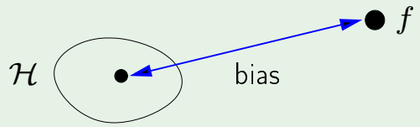
Review of Lecture 8

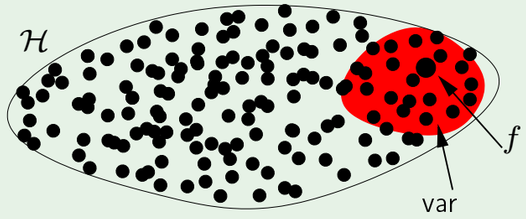
- Bias and variance

Expected value of E_{out} w.r.t. $\mathcal{D}$

$$= \text{bias} + \text{var}$$

The jump from the 'best approximation' from H, which is gbar, to f is characterised by the bias. The jump from the actual final h to gbar, the centroid of the red region, is characterised by the variance. The red region contains all the h we pick based on each possible dataset).





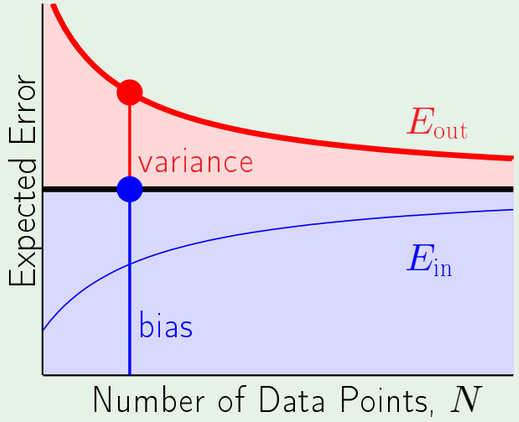
$$g^{(\mathcal{D})}(\mathbf{x}) \rightarrow \bar{g}(\mathbf{x}) \rightarrow f(\mathbf{x})$$

expected value of g w.r.t. D

- Learning curves

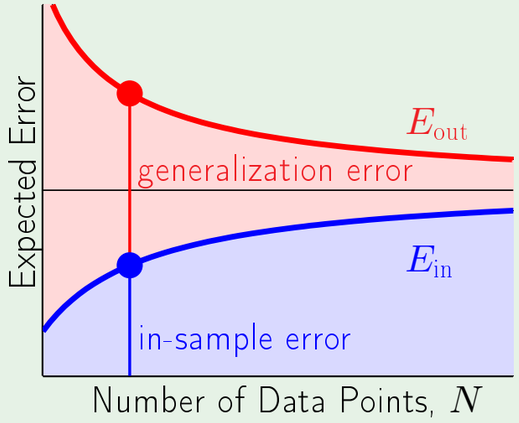
How E_{in} and E_{out} vary with N

B-V:



The bias describes how our best hypothesis gbar compares to the target function - it is therefore a more general measure of the ability of H to model f. The variance then describes the ability to generalize.

VC:



- $N \propto$ "VC dimension"

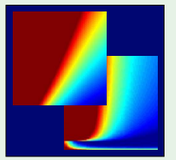
important for practical models in rest of the course

Learning From Data

Yaser S. Abu-Mostafa
California Institute of Technology

Lecture 9: The Linear Model II

Sponsored by Caltech's Provost Office, E&AS Division, and IST • Tuesday, May 1, 2012



Where we are

- Linear classification ✓
- Linear regression ✓
- Logistic regression
- Nonlinear transforms ≈✓

Nonlinear transforms

$$\mathbf{x} = (x_0, x_1, \cdots, x_d) \xrightarrow{\Phi} \mathbf{z} = (z_0, z_1, \cdots\cdots\cdots\cdots, z_{\tilde{d}})$$

The transformation can be quite general; any zi can be an arbitrary non-linear transformation of the entire vector x - computing any formula we want and then assigning to zi (the feature) in the z vector (a point in the feature space). Note the length of z can also be arbitrary - SVM involve infinite dimension feature space. The length of z is only depending on the length of the vector of functions phi.

Each $z_i = \phi_i(\mathbf{x})$ $\qquad$ $\mathbf{z} = \Phi(\mathbf{x})$

Example: $\mathbf{z} = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$

summing with various coefficients, this can model any general second-order surface in the x-space.

