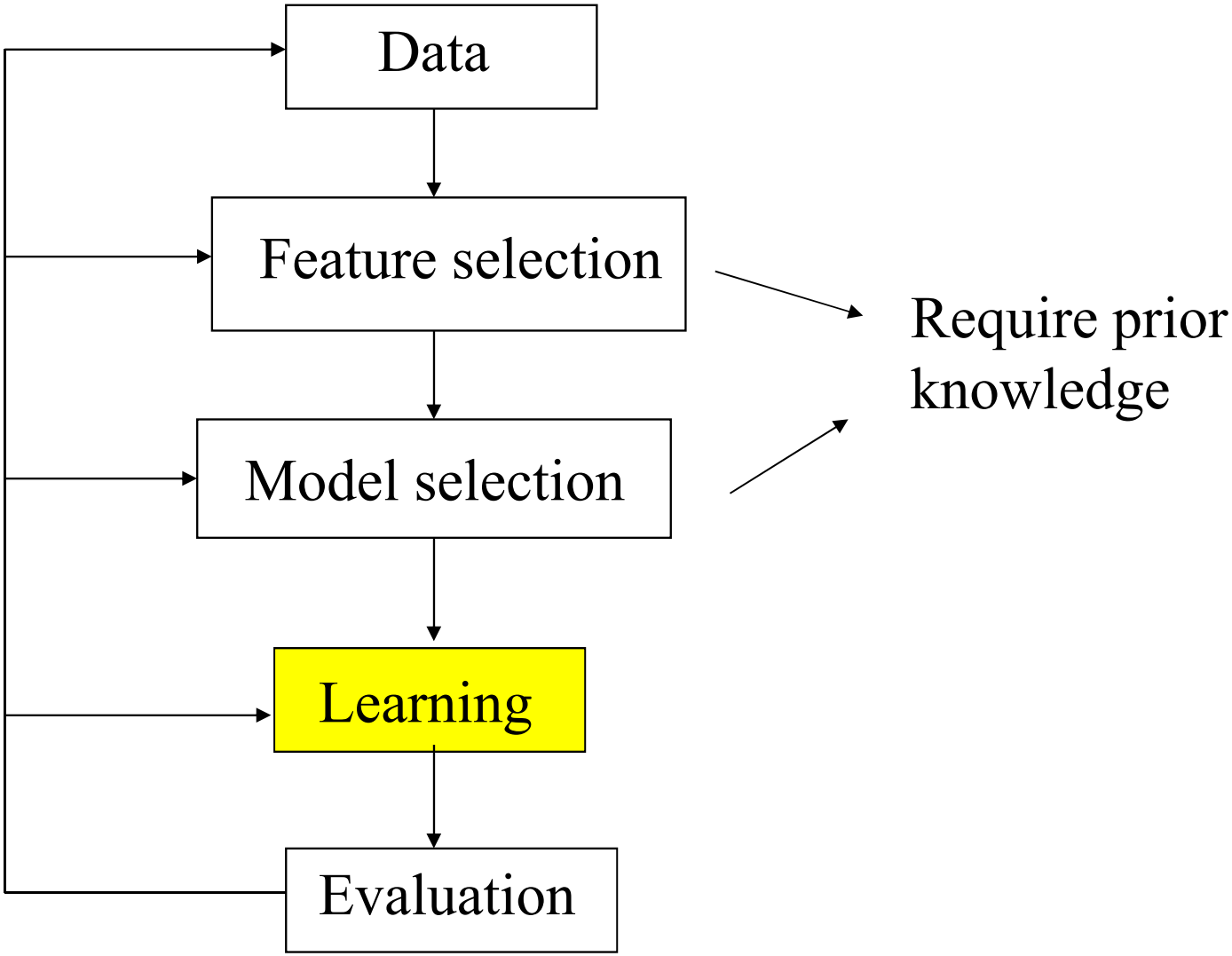
Solutions for overfitting

How to make the learner avoid the overfit?

- **Assure sufficient number of samples** in the training set
  - May not be possible (small number of examples)
- **Hold some data out of the training set = validation set**
  - Train (fit) on the training set (w/o data held out);
  - Check for the generalization error on the validation set, choose the model based on the validation set error

    (random re-sampling validation techniques)
- **Regularization (Occam's Razor)**
  - Explicit preference towards simple models
  - Penalize for the model complexity (number of parameters) in the objective function

Design cycle



Learning

- **Learning = optimization problem**. Various criteria:
  - **Mean square error**

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} Error(\mathbf{w}) \qquad Error\ (\mathbf{w}) = \frac{1}{N} \sum_{i=1,..N} (y_i - f(x_i, \mathbf{w}))^2$$

  - **Maximum likelihood (ML) criterion**

$$\Theta^* = \arg\max_{\Theta}\ P(D \mid \Theta) \qquad Error\ (\Theta) = -\log P(D \mid \Theta)$$

  - **Maximum posterior probability (MAP)**

$$\Theta^* = \arg\max_{\Theta}\ P(\Theta \mid D) \qquad P(\Theta \mid D) = \frac{P(D \mid \Theta) P(\Theta)}{P(D)}$$

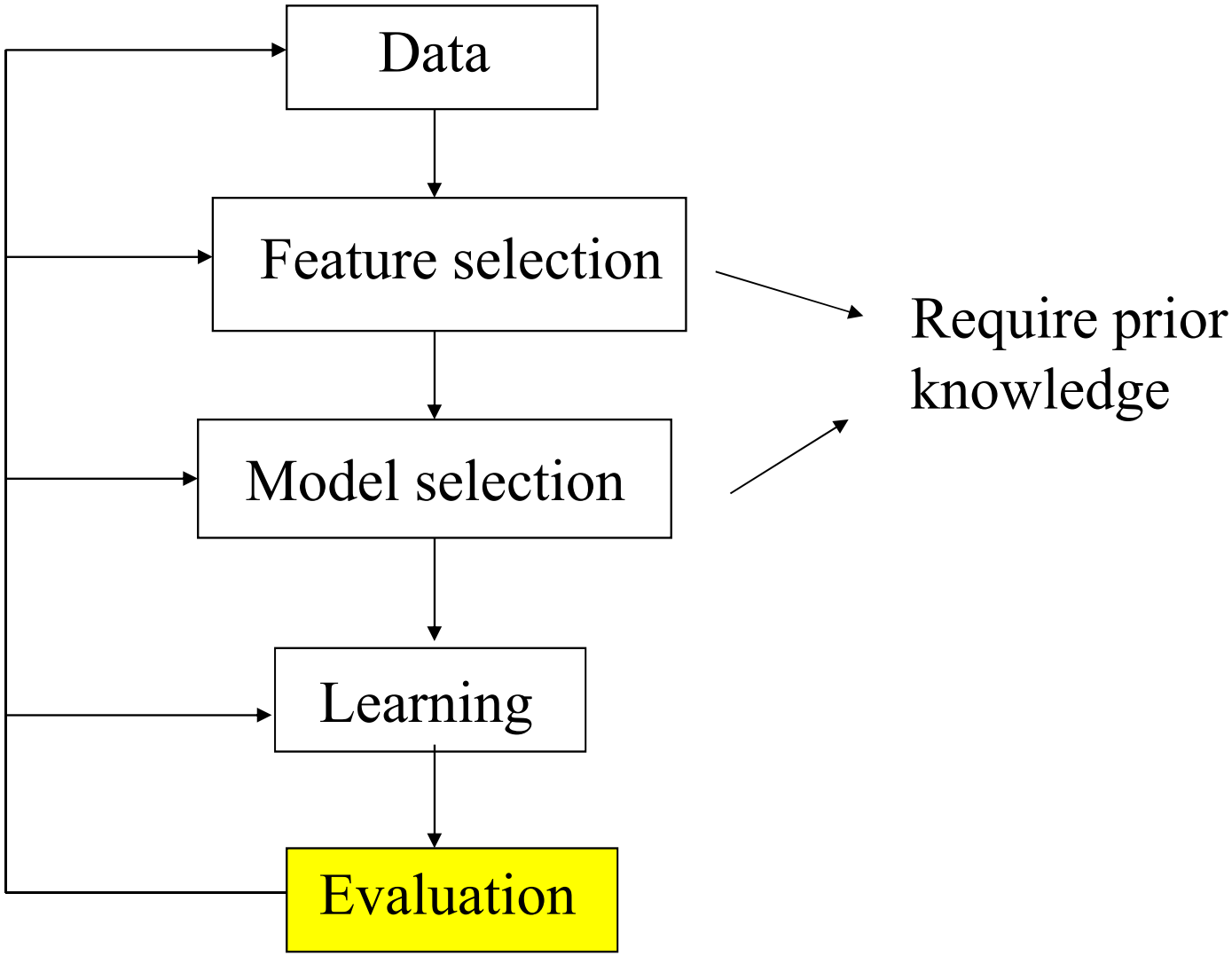
Learning

Learning = optimization problem

- Optimization problems can be hard to solve. Right choice of a model and an error function makes a difference.
- **Parameter optimizations (continuous space)**
  - Linear programming, Convex programming
  - Gradient methods: grad. descent, Conjugate gradient
  - Newton-Rhapson (2^{nd} order method)
  - Levenberg-Marquard

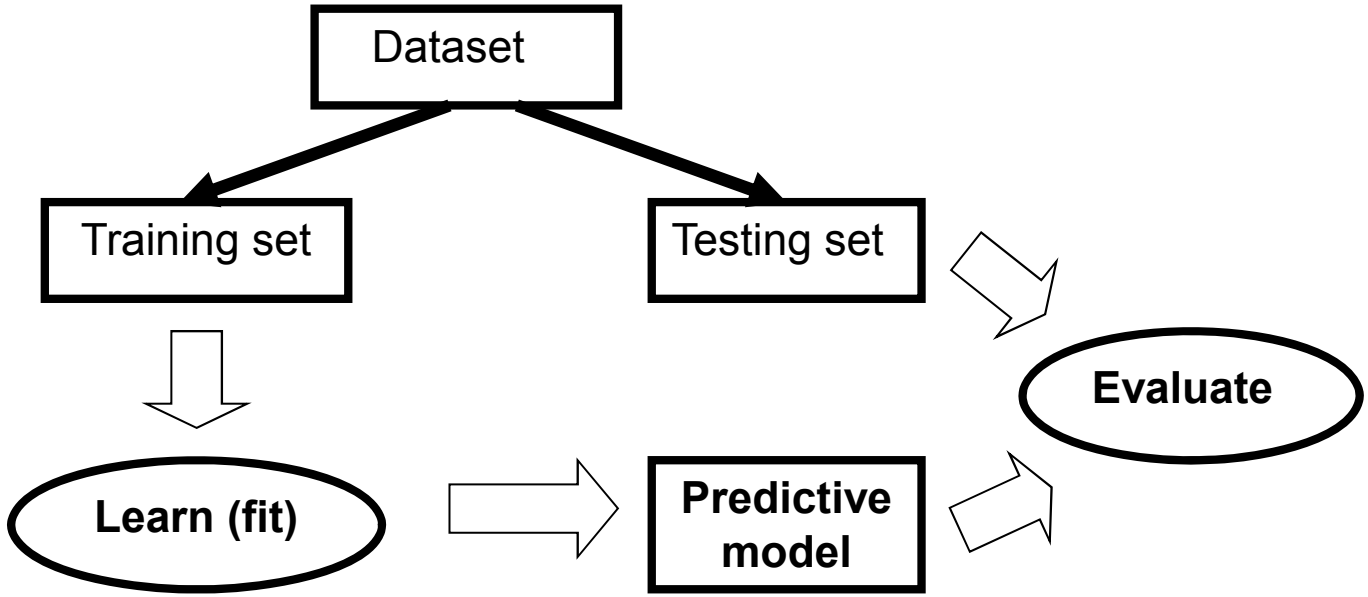
  Some can be carried **on-line** on a sample by sample basis
- **Combinatorial optimizations (over discrete spaces):**
  - Hill-climbing
  - Simulated-annealing
  - Genetic algorithms

