Lecture 8

Classification III

Ioanna Miliou, PhD

Senior Lecturer, Stockholm University

Stockholms universitet

What is Bayesian Classification?

- Bayesian classifiers are statistical classifiers

- For each new sample, they provide a probability that the sample belongs to a class (for all classes)

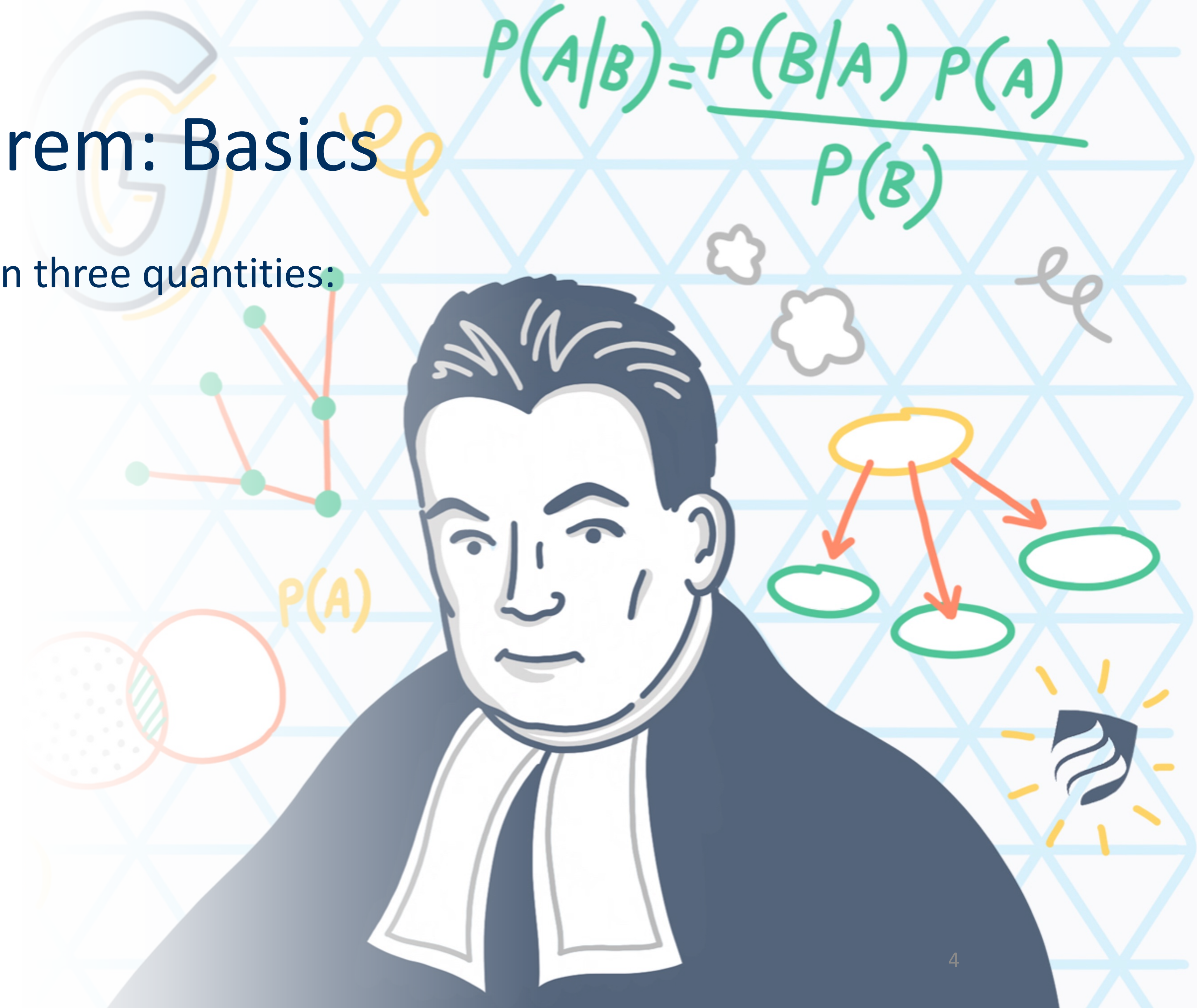
Training Dataset

play tennis?

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

Bayes’ Theorem: Basics

- We are interested in three quantities:
  - Prior
  - Evidence
  - Likelihood



Bayes’ Theorem: Basics

- We are interested in three quantities:
  - Prior

Bayes' Theorem: Basics

- We are interested in three quantities:
  - Prior

P(H) (*prior probability*):

- the initial probability of an example belonging to a class
- e.g., the probability that anyone will play tennis, regardless of weather outlook, temperature, humidity, wind

Bayes’ Theorem: Basics

- We are interested in three quantities:
  - Prior
  - Evidence
  - Likelihood

Bayes' Theorem: Basics

- We are interested in three quantities:
    - Evidence

Bayes’ Theorem: Basics

- We are interested in three quantities:
    - Evidence

P(X) (evidence):

- probability that data sample X (a particular configuration of our data) is observed
- e.g., what is the chance of rain, cool temperature, normal humidity, no wind, and playing tennis?

Bayes' Theorem: Basics

- We are interested in three quantities:
  - Prior
  - Evidence
  - Likelihood

Bayes' Theorem: Basics

- We are interested in three quantities:
  - Likelihood

Bayes' Theorem: Basics

- We are interested in three quantities:
  - Likelihood

P(X|H) (likelihood):

- the probability of observing sample X, given that hypothesis H holds
- e.g., given that X will play tennis, what is the probability that it is sunny, with low temperature, no wind, and normal humidity?

NBS: Naive Bayes Classifier

P(H|**X**) (*posterior probability*):

- the probability that hypothesis H holds, given the observed data sample **X**
- e.g., the probability that **X** will play tennis, given rain, cool temperature, normal humidity, and no wind

NBS: Naive Bayes Classifier

- Given
  - A data sample ("*evidence*") **X**
  - A *hypothesis* H that **X** belongs to class C
- NBS will determine P(H|**X**)
  - the probability that hypothesis H holds, given the observed data sample **X**

Bayes’ Theorem

- Given training data **X** and a hypothesis H, *the posterior probability* P(H|**X**) follows the Bayes theorem

$$\boxed{P(H|\mathbf{X})} = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})}$$

- Informally, this can be written as

posterior = likelihood x prior / evidence

Bayes' Theorem

- Given training data **X** and a hypothesis H, *the posterior probability* P(H|**X**) follows the Bayes theorem

$$\boxed{P(H|\mathbf{X})} = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})}$$

- Predicts that **X** belongs to class C_i
  - if and only if the probability P(C_i|**X**) is the highest among all the P(C_k|X) for all the *m* classes

- **Practical difficulty:** requires initial knowledge of many probabilities, significant computational cost

Towards Naive Bayesian Classifiers

- **D**: a training set of examples and their class labels
- **X** = (A_1, A_2, …, A_n): each example is represented by an n-dimensional attribute vector
- Suppose there are *m* classes C_1, C_2, …, C_m.
- **Classification**: derive the maximum posterior, i.e., the maximum $P(C_i|\mathbf{X})$
- Report the class with the maximum posterior

Towards Naive Bayesian Classifiers

- **Classification**: derive the maximum posterior, i.e., the maximum P(C_i|**X**)
- Bayes' theorem:

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- Since P(**X**) is constant for all classes, we only need to maximize the following:

$$P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)$$

Derivation of Naive Bayesian Classifier

- A simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- Each x_k is a potential value of attribute A_k

- This greatly reduces the computation cost: only counts the class distribution

Derivation of Naive Bayesian Classifier

- If A_k is categorical
  - $P(x_k|C_i)$ is the # of tuples of class C_i having value $A_k = x_k$ divided by $|C_{i, D}|$ (# of tuples of class C_i in D)

- If A_k is continuous-valued
  - $P(x_k|C_i)$ is computed based on a Gaussian distribution with mean μ and standard deviation σ

Naive Bayesian Classifier Example

play tennis?

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

Computing the priors P(H)

Outlook	Temperature	Humidity	Windy	Class
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
overcast	cool	normal	true	P
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P

9

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
rain	cool	normal	true	N
sunny	mild	high	false	N
rain	mild	high	true	N

5

$P(C_i{=}{=}P) = 9/14$

$P(C_i{=}{=}N) = 5/14$

Computing the likelihoods P(X|H)

- Given the training set, we compute $P(x_k|C_i)$ for each attribute

Outlook	P	N		Humidity	P	N
sunny	2/9	3/5		high	3/9	4/5
overcast	4/9	0		normal	6/9	1/5
rain	3/9	2/5				
Tempreature				Windy		
hot	2/9	2/5		true	3/9	3/5
mild	4/9	2/5		false	6/9	2/5
cool	3/9	1/5				

$P(x_k|C_i)$ is the # of tuples of class C_i having value $A_k = x_k$ divided by $|C_{i,\mathbf{D}}|$

(# of tuples of class C_i in **D**)

Example

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- To classify a new sample **X**:

  < **outlook** = sunny, **temperature** = cool, **humidity** = high, **windy** = false >

$$P(C_i|\mathbf{X}) = P(C_i)P(\mathbf{X}|C_i)$$

- Prob(C_i = P|**X**) = Prob(P) * Prob(sunny|P)*Prob(cool|P)* Prob(high|P)*Prob(false|P) =

  9/14*2/9*3/9*3/9*6/9 = 0.01

- Prob(C_i = N|**X**) = Prob(N) * Prob(sunny|N)*Prob(cool|N)* Prob(high|N)*Prob(false|N) =

  5/14*3/5*1/5*4/5*2/5 = 0.013

- Therefore, **X** takes class label **N**

Avoiding the 0-Probability Problem

- Naïve Bayesian prediction requires each conditional probability to be non-zero
- Otherwise, the predicted probability for a class (e.g., C_i) will be zero:
  - Example: assume a dataset with
    - ✓ # of examples = 1000
    - ✓ income = low (0 examples)
    - ✓ income = medium (990 examples)
    - ✓ income = high (10 examples)
  - ○ Any test example for which attribute *income* is “low” will be given a probability of 0

➡ $P(x_1 = low \mid C_i) = 0$

$$P(\mathbf{X} \mid C_i) = \mathbf{0} \times P(sunny \mid C_i) \times P(false \mid C_i)$$

Avoiding the 0-Probability Problem

- **Use Laplacian correction (or Laplacian estimator)**
  - Add 1 to each case

    Prob(income = low) = 1/1003

    Prob(income = medium) = 991/1003

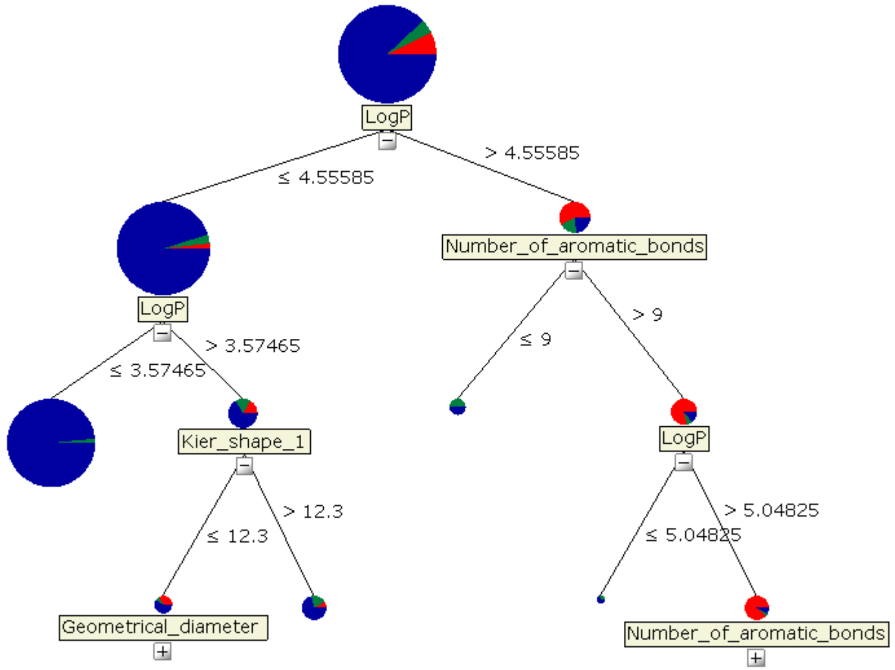
    Prob(income = high) = 11/1003

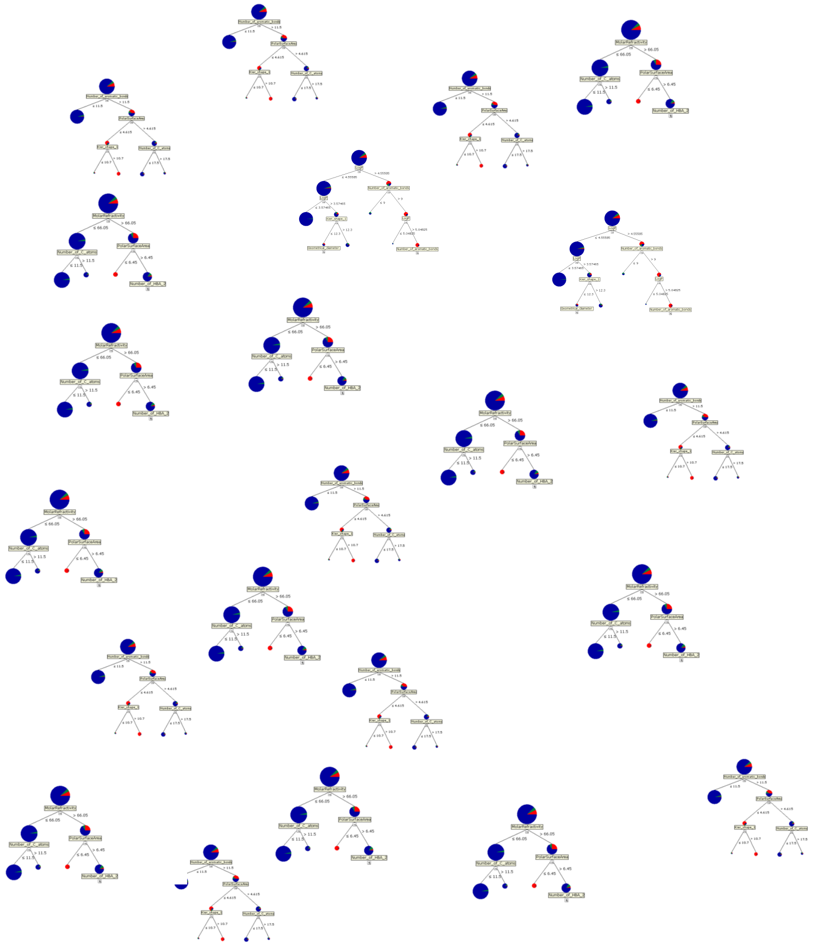
  - The "corrected" probability estimates are close to their "uncorrected" counterparts

