Introduction to Machine Learning (CSCI-UA 473): Fall 2021

Lecture 4: Linear Models for Classification

Sumit Chopra
Courant Institute of Mathematical Sciences
Department of Radiology - Grossman School of Medicine
NYU

Slides derived from materials from Benjamin Peherstorfer, Kyunghyun Cho, Andrew Gordon Wilson

Lecture Outline

Supervised learning setting recap

Three types of problems

Logistic regression model

The error metric and the loss function

Iterative optimization: gradient descent algorithm; stochastic gradient descent; mini-batch gradient descent

Non-linear transformations

Validation and cross validation

Example Problem: Credit from the Bank

As a bank and you receive thousands of credit card applications daily

For every application you want to answer a variety of questions:

1. Whether to extend a credit card to the applicant?
2. If the answer is "yes" then what should be the credit limit?

Applicant age

Applicant gender

Applicant employment status

Applicant annual income (if employed) x

Applicant criminal record

Applicant owns real estate property or not

Applicant owns car or not

Some of these variables are continuous, some are binary, and some are categorical (take only finite number of values)

Example Problem: Credit from the Bank

In a linear setting we make an assumption that function f is linear

$$f(x) \approx w_0 + \sum_{i=1}^{p} w_i x_i$$
$$= w^T x$$

$$w = [w_0, w_1, \ldots, w_p] \; and \; x = [1, x_1, x_2, \ldots, x_p]$$

Question 1: whether to extend credit or not?

y is binary (-1/+1): its a classification task

We use a simple linear classifier (Perceptron)

$$h(x) = sign(w^T x^j)$$

Perceptron Learning Algorithm (PLA)

Example Problem: Credit from the Bank

In a linear setting we make an assumption that function f is linear

$$f(x) \approx w_0 + \sum_{i=1}^{p} w_i x_i$$
$$= w^T x$$

$$w = [w_0, w_1, \ldots, w_p] \; and \; x = [1, x_1, x_2, \ldots, x_p]$$

Question 1: whether to extend credit or not?

y is binary (-1/+1): its a classification task

We use a simple linear classifier (Perceptron)

$$h(x) = sign(w^T x^j) \longleftarrow$$

Perceptron Learning Algorithm (PLA)

Question 2: how much credit to extend?

y is continuous: its a regression task

We use linear regression and squared loss (Lecture 3)

$$Loss(w) = \sum_{j=1}^{N} \left(y^j - w^T x^j\right)^2 \longleftarrow$$

Closed form solution

Example Problem: Credit from the Bank

In a linear setting we make an assumption that function f is linear

Can we ask something in the middle?

What is the probability of the person defaulting on credit?

The output of the model will be a number between 0 and 1

Not a regression problem because as part of the training data you are not given the probabilities. All you are given is whether a person defaulted on the loan or not

$$f(x) \approx w_0 + \sum_{i=1}^{p} w_i x_i \qquad w = [w_0, w_1, \ldots, w_p] \; and \; x = [1, x_1, x_2, \ldots, x_p]$$

$$= w^T x$$

Question 1: whether to extend credit or not?

y is binary (-1/+1): its a classification task

We use a simple linear classifier (Perceptron)

$$h(x) = sign(w^T x)$$ ← Perceptron Learning Algorithm (PLA)

Question 2: how much credit to extend?

y is continuous: its a regression task

We use linear regression and squared loss (Lecture 3)

$$Loss(w) = \sum_{j=1}^{N} \left(y^j - w^T x^j\right)^2$$ ← Closed form solution

Logistic Regression

Question 3: what is the probability of default by the customer?

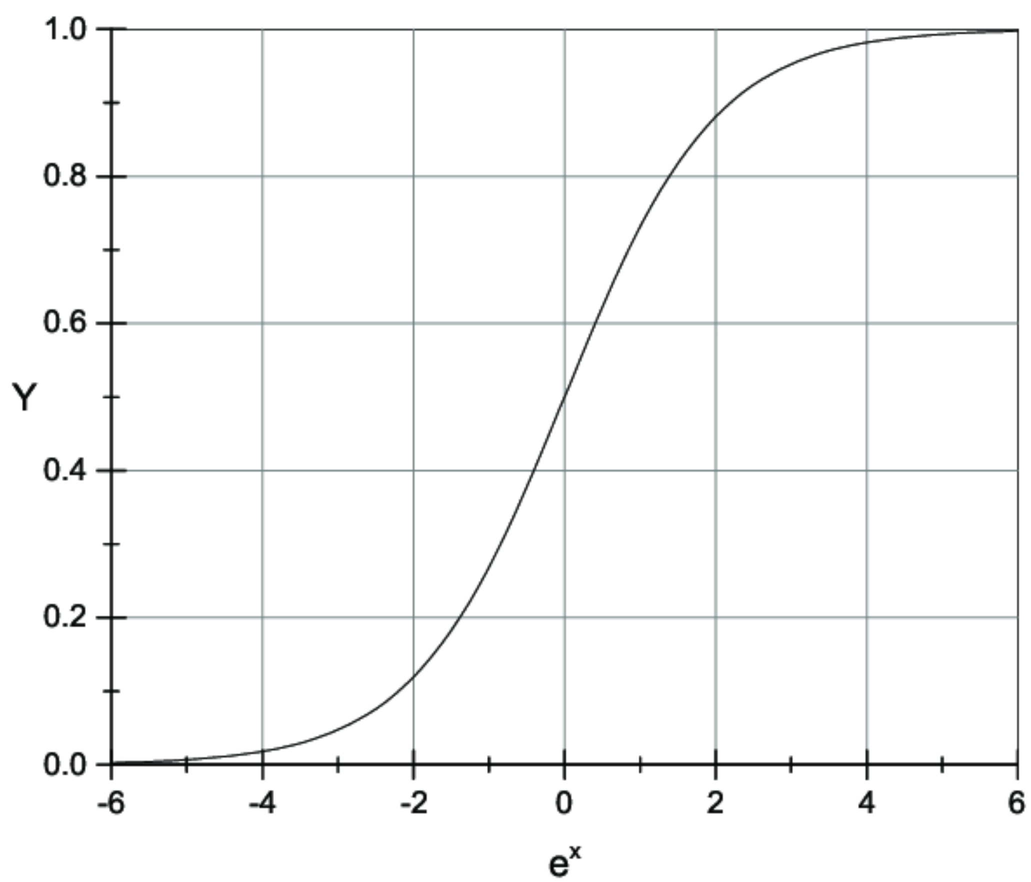
y is continuous and between 0 and 1

We use a logistic function to convert the signal into a number between 0 and 1

Can be interpreted as a probability of a binary event

The function σ can be seen as a soft threshold

$$h(x) = \sigma(w^T x^j)$$



The "signal" $s = \sum_{i=0}^{p} w_i x_i = w^T x$

$$\sigma(s) = \frac{e^s}{1 + e^s}$$ ← Logistic function

Logistic Regression

$$f(x) = \mathbb{P}[y = +1 \,|\, x]$$

We are trying to learn this target function

Note that the target is a probability

The data does not give us explicit probabilities (value of f)

Instead it only provides samples generated from it (binary information)

Thus the data is in fact generated by noisy target $P(y \,|\, x)$

$$P(y \,|\, x) = f(x) \qquad for \;\; y = +1$$
$$P(y \,|\, x) = 1 - f(x) \qquad for \;\; y = -1$$

Logistic Regression: Loss Function

The standard error measure used is based on the notion of "likelihood"

How likely is it that we'll get the output y for a given input x if indeed the target distribution $P(y \mid x)$ was captured by the hypothesis $h(x)$

$$
\begin{aligned}
P(y \mid x) &= h(x) && for \;\; y = +1 \\
P(y \mid x) &= 1 - h(x) && for \;\; y = -1
\end{aligned}
$$

$$P(y \mid x) = h(x) = \sigma(y w^T x)$$

Logistic Regression: Loss Function

$$P(y|x) = \sigma(yw^T x)$$

Using the above formula we can compute the likelihood of the training data
$D = \{(x^1, y^1), (x^2, y^2), \ldots, (x^N, y^N)\}$

Training samples are independently and identically drawn

$$P(Y|X) = \prod_{i=1}^{N} P(y^i | x^i) = \prod_{i=1}^{N} \sigma(y^i w^T x^i)$$

Process of learning reduces to adjusting the parameters w in order to maximize the above likelihood

This is called Maximum Likelihood Estimation!

Logistic Regression: Loss Function

$$P(y|x) = \sigma(yw^T x)$$

Using the above formula we can compute the likelihood of the training data
$D = \{(x^1, y^1), (x^2, y^2), \ldots, (x^N, y^N)\}$

Training samples are independently and identically drawn

$$P(Y|X) = \prod_{i=1}^{N} P(y^i | x^i) = \prod_{i=1}^{N} \sigma(y^i w^T x^i)$$

Process of learning reduces to adjusting the parameters w in order to maximize the above likelihood

This is called Maximum Likelihood Estimation!