Design cycle



Evaluation of learning models

- **Simple holdout method**
  - Divide the data to the training and test data

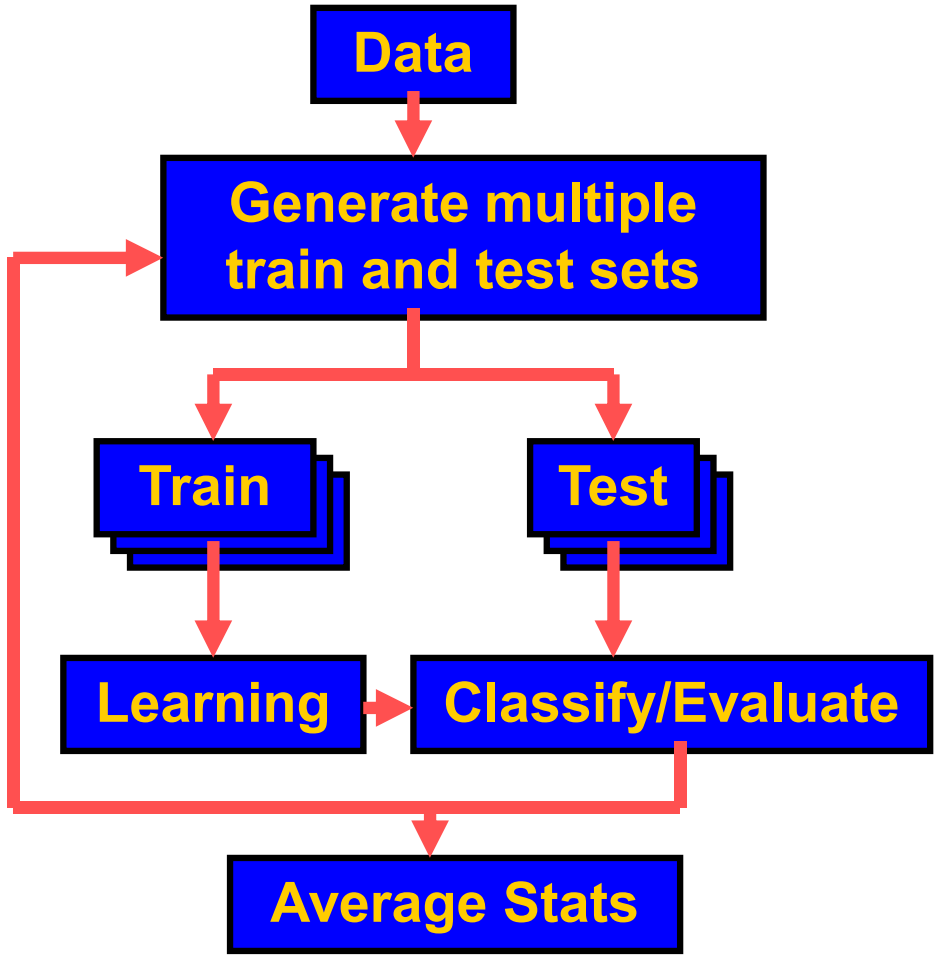


  - Typically 2/3 training and 1/3 testing

Evaluation

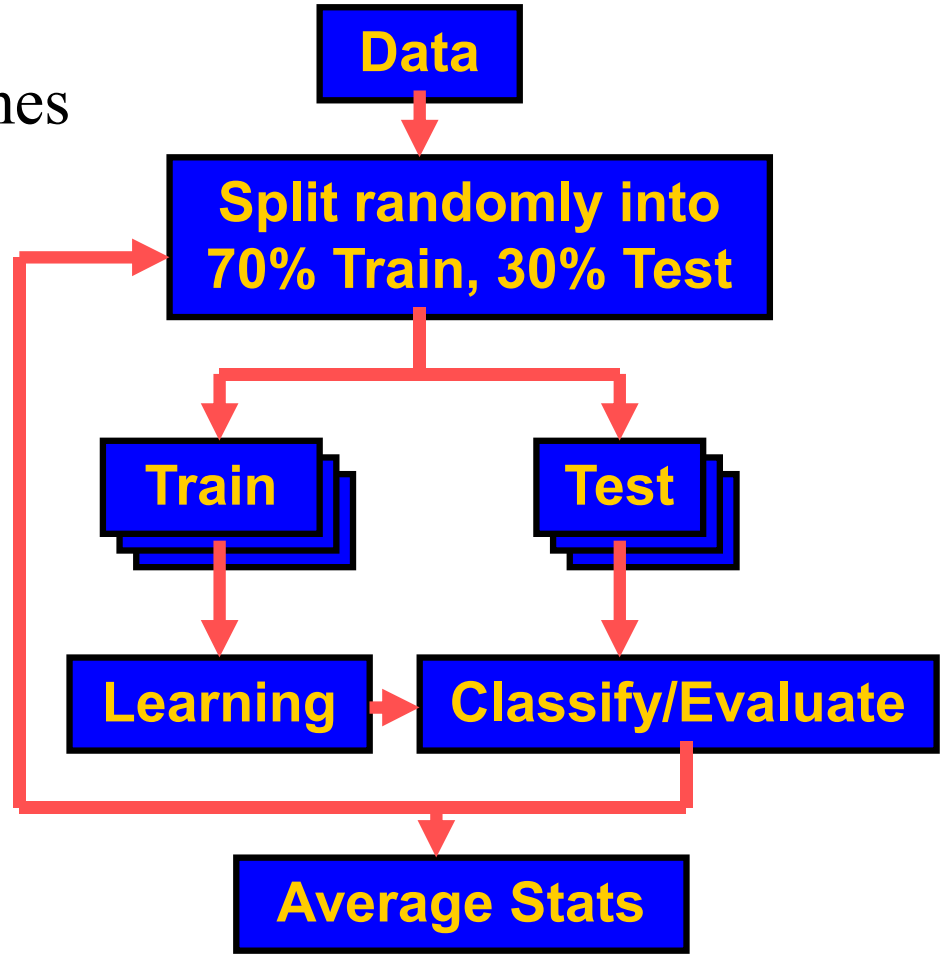
Other more complex methods

- Use multiple train/test sets
- Based on various random re-sampling schemes:
  - Random sub-sampling
  - Cross-validation
  - Bootstrap



Evaluation

- **Random sub-sampling**
  - Repeat a simple holdout method k times

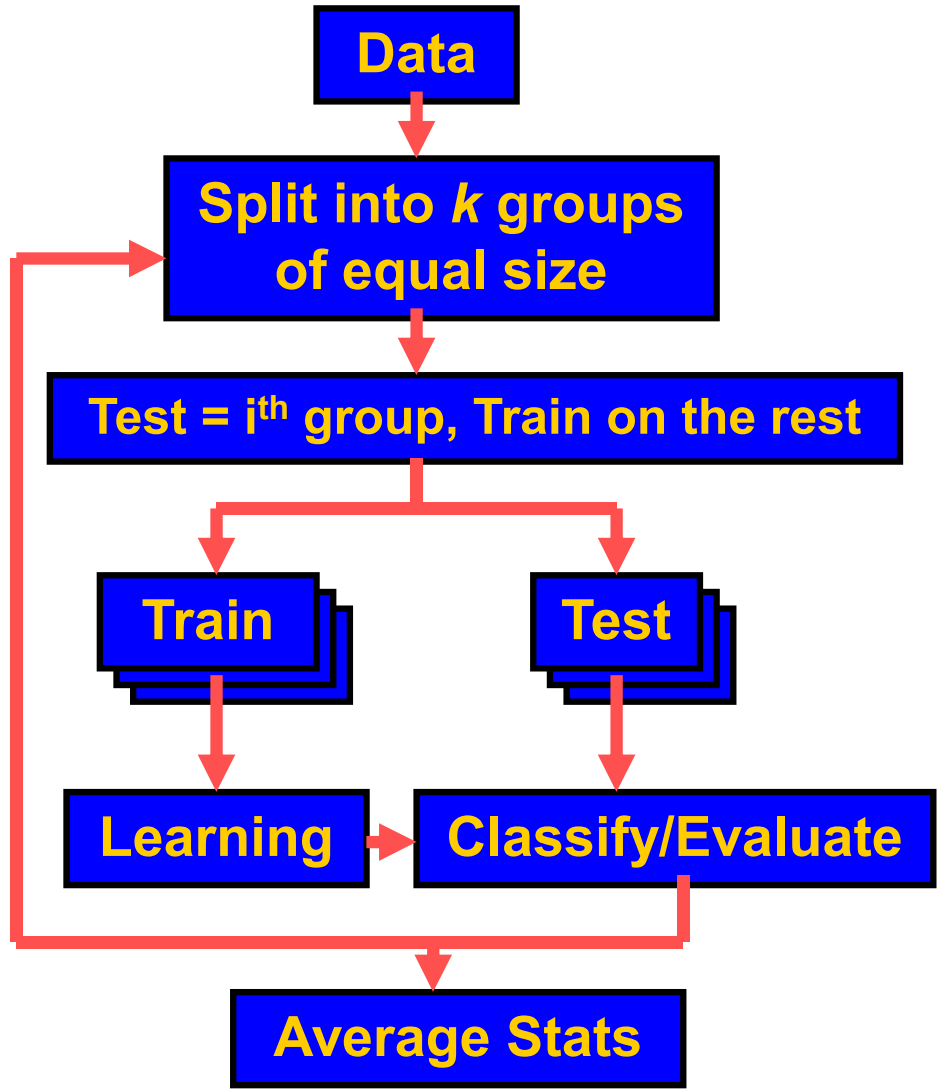


Evaluation

Cross-validation (k-fold)

- Divide data into k disjoint groups, test on k-th group/train on the rest
- Typically 10-fold cross-validation
- Leave one out cross-validation