Final hypothesis $g(\mathbf{x})$ in $\mathcal{X}$ space:

$$\text{sign}\left(\tilde{\mathbf{w}}^\mathsf{T}\Phi(\mathbf{x})\right) \quad \text{or} \quad \tilde{\mathbf{w}}^\mathsf{T}\Phi(\mathbf{x})$$

The price we pay

in terms of generalization

$$\mathbf{x} = (x_0, x_1, \cdots, x_d) \xrightarrow{\Phi} \mathbf{z} = (z_0, z_1, \cdots\cdots\cdots\cdots, z_{\tilde{d}})$$

$$\downarrow \qquad\qquad\qquad\qquad \downarrow$$

$$\mathbf{w} \qquad\qquad\qquad\qquad \tilde{\mathbf{w}}$$

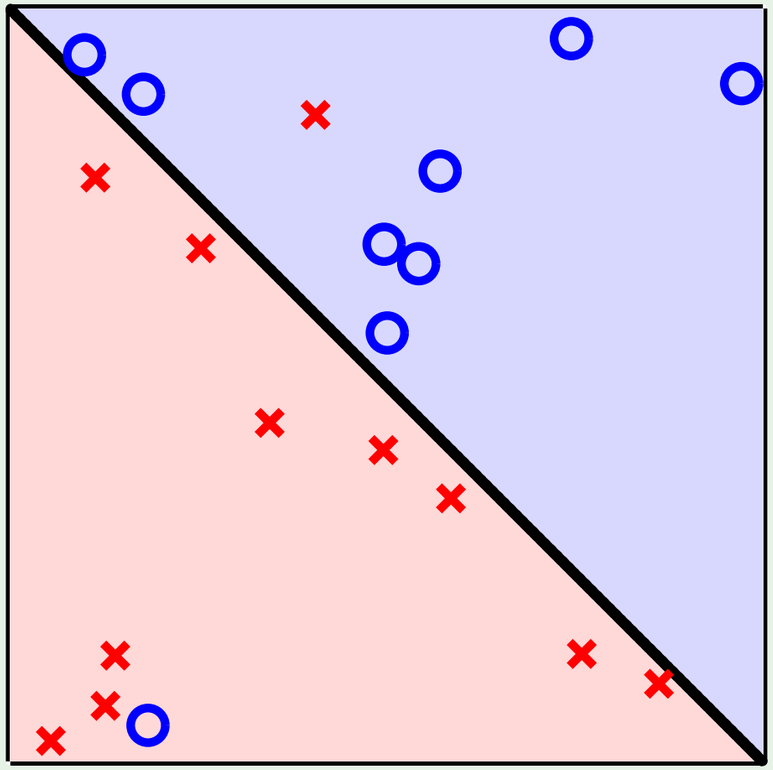
$$d_{\text{VC}} = d + 1 \qquad\qquad d_{\text{VC}} \leq \tilde{d} + 1$$

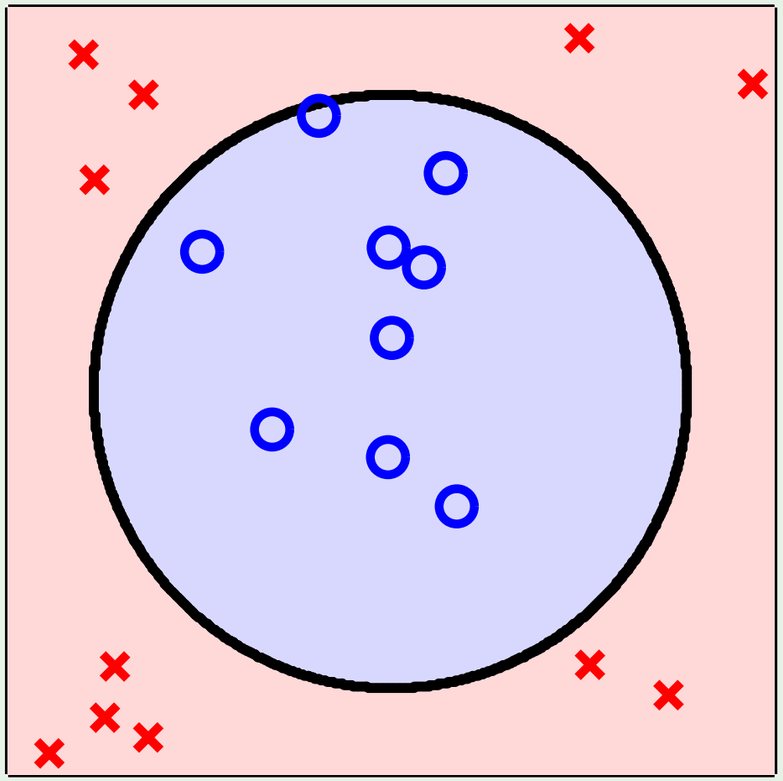
d + 1 free parameters / d.o.f.