NBC: Comments

- Advantages
  - Easy to implement
  - Good results obtained in most of the cases
- Disadvantages
  - Assumption: class conditional independence, therefore loss of accuracy
  - Practically, dependencies exist among variables
- How to deal with these dependencies
  - Bayesian Belief Networks

Single trees vs. forests



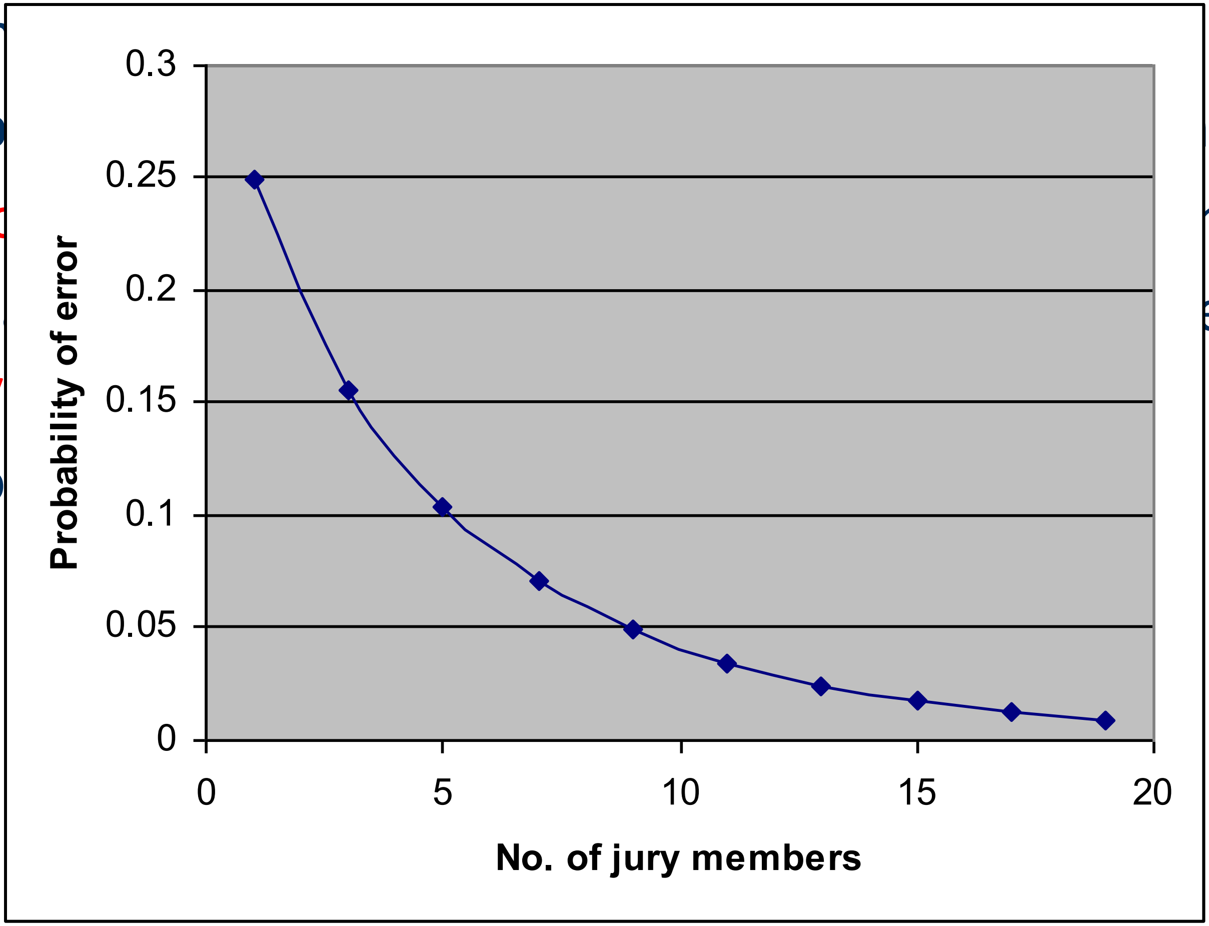


Condorcet’s jury theorem

- Given enough evidence (e.g., training data)
- if each member of a jury is more likely to be right than wrong
- then the majority of the jury, too, is more likely to be right than wrong
- and the probability that a majority of the jury supports the right outcome is an exponentially increasing function of the size of the jury
- converging to 1 as the size of the jury tends to infinity

Condorcet’s jury theorem

- Given enough
- if each memb   g
- then the majo   han wrong
- and the prob   e right outcome is an exponentially
- converging to



Bagging

A *bootstrap replicate* **D'** of a set of examples **D** is created by randomly selecting **n** = |**D**| examples from **D** with replacement.

The probability of an example in **D** appearing in **D'** is:

$$1-(1-\frac{1}{n})^{n}=1-\frac{1}{e}\approx 0.632$$

The examples that are not chosen in a replicate are called ”out-of-bag” examples.

Bootstrap replicate

Ex.	Other	Bar	Fri/Sat	Hungry	Guests	Wait
e2	yes	no	no	yes	full	no
e2	yes	no	no	yes	full	no
e3	no	yes	no	no	some	yes
e4	yes	no	yes	yes	full	yes
e4	yes	no	yes	yes	full	yes
e6	no	yes	no	yes	some	yes

Bagging

Input: examples ***D***, base learner ***BL***, iterations ***n***

Output: combined model ***M***

```
    $i$:= 0
    Repeat
        $i$ := $i$+1
        Generate bootstrap replicate $D_i'$ of $D$
        $M_i$ := $BL(D_i')$
    Until $i = n$
    $M$ := Average*({$M_1$, …, $M_n$})
```

* For **classification**: majority vote or the mean class probability distribution
For **regression**: mean predicted value

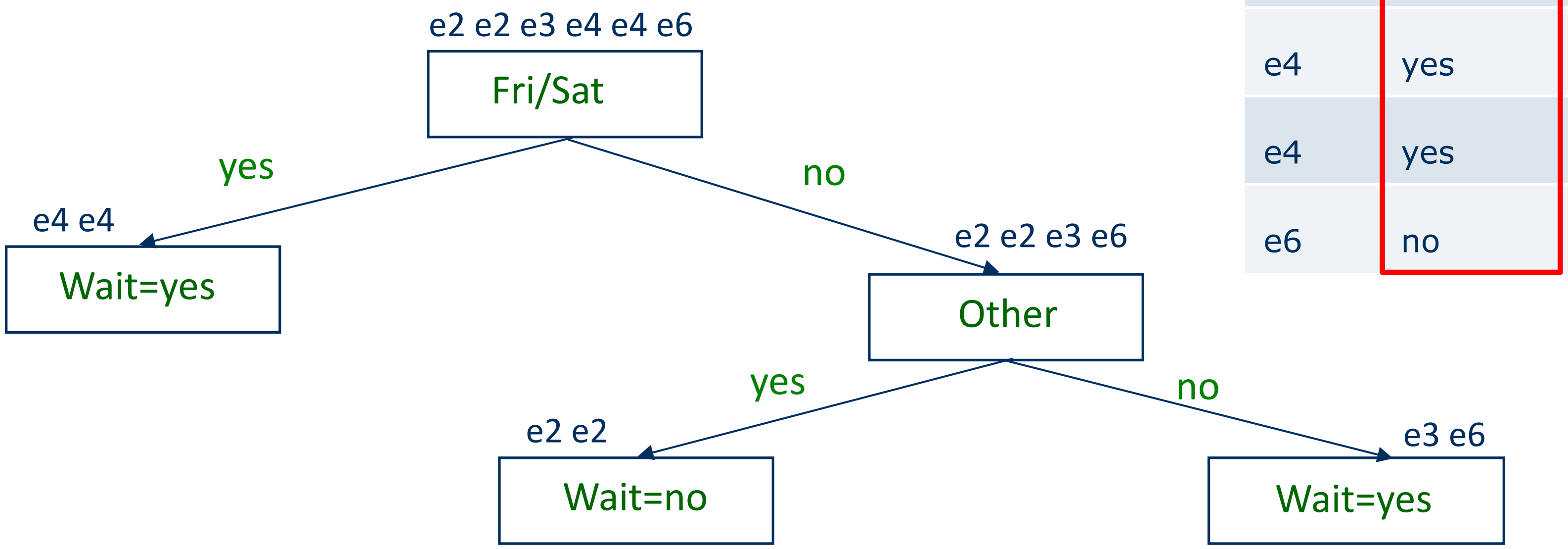
Random forests

Random forests (Breiman 2001) are generated by combining two techniques:

- bagging (Breiman 1996)
- the random subspace method (Ho 1998)

The random subspace method

Ex.	Other	Bar	Fri/Sat	Hungry	Guests	Wait
e2	yes	no	no	yes	full	no
e2	yes	no	no	yes	full	no
e3	no	yes	no	no	some	yes
e4	yes	no	yes	yes	full	yes
e4	yes	no	yes	yes	full	yes
e6	no	yes	no	yes	some	yes



Random forests

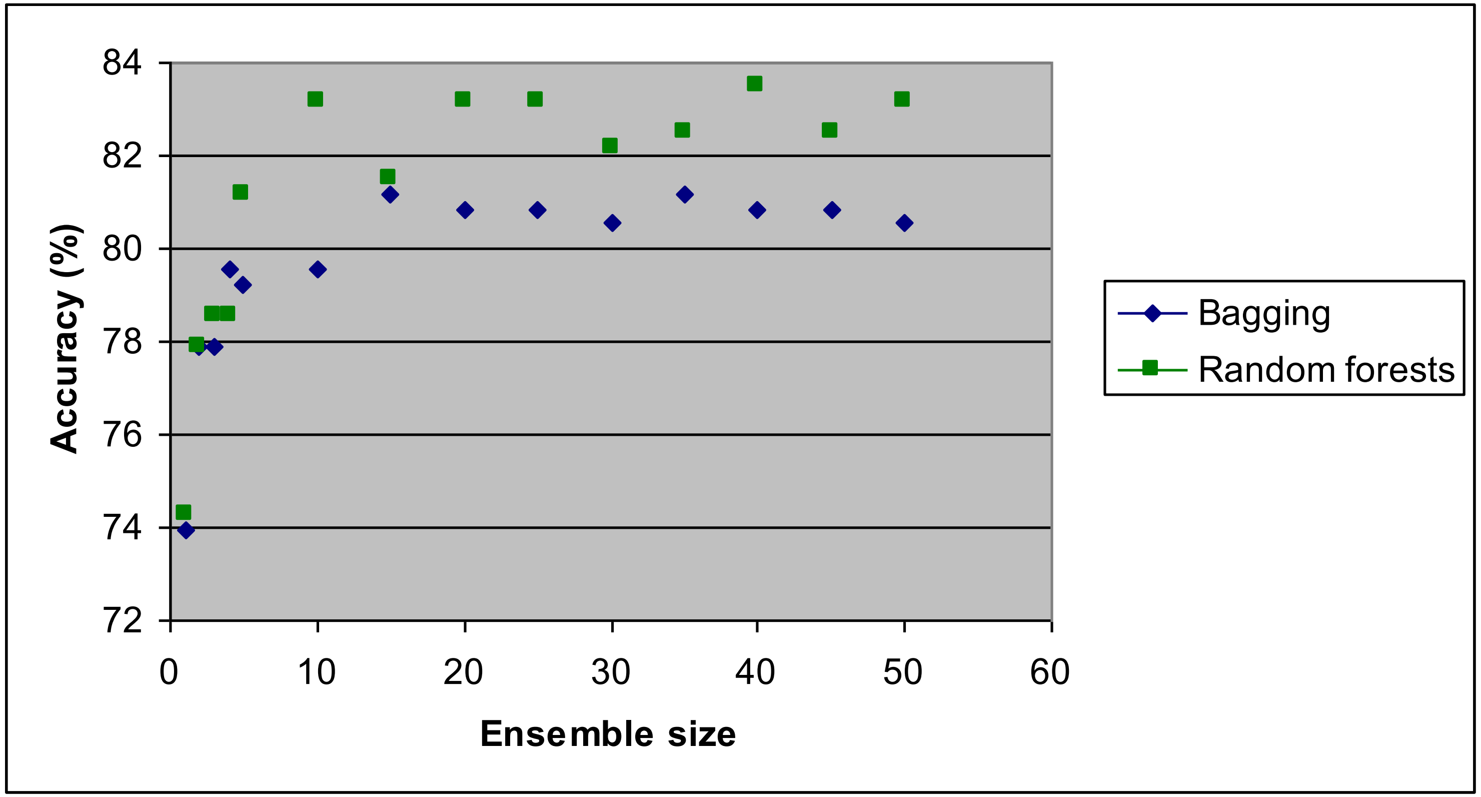
Random forests =

A set of classification trees generated with **bagging** and where only a **randomly chosen subset** of all available features (F) are considered at each node when generating the trees

