# Introduction to Machine Learning
## NPFL 054

http://ufal.mff.cuni.cz/course/npfl054

Barbora Hladká
hladka@ufal.mff.cuni.cz

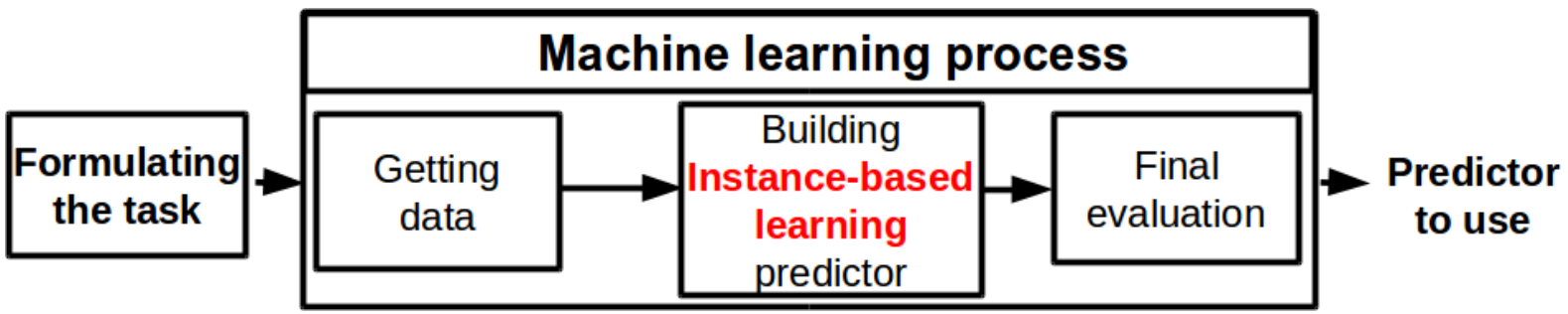
Martin Holub
holub@ufal.mff.cuni.cz

Charles University,
Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics

# Outline

- Instance-based learning
- Bias and variance
- Naïve Bayes algorithm
- Bayesian networks
- Maximum likelihood estimation
- Evaluation of binary classifiers

# Instance-based learning



# Instance-based learning
## Key idea

- IBL methods initially store the training data (that is why IBL methods are often referred to as "**lazy**" methods).
- For a new instance, prediction is based on local similarity, i.e. a set of similar related instances are retrieved and used for prediction
- IBL methods can construct a different approximation of a target function for each distinct test instance.
- Both classification and regression.

# Instance-based learning
## Key points

1. A distance metric
2. How many nearby neighbours look at?
3. A weighting function
4. How to fit with local points?

# Instance-based learning
# Distance metric

$\mathbf{x}_i = \langle x_{i_1}, x_{i_2}, ..., x_{i_m} \rangle, \mathbf{x}_j = \langle x_{j_1}, x_{j_2}, ..., x_{j_m} \rangle$

- for ex. **Euclidean distance**

$$E(\mathbf{x_i}, \mathbf{x_j}) = \sqrt{\sum_{r=1}^{m} (x_{i_r} - x_{j_r})^2} \tag{1}$$

- for ex. **Manhattan distance**

$$M(\mathbf{x_i}, \mathbf{x_j}) = \sum_{r=1}^{m} |x_{i_r} - x_{j_r}| \tag{2}$$

# Instance-based learning
# Learning algorithms

- k-Nearest Neighbour
- Distance weighted k-NN
- Locally weighted linear regression
- ...

# Instance-based learning
# $k$-Nearest Neighbour algorithm

1. **A distance metric**: Euclidian (and many more)
2. **How many nearby neighbours look at?** $k$
3. **A weighting function**: unused
4. **How to fit with local points?**

- **k-NN classification**

$$h(\mathbf{x}) = \text{argmax}_{v \in Y} \sum_{i=1}^{k} \delta(v, y_i), \tag{3}$$

where $\delta(a, b) = 1$ if $a = b$, otherwise 0

- **k-NN regression**

$$h(\mathbf{x}) = \frac{\sum_{i=1}^{k} y_i}{k} \tag{4}$$

# Instance-based learning
# Distance-weighted $k$-NN algorithm

1. **A distance metric**: Euclidian (and many more)
2. **How many nearby neighbours look at?** $k$
3. **A weighting function**: greater weight closer neighbours

$$w_i(\mathbf{x}) \equiv \frac{1}{d(\mathbf{x}, \mathbf{x_i})^2}$$

4. **How to fit with local points?**

- **Classification**

$$h(\mathbf{x}) = \text{argmax}_{v \in Y} \sum_{i=1}^{k} w_i(\mathbf{x}) \delta(v, y_i) \tag{5}$$

- **Regression**

$$h(\mathbf{x}) = \frac{\sum_{i=1}^{k} w_i(\mathbf{x}) y_i}{\sum_{i=1}^{k} w_i(\mathbf{x})} \tag{6}$$

# Instance-based learning
# Distance-weighted $k$-NN algorithm

**Shepard's method**

- **Classification**

$$h(\mathbf{x}) = \operatorname{argmax}_{v \in Y} \sum_{i=1}^{n} w_i(\mathbf{x}) \delta(v, y_i) \tag{7}$$

- **Regression**

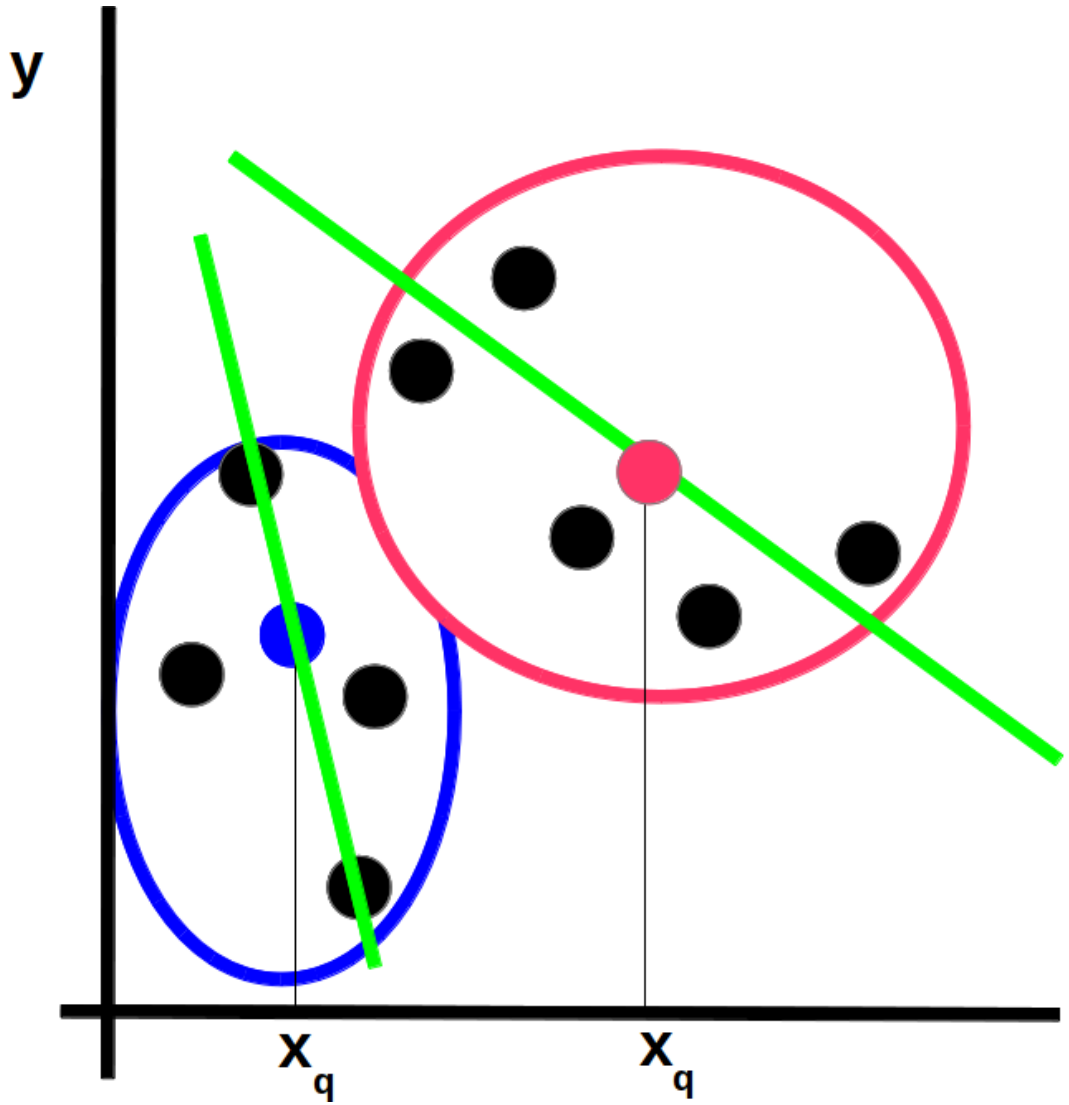
$$h(\mathbf{x}) = \frac{\sum_{i=1}^{n} w_i(\mathbf{x}) y_i}{\sum_{i=1}^{n} w_i(\mathbf{x})} \tag{8}$$

# Instance-based learning
# Locally weighted linear regression

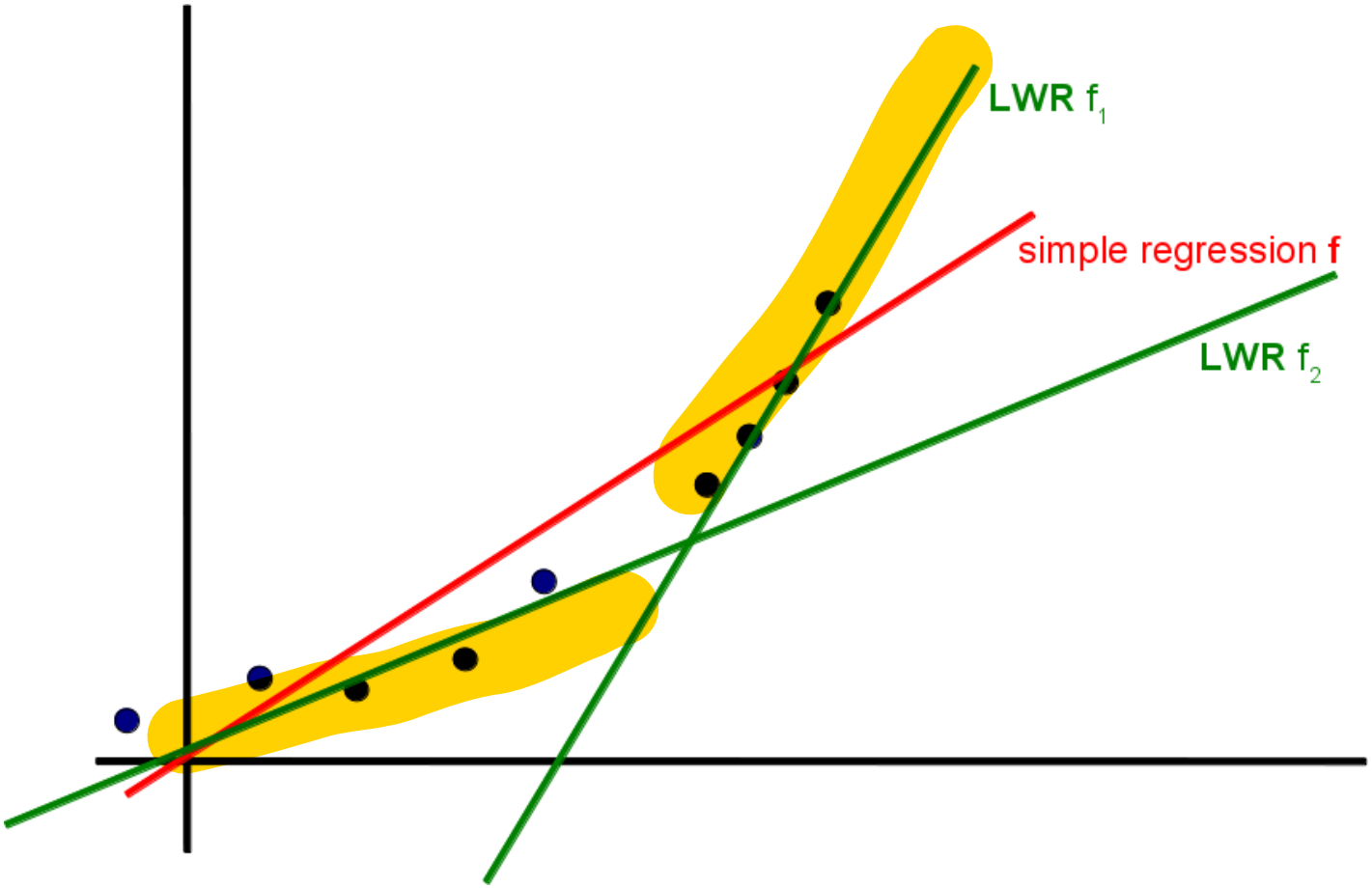
1. **A distance metric**: Euclidian (and many more)
2. **How many nearby neighbours look at?** $k$
3. **A weighting function**: $w_i(\mathbf{x})$
4. **How to fit with local points?**

$$\mathbf{\Theta}^\star = \text{argmin}_{\mathbf{\Theta}} \sum_{i=1}^{k} w_i(\mathbf{x})(\mathbf{\Theta}^T \mathbf{x}_i - y_i)^2 \tag{9}$$