inequality since dVC is measured in the x-space; It is possible that there are certain points in the z-space which are impossible to come by via transformations for x-space (i.e. if two coordinates in z are taken to be identical, so same transformation, it reduces the d.o.f. since one of the points is dictated by the value of another). dVC is usually close to d_tilde + 1 though

w_tilde is much longer in general than w (in the quadratic example with x={x0,x1,x2}, w_tilde has length 6 (see last slide)).

Two non-separable cases





First case

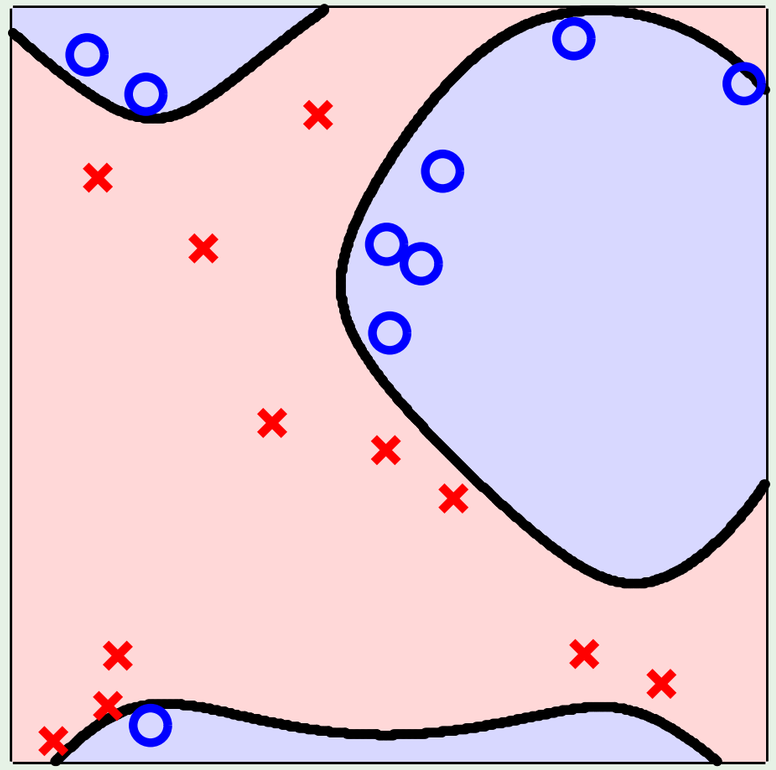
Choices:

1. Use a linear model in $\mathcal{X}$; accept $E_{\text{in}} > 0$

or

2. Insist on $E_{\text{in}} = 0$; go to high-dimensional $\mathcal{Z}$

Here is a straightforward application of the approximation-generalization tradeoff: we go to a more complex model and approximate the data better, but we will generalize worse. So can be better to accept a small training error in order not to use too high a complexity for the hypothesis set.



Second case

$$\mathbf{z} = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$$ general second order surface