In practice a variant of this loss function is minimized

We take the $-\frac{1}{N}\log(\,.\,)$ of the likelihood and minimize it

The two are equivalent since the function is monotonically decreasing

The reasons are grounded in computational stability

Logistic Regression: Loss Function

$$P(Y|X) = \prod_{i=1}^{N} P(y^i|x^i) = \prod_{i=1}^{N} \sigma(y^i w^T x^i)$$

Thus we have

$$E_{in}(w) = -\frac{1}{N} \ln \left(\prod_{i=1}^{N} P(y^i|x^i) \right) = \frac{1}{N} \sum_{i=1}^{N} \ln \left(\frac{1}{P(y^i|x^i)} \right) = \frac{1}{N} \sum_{i=1}^{N} \ln \left(\frac{1}{\sigma(y^i w^T x^i)} \right)$$

$$\boxed{E_{in}(w) = \frac{1}{N} \sum_{i=1}^{N} \ln \left(1 + e^{-y^i w^T x^i} \right)} \longleftarrow$$

Cross-Entropy Loss

This error measure is small when $y^i w^T x^i$ is large and positive

Thus minimizing the error measure pushes w to classify each x^i correctly

Logistic Regression: Loss Function

$$P(Y|X) = \prod_{i=1}^{N} P(y^i|x^i) = \prod_{i=1}^{N} \sigma(y^i w^T x^i)$$

Thus we have

$$E_{in}(w) = -\frac{1}{N} \ln\left(\prod_{i=1}^{N} P(y^i|x^i)\right) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(\frac{1}{P(y^i|x^i)}\right) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(\frac{1}{\sigma(y^i w^T x^i)}\right)$$

$$E_{in}(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y^i w^T x^i}\right) \longleftarrow$$ Cross-Entropy Loss

This error measure is small when $y^i w^T x^i$ is large and positive

Thus minimizing the error measure pushes w to classify each x^i correctly

$$E_{in}(w) = -\sum_{i=1}^{N} \left[y^i \cdot \log \sigma(w^T x^i) + (1 - y^i) \cdot \log(1 - \sigma(w^T x^i))\right]$$

Logistic Regression: Loss Function

$$P(Y|X) = \prod_{i=1}^{N} P(y^i|x^i) = \prod_{i=1}^{N} \sigma(y^i w^T x^i)$$

Thus we have

$$E_{in}(w) = -\frac{1}{N} \ln\left(\prod_{i=1}^{N} P(y^i|x^i)\right) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(\frac{1}{P(y^i|x^i)}\right) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(\frac{1}{\sigma(y^i w^T x^i)}\right)$$

How do we optimize this loss function?

$$E_{in}(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y^i w^T x^i}\right)$$

Cross-Entropy Loss

This error measure is small when $y^i w^T x^i$ is large and positive

Thus minimizing the error measure pushes w to classify each x^i correctly

$$E_{in}(w) = -\sum_{i=1}^{N} \left[y^i \cdot \log \sigma(w^T x^i) + (1 - y^i) \cdot \log(1 - \sigma(w^T x^i))\right]$$

Logistic Regression: Loss Function Optimization

Linear regression has a closed form solution after setting gradient to zero

$$Loss(w) = \sum_{i=1}^{N} \left(y^i - w^T x^i\right)^2 \longrightarrow \hat{w} = (X^T X)^{-1} X^T Y$$

Linear classification (+1/-1) used the Perceptron Learning Algorithm (PLA)

$$h(x) = sign(w^T x) \longrightarrow w(t+1) \leftarrow w(t) + y(t)x(t)$$

$$\boxed{E_{in}(w) = \frac{1}{N} \sum_{i=1}^{N} \ln\left(1 + e^{-y^i w^T x^i}\right)}$$

We neither have a closed form solution nor we can apply the PLA algorithm

We will use an iterative optimization algorithm called Gradient Descent

Gradient Descent Algorithm

$$E_{in}(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y^i w^T x^i}\right)$$

An iterative algorithm that progressively modifies w in a way that decreases the error

Basic Outline

Start with an initial value of parameters w

Compute the direction from that point where there is a decrease in the loss

Update the parameters in that direction

$$w(1) \leftarrow w(0) + \eta\hat{v}$$

Gradient Descent Algorithm

$$E_{in}(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y^i w^T x^i}\right)$$

An iterative algorithm that progressively modifies w in a way that decreases the error

Basic Outline

Start with an initial value of parameters w

Compute the direction from that point where there is a decrease in the loss

Update the parameters in that direction

$$w(1) \leftarrow w(0) + \eta \hat{v}$$

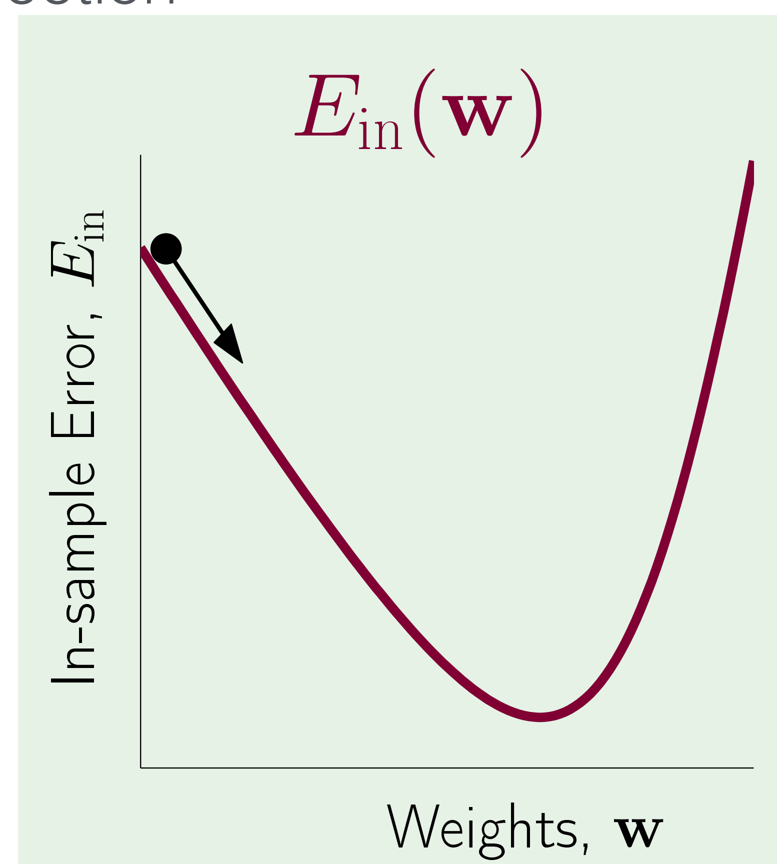


Figure from the book Learning from Data

Gradient Descent Algorithm

$$E_{in}(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y^i w^T x^i}\right)$$

An iterative algorithm that progressively modifies w in a way that decreases the error

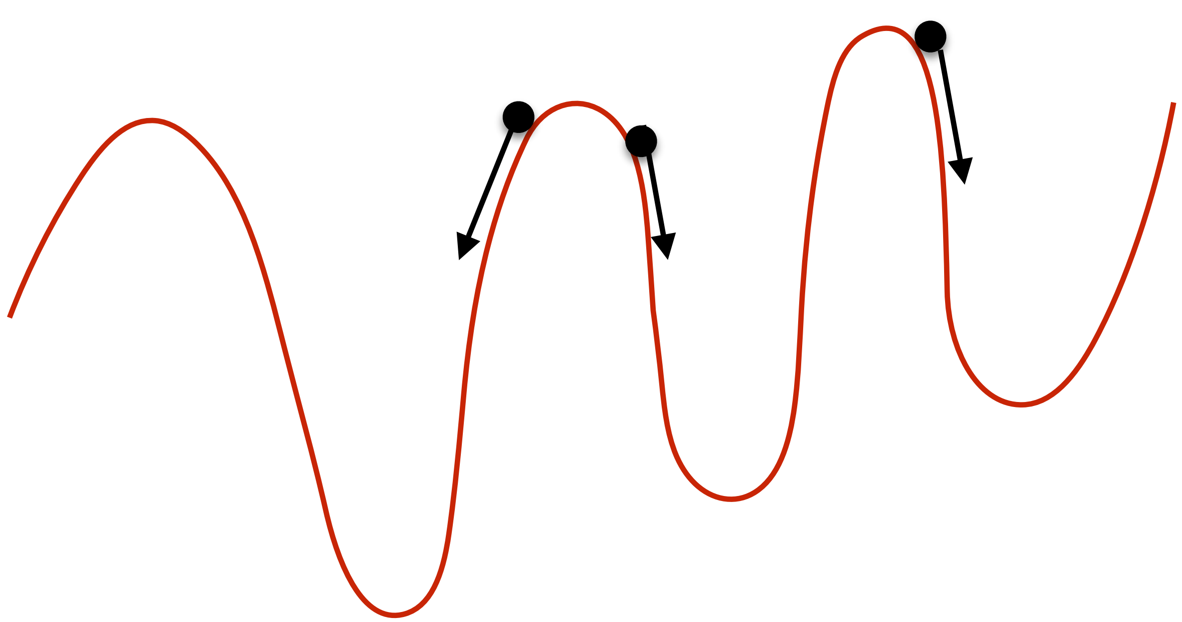
Basic Outline

Start with an initial value of parameters w

Compute the direction from that point where there is a decrease in the loss

Update the parameters in that direction

$$w(1) \leftarrow w(0) + \eta \hat{v}$$



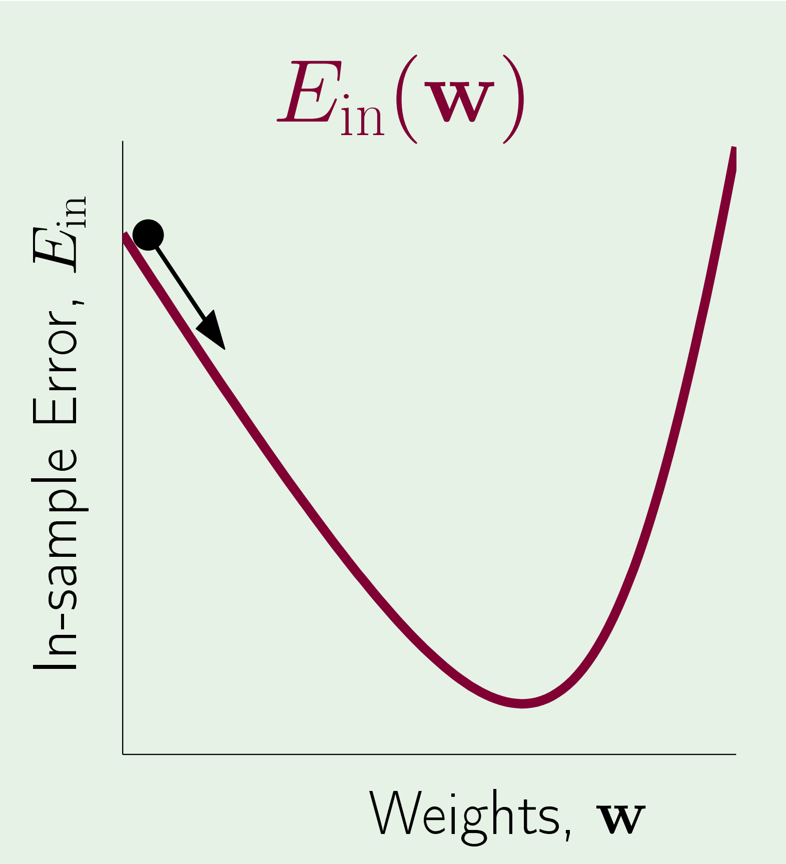


Figure from the book Learning from Data

Gradient Descent Algorithm

$$E_{in}(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y^i w^T x^i}\right)$$