# Instance-based learning
# Locally weighted linear regression



# Instance-based learning
# LW linear regression vs. simple regression



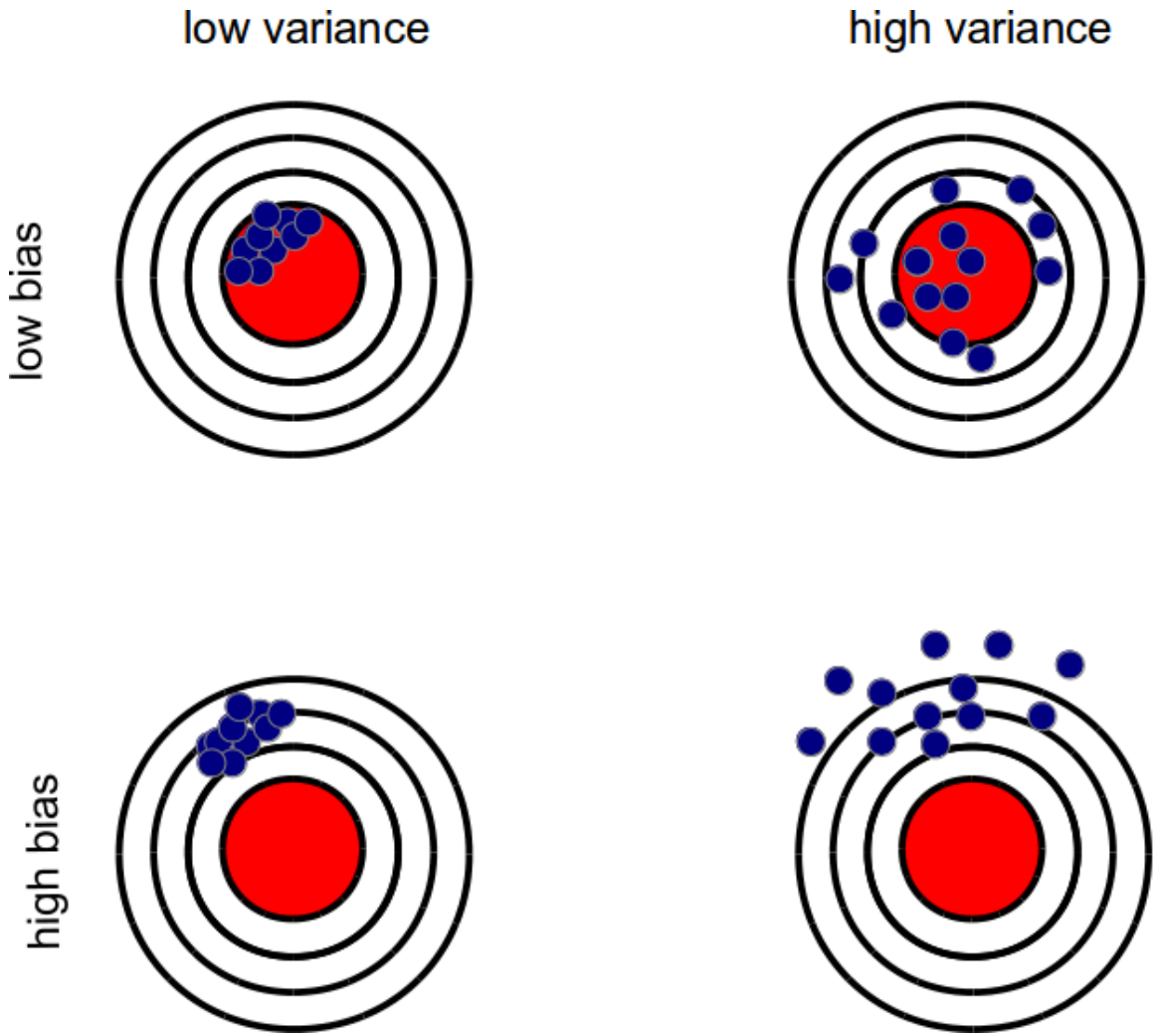
# Bias and variance

1. Select a machine learning algorithm
2. Get $k$ different training sets
3. Get $k$ predictors $h_1^\star, \ldots, h_k^\star$

- **Bias** measures error that originates from the learning algorithm
  – how far off in general the predictions by $k$ predictors are from the true output value

- **Variance** measures error that originates from the training data
  – how much the predictions for a test instance vary between $k$ predictors

# Bias and variance



# Bias and variance

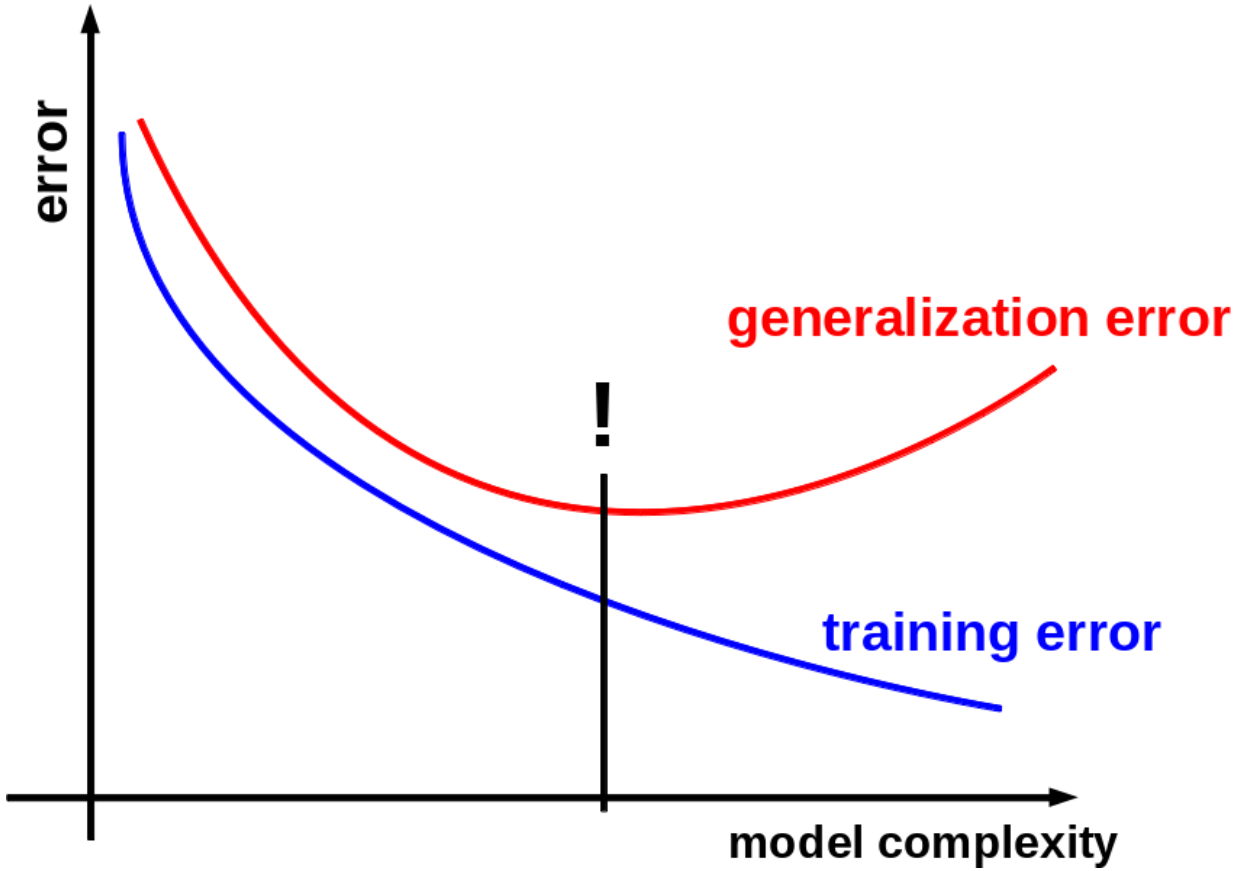
**Generalization error** $\text{error}_{\mathcal{D}}(h)$ measures how well a hypothesis $h$ generalizes beyond the used training data set, to unseen data with distribution $\mathcal{D}$. Usually it is defined as follows

- for **regression**: $\text{error}_{\mathcal{D}}(h) = \mathsf{E}\,(\hat{y}_i - y_i)^2$
- for **classification**: $\text{error}_{\mathcal{D}}(h) = \Pr\,(\hat{y}_i \neq y_i)$

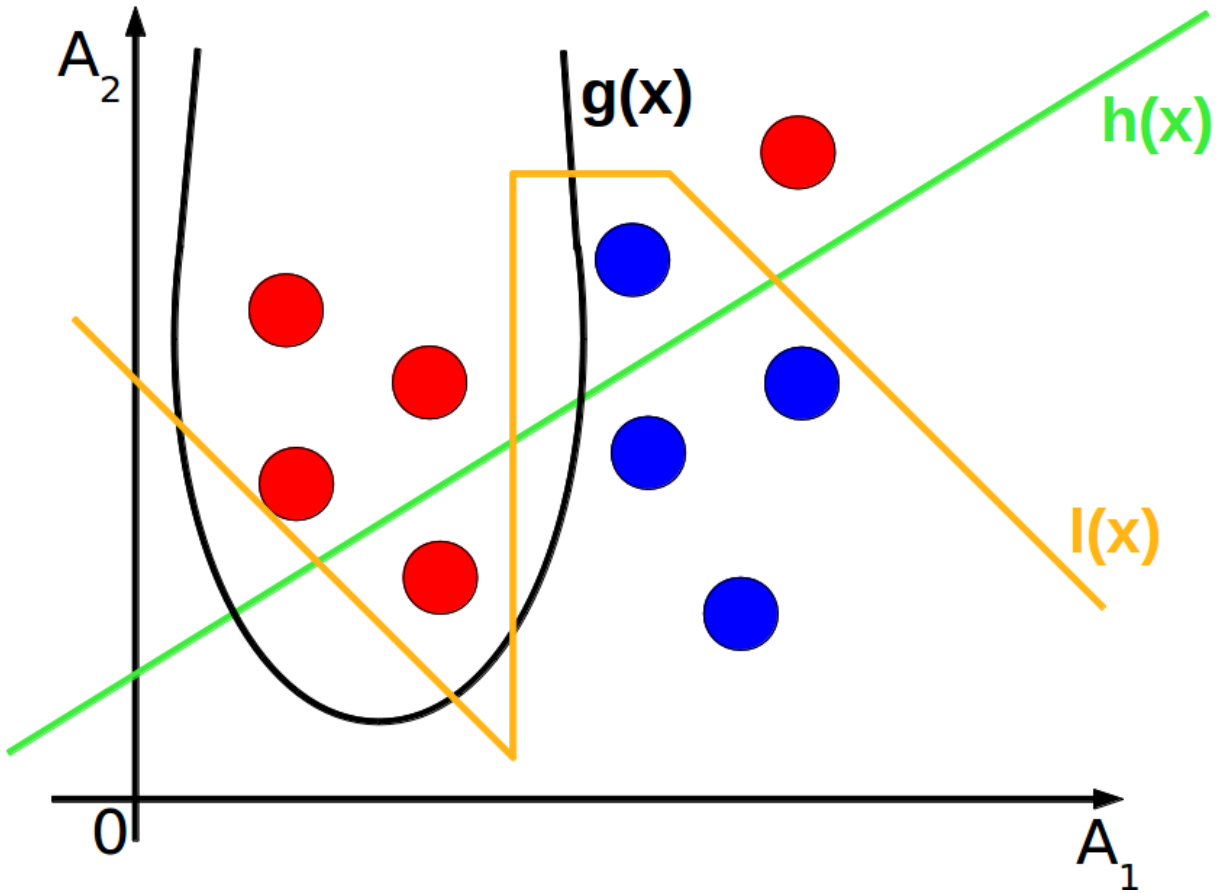
**Decomposition of** $error_{\mathcal{D}}(h)$