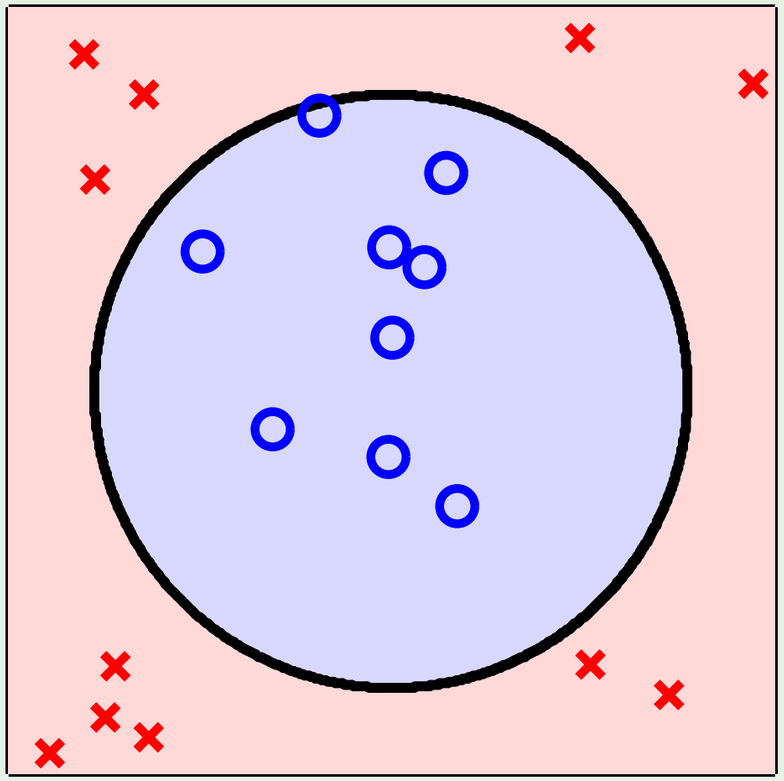
Try to reduce dimensions:

Why not: $\mathbf{z} = (1, x_1^2, x_2^2)$

or better yet: $\mathbf{z} = (1, x_1^2 + x_2^2)$

or even: $\mathbf{z} = (x_1^2 + x_2^2 - 0.6)$

Since using a linear space (say x=1,x1,x2} only involves three weights, while z involves six weights or parameters, we require double the data N to get the same level of performance (not that we have a choice in this case as the linear would not work)



Lesson learned

Looking at the data *before* choosing the model can be hazardous to your E_{out}

The VC inequality can be seen as providing a warranty - if we look at the data before choosing them model, we forfeit the warranty. This is because in the VC inequality we are charged with the VC dimension not of the final (simplified) model like in the previous slide, but with the VC dimension of the entire hypothesis set that was explored in your mind getting there. The person looking at the data acts as a learning algorithm unknowingly. Before looking at the data it may have been decided to use a general 2nd order transform, but looking at the data makes you realize that some coefficients are zero (i.e. that you don't need all of the parameters to learn the data). The hypothesis that was learned was via heirarchecal learning - first you learned, then you passed it onto the algorith to complete the learning. Thus, the effective hypothesis set is the one that was started with. The quantity that describes generalization, the VC dimension, becomes vague since we do not know what the full hypothesis set we explored in the beginning was (however, it is definitely bigger than the VC dimension of the hypothesis set we end up with after snooping.

Data snooping contaminates the data, since it was used to choose the model and can no longer be trusted to reflect the real performance because it has already been used in learning.

Data snooping

With respect to feature selection: did you choose the feature by understanding the problem or by looking at the specific given dataset - the latter is an issue since looking at the data and choosing features involves you learning yourself, so the initial hypothesis set is larger than what we end up with, so VC inequality warranty is forfeit. Note that the warning is not absolute: still thinking that the final H is what will dictate the generalization behavior is where the fallacy lies. Looking at the problem and deriving useful/meaningful features using general understanding rather than the looking at the dataset, we are charged nothing for it.