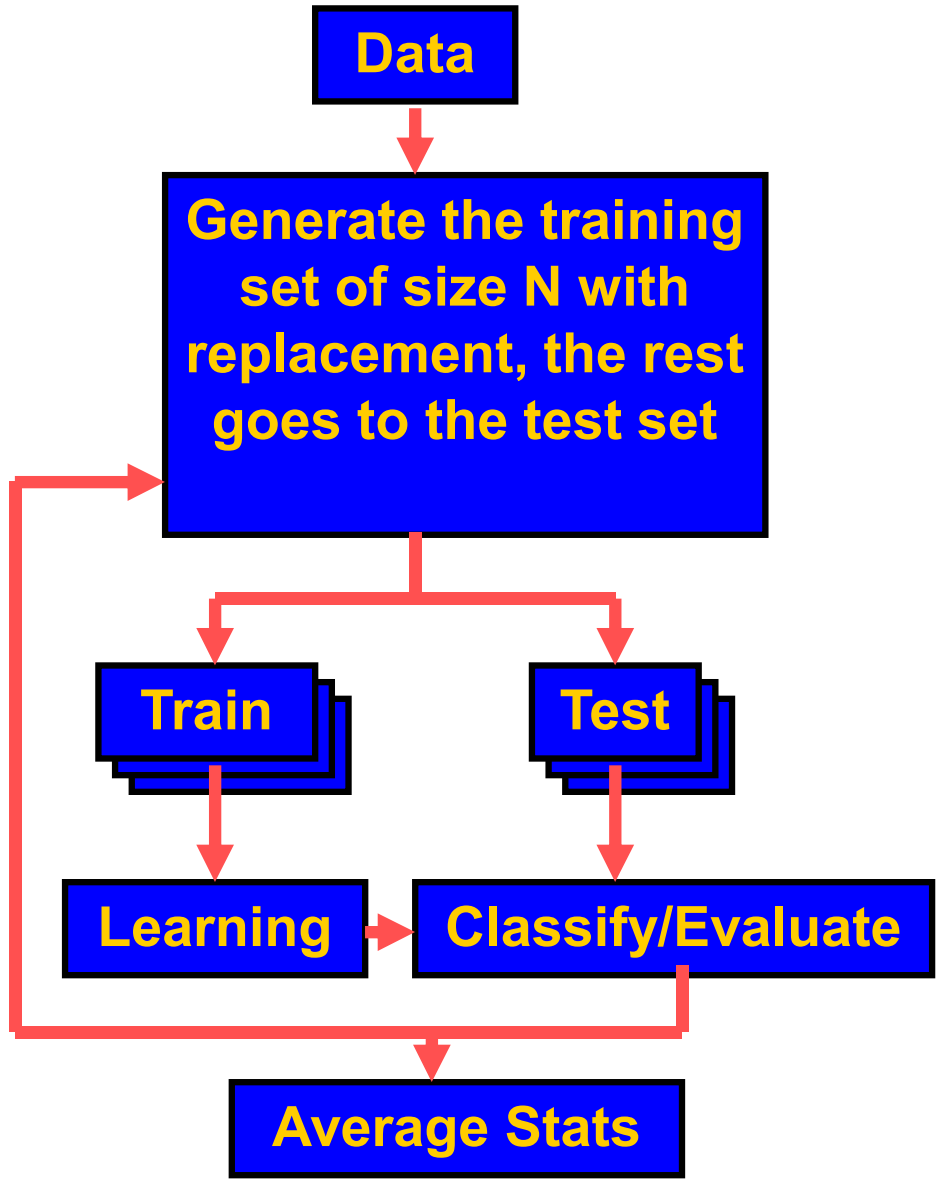
  (k = size of the data D)



Evaluation

Bootstrap

- The training set of size N = size of the data D
- Sampling with the replacement



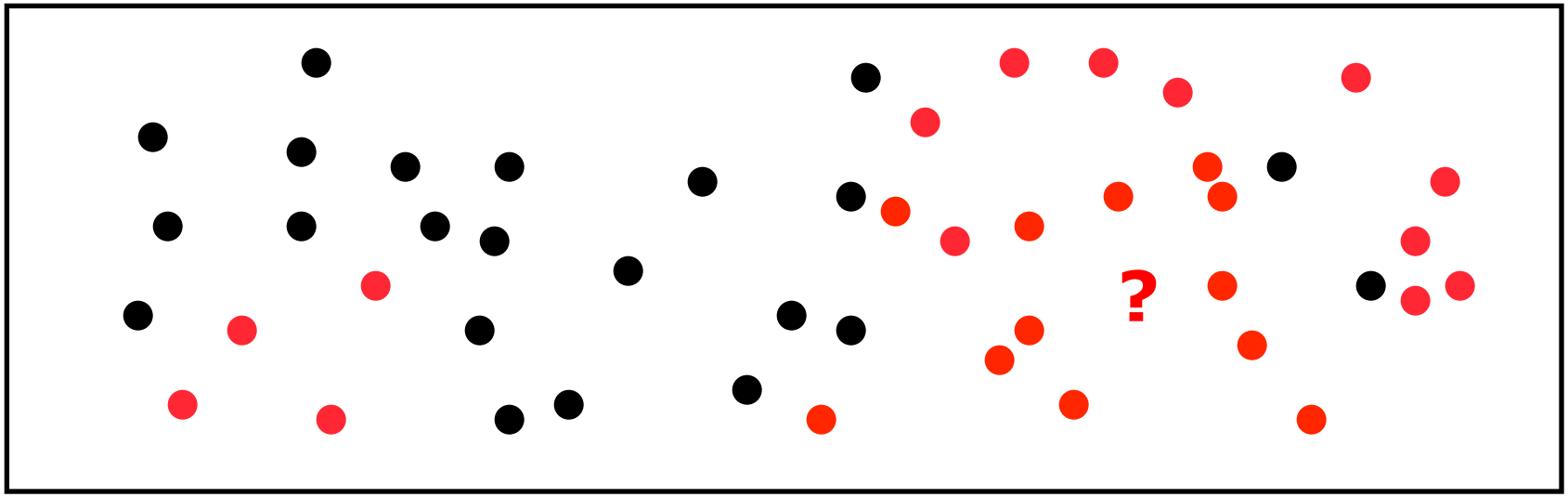
Variety of Learning Methods

Learning methods differ in terms of:

- the form of the hypothesis
- the way the computer finds a hypothesis given the data

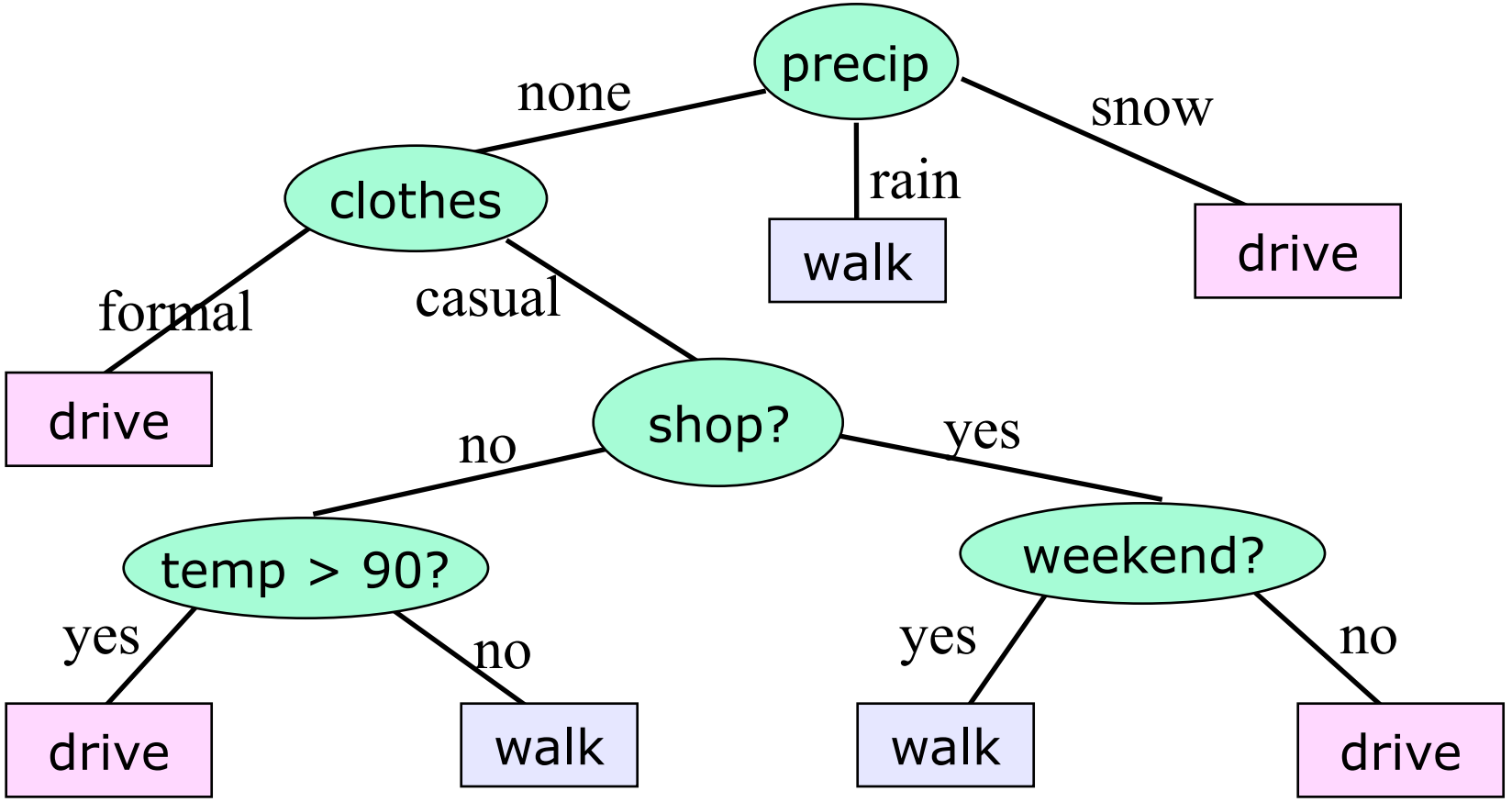
Nearest Neighbor

- Remember all your data
- When someone asks a question,
  - find the nearest old data point
  - return the answer associated with it