$$error_{\mathcal{D}}(h) = \text{Bias}^2 + \text{Variance}$$

# Bias and variance

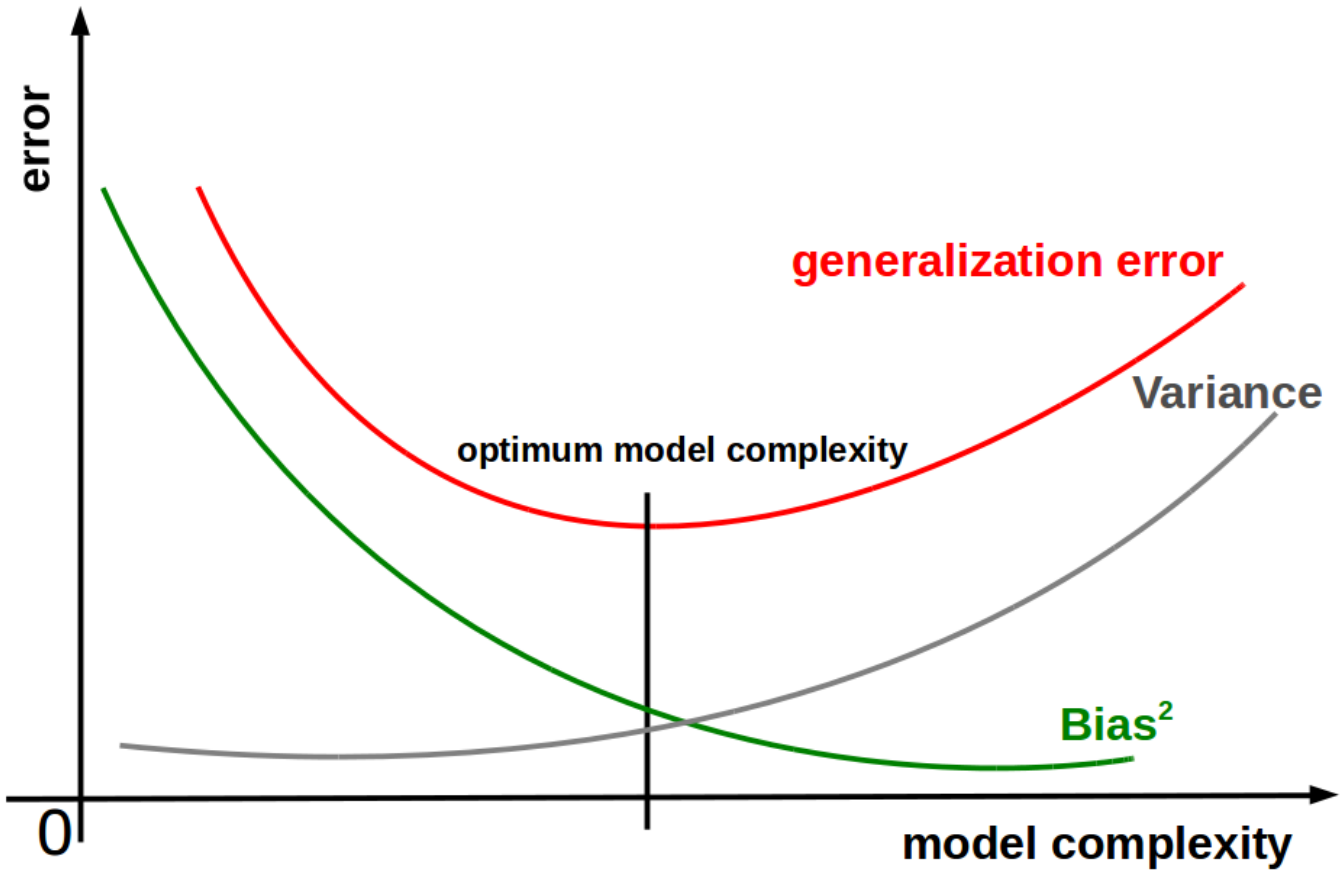


# Bias and variance



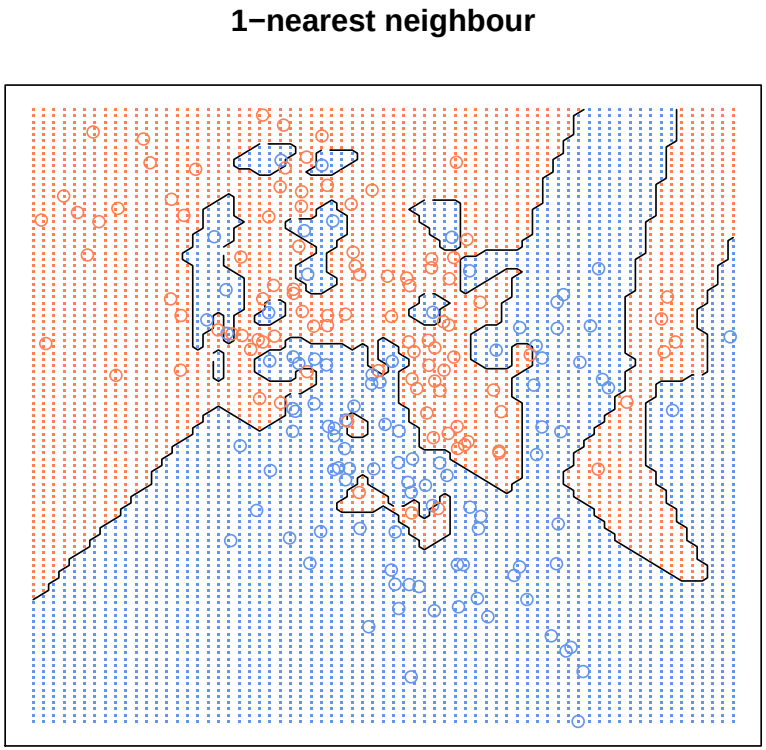
# Bias and variance

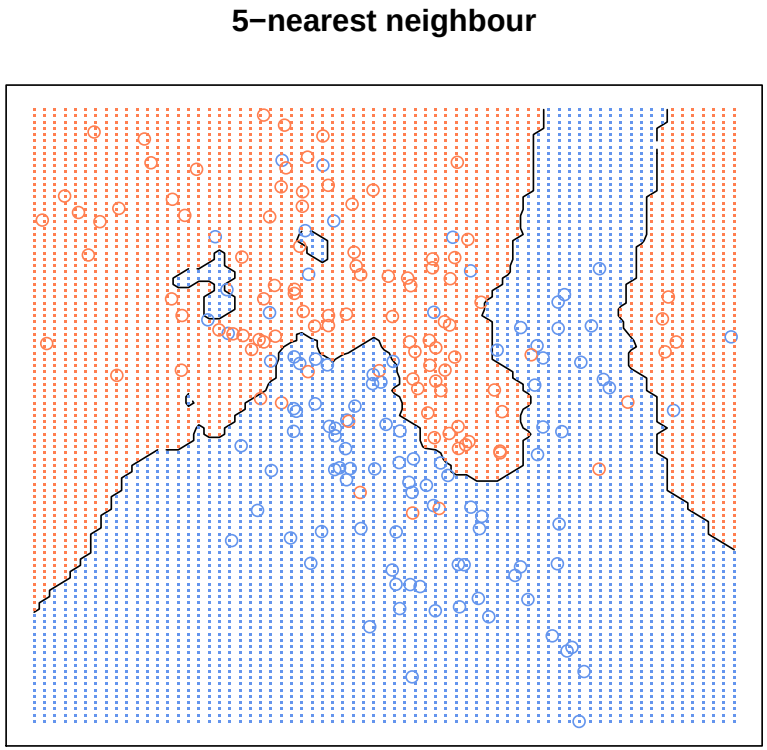
- underfitting = high bias
- overfitting = high variance



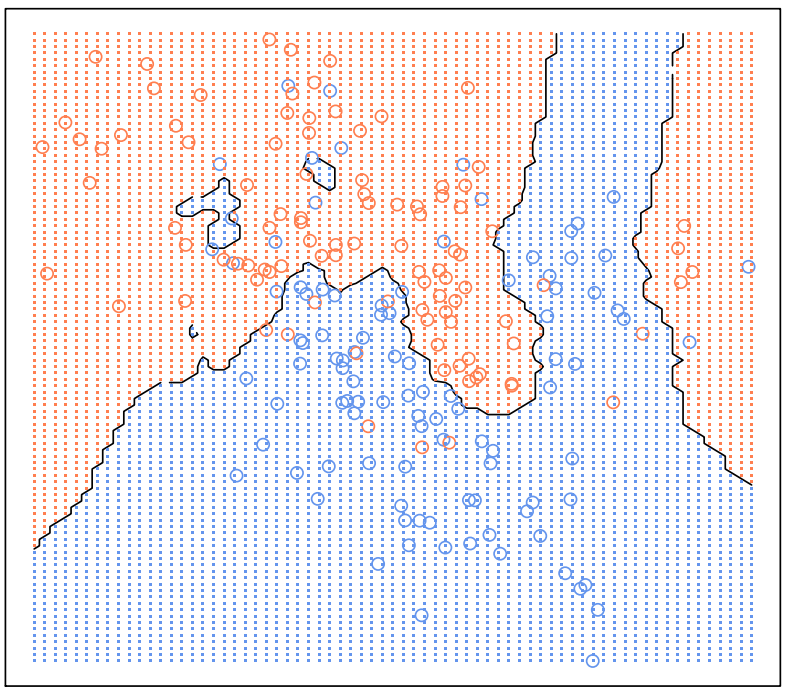
# Bias and variance
# k-Nearest Neighbor

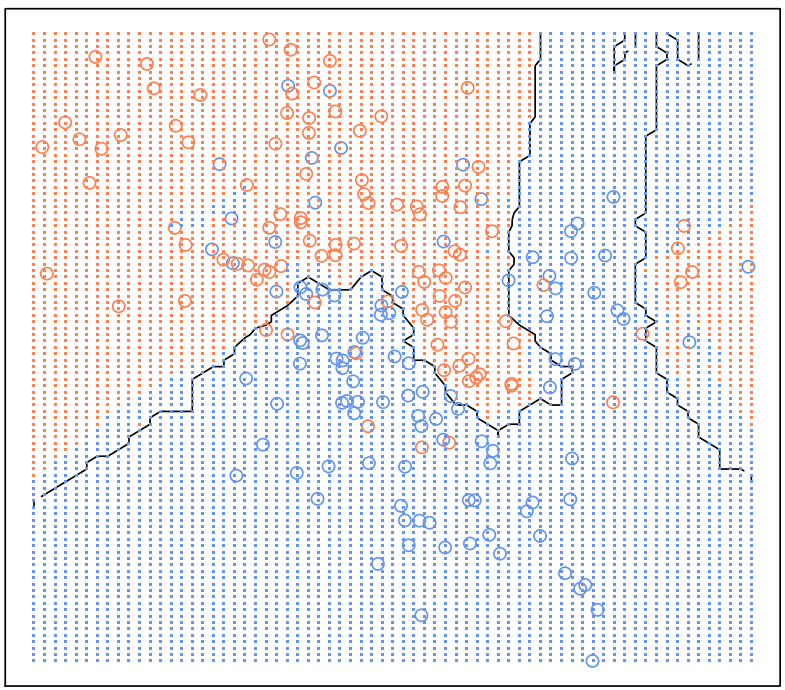
- $\uparrow k \rightarrow \downarrow$ variance and $\uparrow$ bias
- $\downarrow k \rightarrow \uparrow$ variance and $\downarrow$ bias
- Increasing $k$ "simplifies" decision boundary (averaging more instances)