Logistic regression - Outline

- The model
- Error measure
- Learning algorithm

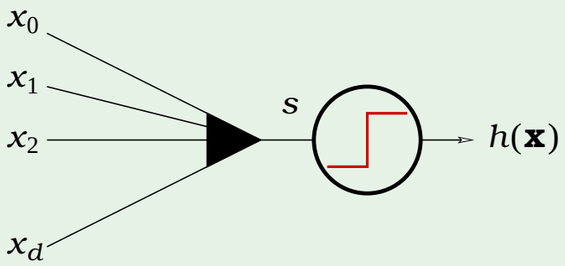
A third linear model

$$s = \sum_{i=0}^{d} w_i x_i$$

sign() and theta() apply non-linearity

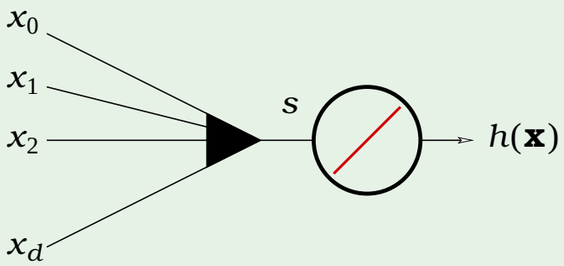
linear classification

$$h(\mathbf{x}) = \text{sign}(s)$$



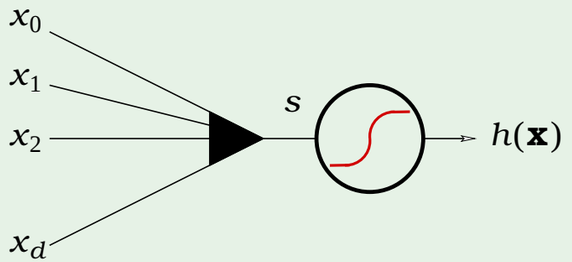
linear regression

$$h(\mathbf{x}) = s$$



logistic regression

$$h(\mathbf{x}) = \theta(s)$$

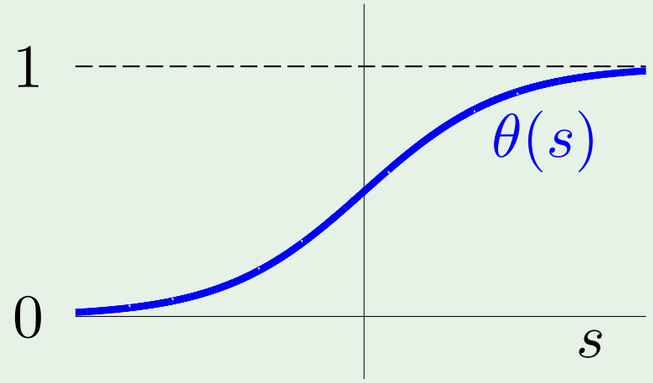


output is treated as a probability, like a combination of the other two models since it returns real values between 0 and 1

The logistic function θ

The formula:

$$\theta(s) = \frac{e^s}{1 + e^s}$$



This formula for a sigmoid is used because it is friendly when we plug it in and get the error measure and optimization - other forms may give us problems

reflects

soft threshold: uncertainty

sigmoid: flattened out 's'

Probability interpretation

$h(\mathbf{x}) = \theta(s)$ is interpreted as a probability

Example. Prediction of heart attacks

Input $\mathbf{x}$: cholesterol level, age, weight, etc.

$\theta(s)$: probability of a heart attack

The signal $s = \mathbf{w}^{\mathsf{T}}\mathbf{x}$ "risk score"

Genuine probability

Even though we produce a probability as output, the data labels (yi) are not themselves probabilities, however these binary output are affected by the probability (or generated by f(x), see below). So we are producing a genuine probability from our model: because the labels are not simply +1 or -1, they have an inherently probabilistic interpretation

Data $(\mathbf{x}, y)$ with binary y, generated by a noisy target:

the probability of a heart attack is the target function itself

Remember, noisy target indicates that two of the same random variable x can produce different outputs y, hence probability distribution P(y | x)

$$P(y \mid \mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{for } y = +1; \\ 1 - f(\mathbf{x}) & \text{for } y = -1. \end{cases}$$

The target $f : \mathbb{R}^d \to [0, 1]$ is the probability

Want to Learn $g(\mathbf{x}) = \theta(\mathbf{w}^\mathsf{T} \mathbf{x}) \approx f(\mathbf{x})$

The only thing under our control is the weights - we change these ro produce different hypotheses. So, our question is: how do we choose w such that g reflects f knowing that f is the way the examples were generated

Error measure

- in this case the error measure has both analytic properties which make it a plausible error measure (i.e. minimizing this error measure means we are genuinely learning well) and it is friendly for the optimizer (easy for the optimizer to optimize)

For each $(\mathbf{x}, y)$, y is generated by probability $f(\mathbf{x})$

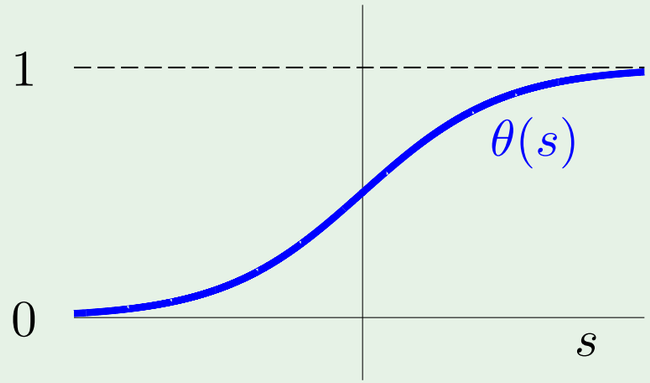
Plausible error measure based on **likelihood**:

If $h = f$, how likely to get y from $\mathbf{x}$?