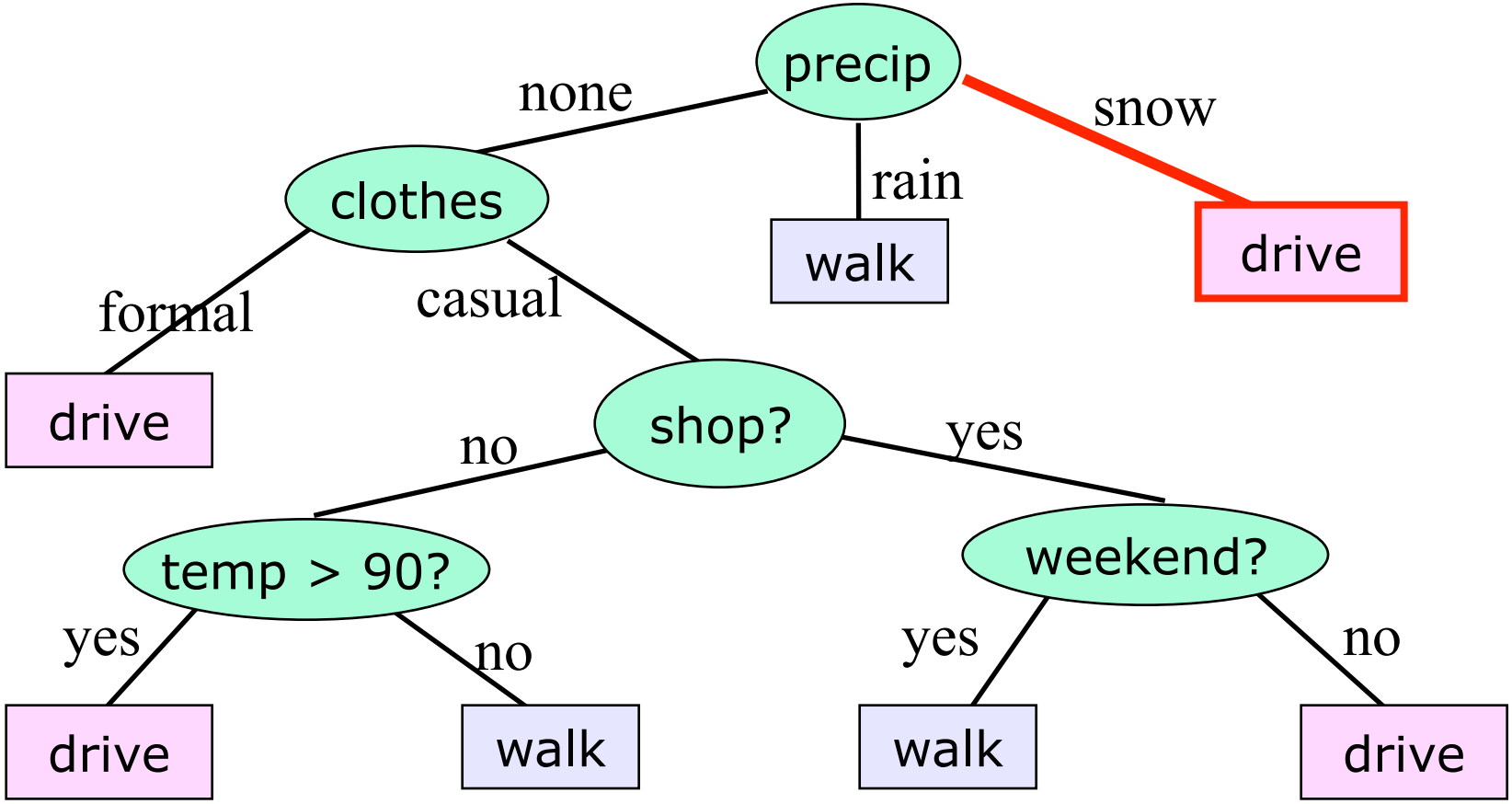
Decision Trees

Use all the data to build a tree of questions with answers at the leaves



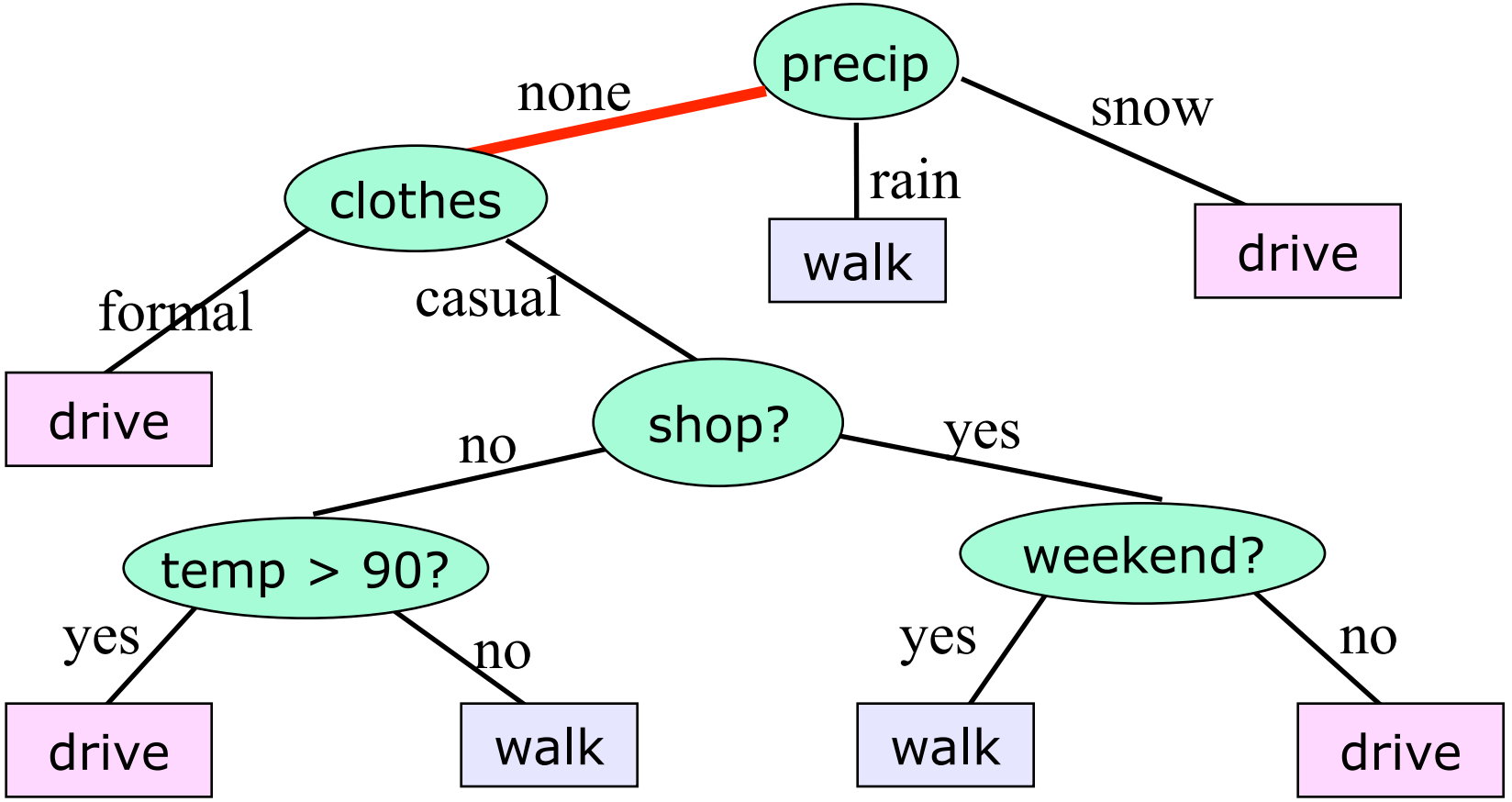
Decision Trees

Use all the data to build a tree of questions with answers at the leaves



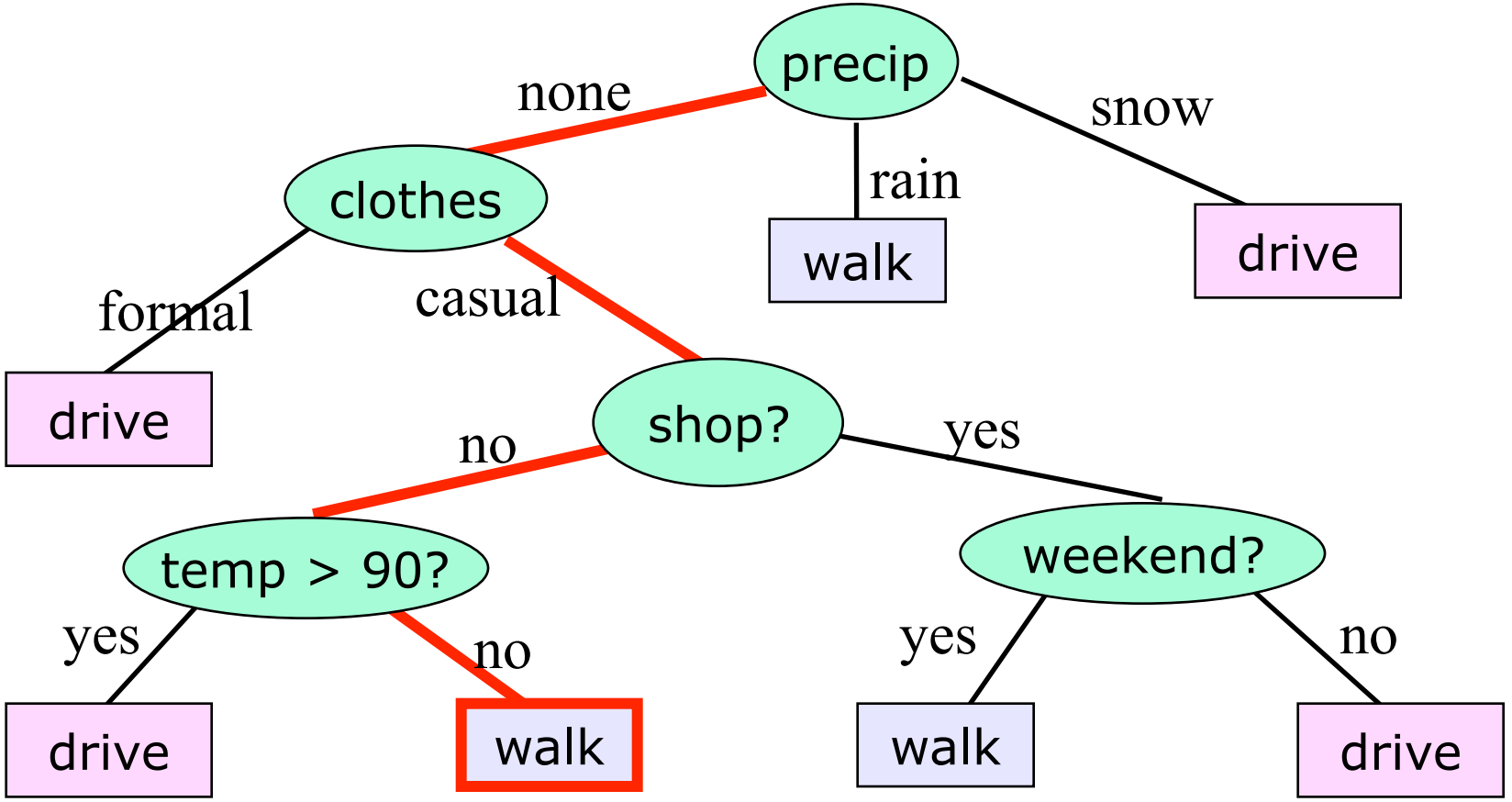
Decision Trees

Use all the data to build a tree of questions with answers at the leaves



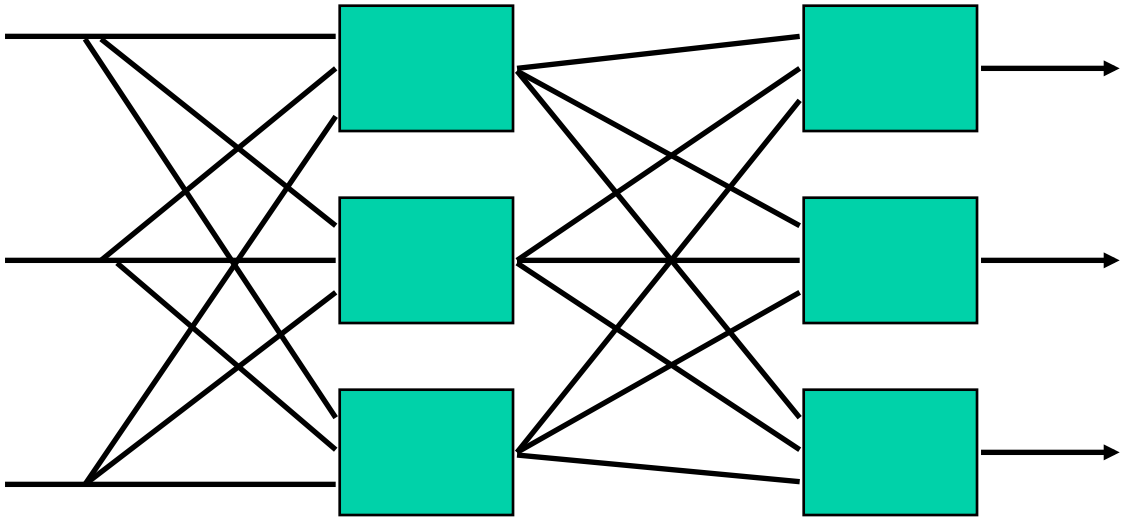
Decision Trees

Use all the data to build a tree of questions with answers at the leaves



Neural Networks

- Represent hypotheses as combinations of simple computations
- Neurophysiologically plausible (sort of)



- Learning through weight adjustment

Machine Learning Successes

- assessing loan credit risk
- detecting credit card fraud
- cataloging astronomical images
- detecting and diagnosing manufacturing faults
- helping NBA coaches analyze performance
- personalizing news and web searches
- steering an autonomous car across the US

Domains

- Congressional voting: given a congressperson's voting record (list of 1s and 0s), predict party
- Gene splice: predict the beginning of a coding section of the genome; input is vector of elements chosen from the set {ACGT}; encode each element with one bit (or possibly with 4)
- Spam filtering: encode every message as a vector of features, one per word; a feature is on if that word occurs in the message; predict whether or not the message is spam
- Marketing: predict whether a person will buy beer based on previous purchases; encode buying habits with a feature for all products, set to 1 if previously purchased