A common choice is to let the number of considered features be equal to the following:

$$\lfloor \log_2 |F| + 1 \rfloor$$

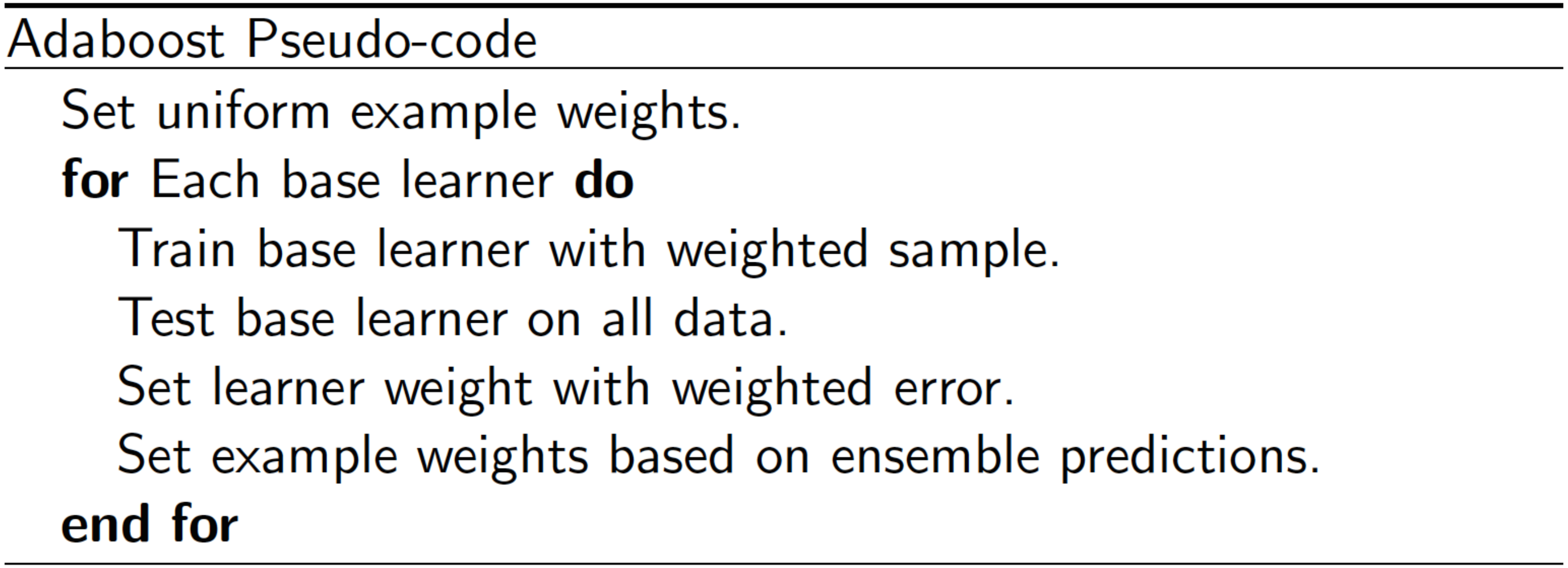
Bagged trees vs. random forests



Heart-disease dataset from the UCI repository
Stratified 10-fold cross-validation
Base learner: **decision trees**

Boosting (AdaBoost) [Schapire & Singer, 1998]

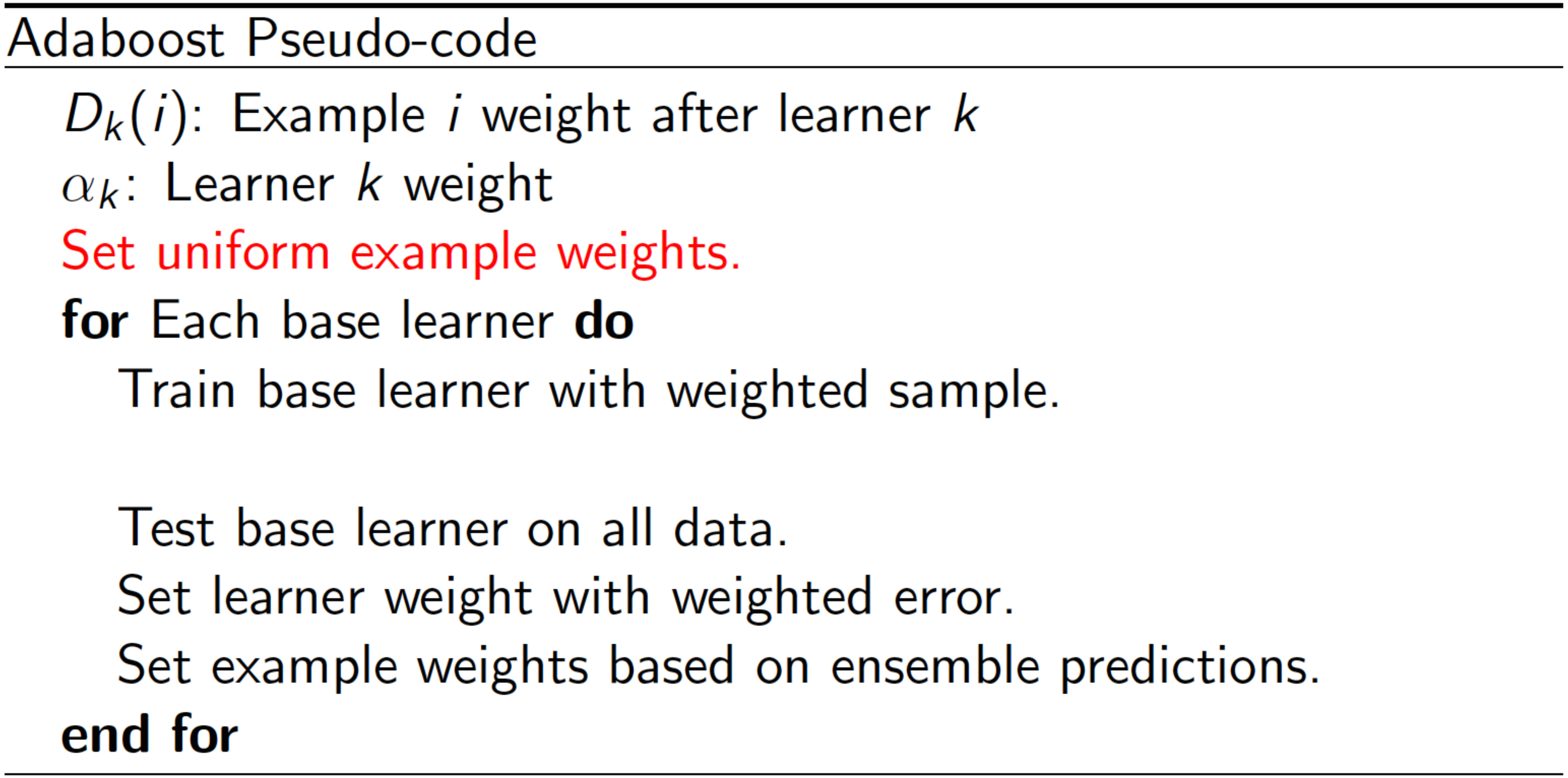
- Use many “weak” (simple) classifiers to come up with a strong one



Adaboost Pseudo-code

```
Set uniform example weights.
for Each base learner do
    Train base learner with weighted sample.
    Test base learner on all data.
    Set learner weight with weighted error.
    Set example weights based on ensemble predictions.
end for
```

Boosting (AdaBoost)

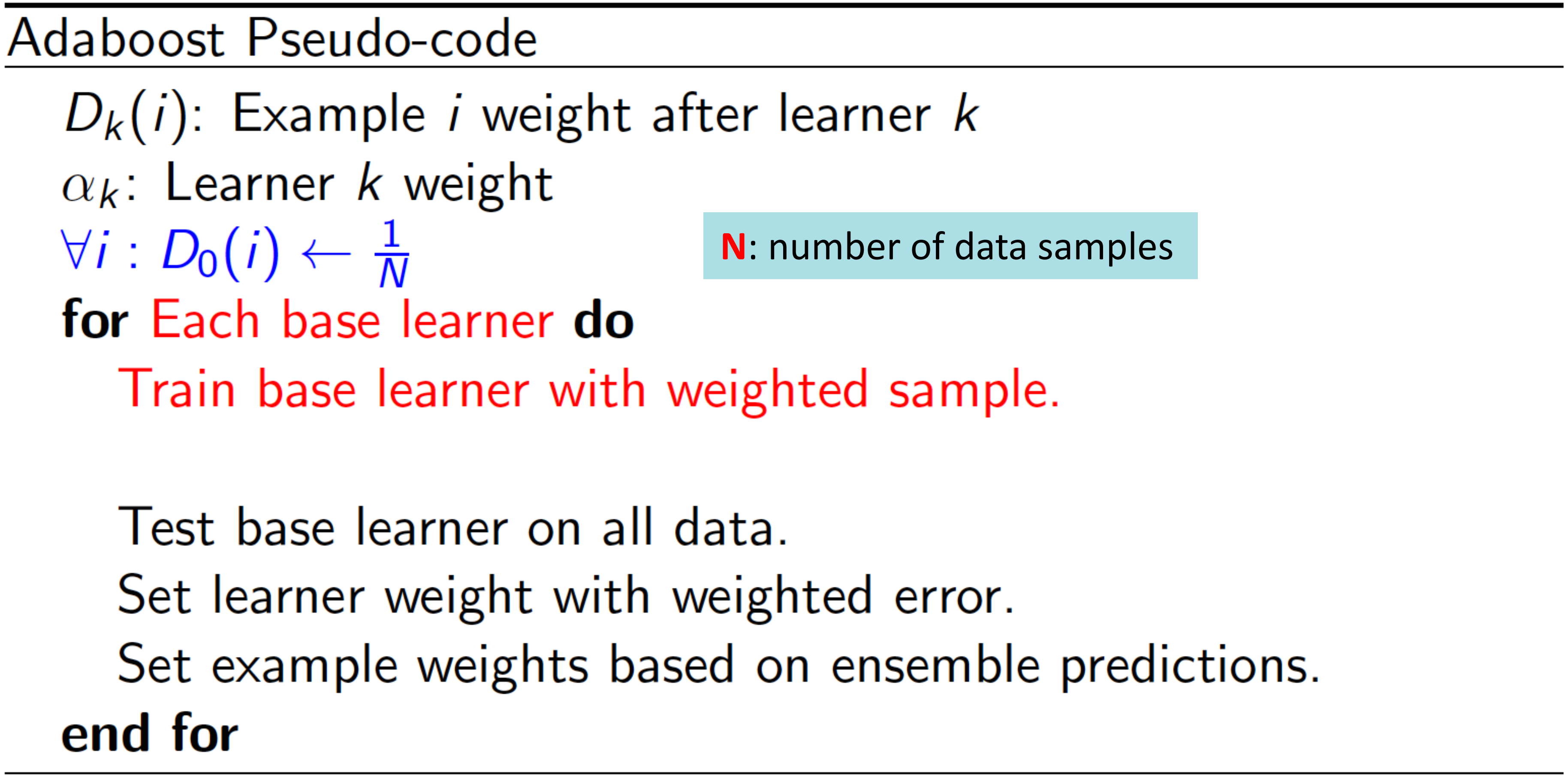


Adaboost Pseudo-code

```
$D_k(i)$: Example $i$ weight after learner $k$
$\alpha_k$: Learner $k$ weight
Set uniform example weights.
for Each base learner do
    Train base learner with weighted sample.

    Test base learner on all data.
    Set learner weight with weighted error.
    Set example weights based on ensemble predictions.
end for
```