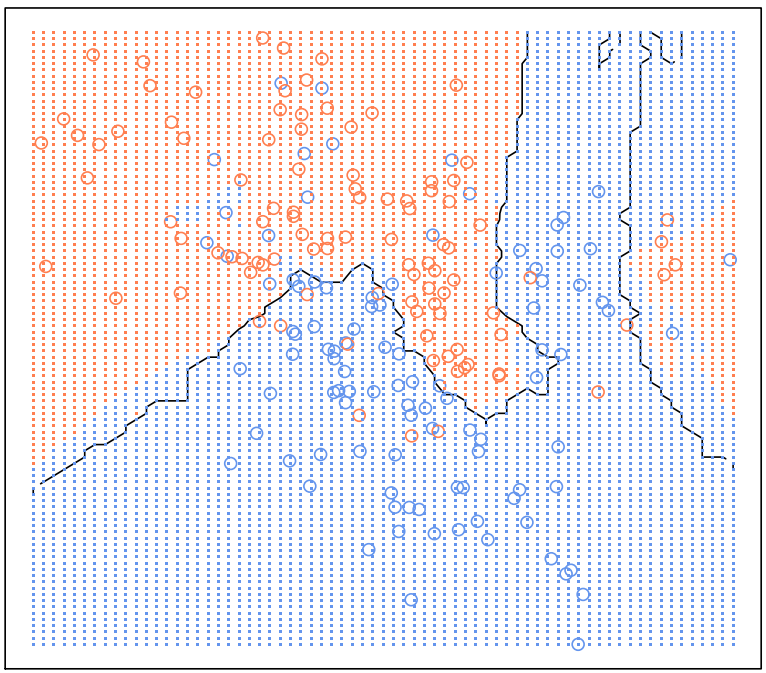
# Bias and variance
# k-Nearest Neighbor



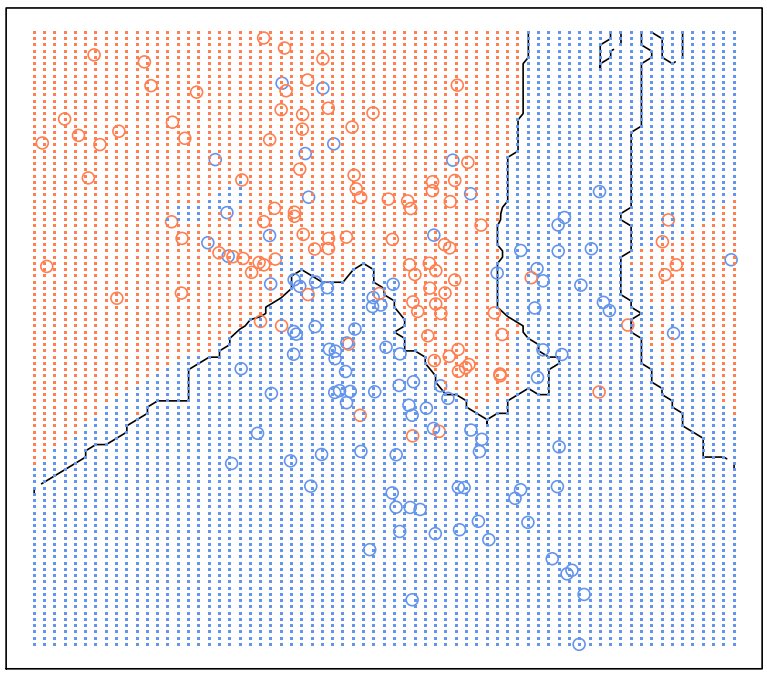



# Bias and variance
# k-Nearest Neighbor

8−nearest neighbour

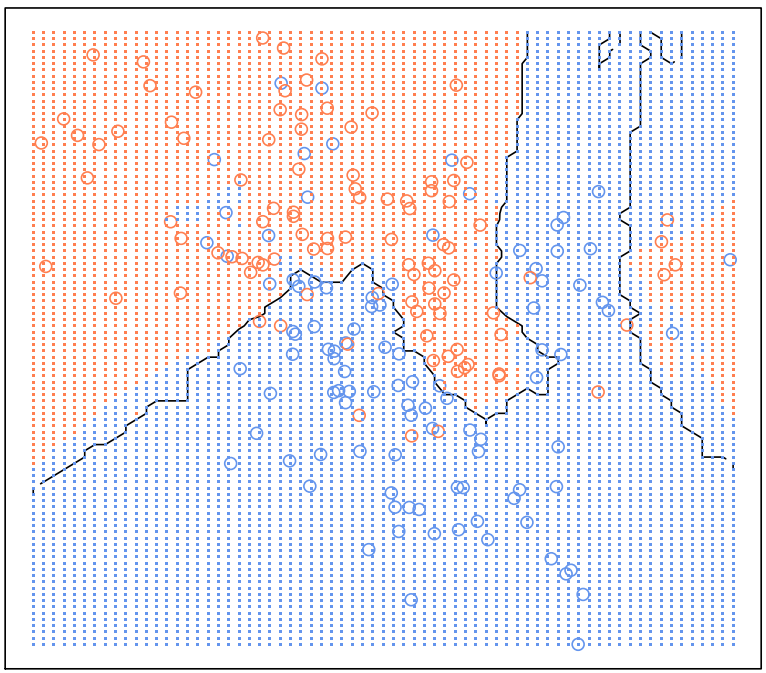


10−nearest neighbour

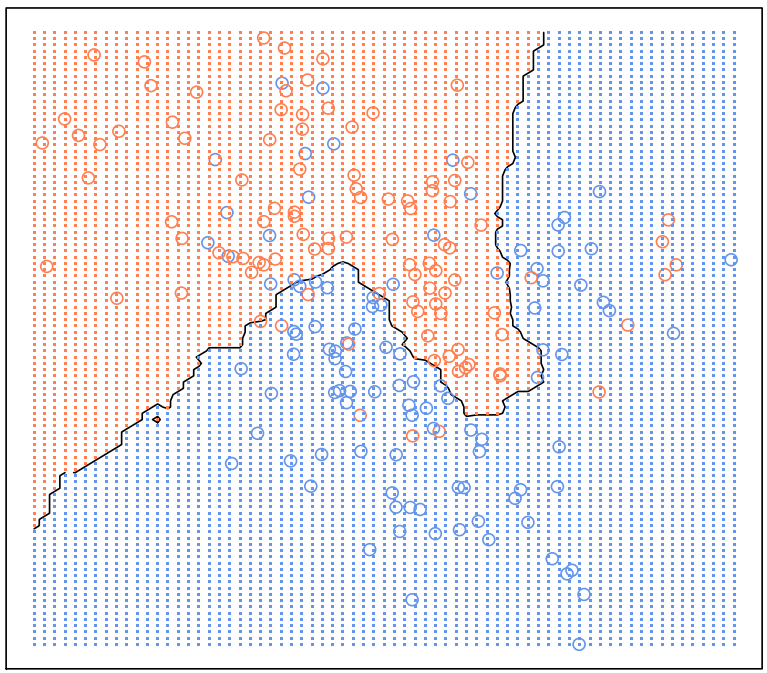


# Bias and variance
# k-Nearest Neighbor

**10−nearest neighbour**



**15−nearest neighbour**



# Naïve Bayes classifier
## Bayes theorem

$$P(A|B) = \frac{P(A,B)}{P(B)}, P(B|A) = \frac{P(A,B)}{P(A)} \tag{10}$$

$$P(A|B) = \frac{P(A) * P(B|A)}{P(B)} \tag{11}$$

- $P(A)$ is the prior probability (marginal) probability of $A$. It does not take into account any information about $B$.
- $P(A|B)$ is the conditional probability of $A$, given $B$. So called the posterior probability because it depends upon the specified value of $B$; $P(B|A)$ is the conditional probability of $B$ given $A$.
- $P(B)$ is the prior (marginal) probability of $B$, and acts as a normalizing constant.

# Naïve Bayes classifier
## Bayes theorem

$$\Pr(Y \mid A_1, \ldots, A_m) = \frac{\Pr(Y)\Pr(A_1, \ldots, A_m \mid Y)}{\Pr(A_1, \ldots, A_m)}$$

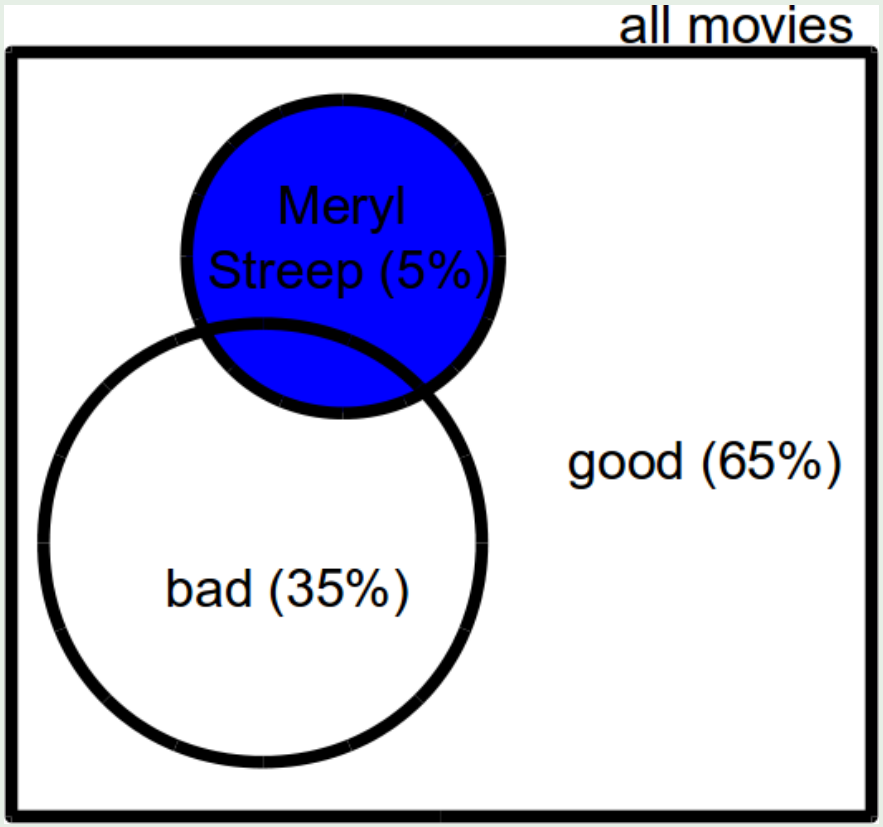
$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

# Naïve Bayes classifier
## Bayes theorem

### Meryl Streep Example

- Assume the discrete random variable *Movie* that has two possible values *good* and *bad* (we do not specify what movie is good/bad). $\Pr(Movie = good)$ denotes the probability that the coming soon movie will be good.
- Assume the discrete random variable *LeadingActress*. $\Pr(LeadingActress = MerylStreep)$ denotes the probability that Maryl Streep will play as a leading actress very soon.

## Meryl Streep Example



# Naïve Bayes classifier
# Bayes theorem

## Meryl Streep Example

| | Meryl Streep | | |
|---|---|---|---|
| | Yes | No | Total |
| bad movie | 1 | 34 | 35 |
| good movie | 4 | 61 | 65 |
| | 5 | 95 | 100 |

- $\Pr(bad, Streep) = ?$
- $\Pr(Streep|bad) = ?$

# Naïve Bayes classifier
# Conditional independence

Let $X, Y$ and $Z$ be three descrete random variables. We say that $X$ is *conditionally independent* of $Y$ given $Z$ if

$\forall x_i, y_j, z_k, x_i \in Values(X), y_j \in Values(Y), z_k \in Values(Z)$ :

$$\Pr(X = x_i | Y = y_j, Z = z_k) = \Pr(X = x_i | Z = z_k) \tag{12}$$

I.e. $\mathrm{P}(X|Y,Z) = \mathrm{P}(X|Z)$.