Congressional Voting

0. handicapped-infants
1. water-project-cost-sharing
2. adoption-of-the-budget-resolution
3. physician-fee-freeze
4. el-salvador-aid
5. religious-groups-in-schools
6. anti-satellite-test-ban
7. aid-to-nicaraguan-contras
8. mx-missile
9. immigration
10. synfuels-corporation-cutback
11. education-spending
12. superfund-right-to-sue
13. crime
14. duty-free-exports
15. export-administration-act-south-africa

232 data points

Supervised Learning

- Given data (training set)

$$D = \left\{ \left\langle x^1, y^1 \right\rangle, \left\langle x^2, y^2 \right\rangle, \ldots . \left\langle x^m, y^m \right\rangle \right\}$$

Supervised Learning

- Given data (training set)

$$D = \left\{ \left\langle x^1, y^1 \right\rangle, \left\langle x^2, y^2 \right\rangle, \ldots . \left\langle x^m, y^m \right\rangle \right\}$$

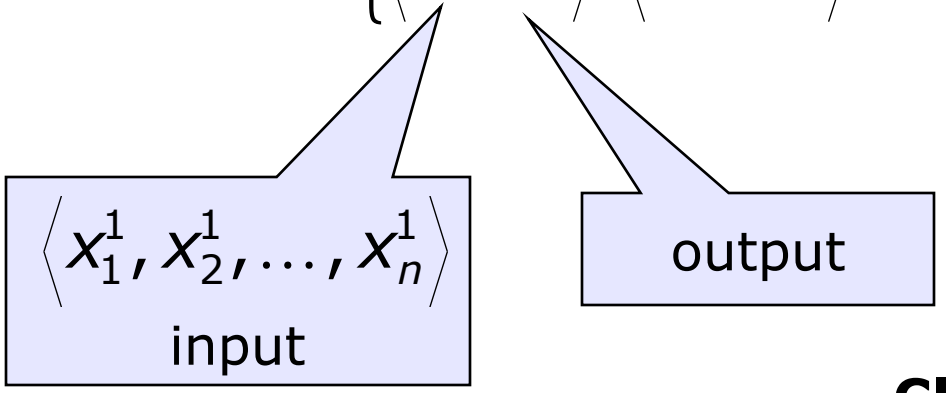
$\left\langle x_1^1, x_2^1, \ldots, x_n^1 \right\rangle$

input

Supervised Learning

- Given data (training set)

$$D = \left\{ \left\langle x^1, y^1 \right\rangle, \left\langle x^2, y^2 \right\rangle, \ldots . \left\langle x^m, y^m \right\rangle \right\}$$



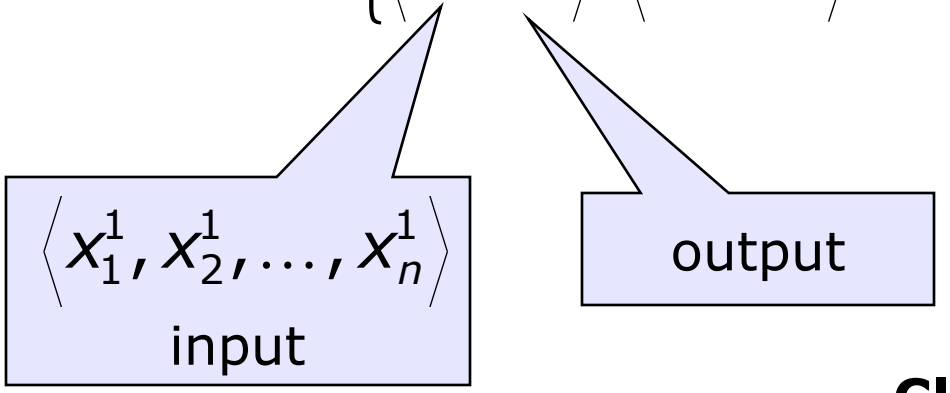
Classification: discrete Y

Regression: continuous Y

Supervised Learning

- Given data (training set)

$$D = \left\{ \left\langle x^1, y^1 \right\rangle, \left\langle x^2, y^2 \right\rangle, \ldots \left\langle x^m, y^m \right\rangle \right\}$$



Classification: discrete Y

Regression: continuous Y

- Goal: find a hypothesis h in hypothesis class H that does a good job of mapping x to y

Best Hypothesis

Hypothesis should

- do a good job of describing the data
- not be too complex

Best Hypothesis

Hypothesis should

- do a good job of describing the data
  - ideally: $h(x^i) = y^i$
  - number of errors: $E(h,D)$

- not be too complex

Best Hypothesis

Hypothesis should

- do a good job of describing the data
  - ideally: $h(x^i) = y^i$
  - number of errors: *E(h,D)*
- not be too complex
  - measure: C(h)

Best Hypothesis

Hypothesis should

- do a good job of describing the data
  - ideally: $h(x^i) = y^i$
  - number of errors: *E(h,D)*

- not be too complex
  - measure: C(h)

William of Ockham

Best Hypothesis

Hypothesis should

- do a good job of describing the data
  - ideally: $h(x^i) = y^i$
  - number of errors: $E(h,D)$

- not be too complex
  - measure: C(h)

Non sunt multiplicanda entia praeter necessitatem

William of Ockham

Minimize $E(h,D) + \alpha C(h)$

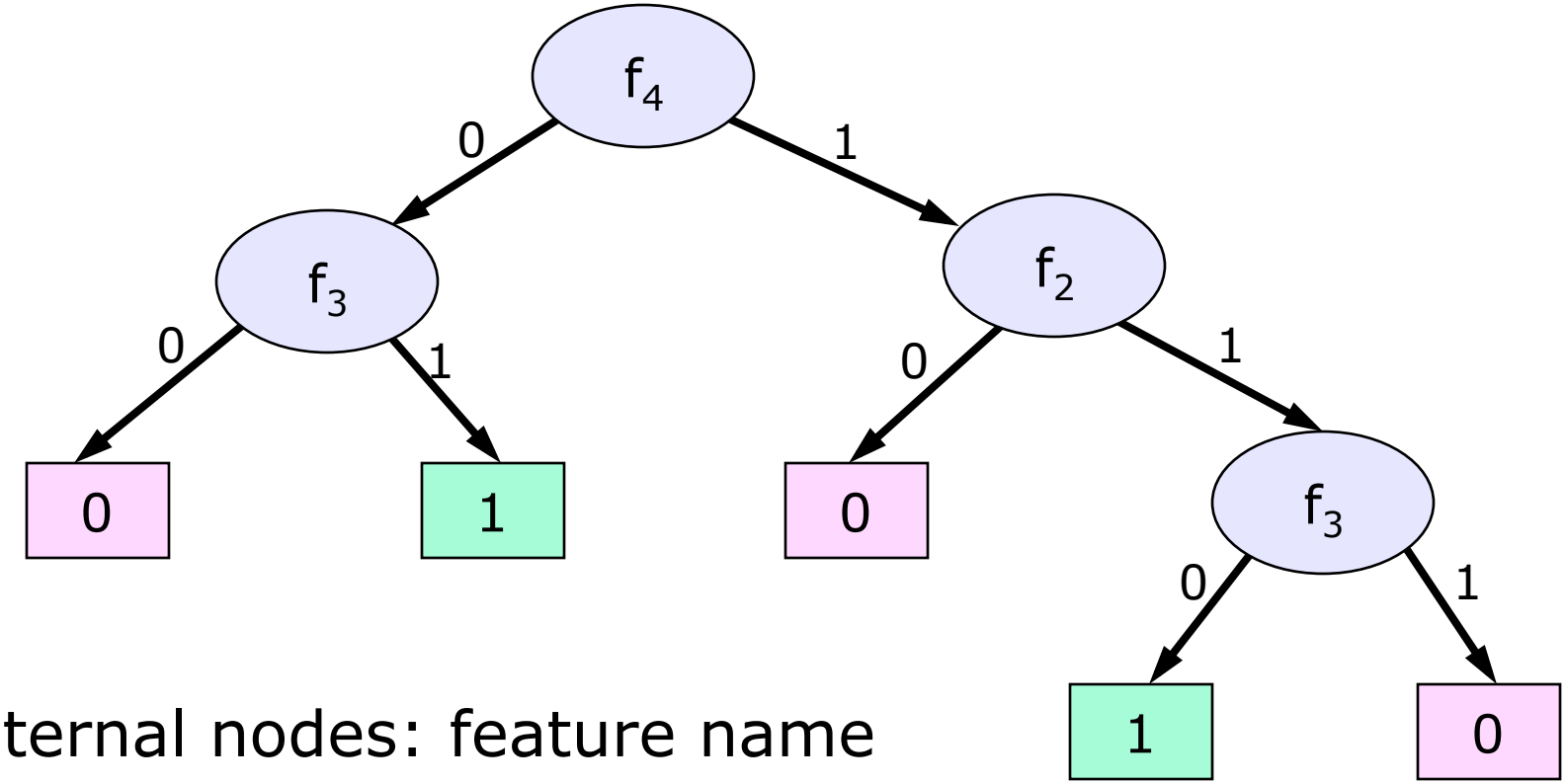
trade-off

Congressional Voting

0. handicapped-infants
1. water-project-cost-sharing
2. adoption-of-the-budget-resolution
3. physician-fee-freeze
4. el-salvador-aid
5. religious-groups-in-schools
6. anti-satellite-test-ban
7. aid-to-nicaraguan-contras
8. mx-missile
9. immigration
10. synfuels-corporation-cutback
11. education-spending
12. superfund-right-to-sue
13. crime
14. duty-free-exports
15. export-administration-act-south-africa