An iterative algorithm that progressively modifies w in a way that decreases the error

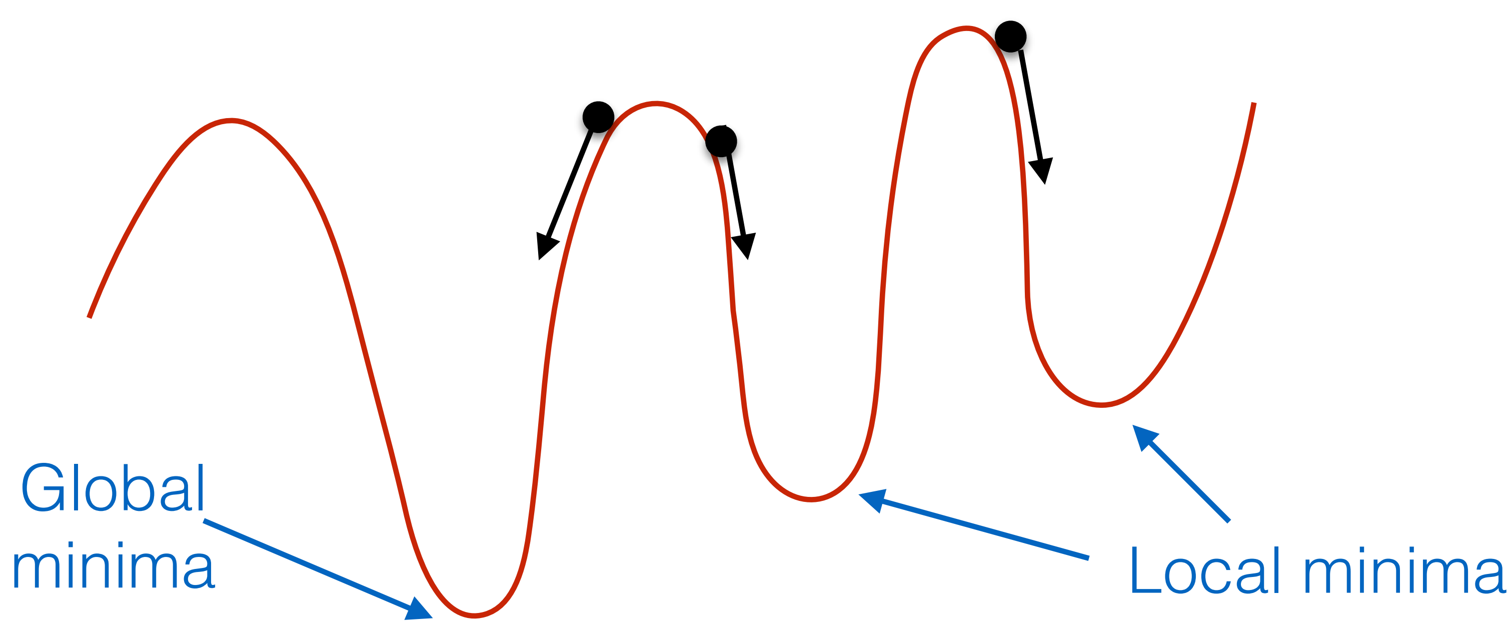
Basic Outline

Start with an initial value of parameters w

Compute the direction from that point where there is a decrease in the loss

Update the parameters in that direction

$$w(1) \leftarrow w(0) + \eta\hat{v}$$



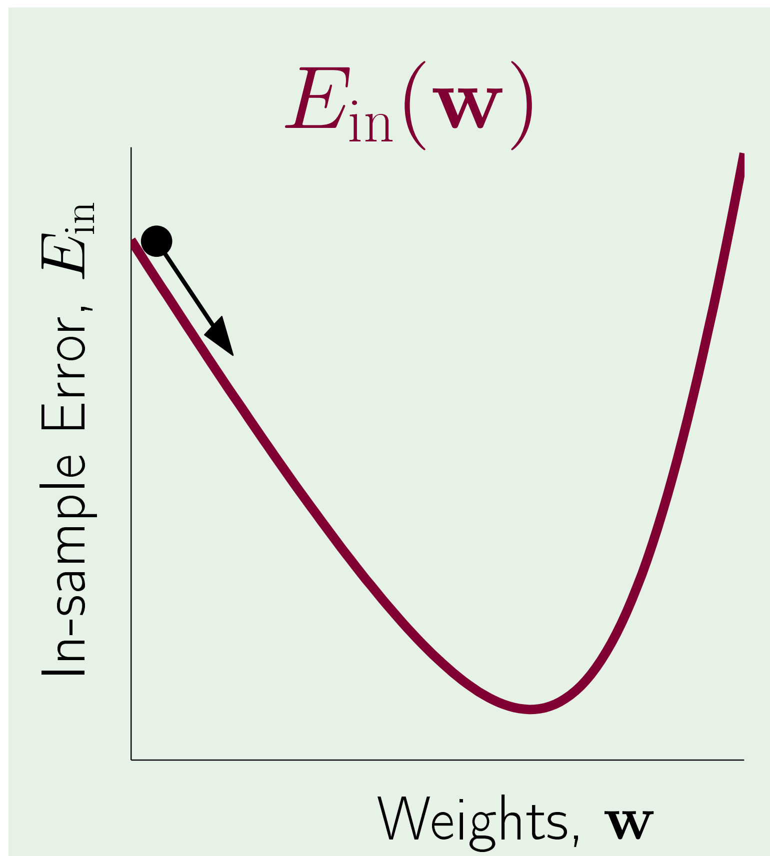


Figure from the book Learning from Data

Gradient Descent Algorithm

$$E_{in}(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y^i w^T x^i}\right)$$

An iterative algorithm that progressively modifies w in a way that decreases the error

Basic Outline

Start with an initial value of parameters w

Compute the direction from that point where there is a decrease in the loss

Update the parameters in that direction

Learning rate

$$w(1) \leftarrow w(0) + \eta\hat{v}$$

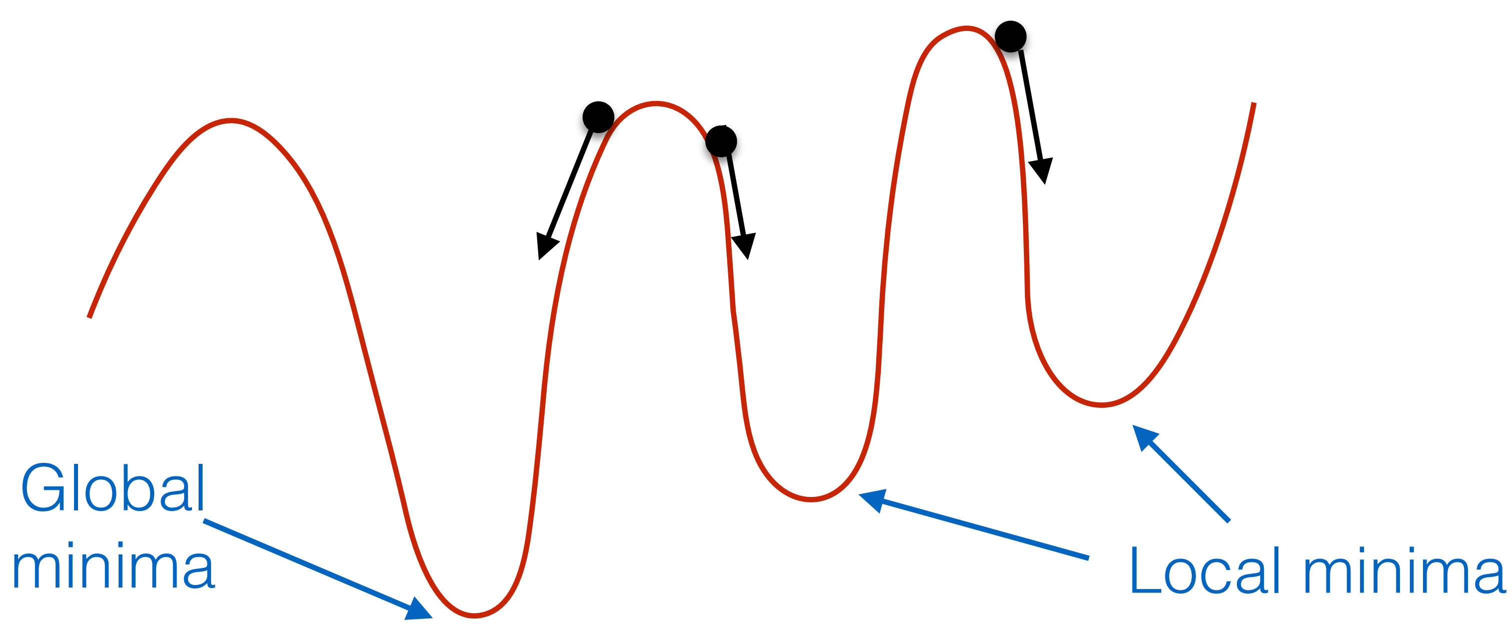
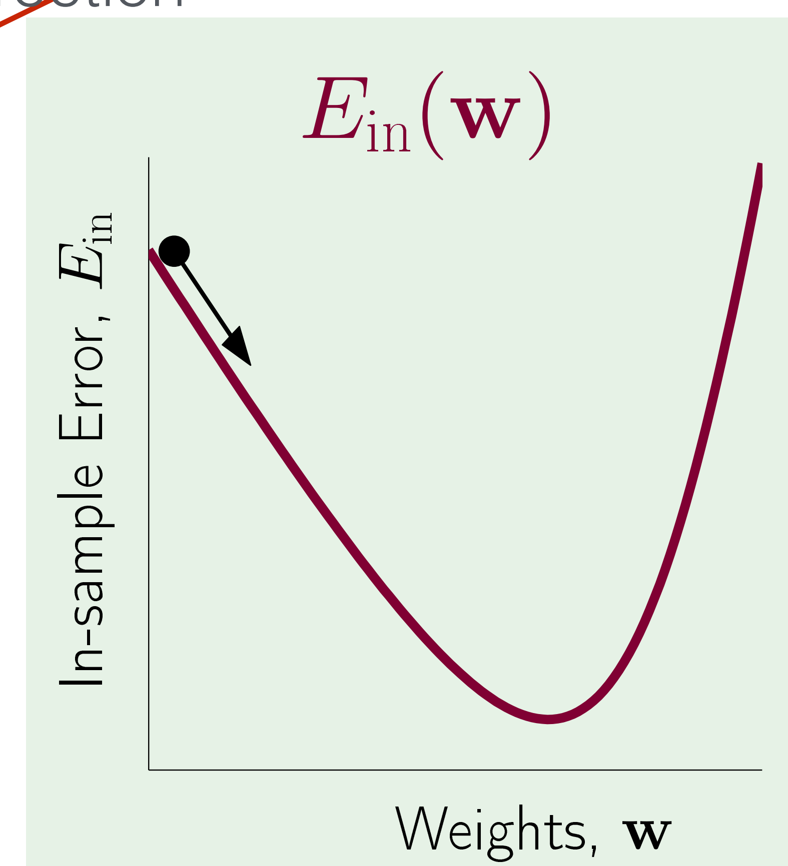