Boosting (AdaBoost)



Adaboost Pseudo-code

```
$D_k(i)$: Example $i$ weight after learner $k$
$\alpha_k$: Learner $k$ weight
$\forall i : D_0(i) \leftarrow \frac{1}{N}$
for Each base learner do
    Train base learner with weighted sample.

    Test base learner on all data.
    Set learner weight with weighted error.
    Set example weights based on ensemble predictions.
end for
```

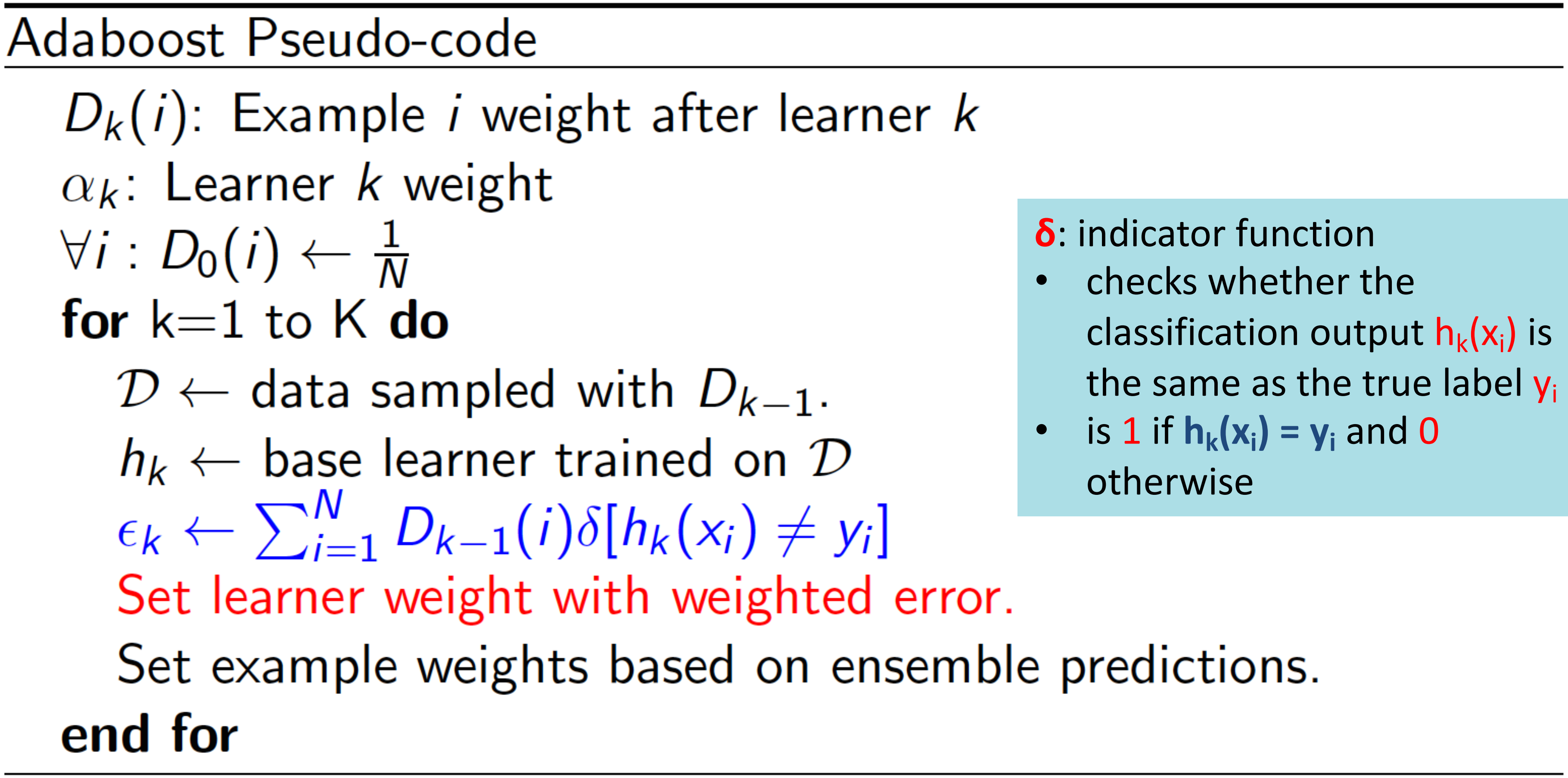
N: number of data samples

Boosting (AdaBoost)

```
Adaboost Pseudo-code
$D_k(i)$: Example $i$ weight after learner $k$
$\alpha_k$: Learner $k$ weight
$\forall i : D_0(i) \leftarrow \frac{1}{N}$
for k=1 to K do
    $\mathcal{D} \leftarrow$ data sampled with $D_{k-1}$.
    $h_k \leftarrow$ base learner trained on $\mathcal{D}$
    Test base learner on all data.
    Set learner weight with weighted error.
    Set example weights based on ensemble predictions.
end for
```

Draw random samples with replacement from original data with the probabilities equal to the sample weights and fit the model

Boosting (AdaBoost)



Adaboost Pseudo-code

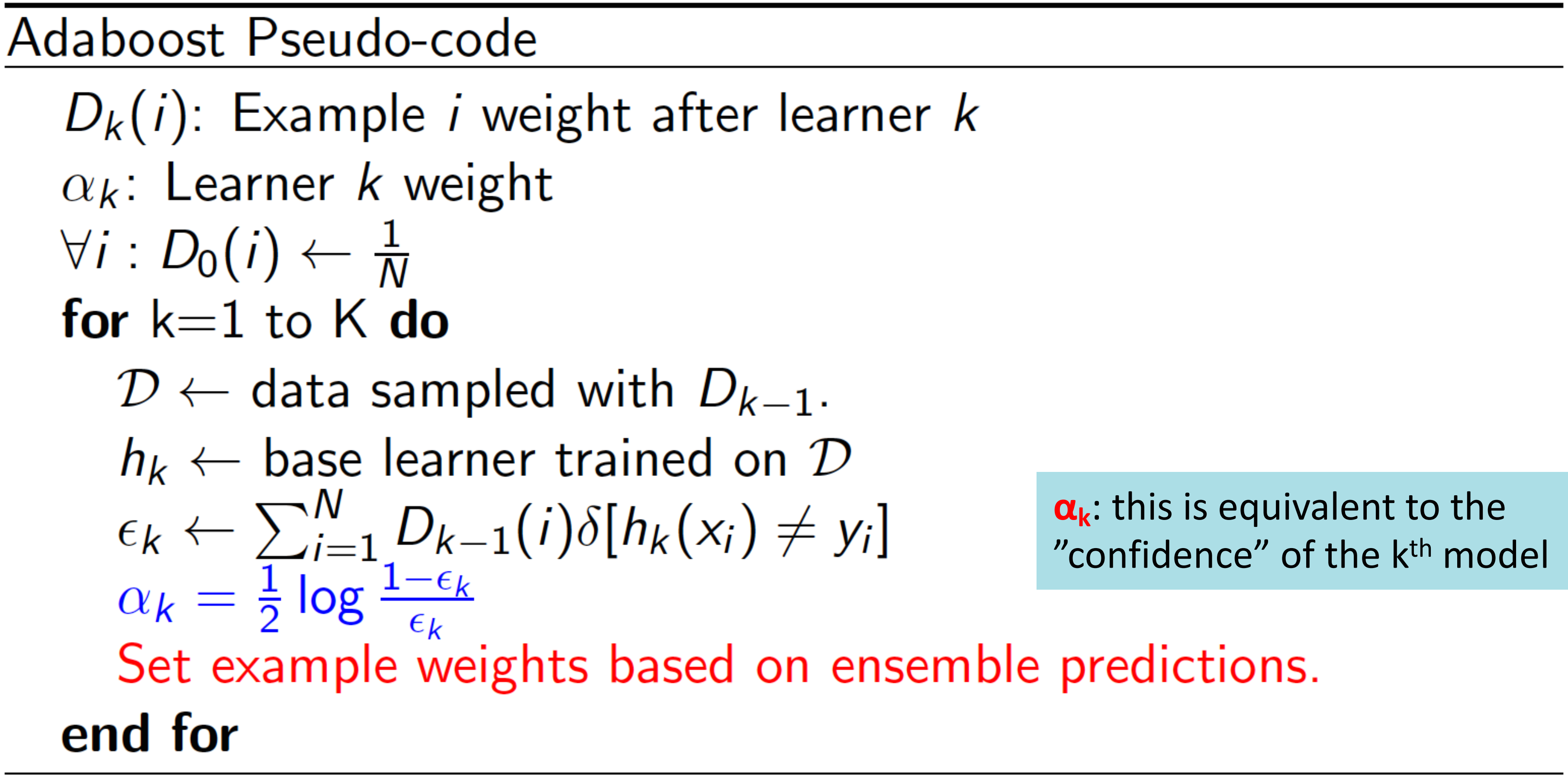
```
$D_k(i)$: Example $i$ weight after learner $k$
$\alpha_k$: Learner $k$ weight
$\forall i : D_0(i) \leftarrow \frac{1}{N}$
for k=1 to K do
    $\mathcal{D} \leftarrow$ data sampled with $D_{k-1}$.
    $h_k \leftarrow$ base learner trained on $\mathcal{D}$
    $\epsilon_k \leftarrow \sum_{i=1}^{N} D_{k-1}(i)\delta[h_k(x_i) \neq y_i]$
    Set learner weight with weighted error.
    Set example weights based on ensemble predictions.
end for
```

δ: indicator function

- checks whether the classification output $h_k(x_i)$ is the same as the true label y_i
- is 1 if $\mathbf{h_k(x_i) = y_i}$ and 0 otherwise

Total error is the **sum of weights** of the **misclassified** samples

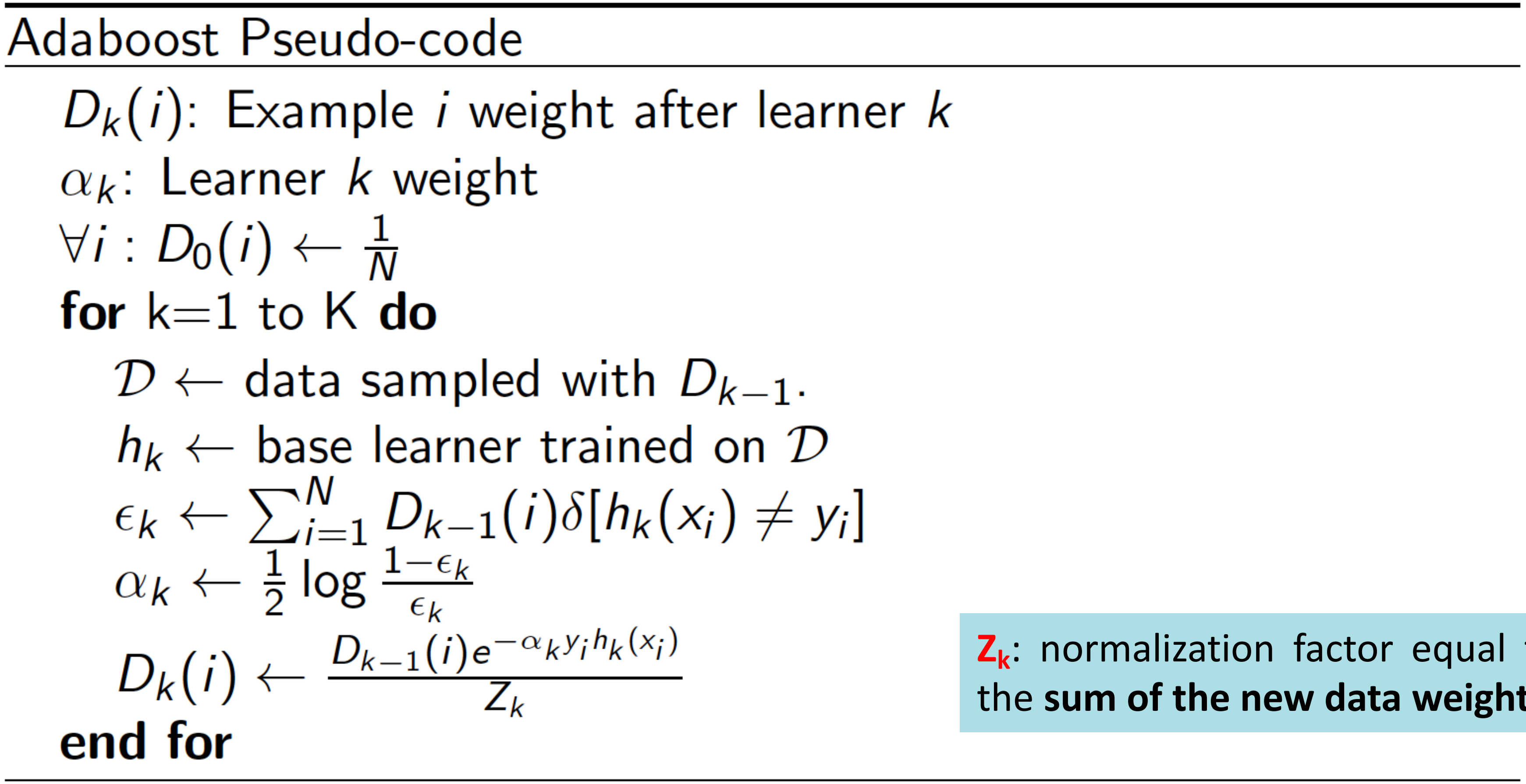
Boosting (AdaBoost)



```
Adaboost Pseudo-code
  $D_k(i)$: Example $i$ weight after learner $k$
  $\alpha_k$: Learner $k$ weight
  $\forall i : D_0(i) \leftarrow \frac{1}{N}$
  for k=1 to K do
    $\mathcal{D} \leftarrow$ data sampled with $D_{k-1}$.
    $h_k \leftarrow$ base learner trained on $\mathcal{D}$
    $\epsilon_k \leftarrow \sum_{i=1}^{N} D_{k-1}(i)\delta[h_k(x_i) \neq y_i]$
    $\alpha_k = \frac{1}{2} \log \frac{1-\epsilon_k}{\epsilon_k}$
    Set example weights based on ensemble predictions.
  end for
```