# Naïve Bayes classifier
# Conditional independence

**Thunder & Rain & Lighting**

Assume three variables: `Thunder`, `Rain`, `Lighting`. `Thunder` is conditionally independent of `Rain` given `Lighting`:

$$\Pr(\text{Thunder}|\text{Rain}, \text{Lighting}) = \Pr(\text{Thunder}|\text{Lighting})$$

# Naïve Bayes classifier
# Discriminative vs. generative classifiers

- Logistic regression classifier is a **discriminative classifier**

$$h_\Theta(\mathbf{x}) = p(y = 1|\mathbf{x}, \Theta)$$

- Naïve Bayes classifier is a **generative classifier**

1. Learn $\Pr(\mathbf{x}|y)$ and $\Pr(y)$
2. Apply Bayes rule to get

$$\Pr(y|\mathbf{x}) = \frac{\Pr(\mathbf{x}|y)\Pr(y)}{\Pr(\mathbf{x})} \sim \Pr(\mathbf{x}|y)\Pr(y)$$

3. 

$$y^\star = \operatorname{argmax}_y \Pr(y|\mathbf{x})$$

# Naïve Bayes classifier
# Discriminative vs. generative classifiers

- **discriminative classifier** does not care about how the data was generated. It simply classifies a given example.
- **generative classifier** models how the data was generated in order to classified an example. It asks the question: based on my generation assumptions, which class is most likely to generate this example?

# Naïve Bayes classifier

If we work with two features $A_1, A_2$ and we assume that they are conditionally independent given the target class $Y$, then

$$\Pr(A_1, A_2|Y) \overset{\text{product rule}}{=} \Pr(A_1|A_2, Y) * \Pr(A_2|Y) \overset{\text{conditional independence assumption}}{=}$$

$$\Pr(A_1|Y) * \Pr(A_2|Y)$$

# Naïve Bayes classifier

Assume conditional independence of features $A_1, \ldots, A_m$ given $Y$. Thus

$$\Pr(x_1, x_2, \ldots, x_m|y) \overset{\text{chain rule}}{=} \prod_{j=1}^{m} \Pr(x_j|x_1, x_2, \ldots, x_{j-1}, y) \overset{\text{c. i. a.}}{=}$$

$$\prod_{j=1}^{m} \Pr(x_j|y)$$

**Naïve Bayes classifier**

$$y^\star = argmax_{y_k \in Y} \Pr(y_k) \prod_{j=1}^{m} \Pr(x_j|y_k) \tag{13}$$

# Naïve Bayes classifier

Naive assumption of feature conditional independence given a target class is rarely true in real world applications. Nevertheless, Naïve Bayes classifier surprisingly often shows good performance in classification.

# Bayesian belief networks
# Motivation

Task: Will students fall asleep during the lecture?

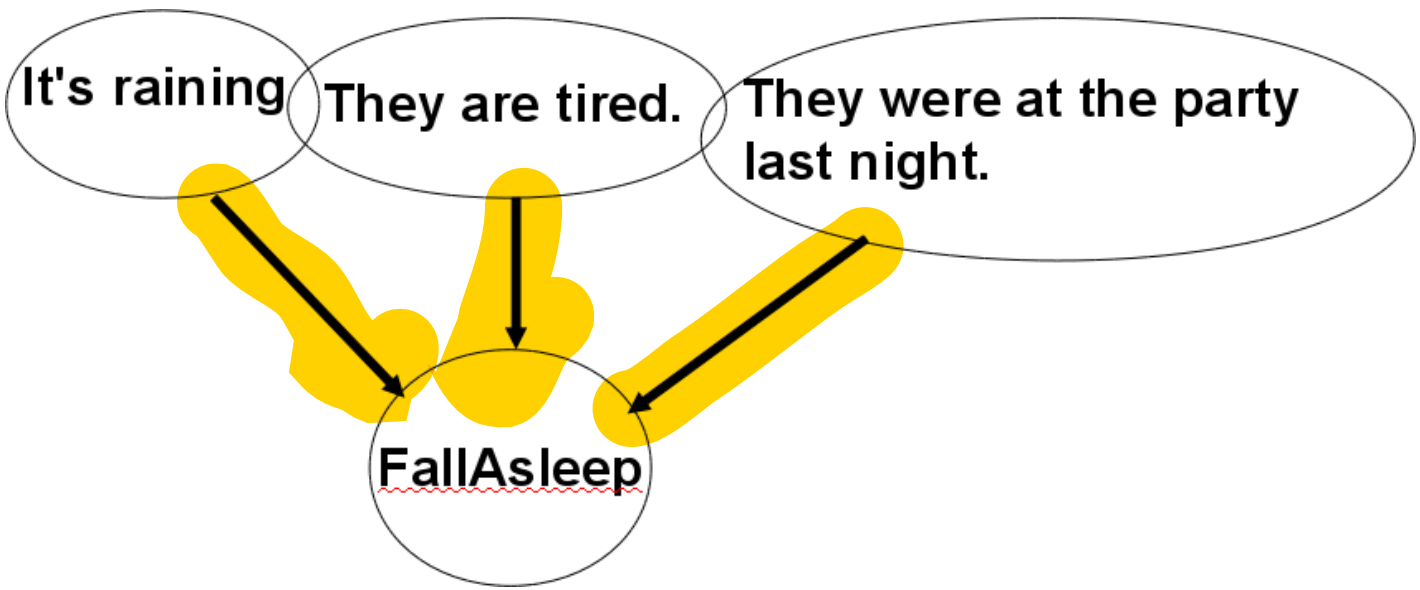
$Attr = \{\texttt{It's raining.}, \texttt{They are tired.}, \texttt{They were at the party last night.}\}$

$Values(\texttt{It's raining.}) = Values(\texttt{They are tired.}) =$
$Values(\texttt{They were at the party last night.}) = \{Yes, No\}$

$Y = \texttt{FallAsleep}$, $Values(\texttt{FallAsleep}) = \{Yes, No\}$

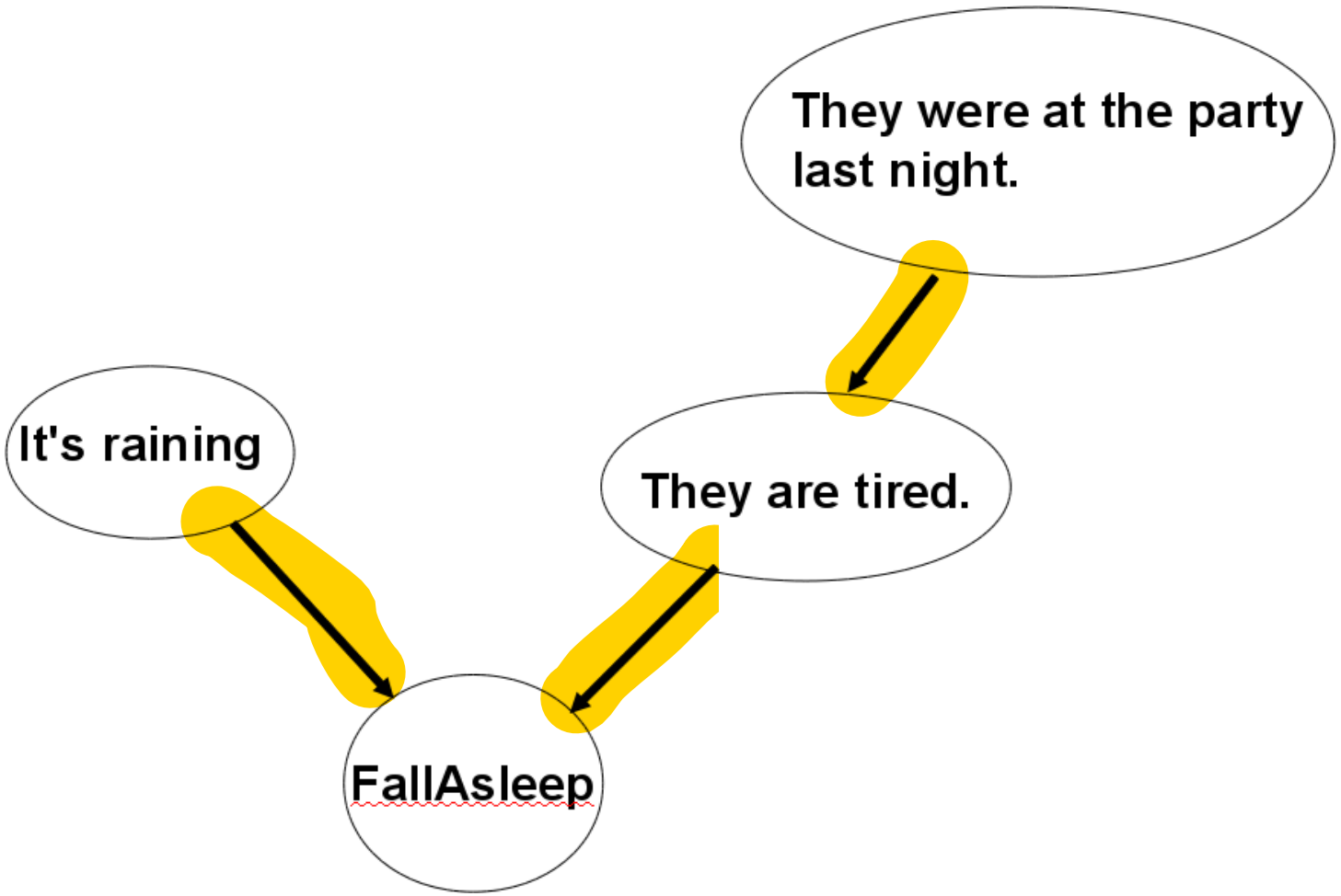
In Naïve Bayes, the features are conditionally independent given the value of the target class.

# Bayesian belief networks
## Motivation



# Bayesian belief networks
# Motivation

... but



# Bayesian belief networks

## Motivation

- Naïve Bayes classifier assumes that ALL features are conditionally independent given the value of the target class.
- A Bayesian network is a graphical model that encodes probabilistic relationships among attributes of interest.
- BBNs allow stating conditional independence assumptions that apply to SUBSETs of the attributes.
- Dependencies are modeled as graph where nodes correspond to attributes and edges go from cause to effect.
- BBNs combine prior knowledge with observed data.
- BBNs are less constraining than the global assumption by NB.

# Bayesian belief networks
## Settings

Consider an arbitrary set of random variables $X_1, X_2, ..., X_m$. Each variable $X_i$ can take on the set of possible values $Values(X_i)$.

We define the **joint space** of the variables $X_1, X_2, ..., X_m$ to be the cross product $Values(X_1) \times Values(X_2) \times Values(X_3) \times ... \times Values(X_m)$.

The probability distribution over the joint space is called the **joint probability distribution**
$P(x_1, x_2, ..., x_m), x_1 \in Values(X_1), x_2 \in Values(X_2), ..., x_n \in Values(X_m)$.

BBN describes the joint probability distribution for a set of variables by specifying a set of conditional independence assumptions together with sets of local conditional probabilities.