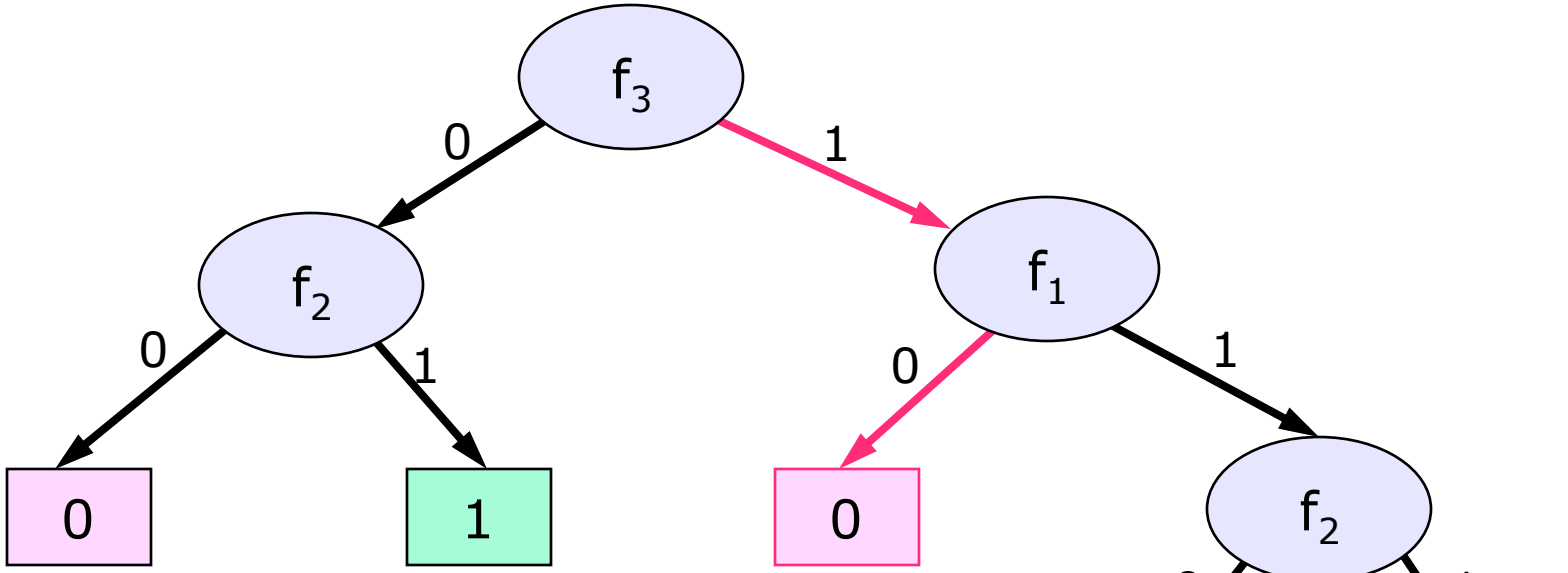
232 data points

Decision Trees: Hypothesis Class



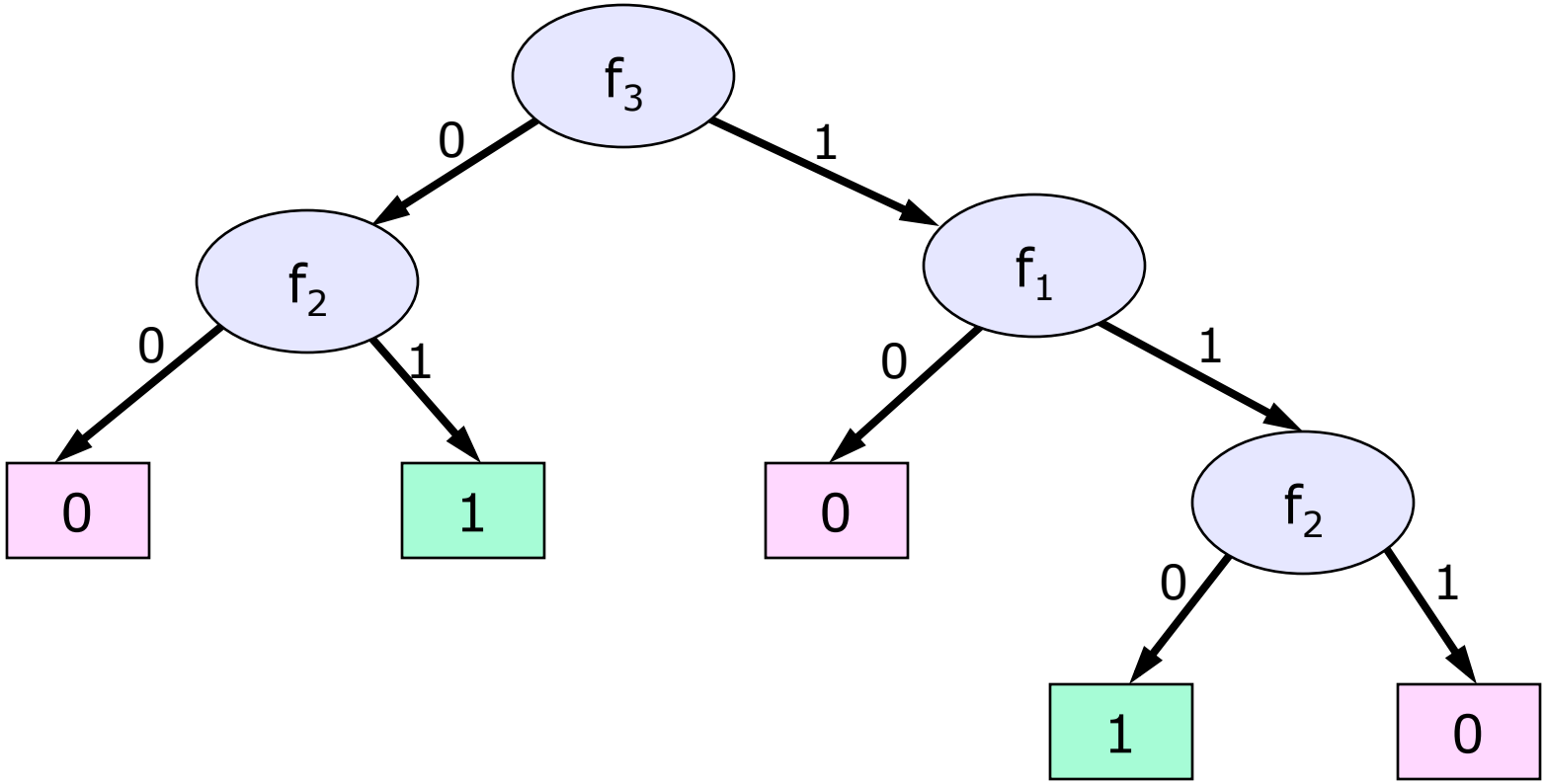
- Internal nodes: feature name
- One child for each value of the feature
- Leaf nodes: output

Hypothesis Class



$h(\langle 0,1,1,0 \rangle) = 0$

Hypothesis Class



$$h = \left(\neg f_3 \wedge f_2\right) \vee \left(f_3 \wedge f_1 \wedge \neg f_2\right)$$

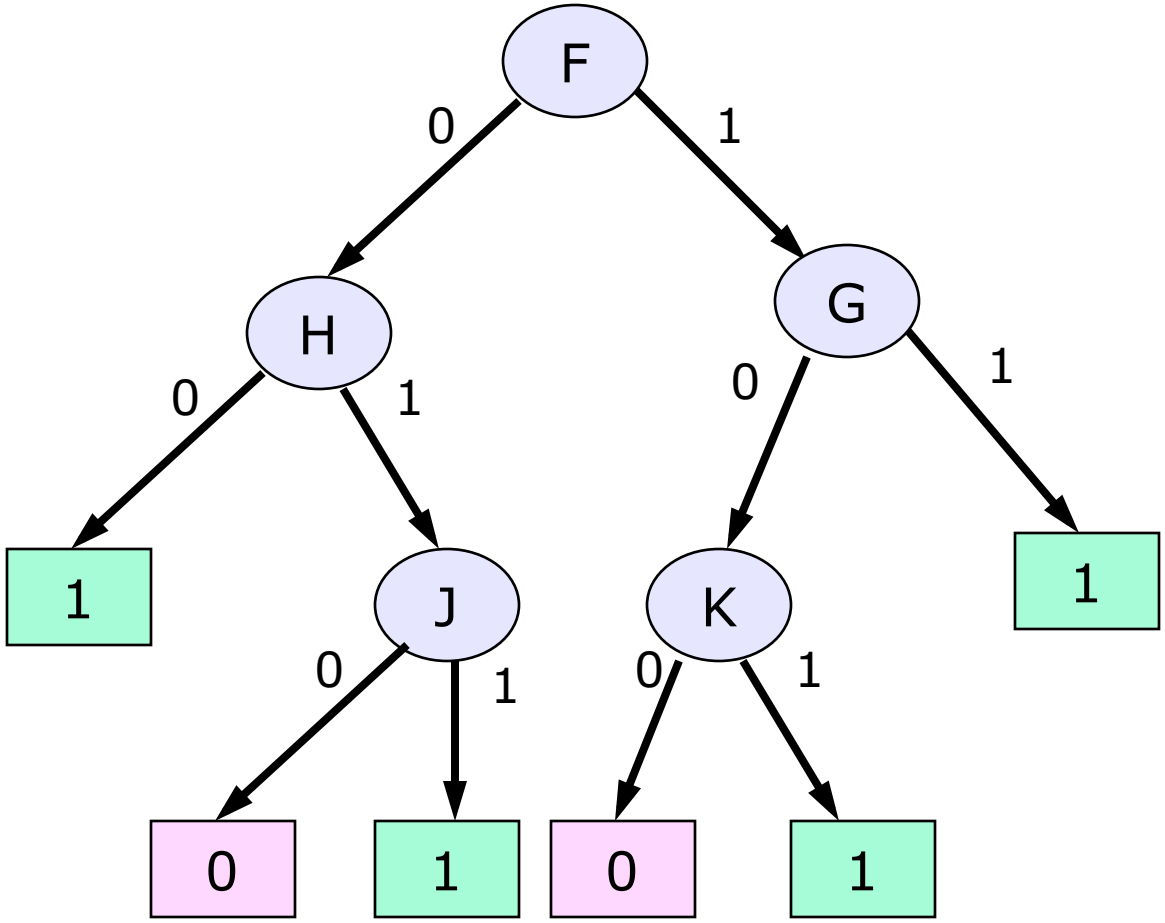
Tree Bias

- Both decision trees and DNF with negation can represent any Boolean function. So why bother with trees?
- Because we have a nice algorithm for growing trees that is consistent with a bias for simple trees (few nodes)

Tree Bias

- Both decision trees and DNF with negation can represent any Boolean function. So why bother with trees?
- Because we have a nice algorithm for growing trees that is consistent with a bias for simple trees (few nodes)
- Too hard to find the smallest good tree, so we'll be greedy again
- Have to watch out for overfitting

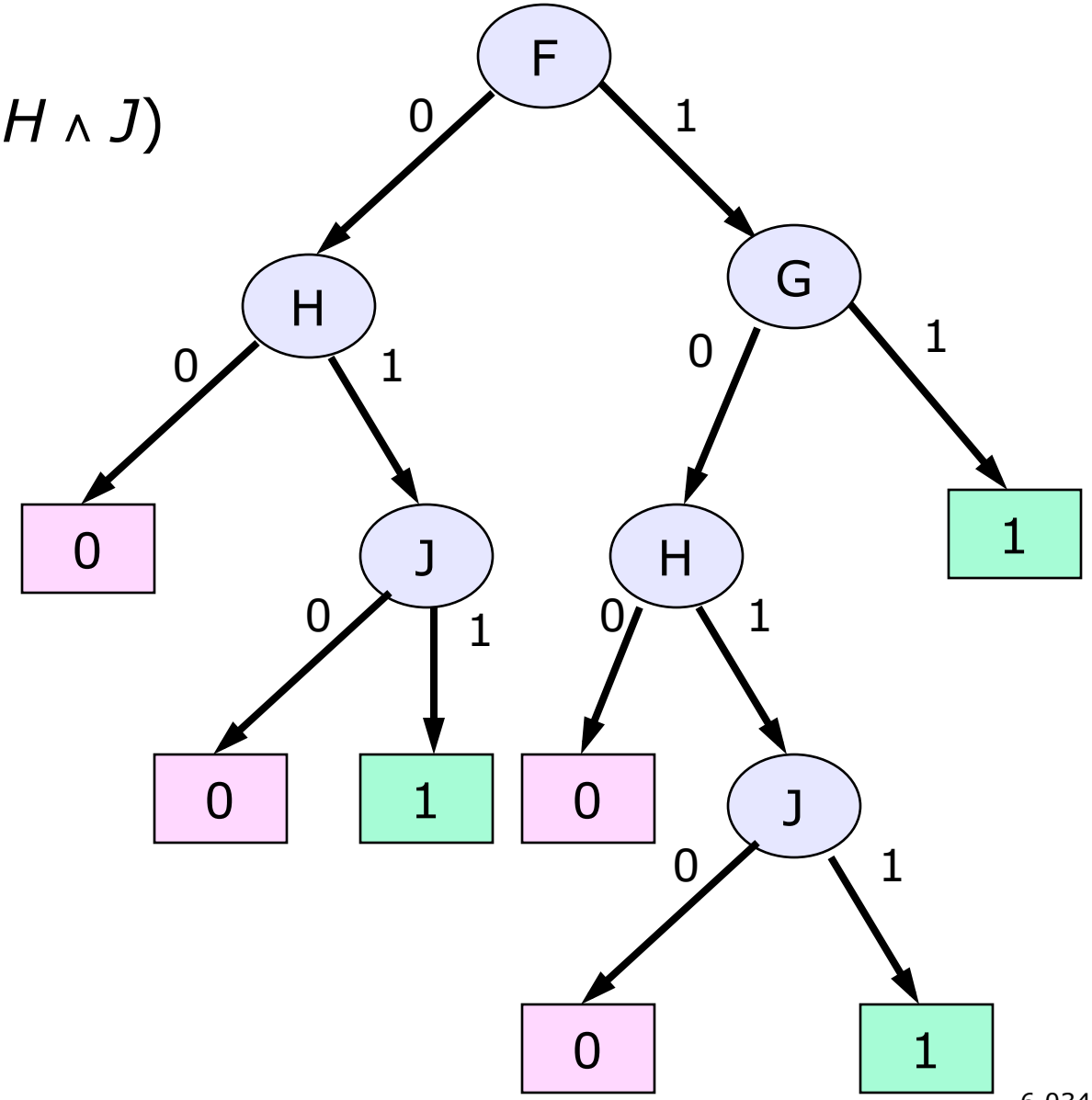
Trees vs DNF



$$(\neg F \wedge \neg H) \vee (\neg F \wedge H \wedge J) \vee (F \wedge \neg G \wedge K) \vee (F \wedge G)$$