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

Formula for likelihood

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



Substitute $h(\mathbf{x}) = \theta(\mathbf{w}^\mathsf{T}\mathbf{x})$, noting $\theta(-s) = 1 - \theta(s)$

$$P(y \mid \mathbf{x}) = \theta(y\ \mathbf{w}^\mathsf{T}\mathbf{x})$$

Likelihood of $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ is

$$\prod_{n=1}^{N} P(y_n \mid \mathbf{x}_n) = \prod_{n=1}^{N} \theta(y_n \mathbf{w}^\mathsf{T}\mathbf{x}_n)$$

The same vector w of the hypothesis contributes to all of the terms in the product sum. So there will be a compromise: if w is chosen to favor one example, it will be slightly worse on another etc. so we will have to find a compromise. This compromise of w is likely to reflect that we are catching something for the underlying probability distribution that generated these examples in the first place.

Maximizing the likelihood

Equivalent to

$$\text{Minimize} \qquad -\frac{1}{N}\ln\left(\prod_{n=1}^{N}\theta(y_n\ \mathbf{w}^{\mathsf{T}}\ \mathbf{x}_n)\right)$$

$$= \frac{1}{N}\sum_{n=1}^{N}\ln\left(\frac{1}{\theta(y_n\ \mathbf{w}^{\mathsf{T}}\ \mathbf{x}_n)}\right)$$

Simply divide the other representation of theta by e^s

$$\left[\theta(s) = \frac{1}{1+e^{-s}}\right]$$

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\underbrace{\ln\left(1+e^{-y_n\mathbf{w}^{\mathsf{T}}\mathbf{x}_n}\right)}_{\mathrm{e}\left(h(\mathbf{x}_n),y_n\right)}$$

similar look to more familiar forms of error measure

"cross-entropy" error

where wTxn is the risk factor from before

Logistic regression - Outline

- The model
- Error measure
- Learning algorithm

How to minimize E_{in}

For logistic regression,

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1 + e^{-y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n}\right)$$

⟵ **iterative** solution

since no closed form solution
(like perceptrons)

Compare to linear regression:

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} (\mathbf{w}^{\mathsf{T}} \mathbf{x}_n - y_n)^2$$

⟵ closed-form solution

(pseudo-inverse one-step learning)

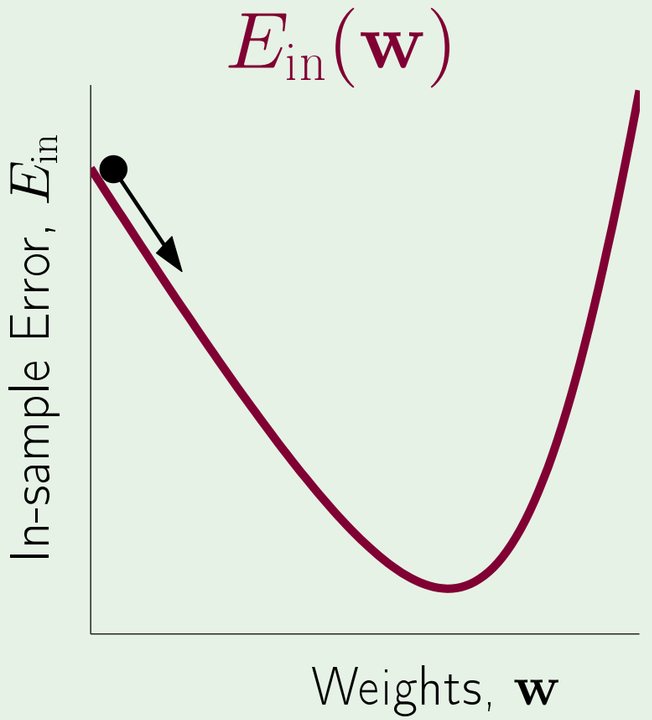
Iterative method: gradient descent

General method for nonlinear optimization

Start at $\mathbf{w}(0)$; take a step along steepest slope

Fixed step size: $\mathbf{w}(1) = \mathbf{w}(0) + \eta\,\hat{\mathbf{v}}$

What is the direction $\hat{\mathbf{v}}$?