Figure from the book Learning from Data



Gradient Descent Algorithm

$$E_{in}(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y^i w^T x^i}\right)$$

So how do we role down the surface E_{in}?

We would like to take a step in the direction of steepest descent to gain the biggest bang for the buck

$$w(1) \leftarrow w(0) + \eta \hat{v}$$

Gradient Descent Algorithm

$$\boxed{E_{in}(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y^i w^T x^i}\right)}$$

So how do we role down the surface E_{in}?

We would like to take a step in the direction of steepest descent to gain the biggest bang for the buck

$$w(1) \leftarrow w(0) + \eta\hat{v}$$

Using first order Taylor expansion we compute the change in E_{in}

$$\begin{aligned}\Delta E_{in} &= E_{in}(w(0) + \eta\hat{v}) - E_{in}(w(0)) \\ &= \eta \nabla E_{in}(w(0))^T \hat{v} + O(\eta^2) \\ &\geq -\eta \,||\nabla E_{in}(w(0))||\end{aligned}$$

Since $\hat{v}$ is a unit vector, this equality holds if and only if

$$\boxed{\hat{v} = -\frac{\nabla E_{in}(w(0))}{||\nabla E_{in}(w(0))||}}$$

Direction of steepest descent

Gradient Descent Algorithm

How about η? How do we choose it?

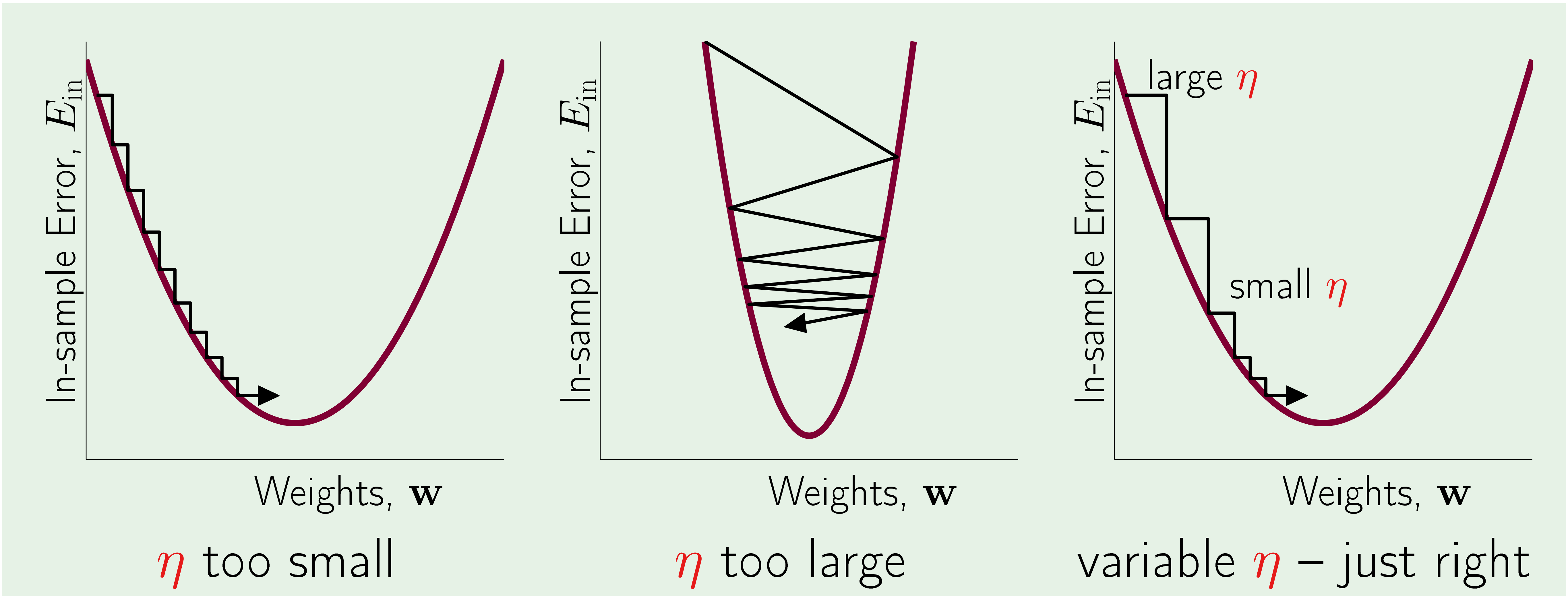
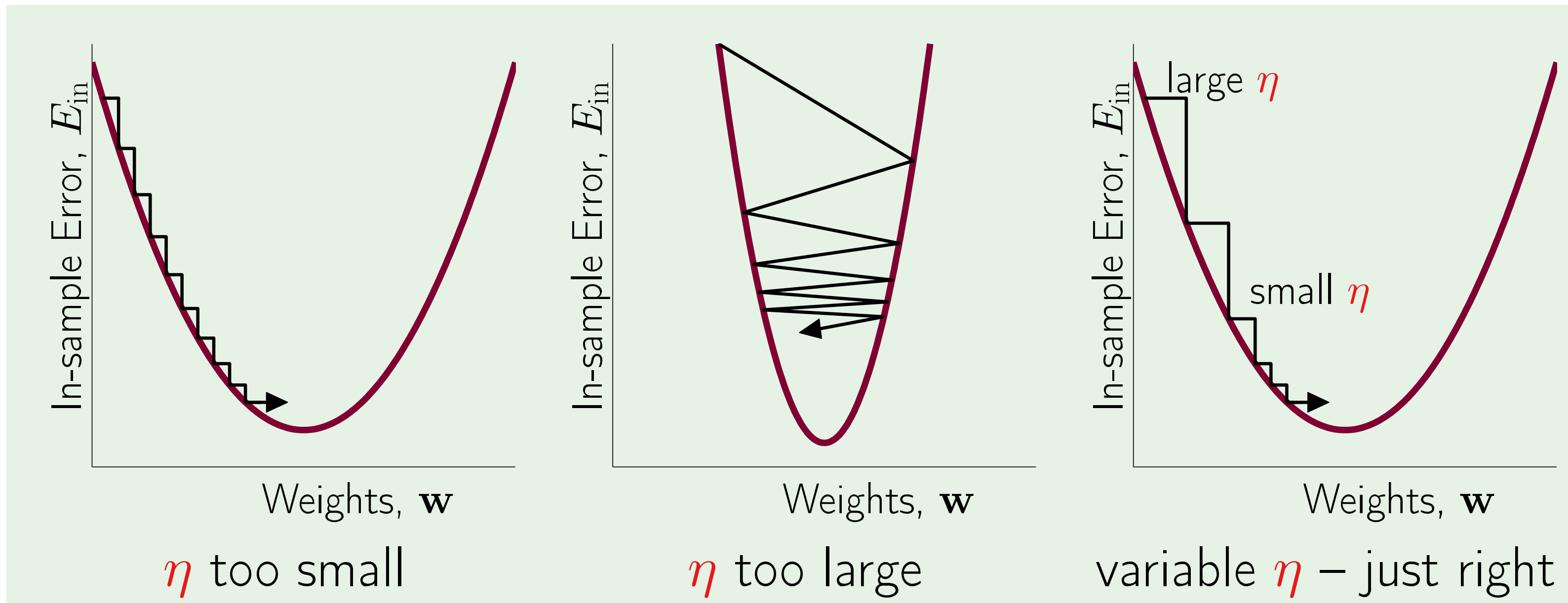


Figure from the book Learning from Data

Gradient Descent Algorithm

How about η? How do we choose it?