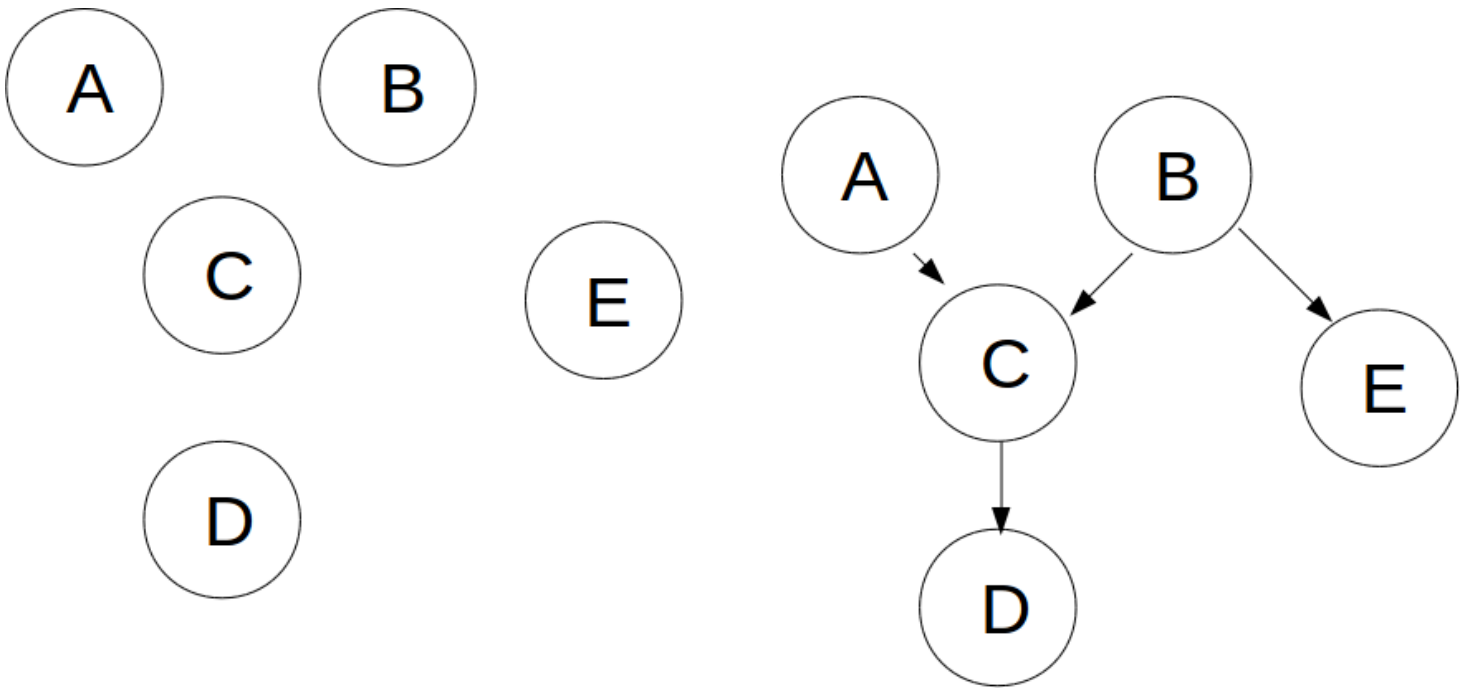
# Bayesian belief networks

**Representation**

1. A directed acyclic graph $G = (V, E)$: a network of nodes, one for each variable, and arcs between nodes represent probabilistic dependencies - $Attr$, $i \in V$ corresponds to $X_i \in Attr$; arcs are drawn from cause to effect.
2. The network arcs represent the assertion that the variable $X$ is conditionally independent of its nondescendants given its immediate predecessors $Parents(X)$; $\Pr(X|X_i)_{X_i \in Parents(X)}$
   We say $Y$ is a *descendant* of $X$ if there is a directed path from $X$ to $Y$.
3. A set of tables for each node in the graph - a conditional probability table is given for each variable; it describes the probability distribution for that variable given the values of its immediate predecessors.

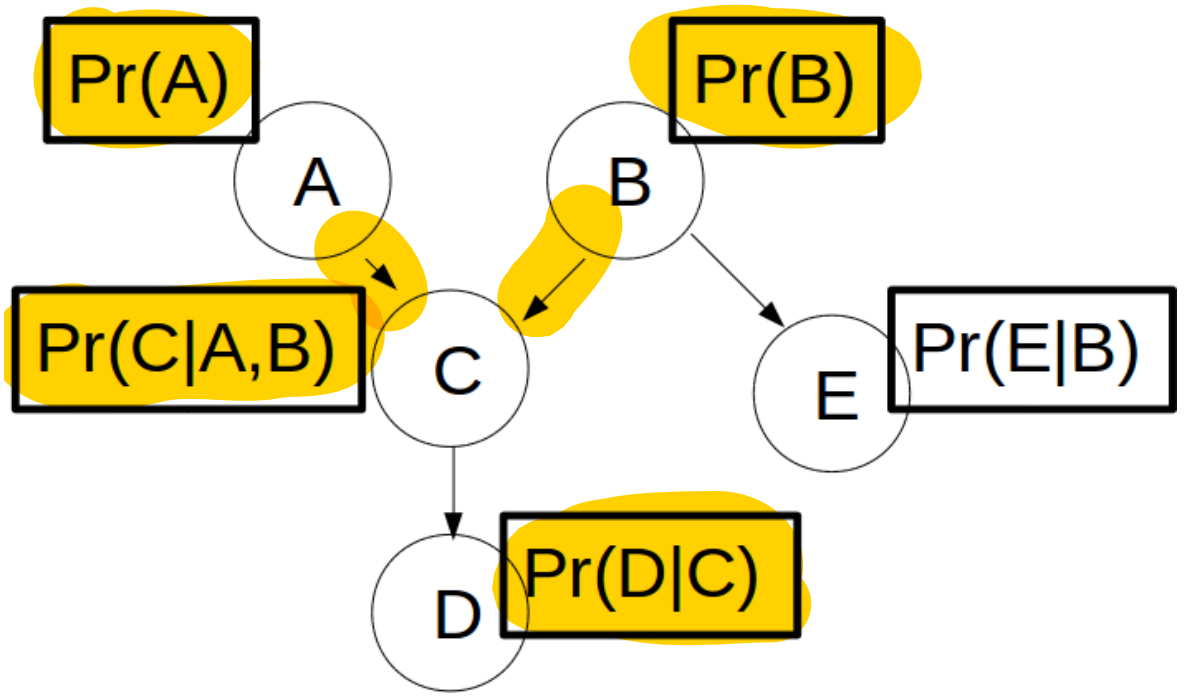
# Building a Bayes net

1. Choose the variables to be included in the net: $A, B, C, D, E$
2. Add the links.



# Building a Bayes net

**3.** Add a probability table for each node $\Pr(X|X_i)_{X_i \in Parents(X)}$.



# Once the net is built ...

The join probability of any assignment of values $x_1, x_2, ..., x_m$ to the tuple of network variables $X_1, X_2, ..., X_m$ can be computed by the formula

$$\Pr(x_1, x_2, ..., x_m) = \prod_{i=1}^{m} \Pr(X|X_i)_{X_i \in Parents(X)} \tag{14}$$

# Bayesian belief networks

Two components:

1. A function for evaluating a given network based on the data.
2. A method for searching through the space of possible networks.

Learning the network structure:

searching through the space of possible sets of edges, estimating the conditional probability tables for each set, and computing the quality of the network (network scoring; probability of the data given the network).

# K2 algorithm

This 'search and score' algorithm heuristically searches for the most probable belief-network structure given a training data.

It starts by assuming that a node has no parents, after which, in every step it adds incrementally the parent whose addition mostly increase the probability of the resulting structure. K2 stops adding parents to the nodes when the addition of a single parent cannot increase the probability of the network given the data.

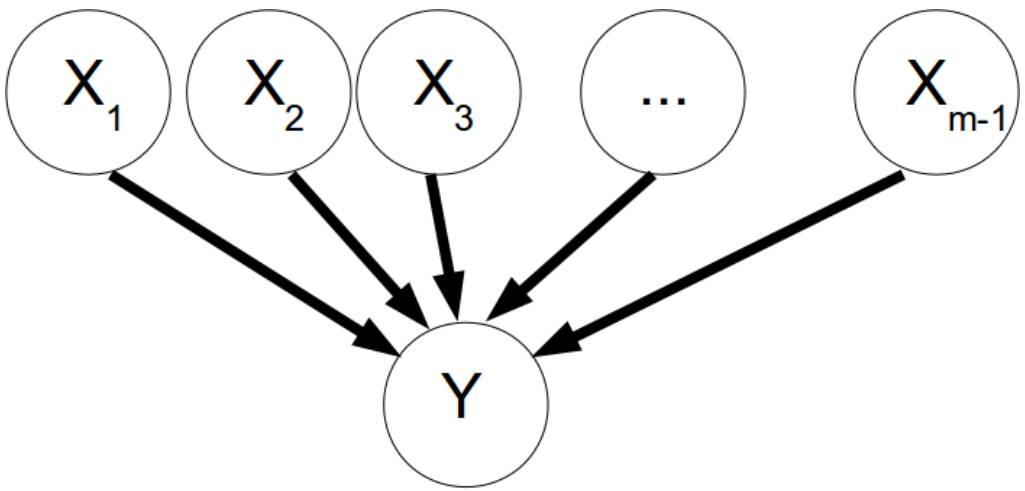
In general, the BBNs deal with probability propagation that consists of updating the probability values of the variables in a dependence graph, given some variables that have been observed.

# K2 algorithm

- **INPUT:** a set of $m$ nodes (i.e., attributes), an ordering on the nodes $X_1, X_2, ..., X_m$ (parents are before their kids), an upper bound $u$ on the number of parents a node may have, training data $D$, $|D| = n$
- **OUTPUT:** for each node, a printout of the parent nodes

Note on the initial nodes ordering:

the Naïve Bayes Classifier is a network with an edge leading from the target feature to each other features. This network can be used as a starting point for the search.



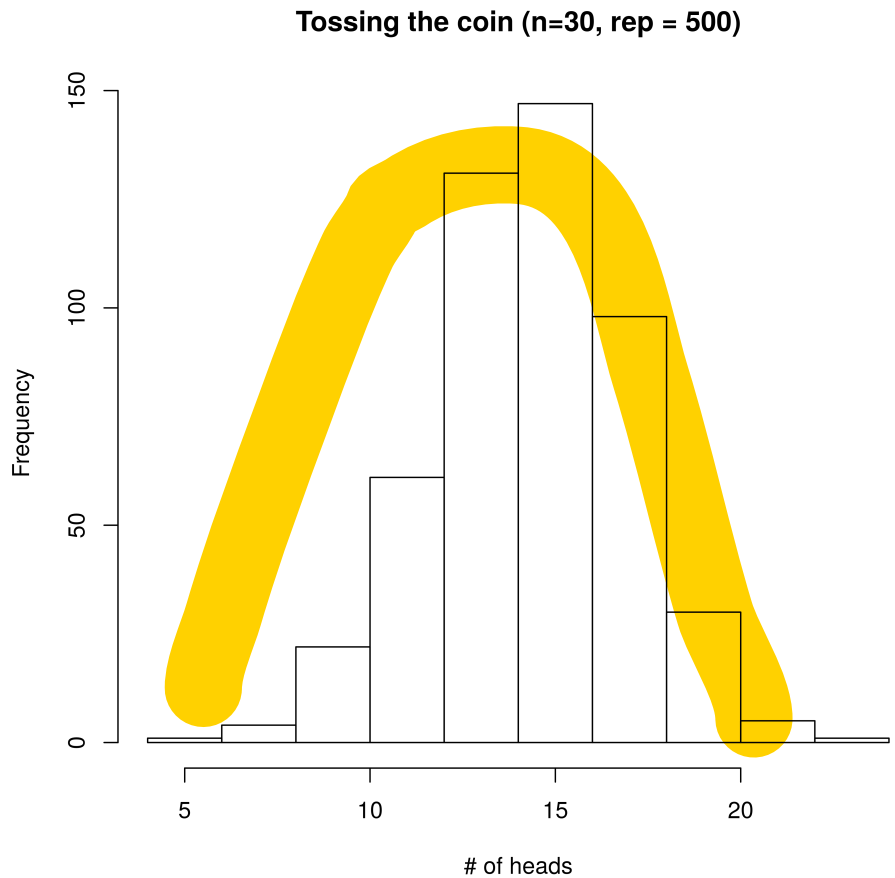
# Maximum likelihood estimation
# Example #1

**Tossing the coin 30 times**

```
> toss.coin <- function(n) sample(0:1,n,rep=T)
> e <- toss.coin(30)
> f.1 <- sum( e == 1)
> rf.1 <- sum( e == 1)/length(e)
> # repeat the experiment 500 times
> rep <- 500
> for (i in seq(1,rep,1)){
> e <- toss.coin(30)
> f.1[i] <- sum( e == 1)
> }
> hist(f.1, xlab="# of heads", ylab = "Frequency",
       main = "Tossing the coin (n=30, rep = 500)")
```