α_k: this is equivalent to the ”confidence” of the kth model

Boosting (AdaBoost)



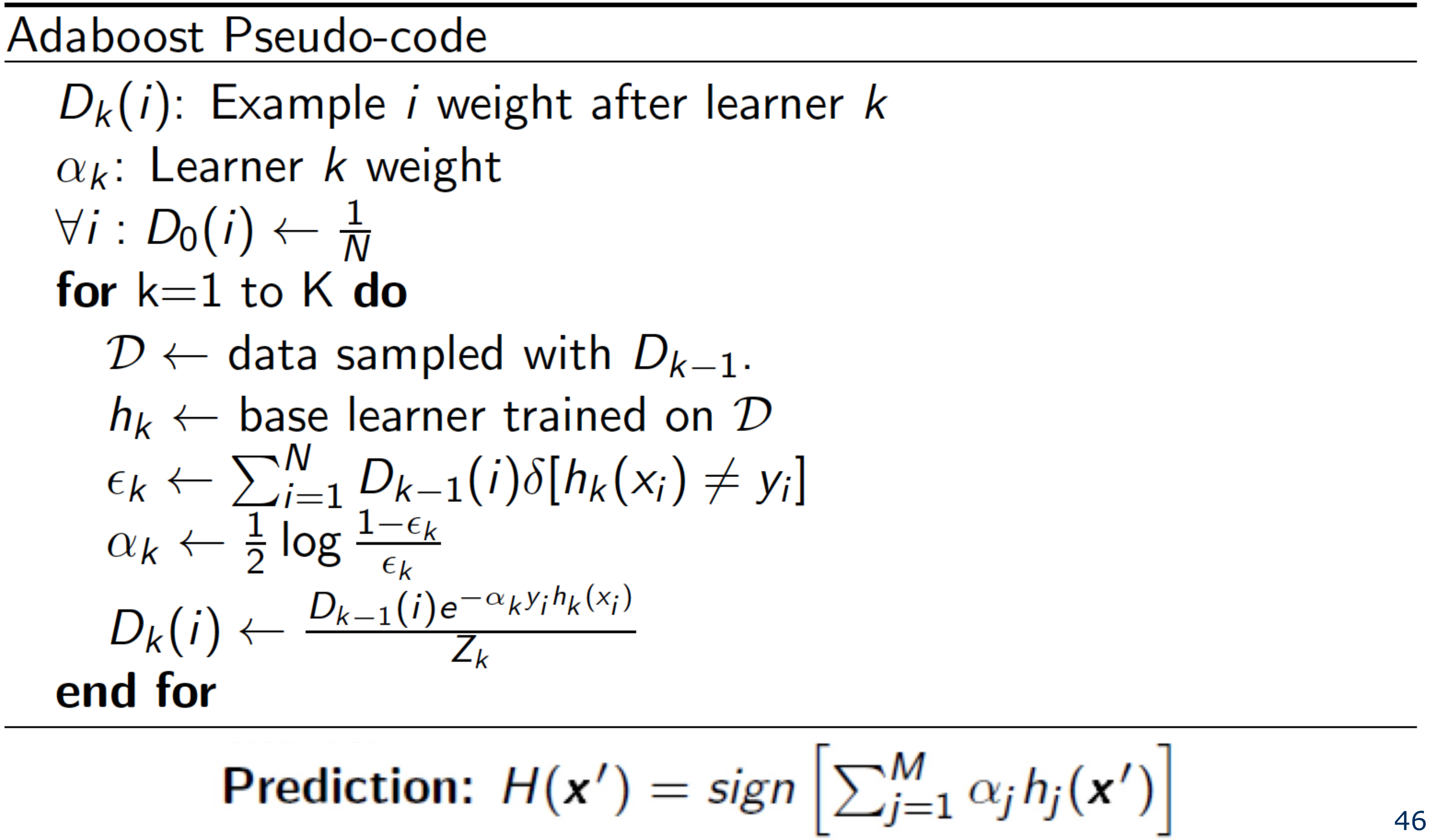
Adaboost Pseudo-code

```
$D_k(i)$: Example $i$ weight after learner $k$
$\alpha_k$: Learner $k$ weight
$\forall i : D_0(i) \leftarrow \frac{1}{N}$
for k=1 to K do
    $\mathcal{D} \leftarrow$ data sampled with $D_{k-1}$.
    $h_k \leftarrow$ base learner trained on $\mathcal{D}$
    $\epsilon_k \leftarrow \sum_{i=1}^{N} D_{k-1}(i)\delta[h_k(x_i) \neq y_i]$
    $\alpha_k \leftarrow \frac{1}{2} \log \frac{1-\epsilon_k}{\epsilon_k}$
    $D_k(i) \leftarrow \frac{D_{k-1}(i)e^{-\alpha_k y_i h_k(x_i)}}{Z_k}$
end for
```

$\mathbf{Z_k}$: normalization factor equal to the **sum of the new data weights**

Weights of misclassified samples are increased, while weights of correctly classified samples are decreased

Boosting (AdaBoost)

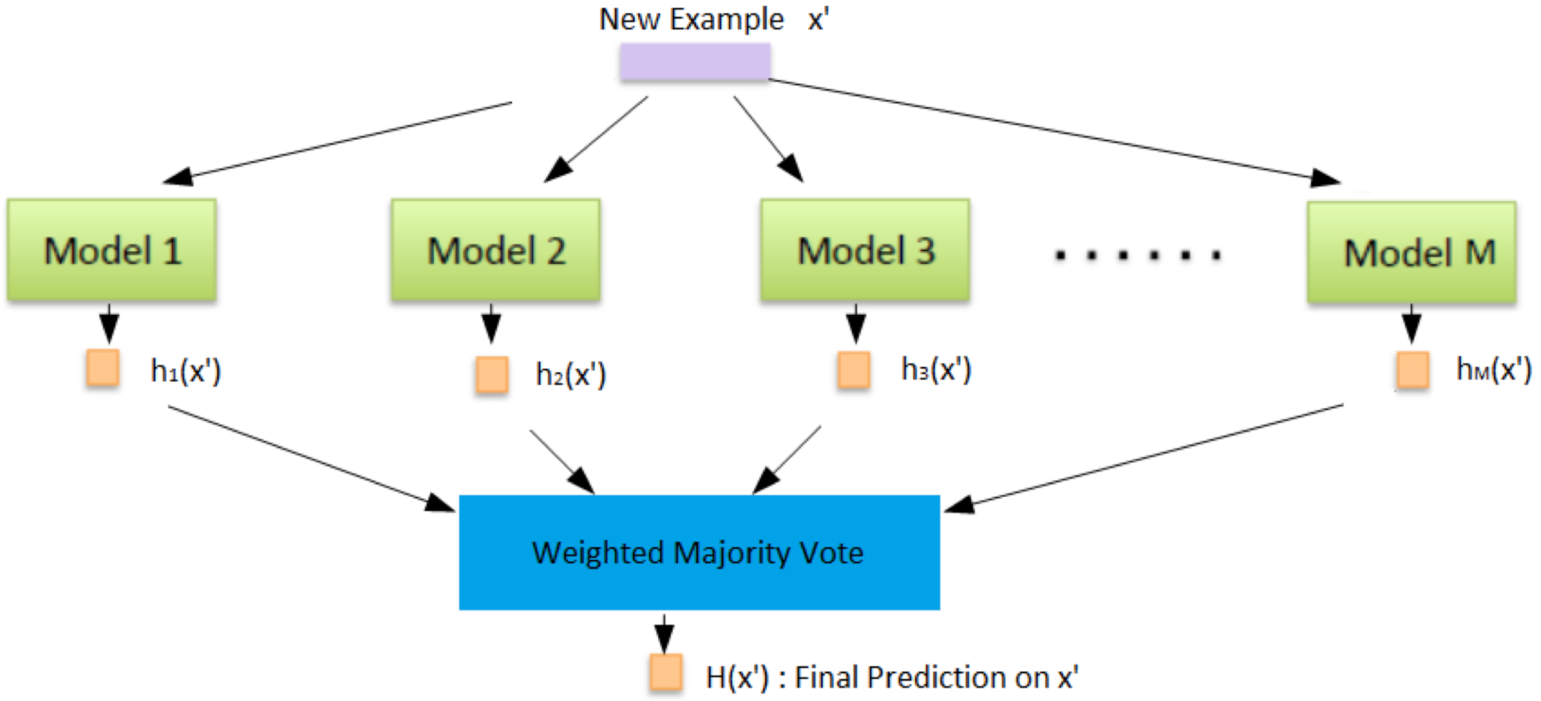


```
Adaboost Pseudo-code
$D_k(i)$: Example $i$ weight after learner $k$
$\alpha_k$: Learner $k$ weight
$\forall i : D_0(i) \leftarrow \frac{1}{N}$
for k=1 to K do
    $\mathcal{D} \leftarrow$ data sampled with $D_{k-1}$.
    $h_k \leftarrow$ base learner trained on $\mathcal{D}$
    $\epsilon_k \leftarrow \sum_{i=1}^{N} D_{k-1}(i)\delta[h_k(x_i) \neq y_i]$
    $\alpha_k \leftarrow \frac{1}{2} \log \frac{1-\epsilon_k}{\epsilon_k}$
    $D_k(i) \leftarrow \frac{D_{k-1}(i)e^{-\alpha_k y_i h_k(x_i)}}{Z_k}$
end for
```

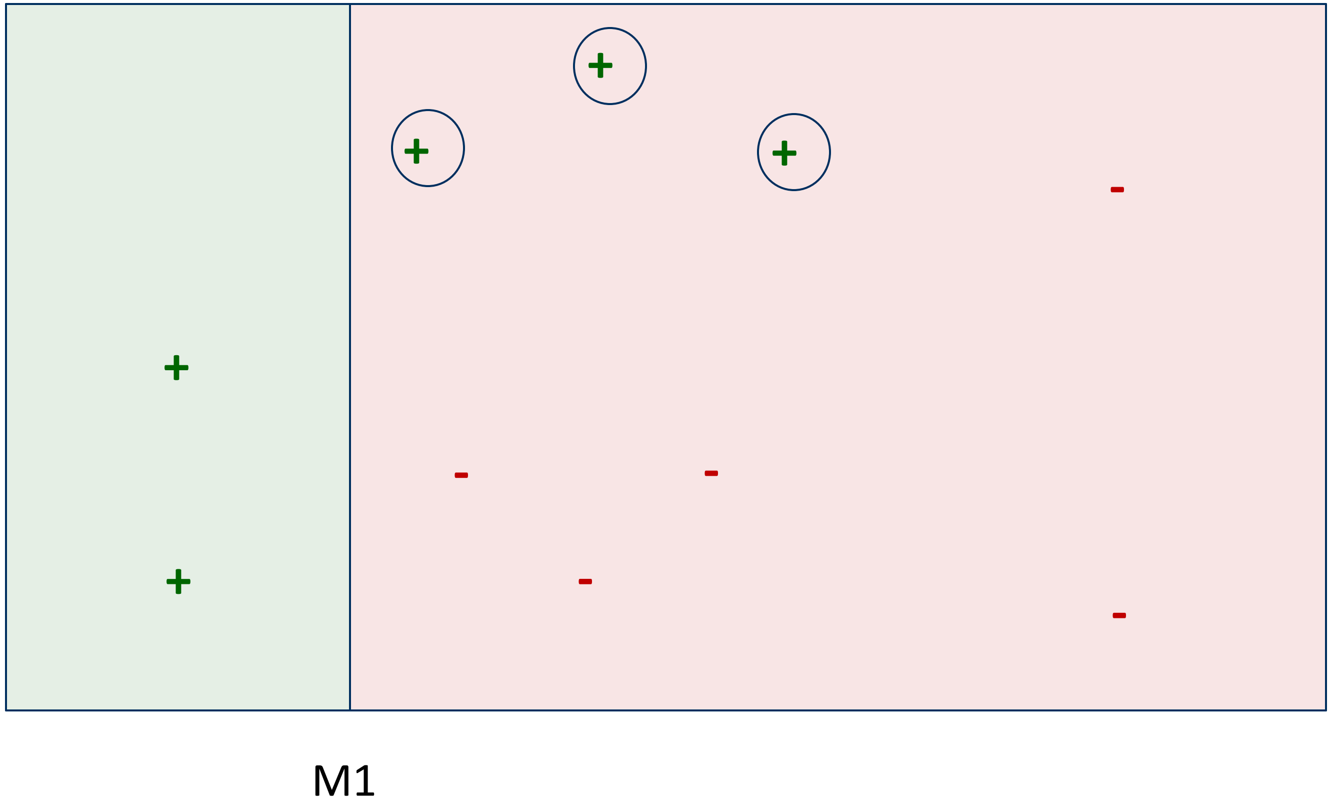
Prediction: $H(\mathbf{x}') = sign\left[\sum_{j=1}^{M} \alpha_j h_j(\mathbf{x}')\right]$