Trees vs DNF

$(F \wedge G) \vee (H \wedge J)$



Algorithm

- Developed in parallel in AI by Quinlan and in statistics by Breiman, Friedman, Olsen and Stone

Algorithm

- Developed in parallel in AI by Quinlan and in statistics by Breiman, Friedman, Olsen and Stone

BuildTree (Data)

Algorithm

- Developed in parallel in AI by Quinlan and in statistics by Breiman, Friedman, Olsen and Stone

BuildTree (Data)

if all elements of Data have the same y value, then

MakeLeafNode(y)

Algorithm

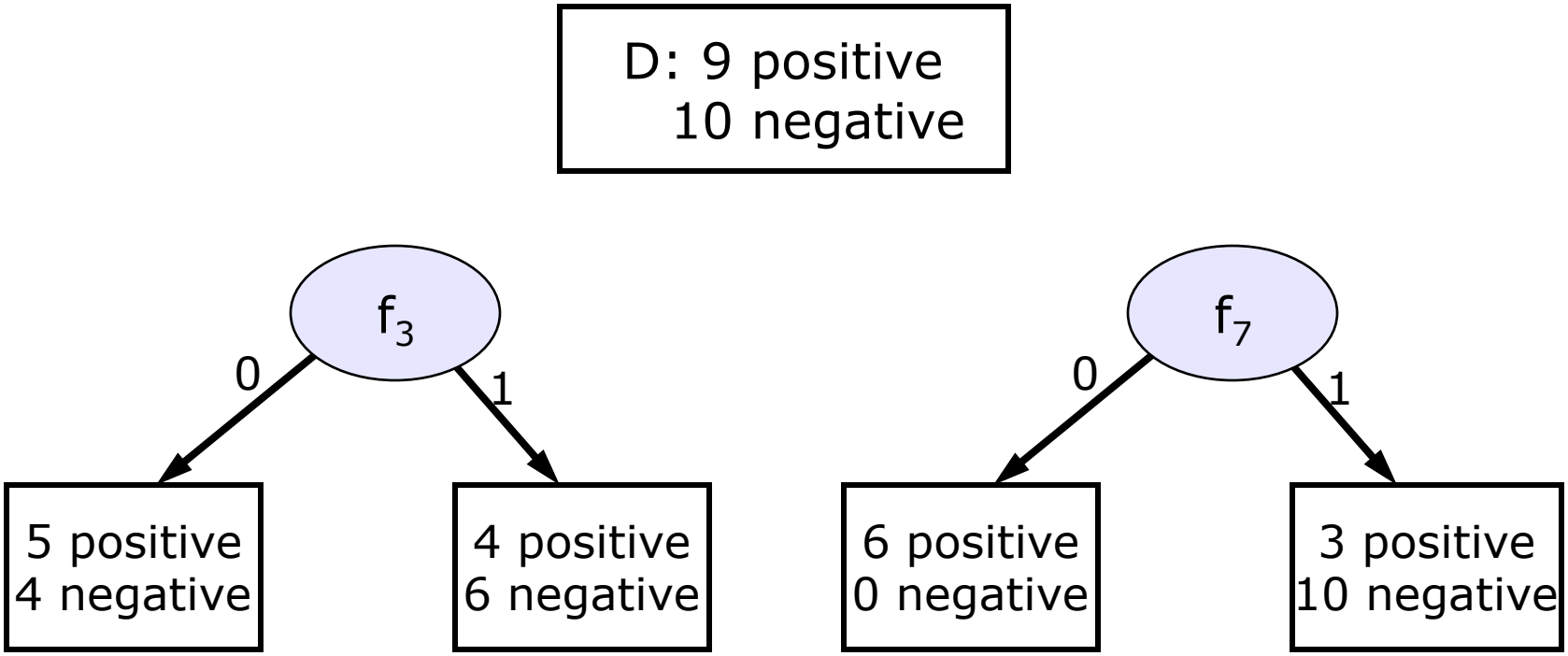
- Developed in parallel in AI by Quinlan and in statistics by Breiman, Friedman, Olsen and Stone

```
BuildTree (Data)
    if all elements of Data have the same y value, then
        MakeLeafNode(y)
    else
        feature := PickBestFeature(Data)
        MakeInternalNode(feature,
            BuildTree(SelectFalse(Data, feature)),
            BuildTree(SelectTrue(Data, feature)))
```

Let's Split

D: 9 positive
10 negative

Let's Split


D: 9 positive
10 negative
f_3
0
1
5 positive
4 negative
4 positive
6 negative
f_7
0
1
6 positive
0 negative
3 positive
10 negative

Entropy

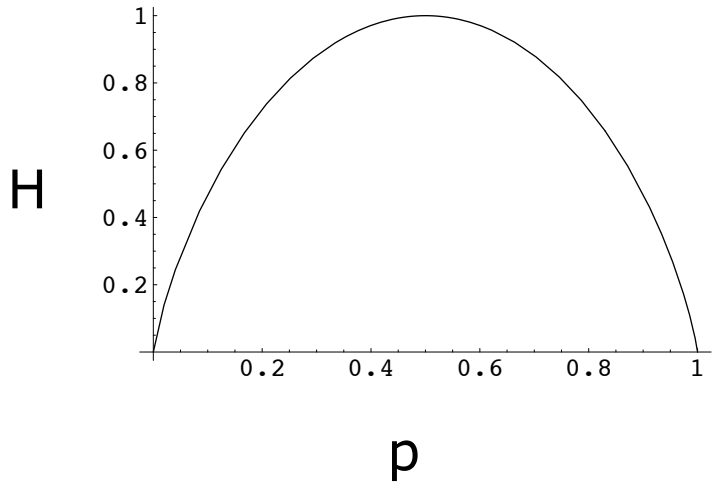
p : proportion of positive examples in a data set

$$H = -p \log_2 p - (1 - p) \log_2 (1 - p)$$

Entropy

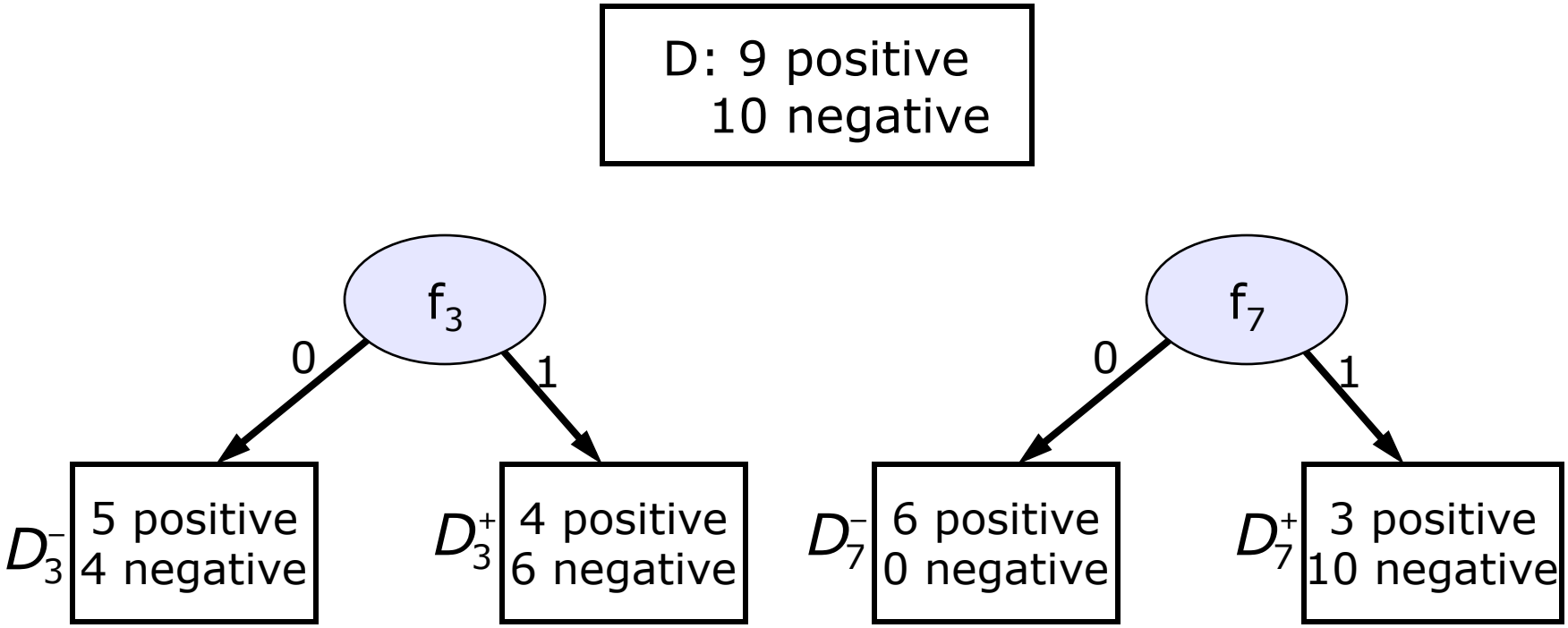
p : proportion of positive examples in a data set