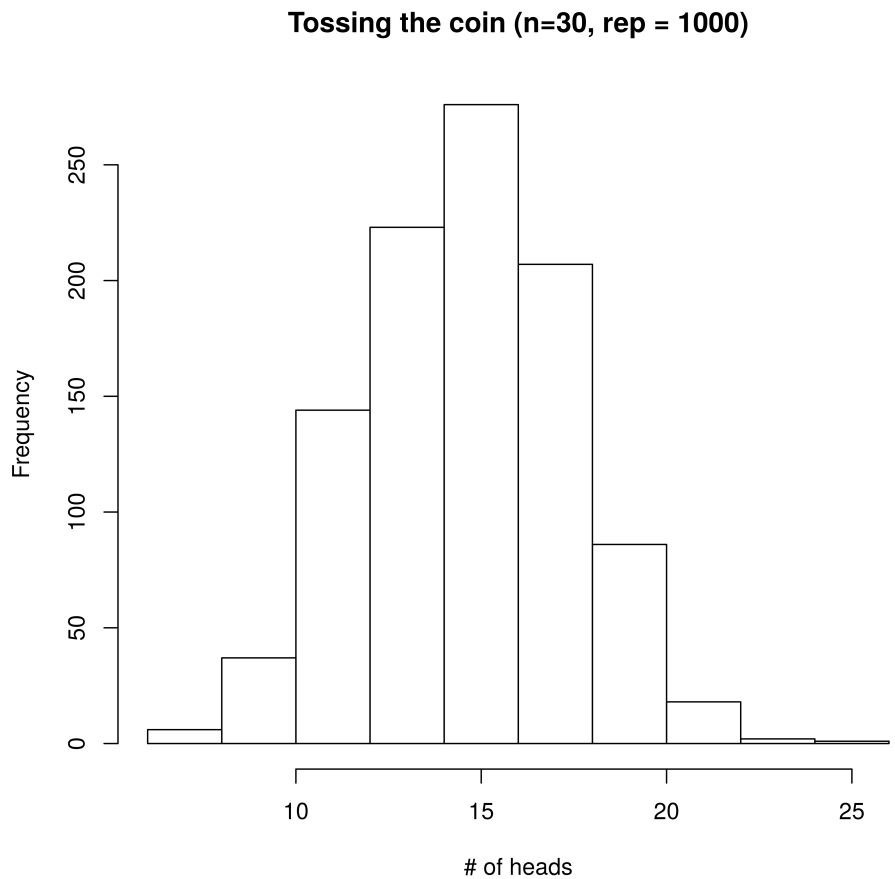
# Maximum likelihood estimation
# Example #1

## Tossing the coin 30 times



# Maximum likelihood estimation
# Example #1

## Tossing the coin 30 times



# Maximum likelihood estimation
# Example #1

The binomial distribution is the discrete probability distribution of the number of successes in a sequence of $n$ independent yes/no experiments, each of which yields success with probability $p$, $X \sim Bin(n, p)$.

**Probabilistic mass function** $\Pr(X = k) = f(k; n, p) = \frac{n!}{k!(n-k)!} p^k (1-p)^{(n-k)}$

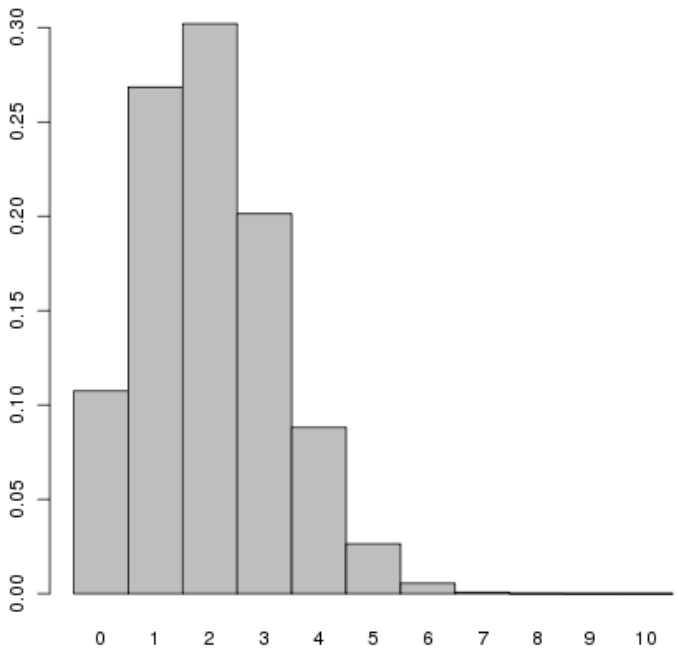
## Coin tossing

Let $n = 10$, $x$ represents the number of successes in $n$ trials and probability of head on one trial is $p = 0.2$. Then

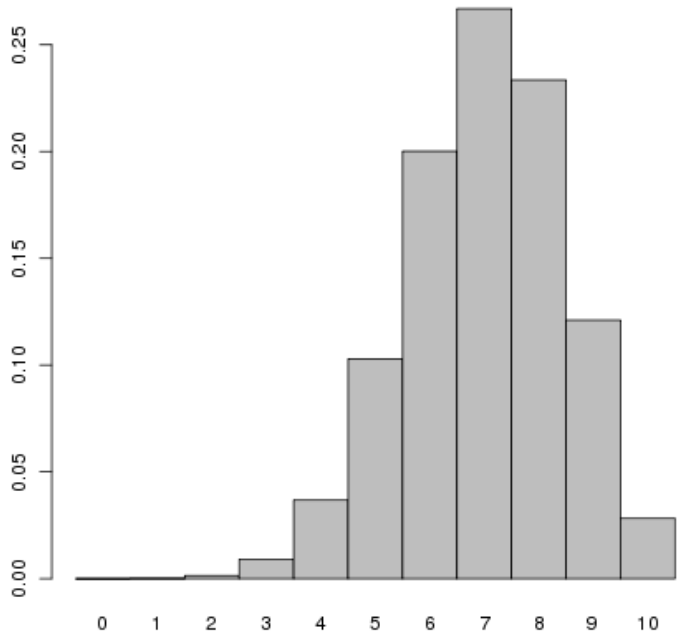
$$f(x; 10, 0.2) = \frac{10!}{x!(10-x)!} 0.2^x (0.8)^{(10-x)}$$

# Maximum likelihood estimation
# Example #1

$p = 0.2$




$p = 0.7$



# Maximum likelihood estimation

- $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$

**Assumption**

- $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are independent and identically distributed
  - with an unknown probability density function (PDF) $f(\mathbf{X}; \Theta)$
  - joint density function $f(\mathbf{x}_1, \ldots, \mathbf{x}_n; \Theta) \overset{i.i.d.}{=} \prod_{i=1}^{n} f(\mathbf{x}_i; \Theta)$
  - unknown $\Theta$

We determine what value of $\Theta$ would make the data that we observed most likely.

# Maximum likelihood estimation

MLE is a general method for estimating parameters of interest from data.
**Goal:** identify the population that is most likely to have generated the sample.
Given the sample and a type of probability distribution, we find the PDF that is most likely to have generated the sample.

**Likelihood function**

$$\mathcal{L}(\Theta|\mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{i=1}^{n} f(\mathbf{x}_i; \Theta) \tag{15}$$

**Log-likelihood function**

$$\log \mathcal{L}(\Theta|\mathbf{x}_1, \ldots, \mathbf{x}_n) = \sum_{i=1}^{n} \log f(\mathbf{x}_i; \Theta) \tag{16}$$

**Maximum likelihood estimate of $\Theta$**

$$\Theta^{\star}_{MLE} = \operatorname{argmax}_{\Theta} \log \mathcal{L}(\Theta|\mathbf{x}_1, \ldots, \mathbf{x}_n) \tag{17}$$

# Maximum likelihood estimation

**MLE analytically**

- Likelihood equation: $\frac{\partial \log \mathcal{L}(\Theta|X)}{\partial \Theta_i} = 0$ at $\Theta_i$ for all $i = 1, \ldots, m$
- Maximum, not minimum: $\frac{\partial^2 \mathcal{L}(\theta|\mathbf{x})}{\partial \Theta_i^2} < 0$

**Numerically**

- Use an optimization algorithm (for ex. gradient descent)

# Maximum likelihood estimation
# Discrete uniform distribution

**Students from Prague in Paris**

We meet four students of the Faculty of Mathematics and Physics on the top of the Eiffel tower in Paris. Each of them has got his/her student id card: #258, #103, #498, #99. Estimate the number of students at the Faculty of Mathematics and Physics.

- **x** is the ID of a randomly chosen student
- **X** is uniformly distributed on $1, 2, \ldots, N$
- $f(x; n) = 1/n$

**Goal:** estimate N

- $\mathcal{L}(N|\boldsymbol{X}) = f(258; N)f(103; N)f(498; N)f(99; N) = (1/N)^4$
- $N^{\star}_{MLE} = \operatorname{argmax}_N \log \mathcal{L}(N|\boldsymbol{X}) = \operatorname{argmax}_N \log(1/N)^4$
- choose $N$ as small as possible, i.e $N^{\star}_{MLE} = 498$

# Maximum likelihood estimation
# Binomial distribution

Estimate the probability $p$ that a coin lands heads using the result of $n$ coin tosses, $k$ of which resulted in heads.

- $f(k; n, p) = \frac{n!}{k!(n-k)!}(p)^k(1-p)^{(n-k)}$
- $\mathcal{L}(p|n, k) = \frac{n!}{k!(n-k)!}(p)^k(1-p)^{(n-k)}$
- $\log \mathcal{L}(p|n, k) = \log \frac{n!}{k!(n-k)!} + k \log p + (n-k) log(1-p)$
- $\frac{\partial \log \mathcal{L}(p|n,k)}{\partial p} = \frac{k}{p} + \frac{n-k}{1-p} = 0$
- $p^{\star}_{MLE} = \frac{k}{n}$

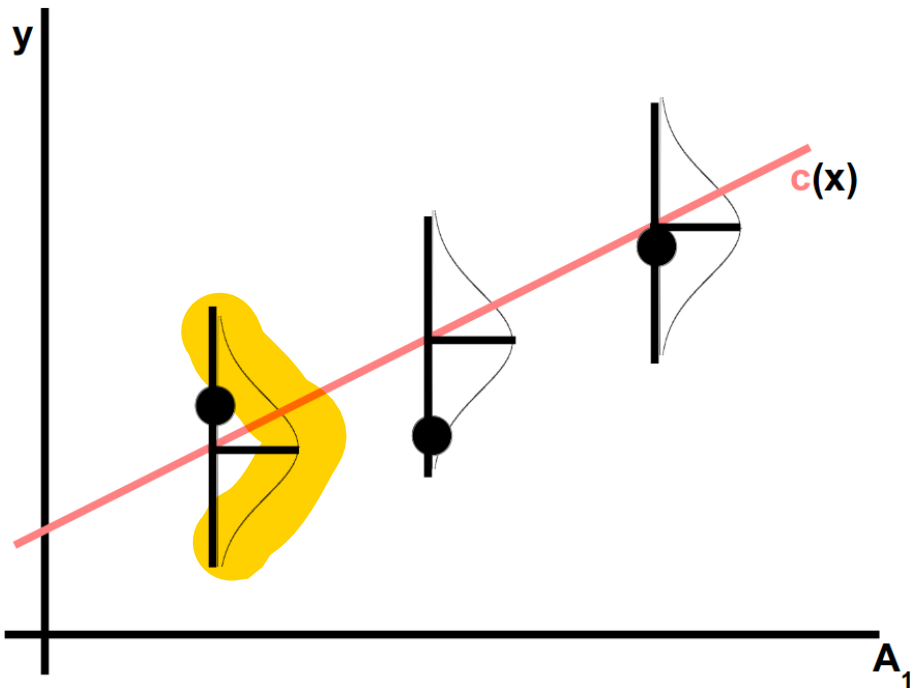
# Maximum likelihood estimation
## Least squares

$$y_i = \Theta^T \mathbf{x}_i + \epsilon_i$$

At each value of $A_1$, the output value $y$ is subject to random error $\epsilon$ that is distributed normally with mean $\mu = 0$ and standard deviation $\sigma$ ($N(0, \sigma^2)$).

Learn $\Theta^\star$ from
$Data = \{\langle \mathbf{x}_i, y \rangle, y \in \mathcal{R}, i = 1, ..., n\}$.