Large step size when far away from local minima

Small step size when close to the local minima

The following simple heuristic does the trick

$$\eta_t = \eta \,||\nabla E_{in}||$$

Thus we have a fixed learning rate algorithm

$$\Delta w = -\eta \nabla E_{in}$$

Figure from the book Learning from Data

The Logistic Regression Algorithm

1. Initialize the weights at time step $t = 0$ to $w(0)$
2. For $t = 0,1,2,\ldots$ do
3. Compute the gradient $g_t = -\frac{1}{N}\sum_{n=1}^{N}\frac{y^n x^n}{1 + e^{y^n w^T(t) x^n}}$
4. Set the direction to move $v_t = -g_t$
5. Update the weights:; $w(t+1) = w(t) + \eta v_t$
6. Return the final weights w

Stochastic Gradient Descent

We computed gradient on the entire training set and then made a weight update

$$g_t = -\frac{1}{N}\sum_{n=1}^{N}\frac{y^n x^n}{1+e^{y^n w^T(t) x^n}}$$

This is called the batch gradient descent

In stochastic gradient descent we do the following:

1. Randomly pick a training sample (x^i, y^i)
2. Compute the gradient of the loss associated with this training sample

$$\nabla e_i(w) = \frac{-y^i x^i}{1+e^{y^i w^T x^i}}$$

3. Update the weights
4. $w(t+1) \leftarrow w(t) - \eta \nabla e_i(w)$

Mini-batch Gradient Descent

Mini-batch gradient descent is somewhere in between batch and stochastic

We do the following:

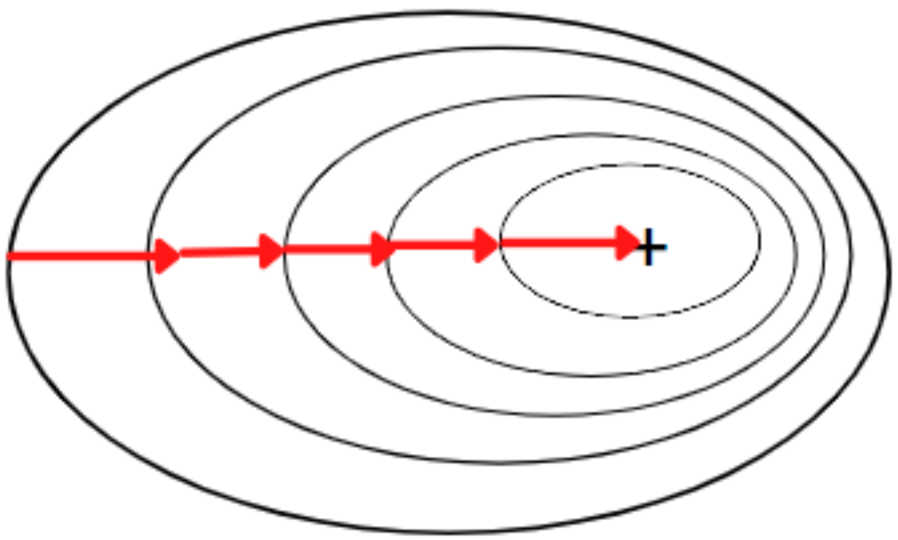
1. Randomly pick M training samples: $B = (x^i, y^i), \ldots, (x^{i+M}, y^{i+M})$
2. Compute the gradient of the loss associated with this mini-batch

$$g_B = -\frac{1}{M} \sum_{(x^j, y^i) \in B} \frac{-y^j x^j}{1 + e^{y^j w^T x^j}}$$

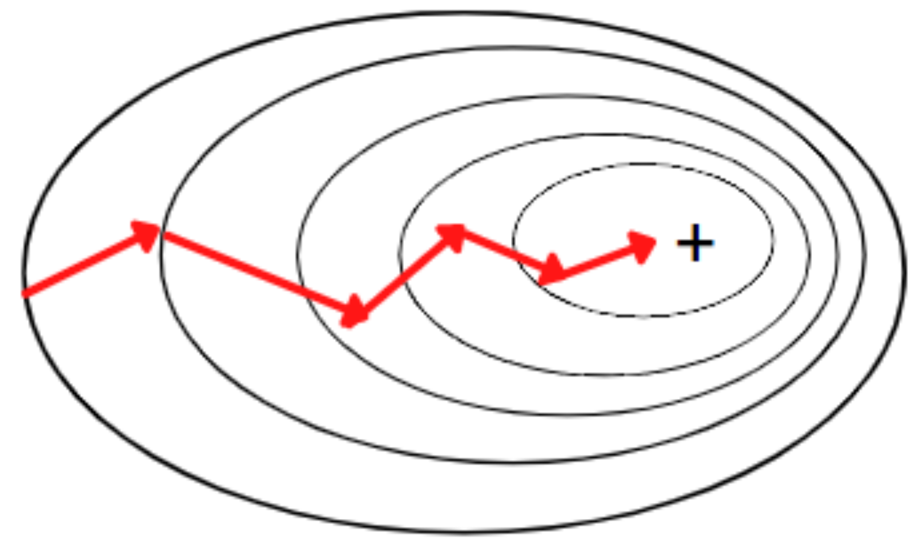
3. Update the weights
4. $w(t+1) \leftarrow w(t) - \eta g_B$