N.B. gradient descent requires a twice differentiable surface.

We take the magnitude of the step in w-space (eta) to be fixed and small; we apply local approximations based on calculus (Taylor series) and we know this approximation will apply well if the move is not big - if we go very far, the higher order terms kick in and we cannot be sure the conclusion we got locally will apply.

Formula for the direction $\hat{\mathbf{v}}$

We want delta(Ein) to be as negative as possible

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

1st term of series = derivative * [difference between w(1) and w(0)]. Note that the grad is with respect to the weight vector w

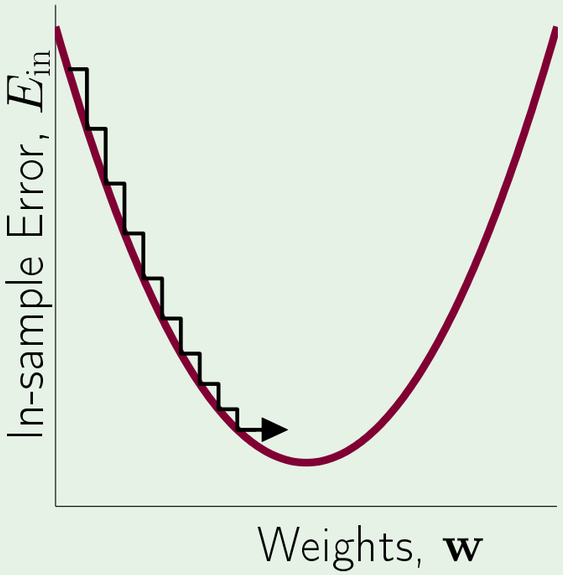
This term is the least possible value we can get for any choice of v_hat (which is when v_hat is opposite grad(Ein))

Since $\hat{\mathbf{v}}$ is a unit vector, it must be normalised:

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

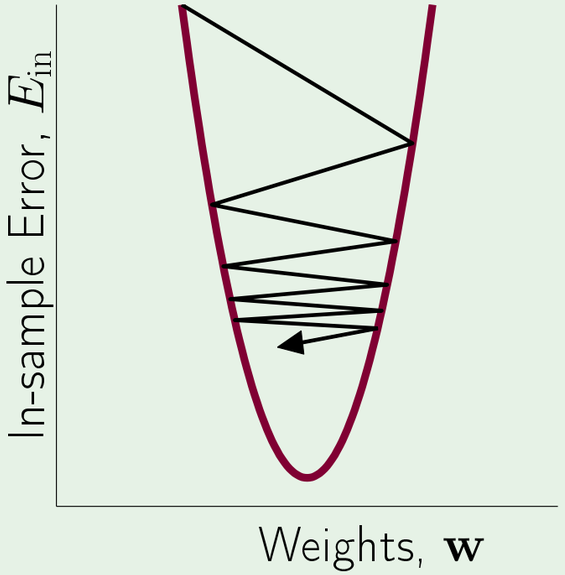
Fixed-size step?

How η affects the algorithm:

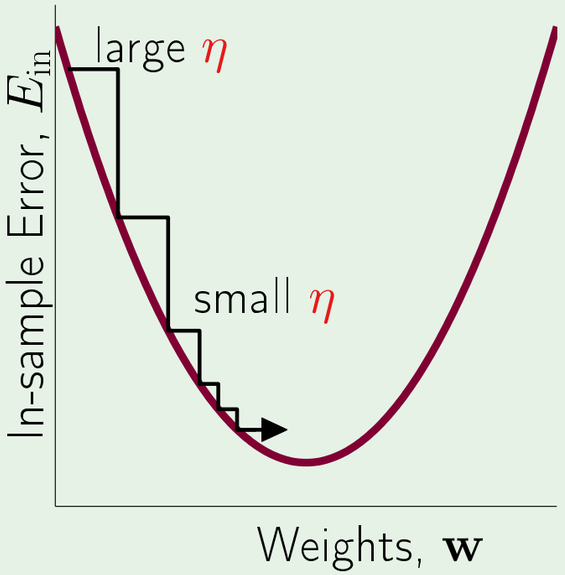


η too small

so it takes a long time to minimize Ein



η too large



variable η – just right

η should increase with the slope

Easy implementation

Instead of

$$\Delta \mathbf{w} = \eta \, \hat{\mathbf{v}}$$

$$= -\eta \, \frac{\nabla E_{\text{in}}(\mathbf{w}(0))}{\|\nabla E_{\text{in}}(\mathbf{w}(0))\|}$$

We want eta proportional to the size of the gradient so it is larger for a larger slope. Since the magnitude of the slope is in the denominator, if eta is proportional it will cancel the slope magnitude and leave a constant of proportionality, which we denote with a purple eta. Now, in the formula for the change in w, we don't have a fixed step anymore but a fixed learning rate: eta.

Have

$$\Delta \mathbf{w} = -\eta \; \nabla E_{\text{in}}(\mathbf{w}(0))$$

Fixed **learning rate** η

Logistic regression algorithm

1: Initialize the weights at $t = 0$ to $\mathbf{w}(0)$

all weights set to 0, so effectively 50:50 probability to start

2: **for** $t = 0, 1, 2, \ldots$ **do**

3: Compute the gradient

$$\nabla E_{\text{in}} = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^{\mathsf{T}}(t) \mathbf{x}_n}}$$

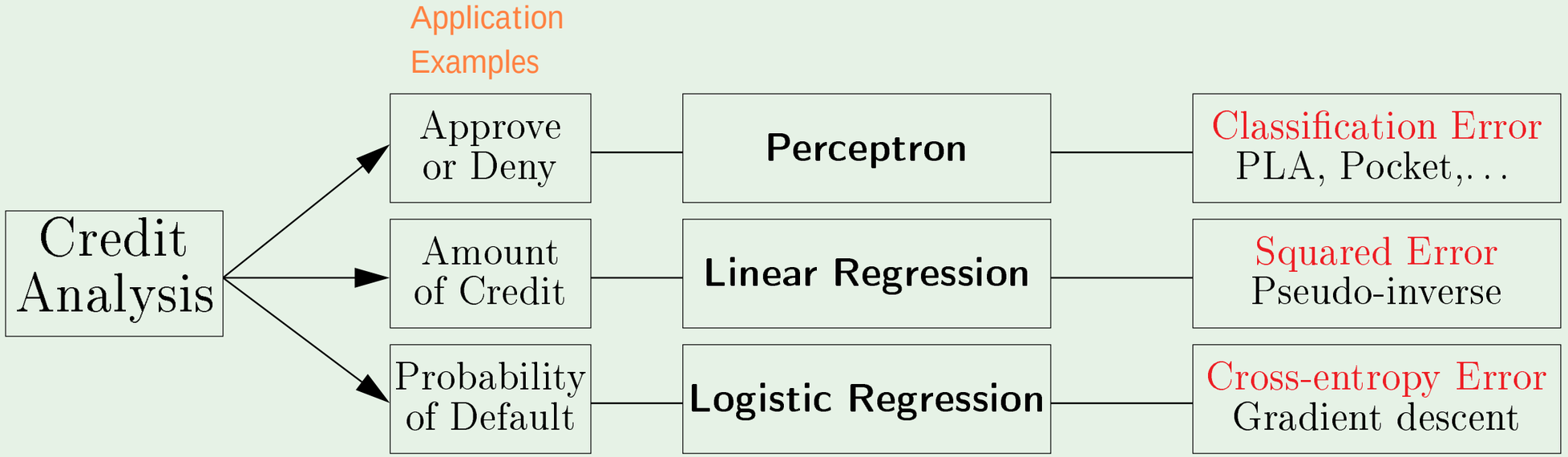
Note: grad taken w.r.t. weight vector

4: Update the weights: $\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla E_{\text{in}}$

5: Iterate to the next step until it is time to stop

6: Return the final weights $\mathbf{w}$

Summary of Linear Models



Learning algorithm resulted from the error measure we chose