$$H = -p \log_2 p - (1 - p) \log_2 (1 - p)$$

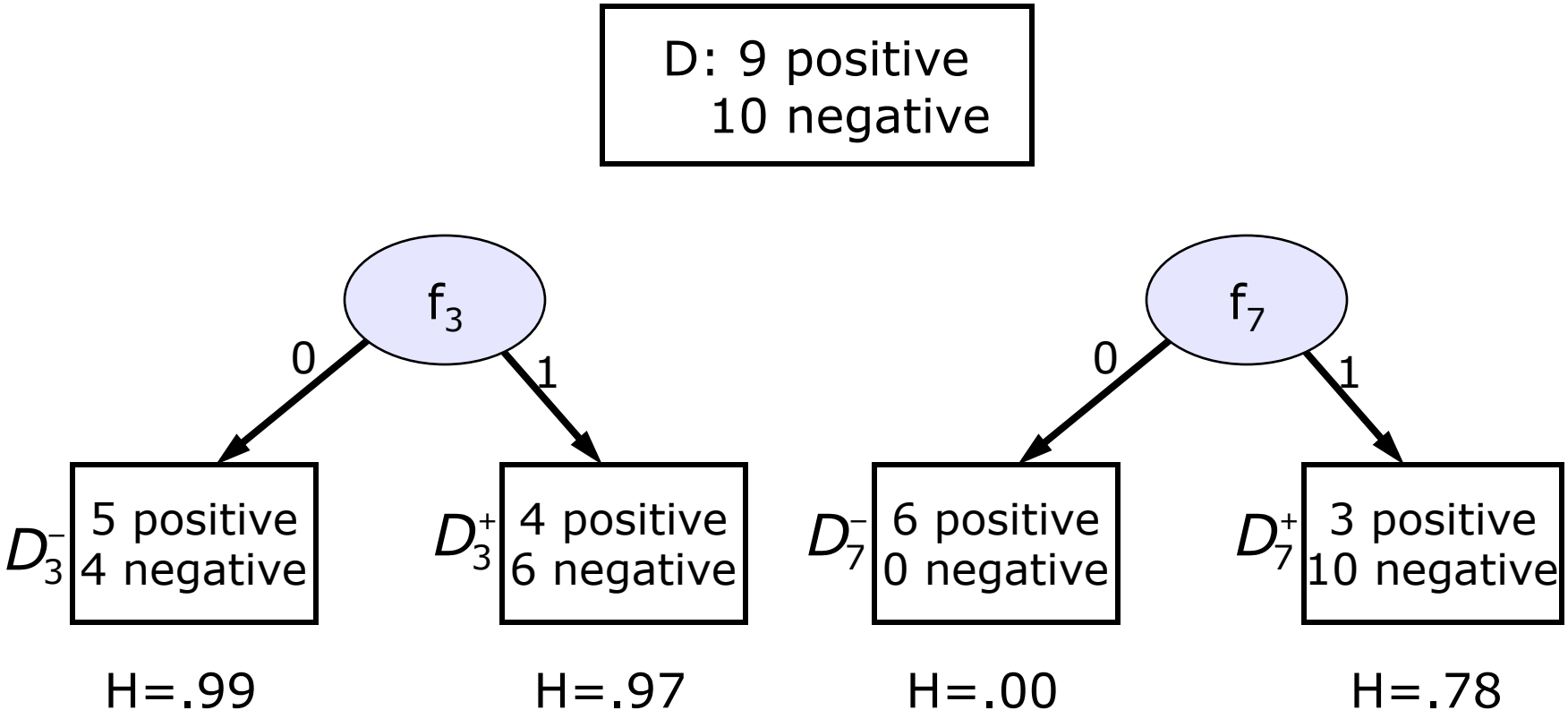


$$0 \log_2 0 = 0$$

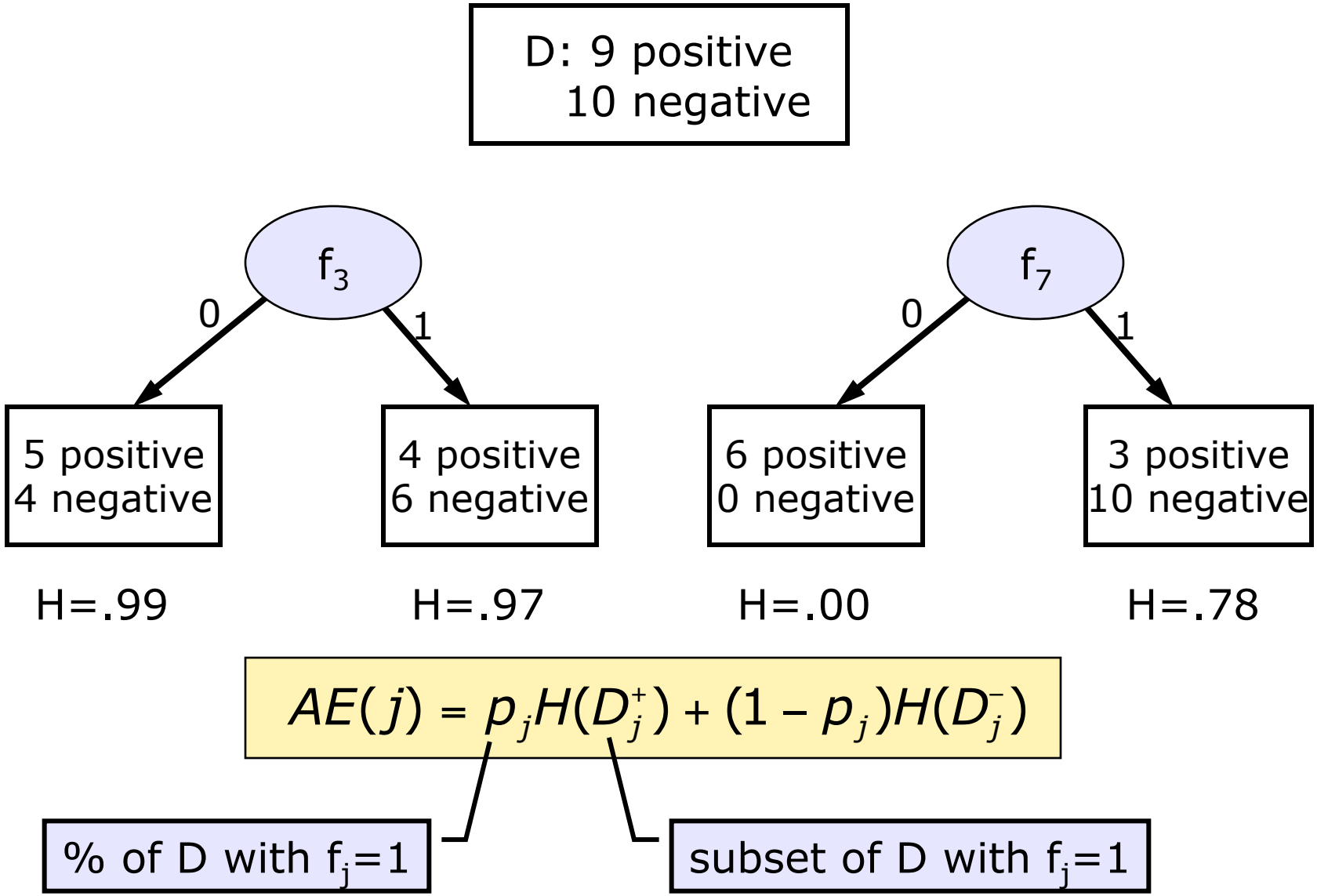
Let's Split


D: 9 positive
10 negative
f_3
0
1
D_3^-
5 positive
4 negative
D_3^+
4 positive
6 negative
f_7
0
1
D_7^-
6 positive
0 negative
D_7^+
3 positive
10 negative

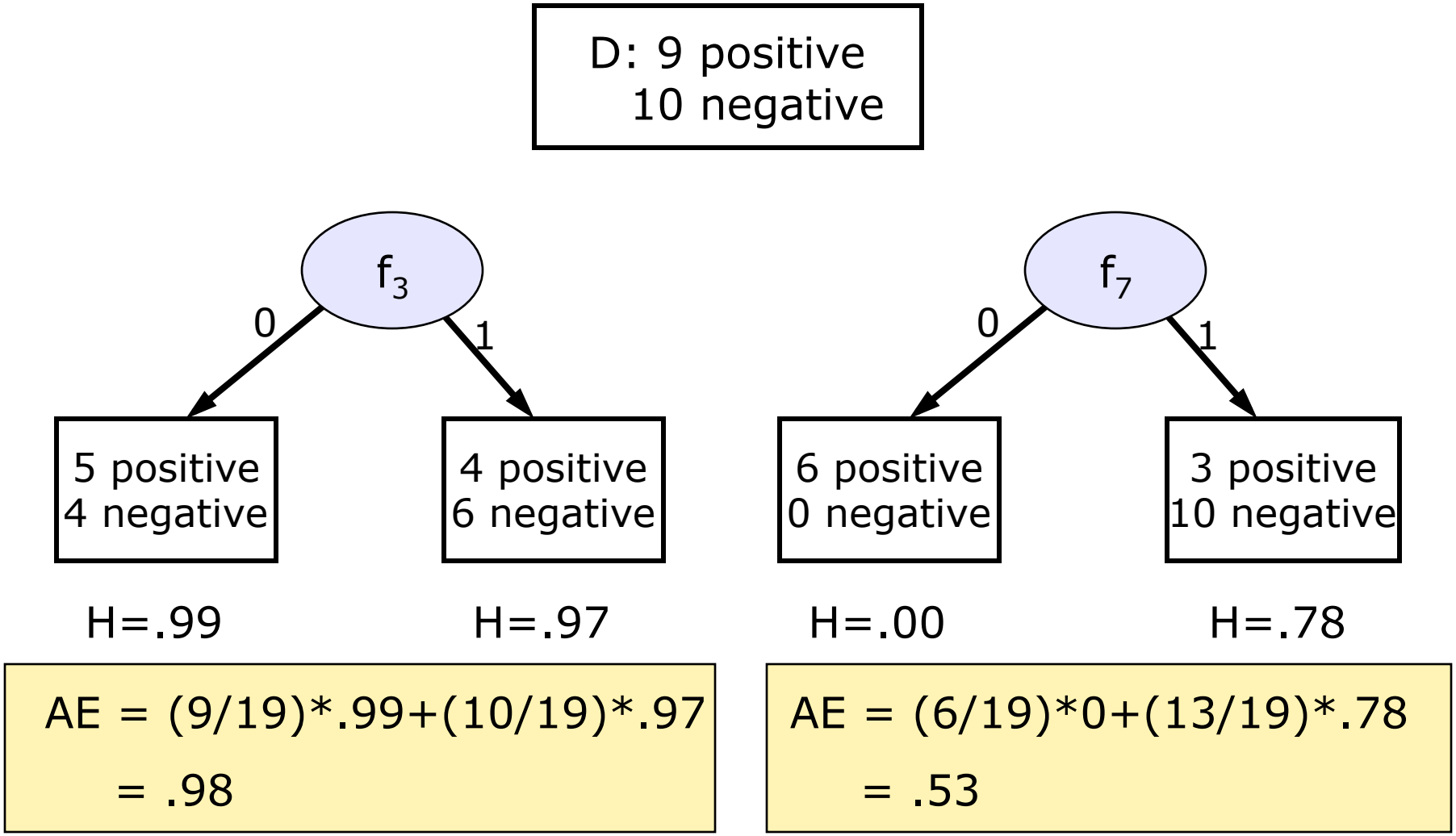
Let's Split


D: 9 positive
10 negative
f_3
0
1
D_3^-
5 positive
4 negative
D_3^+
4 positive
6 negative
H=.99
H=.97
f_7
0
1
D_7^-
6 positive
0 negative
D_7^+
3 positive
10 negative
H=.00
H=.78

Let's Split


D: 9 positive
10 negative
f3
0
1
f7
0
1
5 positive
4 negative
4 positive
6 negative
6 positive
0 negative
3 positive
10 negative
H=.99
H=.97
H=.00
H=.78
$AE(j) = p_j H(D_j^+) + (1 - p_j) H(D_j^-)$
% of D with f_j=1
subset of D with f_j=1

Let's Split


D: 9 positive
10 negative
f_3
0
1
5 positive
4 negative
4 positive
6 negative
H=.99
H=.97
AE = (9/19)*.99+(10/19)*.97
= .98
f_7
0
1
6 positive
0 negative
3 positive
10 negative
H=.00
H=.78
AE = (6/19)*0+(13/19)*.78
= .53

Algorithm

- Developed in parallel in AI by Quinlan and in statistics by Breiman, Friedman, Olshen and Stone

```
BuildTree (Data)
    if all elements of Data have the same y value, then
        MakeLeafNode(y)
    else
        feature := PickBestFeature(Data)
        MakeInternalNode(feature,
            BuildTree(SelectFalse(Data, feature)),
            BuildTree(SelectTrue(Data, feature)))
```

- Best feature minimizes average entropy of data in the children