Gradient Descents

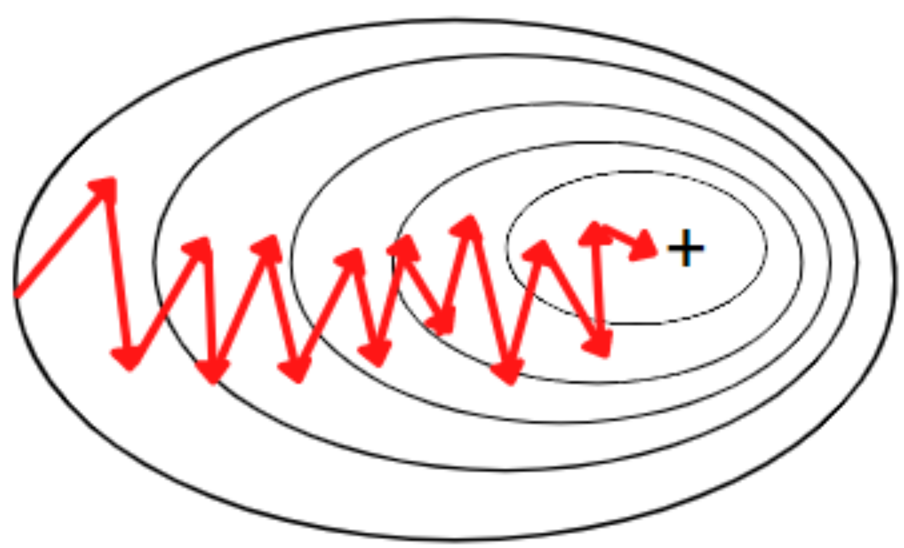
Batch Gradient Descent



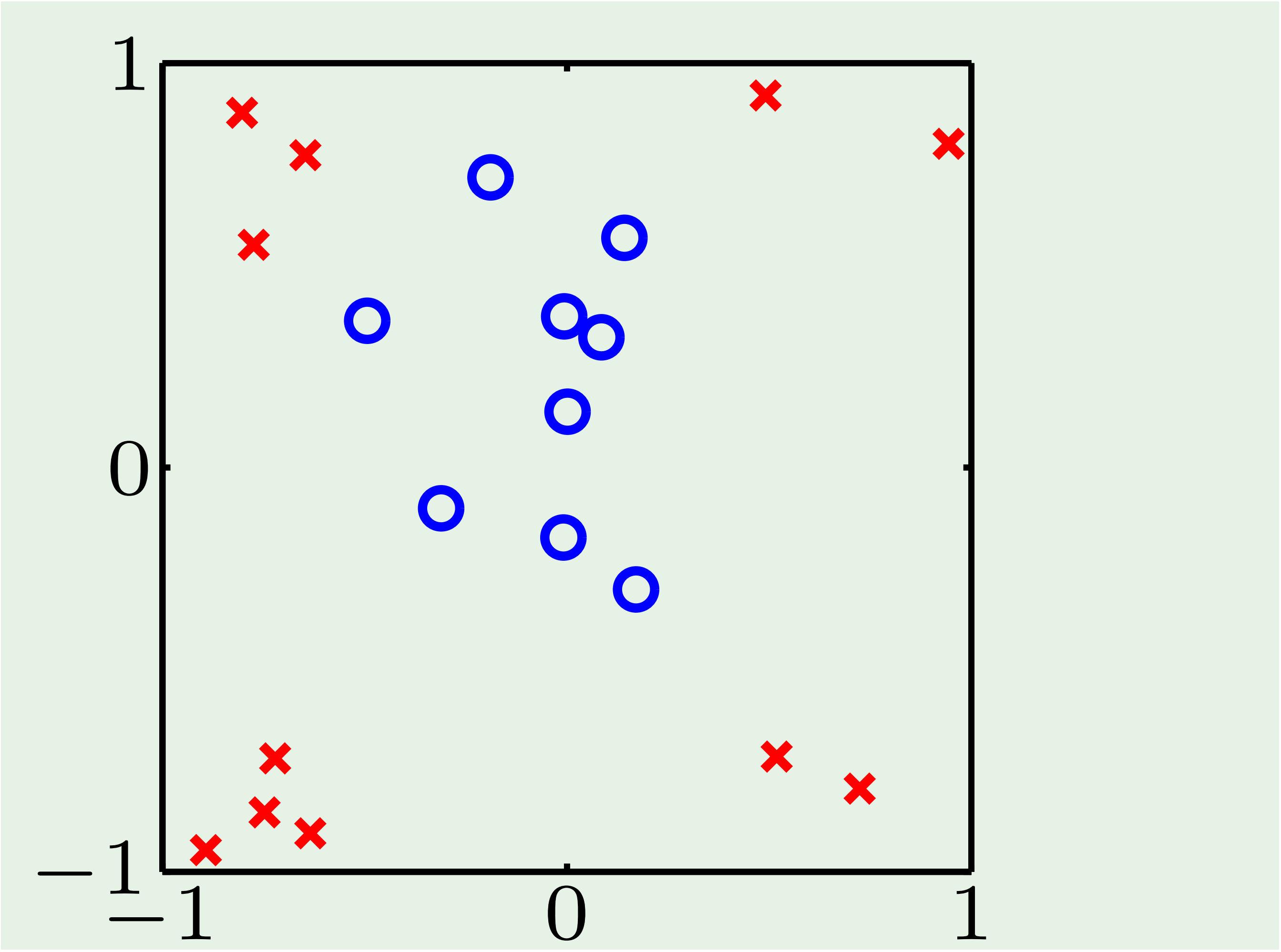
Mini-Batch Gradient Descent



Stochastic Gradient Descent

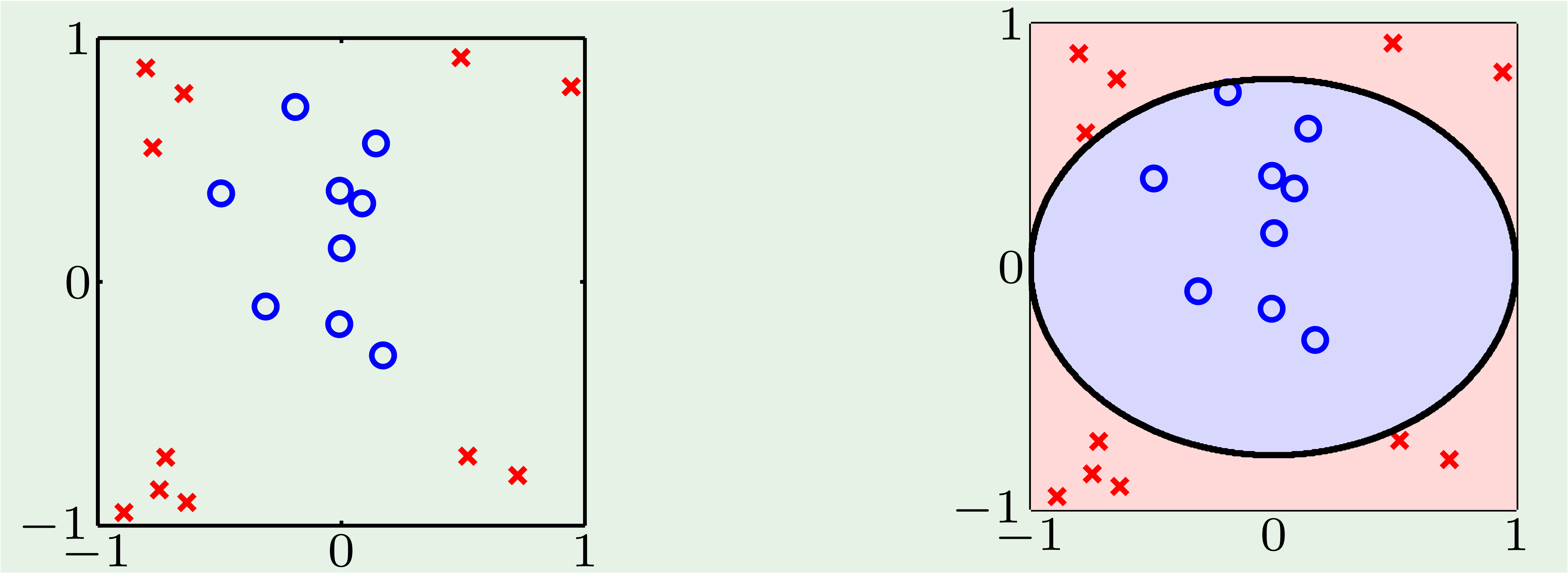


Non Linear Transformation



Slide adapted from the book Learning from Data

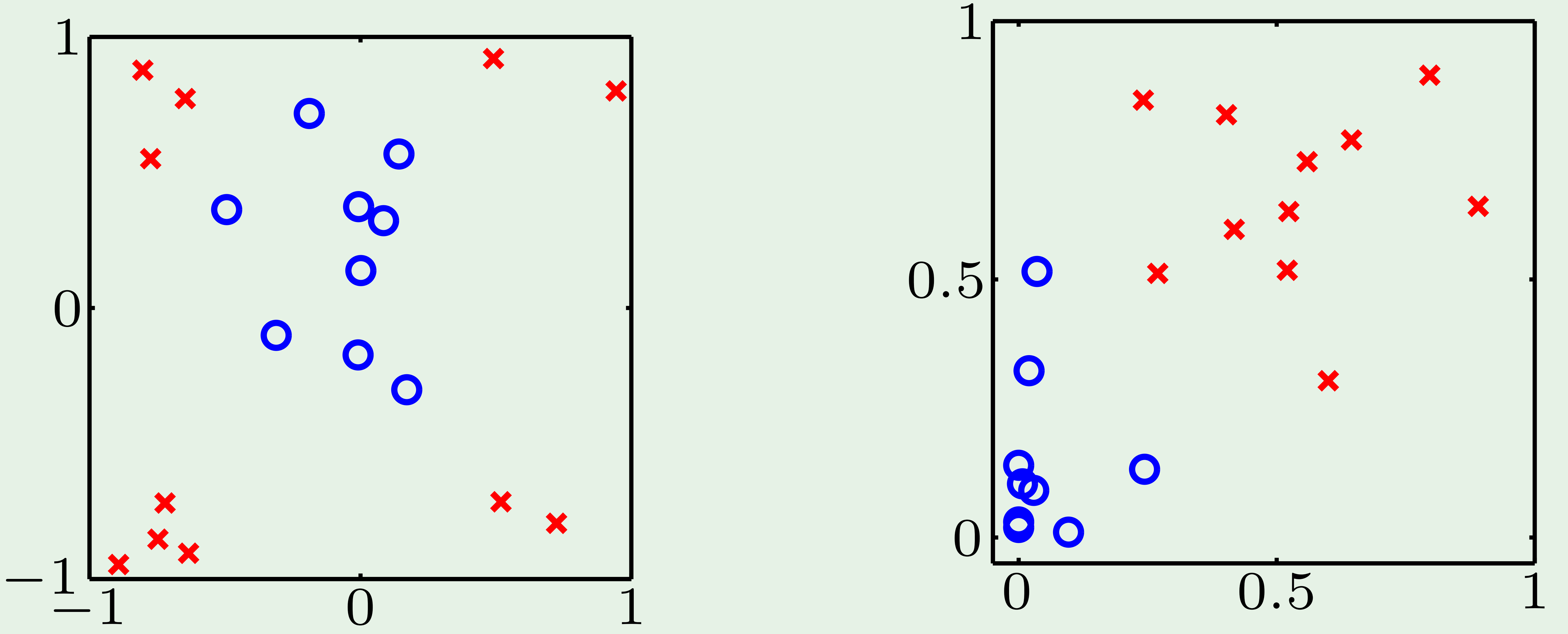
Non Linear Transformation



Slide adapted from the book Learning from Data

Non Linear Transformation

$$(x_1, x_2) \longrightarrow (x_1^2, x_2^2)$$



You need to come up with a transformation before looking at the data

Slide adapted from the book Learning from Data

Validation

For any hypothesis h

$E_{out}(h) = E_{in}(h)$ + Penalty for overfitting

Regularization
estimates this
quantity

Validation

For any hypothesis h

$E_{out}(h) = E_{in}(h)$ + Penalty for overfitting

Validation cuts to the chase and tries to directly estimate this

Regularization estimates this quantity

Validation

We briefly discussed carving out a validation set from your training set to estimate out-of-sample error

Can we say something more formally?

For any hypothesis h

$$E_{out}(h) = E_{in}(h) + \text{Penalty for overfitting}$$

Validation cuts to the chase and tries to directly estimate this

Regularization estimates this quantity

Validation

Dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$

Training set $D_{train} \in D \leftarrow N - K$ samples

Validation set $D_{val} \in D \leftarrow K$ samples

g^-: the hypothesis selected after training the model on D_{train}

$$E_{val}(g^-) = \frac{1}{K} \sum_{x_n \in D_{val}} e\left(g^-(x_n), y_n\right)$$

How do we know whether $E_{val}(g^-)$ is a reasonable estimate of $E_{out}(g^-)$?

Validation

$$
\begin{aligned}
\mathbb{E}_{D_{val}}\left[E_{val}(g^-)\right] &= \mathbb{E}_{D_{val}}\left[\frac{1}{K}\sum_{x_n \in D_{val}} e\left(g^-(x_n), y_n\right)\right] \\
&= \frac{1}{K}\sum_{x_n \in D_{val}} \mathbb{E}_{D_{val}}\left[e\left(g^-(x_n), y_n\right)\right] \\
&= \frac{1}{K}\sum_{x_n \in D_{val}} E_{out}(g^-) \\
&= E_{out}(g^-)
\end{aligned}
$$

This is because

$$
\mathbb{E}_{D_{val}}\left[e\left(g^-(x_n), y_n\right)\right] = E_{x_n}\left[e\left(g^-(x_n), y_n\right)\right] = E_{out}(g^-)
$$

Validation

Thus we have

$$\mathbb{E}\left[E_{val}(g^-)\right] = \frac{1}{K}\sum_{x_n \in D_{val}} \mathbb{E}\left[e\left(g^-(x_n), y_n\right)\right] = E_{out}(g^-)$$

$$\mathbb{V}\left[E_{val}(g^-)\right] = \frac{1}{K^2}\sum_{x_n \in D_{val}} \mathbb{V}\left[e\left(g^-(x_n), y_n\right)\right] = \frac{\sigma^2}{K}$$

$$E_{out}(g^-) \leq E_{val}(g^-) + O\left(\frac{1}{\sqrt{K}}\right)$$

Small K will lead to a poor estimate of the error

But what about large K ?