Boosting (AdaBoost)

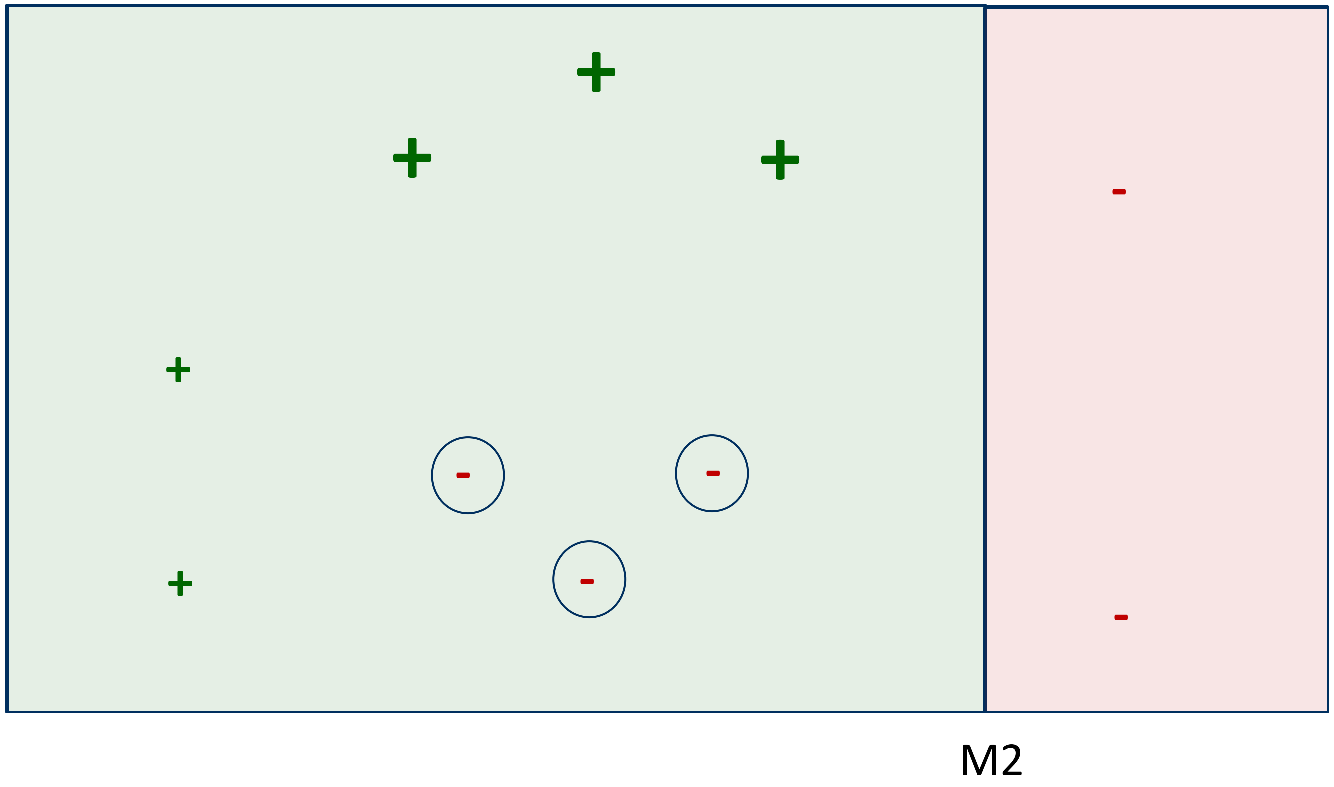
$$\textbf{Prediction: } H(\boldsymbol{x}') = sign\left[\sum_{j=1}^{M} \alpha_j h_j(\boldsymbol{x}')\right]$$



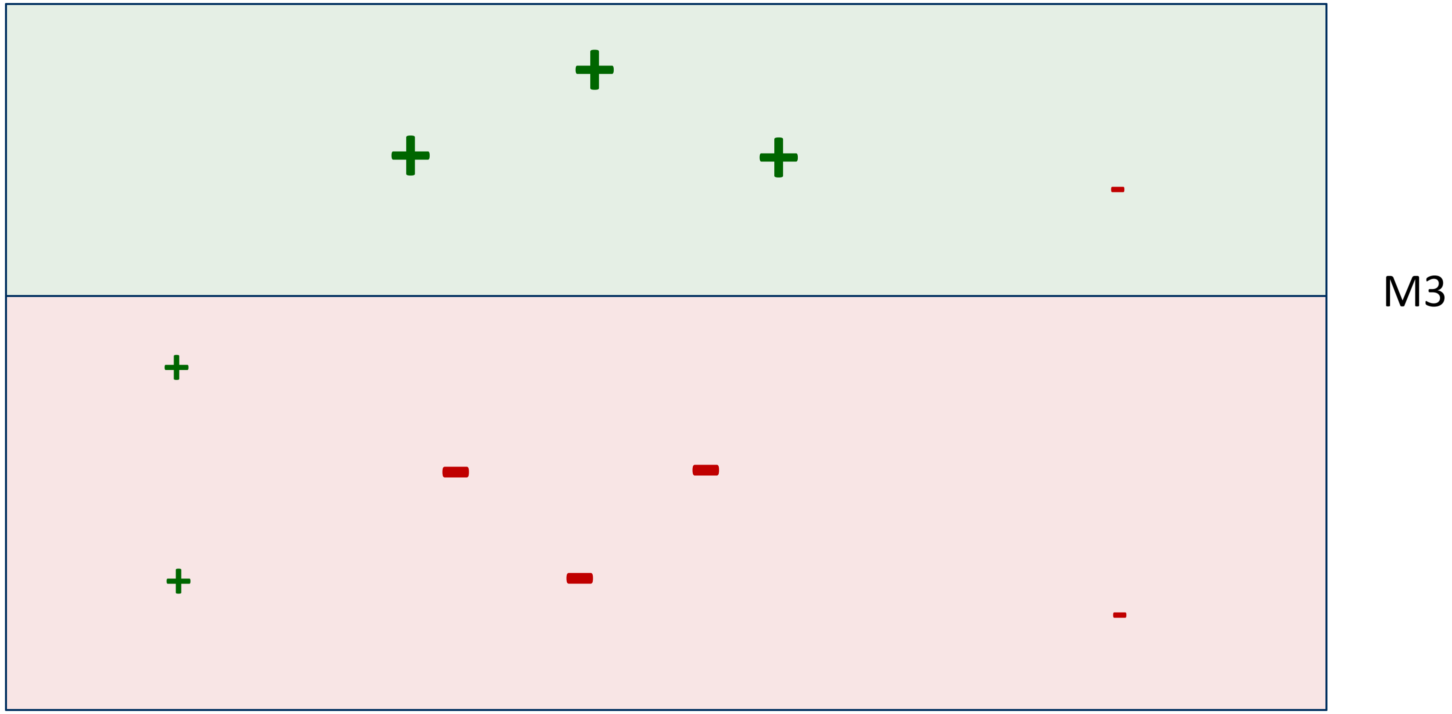
Boosting (example)



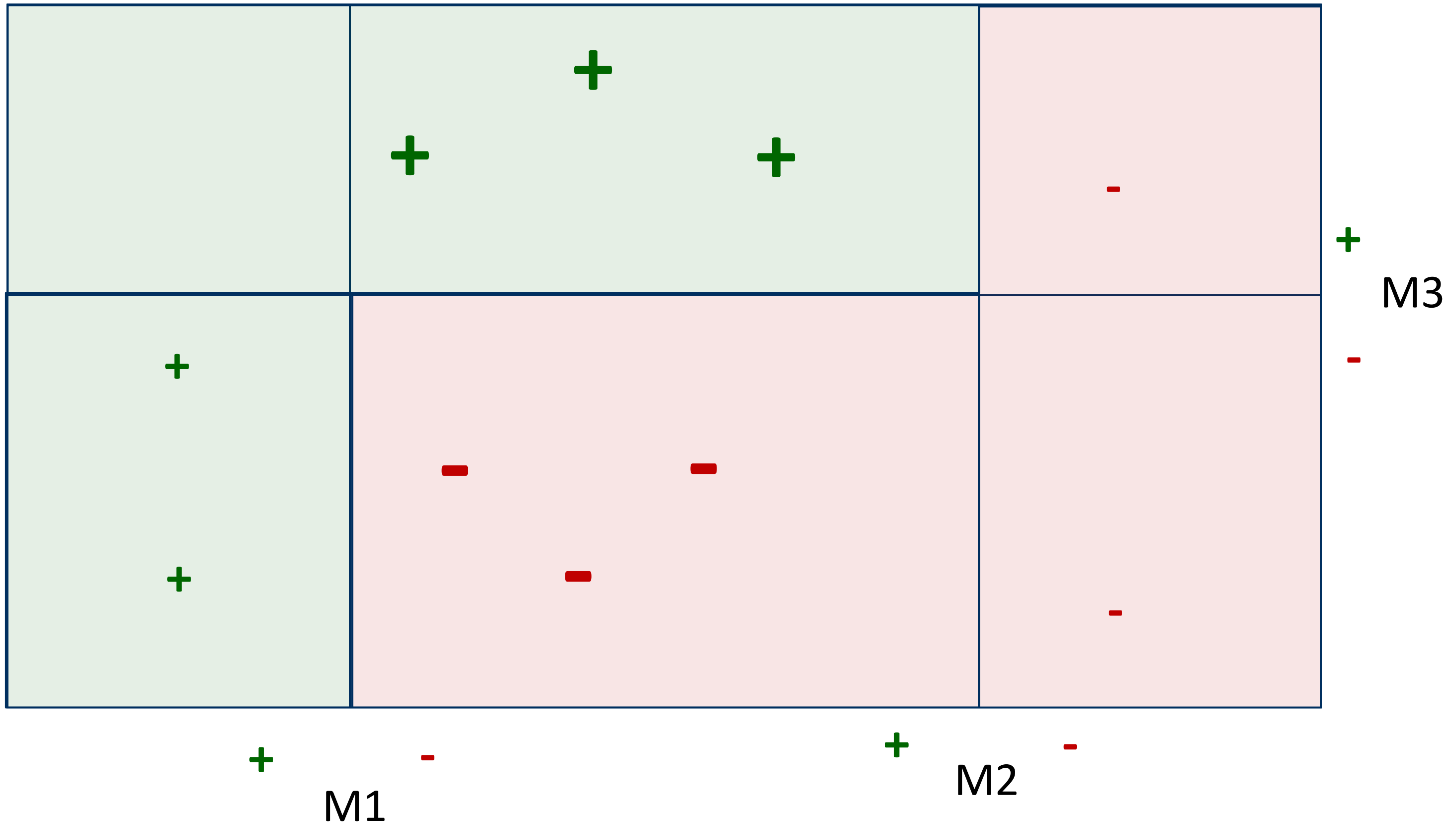
Boosting (example)



Boosting (example)