Use MLE.



# Maximum likelihood estimation
# Least squares

- probability density function of the Normal distribution

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

-

$$\mathcal{L}(\mu,\sigma|\epsilon) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\sum_{i=1}^{n}(\epsilon_i-\mu)^2}{2\sigma^2}}$$

- $\epsilon_i = y_i - \boldsymbol{\Theta}^T \mathbf{x}_i \sim N(0,\sigma^2)$. The likelihood distribution function is

$$\mathcal{L}(\Theta,\sigma|\mathbf{X},\mathbf{y}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i-\boldsymbol{\Theta}^T\mathbf{x}_i)^2}{2\sigma^2}}$$

# Maximum likelihood estimation
# Least squares

- $$\log \mathcal{L}(\Theta, \sigma | \mathbf{X}, \mathbf{y}) = \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(y_i - \mathbf{\Theta}^T \mathbf{x}_i)^2$$
- $$\max \log \mathcal{L}(\Theta, \sigma | \mathbf{X}, \mathbf{y}) = \max \sum_{i=1}^{n} -\frac{1}{2\sigma^2}(y_i - \mathbf{\Theta}^T \mathbf{x}_i)^2$$
- $$\min \log \mathcal{L}(\Theta, \sigma | \mathbf{X}, \mathbf{y}) = \min \sum_{i=1}^{n} (y_i - \mathbf{\Theta}^T \mathbf{x}_i)^2$$
- The least square estimates are equivalent to the maximum likelihood estimates under the assumption that $Y$ is generated by adding random noise to the true target values characterized by the Normal distribution $N(0, \sigma^2)$

# Maximum likelihood estimation
# Logistic regression

Logistic regression models conditional probability using linear function.

$$h(\mathbf{x}) = \frac{1}{1 + e^{-\Theta^T \mathbf{x}}} = \Pr(y = 1|\mathbf{x})$$

Learn $\Theta^\star$ from $Data = \{\langle \mathbf{x}_i, y \rangle, y \in \{0, 1\}, i = 1, ..., n\}$. Use MLE.

# Maximum likelihood estimation
# Logistic regression

- $h(\mathbf{x}; \Theta) = \Pr(y = 1|\mathbf{x})$
- $\prod_{i=1}^{n} \Pr(y = y_i|\mathbf{x}_i) = \prod_{i=1}^{n} h(\mathbf{x}_i; \Theta)^{y_i}(1 - h(\mathbf{x}_i; \Theta))^{1-y_i}$
- $$\mathcal{L}(\Theta|\mathbf{X}, \mathbf{y}) = \prod_{i=1}^{n} h(\mathbf{x}_i; \Theta)^{y_i}(1 - h(\mathbf{x}_i; \Theta))^{1-y_i}$$
- $$\mathcal{L}(\Theta|Data) = \sum_{i=1}^{n} y_i \log h(\mathbf{x}_i; \Theta) + (1 - y_i) \log(1 - h(\mathbf{x}_i; \Theta))$$
- $\Theta^{\star}_{MLE} = \operatorname{argmax}_{\boldsymbol{\Theta}} \sum_{i=1}^{n} y_i \log h(\mathbf{x}_i; \Theta) + (1 - y_i) \log(1 - h(\mathbf{x}_i; \Theta))$

### Categorical features

- $\Theta_j(x|y) = \Pr(x|y)$, $x \in A_j, y \in Y$
- $\Theta(y) = \Pr(y)$, $y \in Y$

Where to get $\Theta_j(x|y)$ and $\Theta(y)$?

# Maximum likelihood estimation
# Naïve Bayes classifier

**Theorem**

*The Maximum likelihood estimates for NB take the form*

- $\Theta(y) = \frac{c_y}{n}$ *where* $c_y = \sum_{i=1}^{n} \delta(y_i, y)$
- $\Theta_j(x|y) = \frac{c_{j_{x,y}}}{c_y}$ *where* $c_{j_{x|y}} = \sum_{i=1}^{n} \delta(y_i, y)\delta(\mathbf{x}_{i_j}, x)$

# Maximum likelihood estimation
# Naïve Bayes classifier

## Categorical features

### Laplace smoothing

A new instance $\mathbf{x} = \langle x_1, \ldots, x_j, \ldots, x_m \rangle$. If $x_j$ does not occur in training data, then $\sum_{i=1}^{n} \delta(\mathbf{x}_{i_j}, x_j) = 0$ and $\mathrm{P}(y|\mathbf{x}) = \frac{0}{0}$!

We add "hallucinated" instances that are spread evenly over $Values(A_j)$. Then the ML estimates take the form

- $\Theta(y) = \frac{c_y}{n}$ where $c_y = \sum_{i=1}^{n} \delta(y_i, y)$
- $\Theta_j(x|y) = \frac{c_{j_{x|y}}}{c_y + l|Values(A_j)|}$ where $c_{j_{x|y}} = \sum_{i=1}^{n} \delta(y_i, y)\delta(\mathbf{x}_{i_j}, x) + l$

$l$ is the strength of the smoothing.

# Maximum likelihood estimation
# Naïve Bayes classifier

**Continuous features**

Typical assumption, each continuous feature has a Gaussian distribution.

**Theorem**

*The ML estimates for NB take the form*

- $\overline{\mu_k} = \frac{\sum_{j=1}^{n} x_i^j \delta(Y^j = y_k)}{\sum_{j=1}^{n} \delta(Y^j = y_k)}$
- $\overline{\sigma_k}^2 = \frac{\sum_j (x_i^j - \overline{\mu_k})^2 \delta(Y^j = y_k)}{\sum_j \delta(Y^j = y_k)}$

$$\Pr(\mathbf{X}_i = x | Y = y_k) = \frac{1}{\sqrt{2\pi\overline{\sigma_k}^2}} e^{\frac{-(x-\overline{\mu_k})^2}{2\overline{\sigma_k}^2}}$$

# Evaluation of binary classifiers
# Confusion matrix

**Binary classification** $\overset{aka}{=}$ **2-class classification** $\overset{aka}{=}$ **0/1 classification**

In binary classification tasks, examples are divided into two disjoint subsets:

- **positive examples** – "to be retrieved" (ones)
- **negative examples** – "not to be retrieved" (zeros)

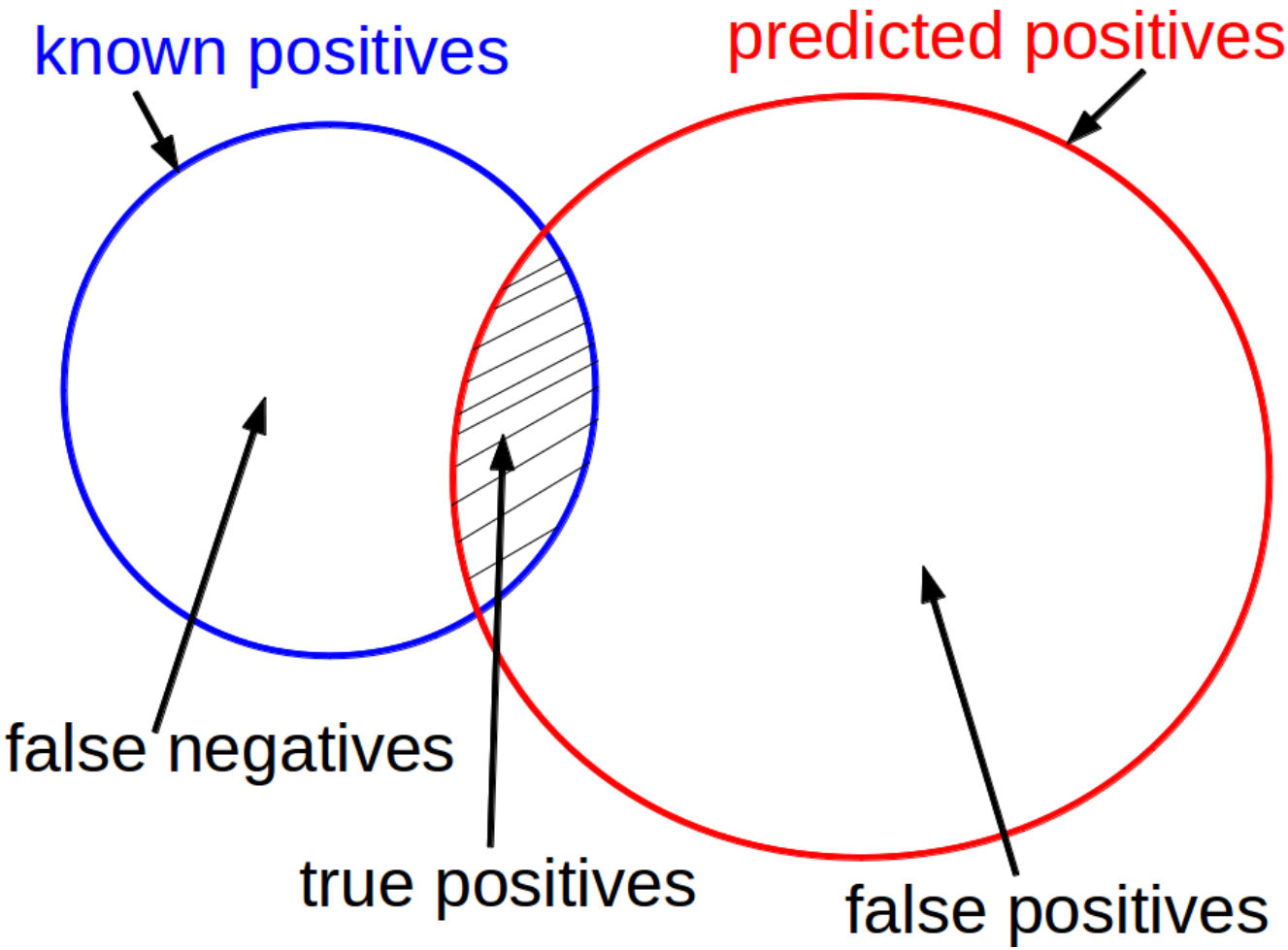
```
# Example of confusion matrix for binary classification
> table(cv.test$Class, pred.test)
       prediction
          0   1
 true  0 580  69
       1  37 144
>
```

# Evaluation of binary classifiers
# Confusion matrix

| | | Predicted class | |
|---|---|---|---|
| | | **Positive** | **Negative** |
| **True class** | **Positive** | True Positive (TP) | False Negative (FN) |
| | **Negative** | False Positive (FP) | True Negative (TN) |

- **'Trues'** are examples correctly classified
- **'Falses'** are examples incorrectly classified
- **'Positives'** are examples predicted as positives (correctly or incorrectly)
- **'Negatives'** are examples predicted as negatives (correctly or incorrectly)

# Evaluation of binary classifiers
# Confusion matrix



# Evaluation of binary classifiers
# Basic performance measures

| Measure | Formula |
|---|---|
| Precision | TP/(TP+FP) |
| Recall/Sensitivity | TP/(TP+FN) |
| Specificity | TN/(TN+FP) |
| Accuracy | (TP+TN)/(TP+FP+TN+FN) |

Very often you need to **combine both good precision and good recall**. Then you usually use **balanced F-score**, so called **F-measure**

$$F = 2\,\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

# Evaluation of binary classifiers
# Basic performance measures

**Example** (see lab # 5)

```
# training
> m <- glm(train$hon ~ train$math + train$female + train$read,
             data = train, family = binomial(link = 'logit'))

# prediction
> out <- predict.glm(m, test, type = "response")

# prediction rule
> threshold <- .5
> class <- vector()
> for(i in 1:length(out)) {
+ if(out[i] >= threshold) {   class[i] <- 1   }
+ else                       {   class[i] <- 0   }
+ }
```

# Evaluation of binary classifiers
# Basic performance measures

**Example** (see lab # 5) – cntnd

```
# evaluation -- confusion matrix
> r <- table(class, test$hon)
class  0  1
    0 12  3
    1  2  3

# evaluation -- accuracy
> sum(diag(r))/sum(r)
[1] 0.75

# evaluation -- precision
> r[[4]]/(r[[4]] + r[[3]])
[1] 0.5

# evaluation -- recall
> r[[4]]/(r[[4]] + r[[2]])
[1] 0.6
```