Validation

Note that the validation set is carved out of the full data set

Large K means small $N - K$ (size of D_{train})

So your chosen hypothesis g^- is poor and hence the estimates will be completely off

Practical rule of thumb: use 20% of D as D_{val}

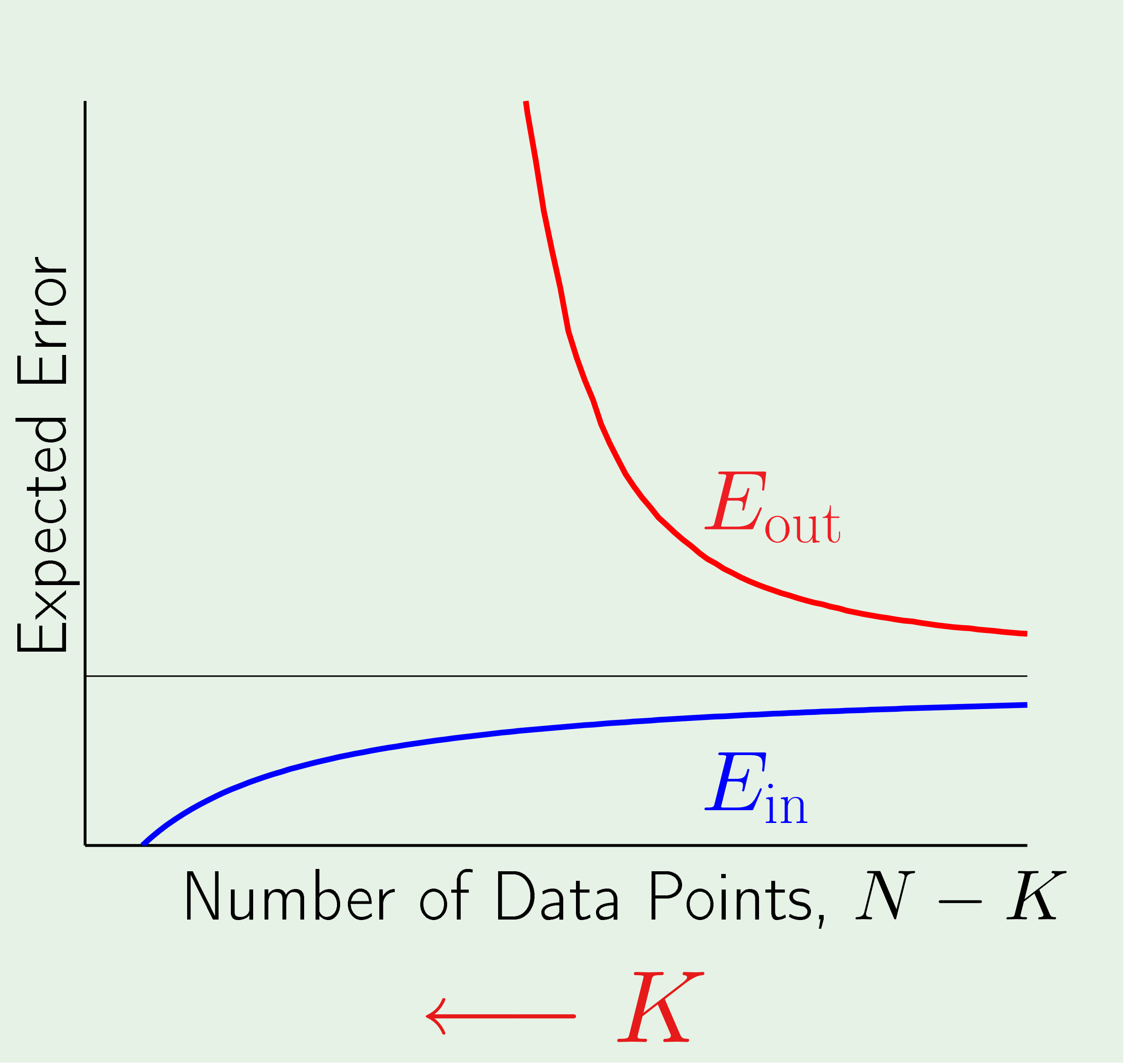


Figure from the book Learning from Data

Validation: Fold Back In

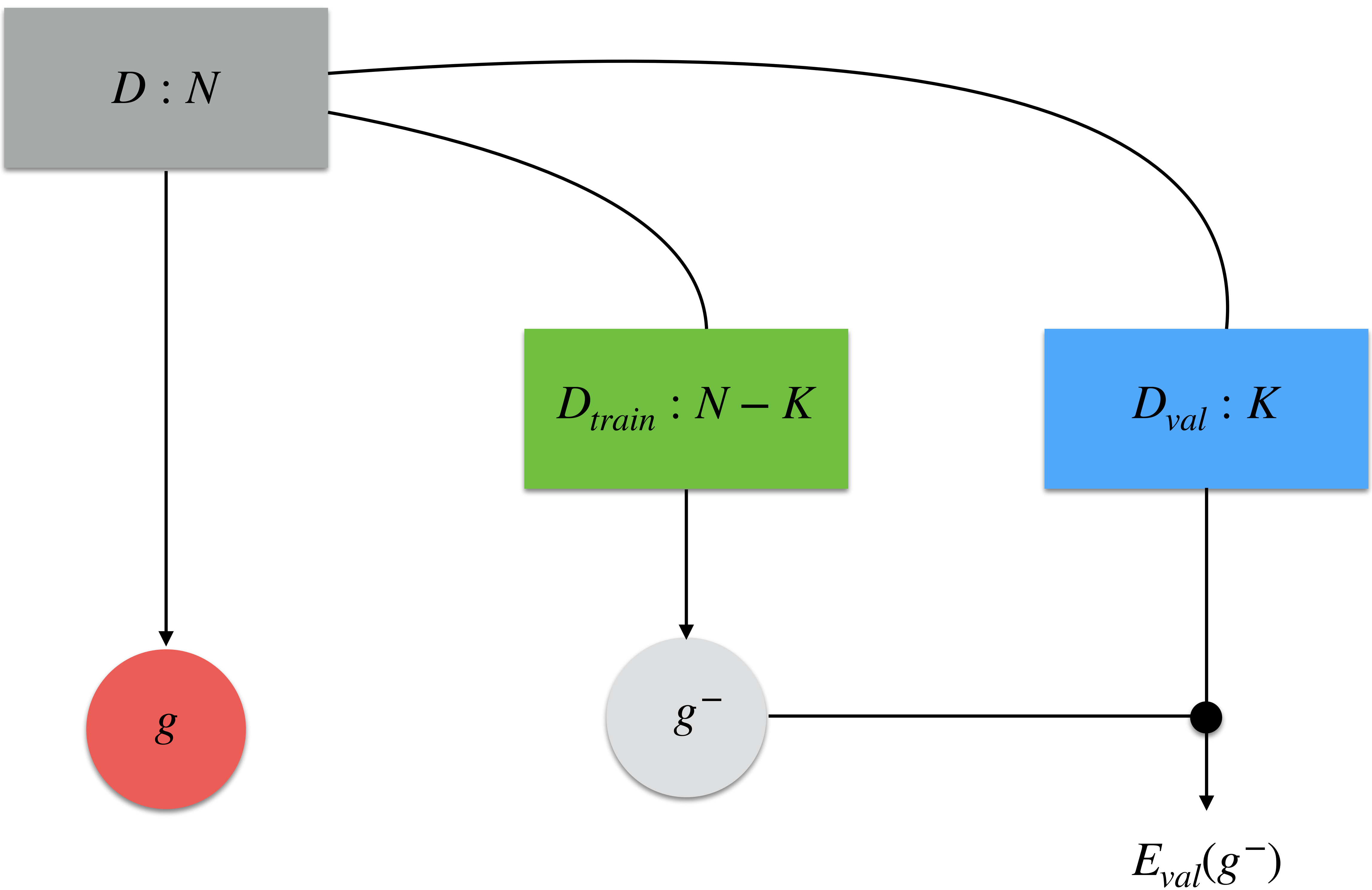


Figure adapted from the book Learning from Data

Model Selection

Turns out that you can use the same validation set multiple times without loosing guarantees

Assume you have M models (hypothesis sets): $\{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_M\}$