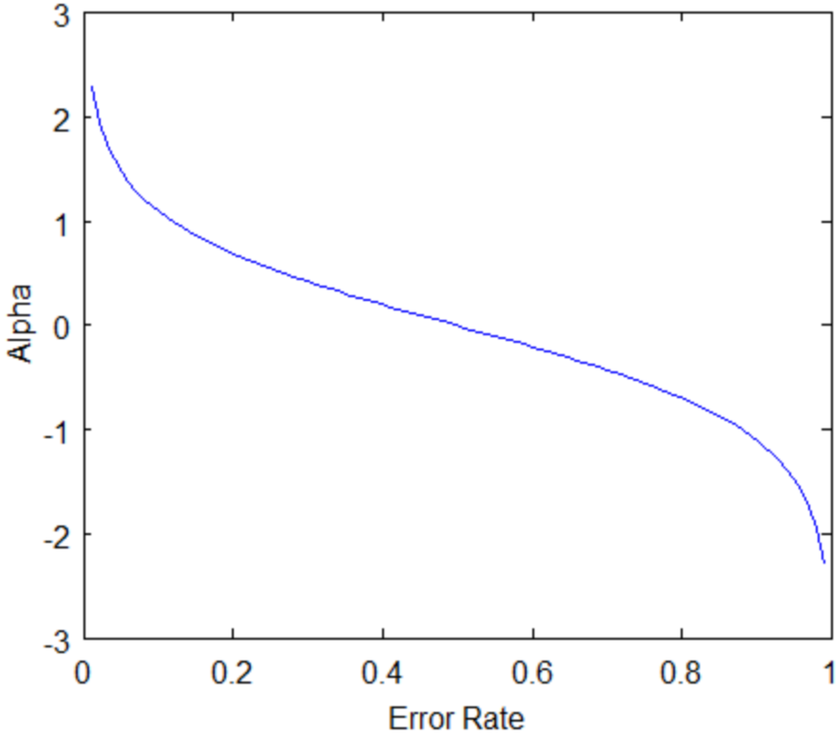


Boosting (example)



Boosting (AdaBoost)

- α_k grows exponentially as the error approaches 0
- α_k = 0 if the error rate is 0.5:
  - a classifier with 50% accuracy is no better than random guessing, so we ignore it
- α_k grows exponentially negative as the error approaches 1:
  - negative weight to classifiers with worse than 50% accuracy
  - equivalent to flipping their predictions



Stockholms universitet

Base learners

- Simple linear classifiers: e.g., lines
- Simple decision trees:
  - Decision stumps: a decision tree with only one decision node (the root)
- More complex classifiers…

Using Boosting

- For multi-class problems, it may be difficult to reduce the error below 50% with weak base learners (e.g., decision stumps)
  - More **powerful** base learners may be employed
  - The problem may be transformed into **multiple binary classification** problems
  - Specific versions of AdaBoost have been developed for **multi-class** problems

- Boosting can be sensitive to noise (i.e., erroneously labeled training examples)
  - More **robust** versions have been developed

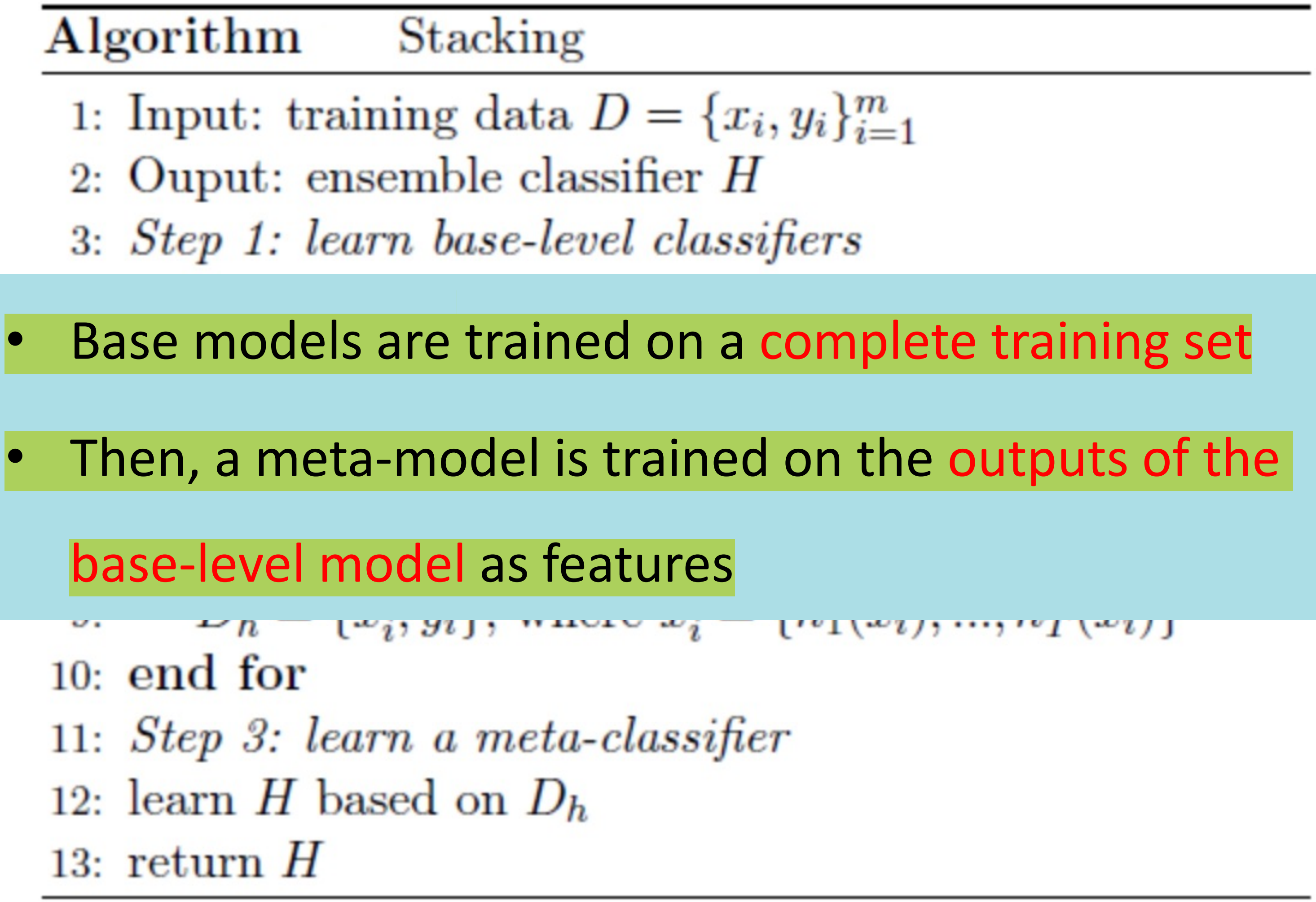
Using Boosting

- When to stop?
  - Most improvement for first 5 to 10 classifiers
  - Significant gains up to 25 classifiers
  - **Generalization error can continue to improve even after the training error is zero!**
- When can boosting have problems?
  - not enough data
  - really weak learner
  - really strong learner
  - very noisy data
- Although this can be mitigated: e.g., by detecting noise or outliers, how?
  - ✓ look for very high weights!

Boosting vs Bagging?

BAGGING	ADABOOST
Resample dataset	Resample or reweight dataset
Builds base models in parallel	Builds base models sequentially
Reduces variance (doesn’t work well with e.g. decision stumps)	Also reduces bias (works well with stumps)

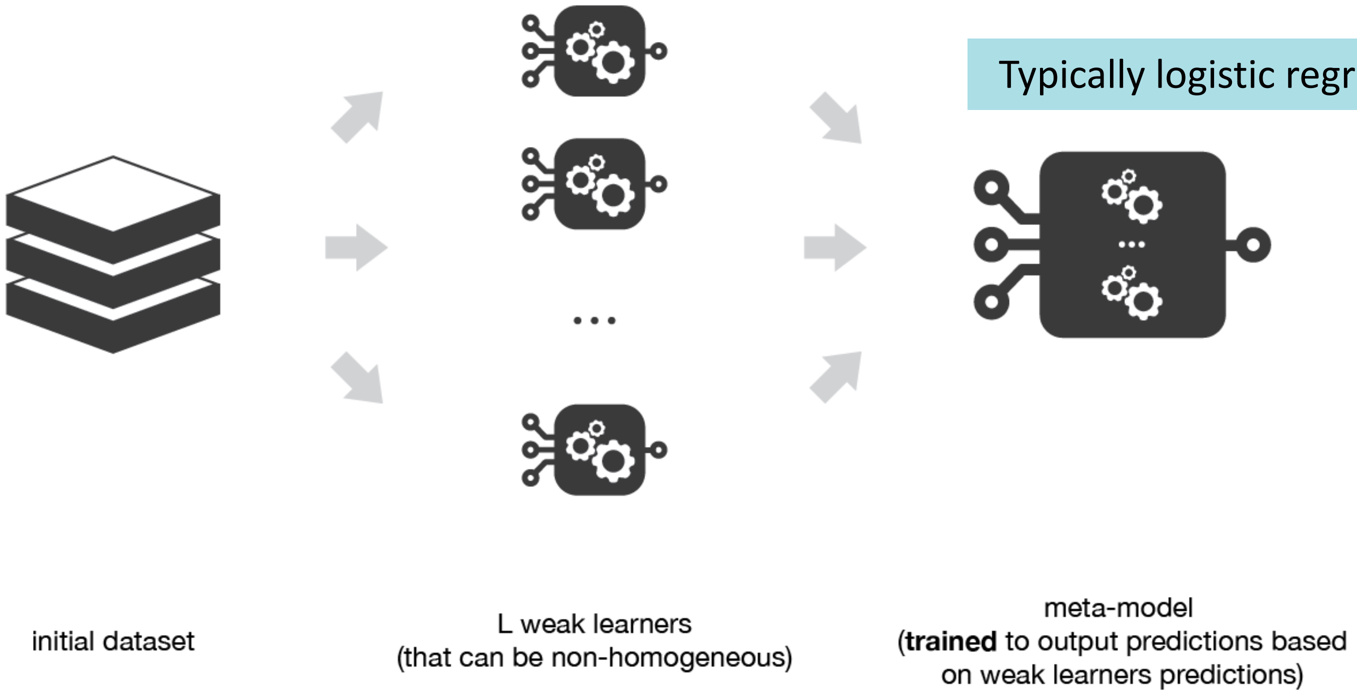
Stacking



```
Algorithm    Stacking
Input: training data $D = \{x_i, y_i\}_{i=1}^{m}$
Ouput: ensemble classifier $H$
Step 1: learn base-level classifiers
    $D_h = \{x'_i, y_i\}$, where $x'_i = \{h_1(x_i), ..., h_T(x_i)\}$
end for
Step 3: learn a meta-classifier
learn $H$ based on $D_h$
return $H$
```

- Base models are trained on a complete training set
- Then, a meta-model is trained on the outputs of the base-level model as features

Stacking - overview



Typically logistic regression

initial dataset

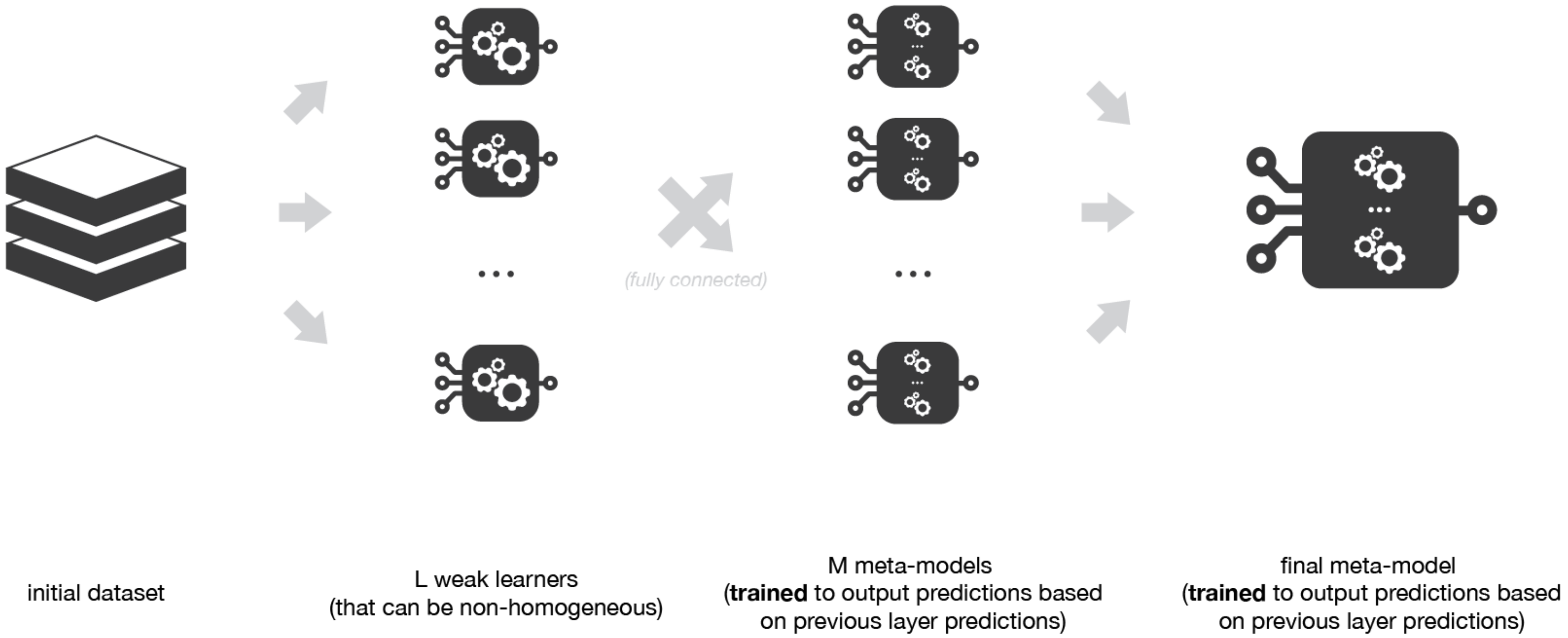
L weak learners
(that can be non-homogeneous)

meta-model
(**trained** to output predictions based on weak learners predictions)

Stacking consists in training a meta-model to produce outputs based on the outputs returned by some lower layer weak learners.

Multi-level stacking

Fit M meta-models instead of one, and then, combine them into a final meta-model



(fully connected)

initial dataset

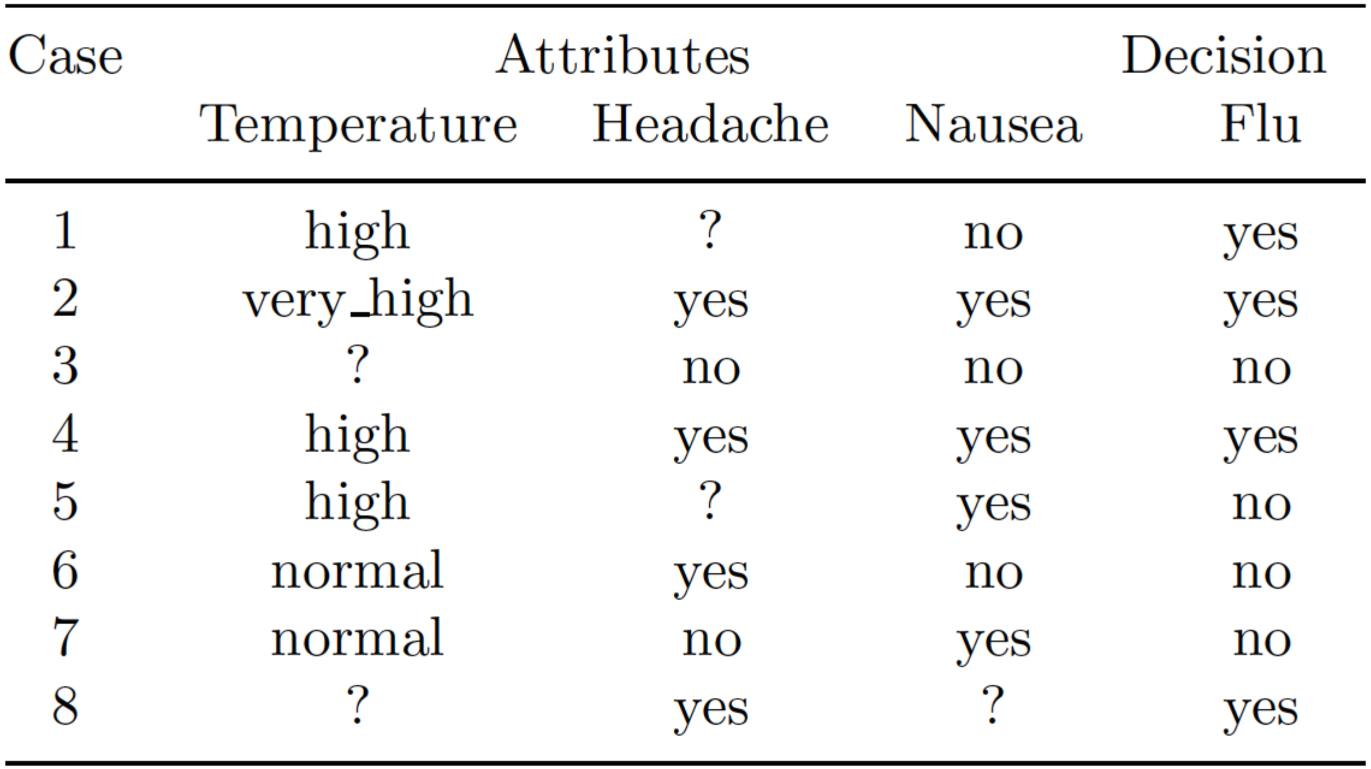
L weak learners
(that can be non-homogeneous)

M meta-models
(**trained** to output predictions based on previous layer predictions)

final meta-model
(**trained** to output predictions based on previous layer predictions)

Handling missing values

- Missing values are indicated with a ‘?’



Case	Attributes			Decision
	Temperature	Headache	Nausea	Flu
1	high	?	no	yes
2	very_high	yes	yes	yes
3	?	no	no	no
4	high	yes	yes	yes
5	high	?	yes	no
6	normal	yes	no	no
7	normal	no	yes	no
8	?	yes	?	yes

Handling missing values

Imputation:

An estimation of the missing value or of its distribution is used to generate predictions from a given model:

- a missing value is replaced with an estimation of the value or
- the distribution of possible missing values is estimated, and corresponding model predictions are combined probabilistically

Handling missing values

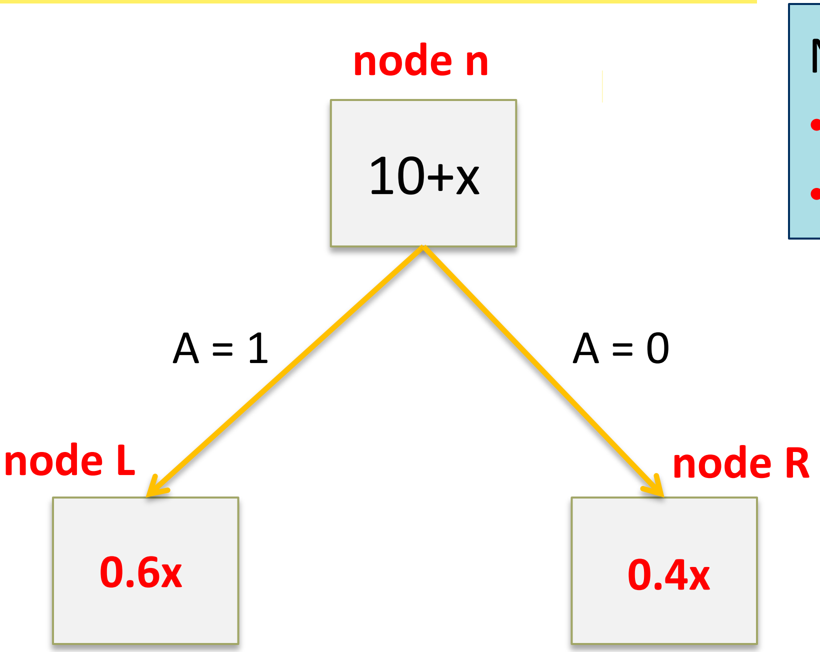
- Remove attributes with missing values
- Remove examples with missing values
- Assume most frequent value
- Assume most frequent value given a class
- Learn the distribution of a given attribute
- Induce relationships between the available attribute values and the missing feature

Missing values in Decision Trees

- Naïve:
  - Simply ignore them
  - Treat them as another category (if the attribute is nominal)
- Smarter:
  - Assign a probability to each possible value
  - Distribute the example to all branches using these probabilities

Missing values in Decision Trees

- Suppose that x is an example, and it has a missing value for attribute A
- Let x.A denote the value of attribute A for example x
- Attribute A is binary: it can be 0 or 1



Node n contains 10 examples + x:

- 6 (out of 10) examples with value 1 for A
- 4 (out of 10) examples with value 0 for A

- x.A = 1 with 0.6 probability
- x.A = 0 with 0.4 probability

A fraction of 60% of x goes to L and a fraction of 40% of x goes to R

Imputing missing values

- Expectation Maximization (EM):
  - Build model of data values (ignore missing ones)
  - Use model to estimate missing values
  - Build new model of data values (including estimated values from previous step)
  - Use new model to re-estimate missing values
  - Re-estimate model
  - Repeat until convergence

Potential Problems

- Imputed values may be **inappropriate**:
  - in medical databases, if missing values are not imputed separately for male and female patients, it may end up with male patients with 1.3 prior pregnancies and female patients suffering from a prostate infection
  - many of these situations will not be so obvious
- If some attributes are difficult to predict, filled-in values may be **random** (or worse)

Today…

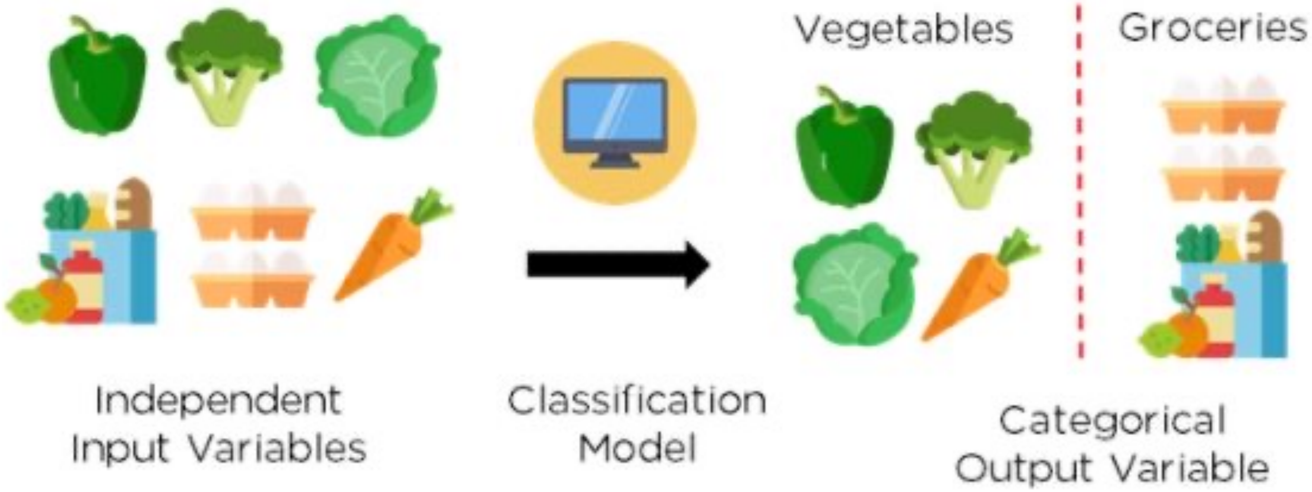
What is a **Naïve Bayes Classifier**?

What are **Random Forests**?

What is the difference between **Boosting** and **Bagging?**

What is **Stacking**?

How do we handle **Missing Values**?



TODOs

Reading:

Main course book:

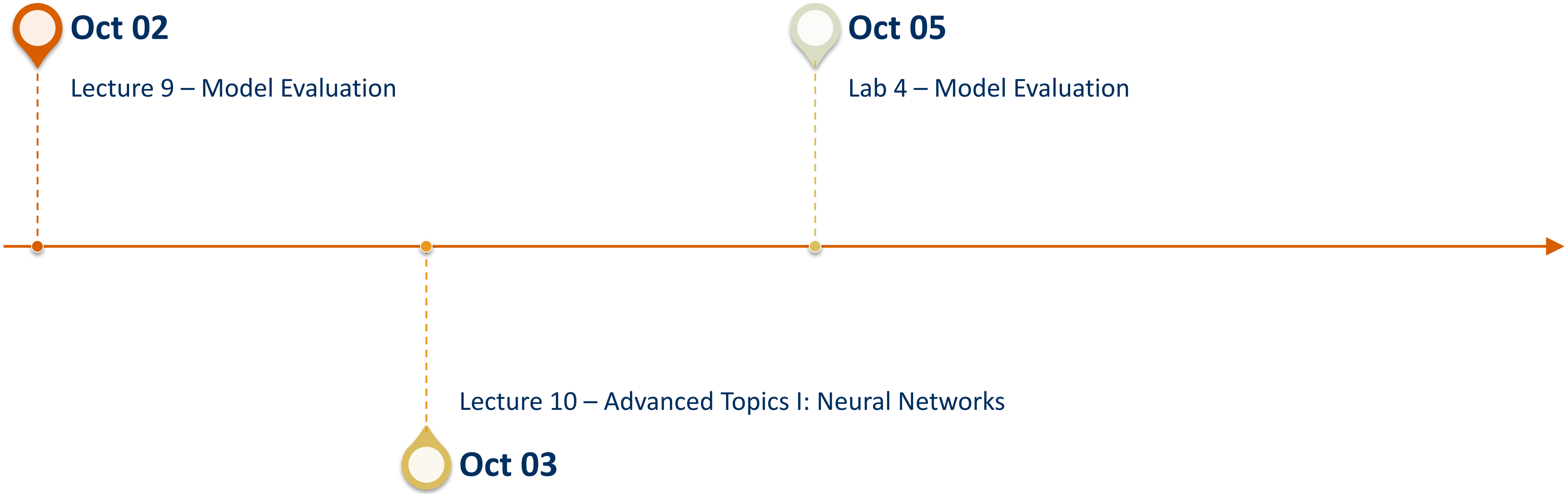
Chapter 18

Lab 3

Recommended to complete the lab before the end of the week

Quiz 3

Coming up next



Oct 02

Lecture 9 – Model Evaluation

Oct 03

Lecture 10 – Advanced Topics I: Neural Networks

Oct 05

Lab 4 – Model Evaluation