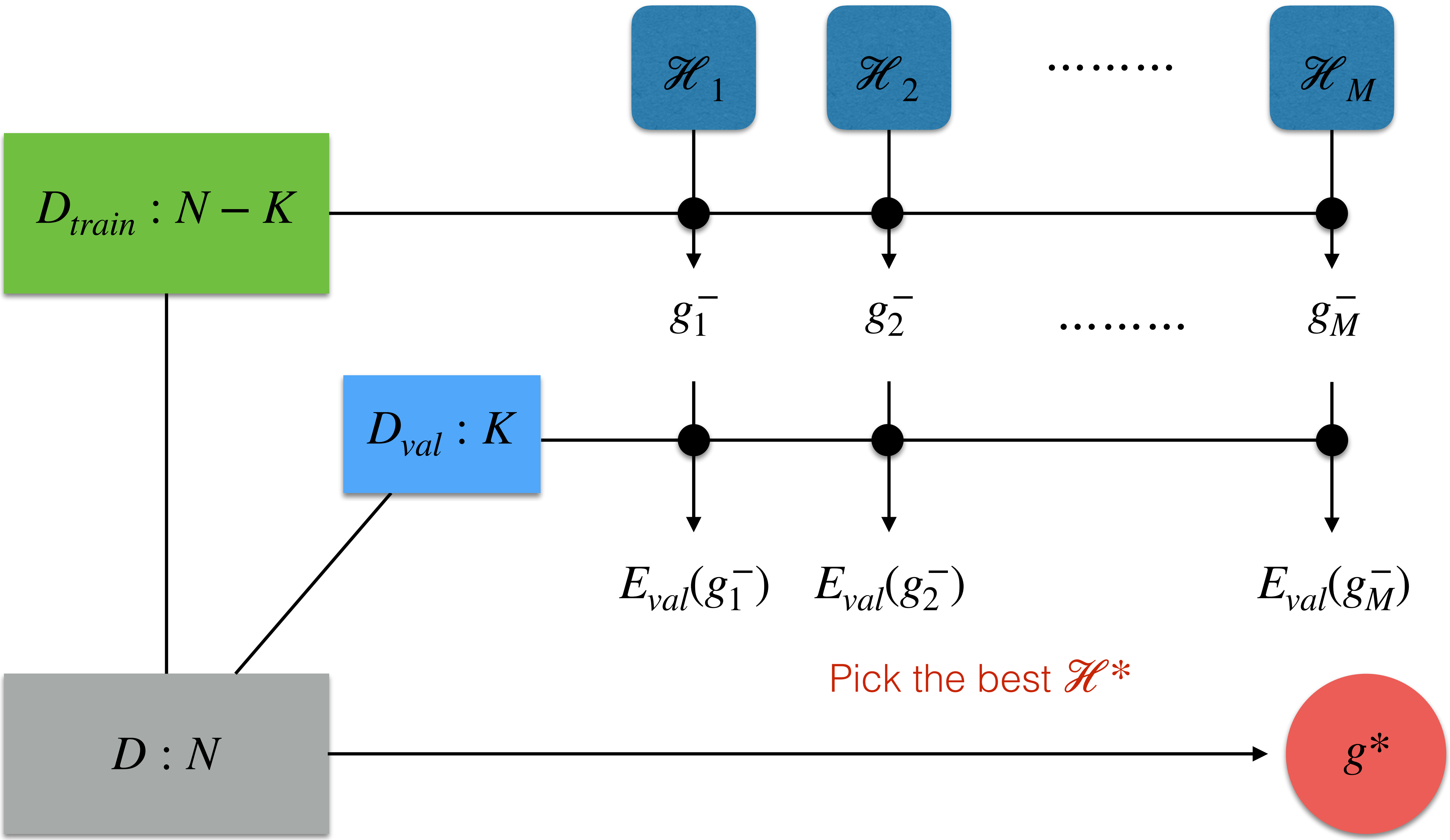


Figure adapted from the book Learning from Data

Cross Validation

$$\underbrace{E_{out}(g) \approx E_{out}(g^{-})}_{\text{Small K}}$$

Cross Validation

$$\overbrace{E_{out}(g) \approx \underbrace{E_{out}(g^-)}_{} }^{}$$

Large K

$$\underbrace{E_{out}(g) \approx E_{out}(g^-)}_{\text{Small K}} \approx E_{val}(g^-)$$

$$E_{out}(g) \approx \overbrace{E_{out}(g^-) \approx E_{val}(g^-)}^{\text{Large K}}$$

Cross Validation

Large K

$$\underbrace{E_{out}(g) \approx \overbrace{E_{out}(g^-)}_{\text{Small K}} \approx E_{val}(g^-)}$$

Leave One Out Analysis

$$D = (x_1, y_1), \ldots (x_{n-1}, y_{n-1}), (x_n, y_n), (x_{n+1}, y_{n+1}), \ldots, (x_N, y_N)$$

Cross Validation

Large K

$$E_{out}(g) \approx E_{out}(g^-) \approx E_{val}(g^-)$$

Small K

Leave One Out Analysis

$$D = (x_1, y_1), \ldots (x_{n-1}, y_{n-1}), \cancel{(x_n, y_n)}, (x_{n+1}, y_{n+1}), \ldots, (x_N, y_N)$$

$$E_{val}(g^-) = e_n = e(g_n^-(x_n), y_n)$$

$$E_{cv} = \frac{1}{N} \sum_{n=1}^{N} e_n$$

Leave One Out Analysis

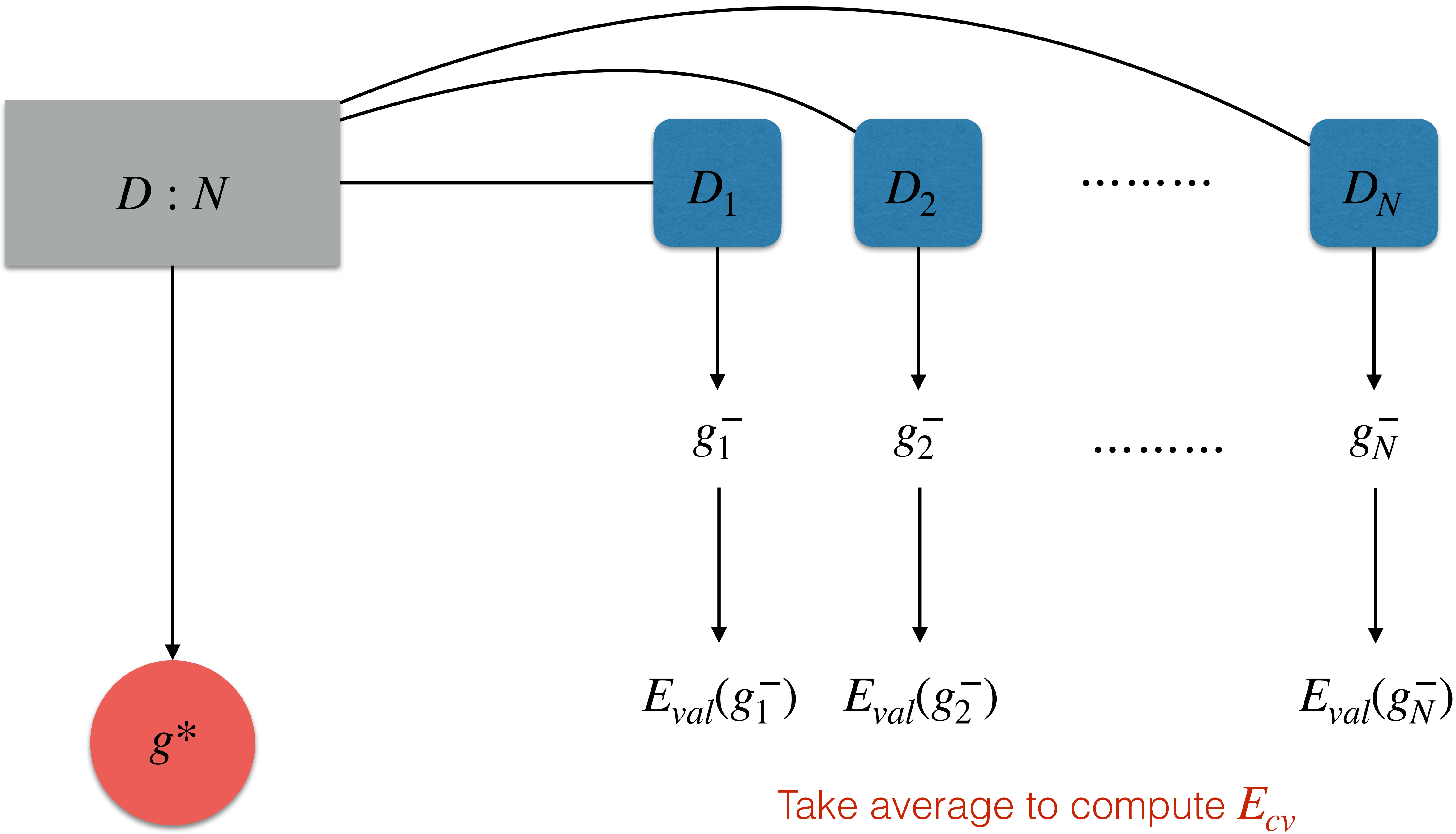
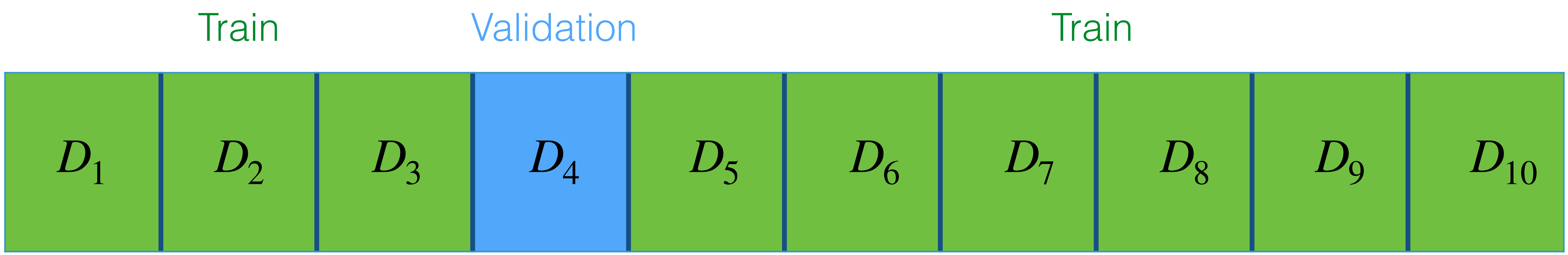


Figure adapted from the book Learning from Data

K-fold Cross Validation



Model Selection with Cross Validation

Define M models by choosing different values of λ: $(\mathscr{H}, \lambda_1), (\mathscr{H}, \lambda_2)\ldots, (\mathscr{H}, \lambda_M)$

For each model $m = 1,2,\ldots,M$ do

Run the cross validation module to get an estimate of the cross validation error $E_{cv}(g^m)$

Pick the model $(\mathscr{H}, \lambda^*)$ with the smallest error

Train the model $(\mathscr{H}, \lambda^*)$ on the entire training set D to obtain the final hypothesis g^{m^*}