Review of Lecture 3

- Linear models use the 'signal':

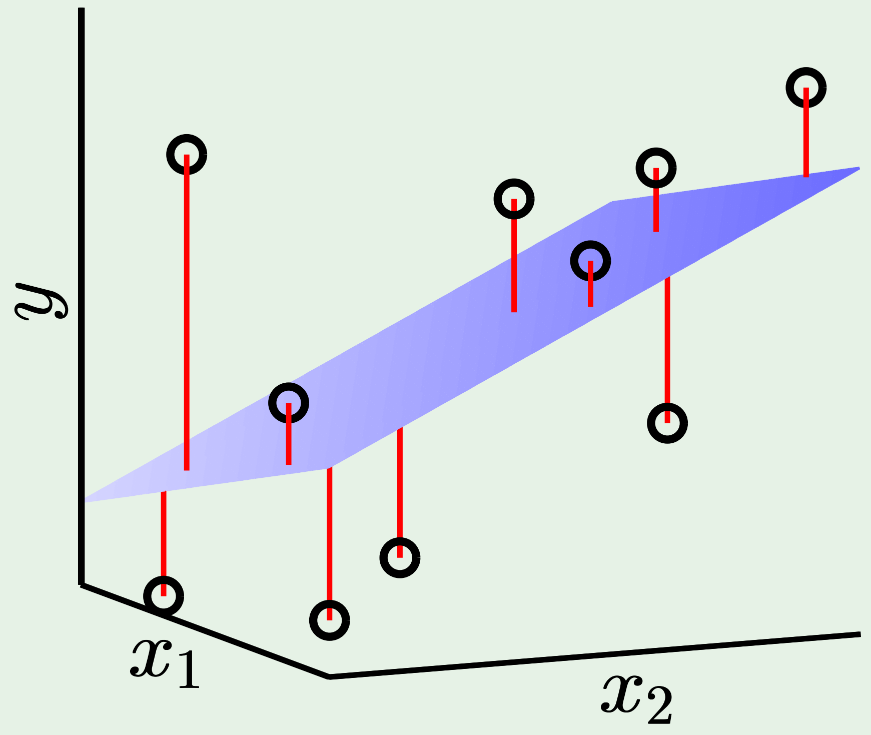
$$\sum_{i=0}^{d} w_i x_i = \mathbf{w}^\intercal \mathbf{x}$$

  - Classification: $h(\mathbf{x}) = \text{sign}(\mathbf{w}^\intercal \mathbf{x})$
  - Regression: $h(\mathbf{x}) = \mathbf{w}^\intercal \mathbf{x}$

- Linear regression algorithm:

$$\mathbf{w} = (\mathrm{X}^\intercal \mathrm{X})^{-1} \mathrm{X}^\intercal \mathbf{y}$$

"one-step learning"



- Nonlinear transformation:
  - $\mathbf{w}^\intercal \mathbf{x}$ is linear in $\mathbf{w}$
  - Any $\mathbf{x} \xrightarrow{\Phi} \mathbf{z}$ preserves this linearity.
  - Example: $(x_1, x_2) \xrightarrow{\Phi} (x_1^2, x_2^2)$

Learning From Data

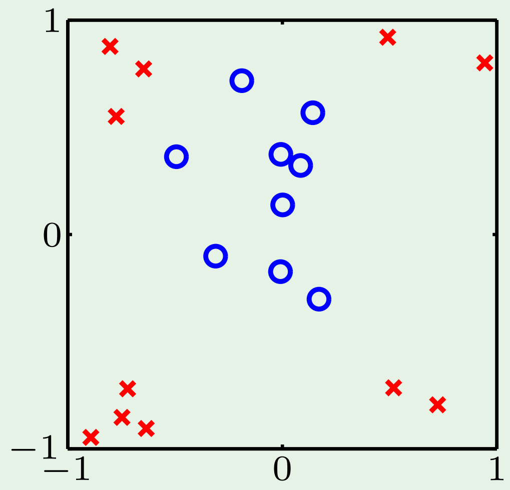
Yaser S. Abu-Mostafa
California Institute of Technology

Lecture 4: Error and Noise

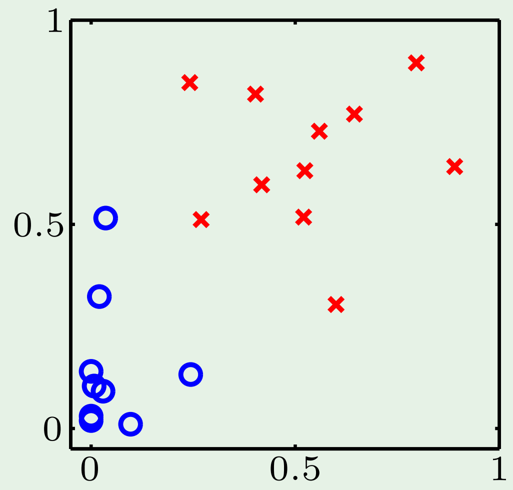
Sponsored by Caltech's Provost Office, E&AS Division, and IST • Thursday, April 12, 2012

Outline

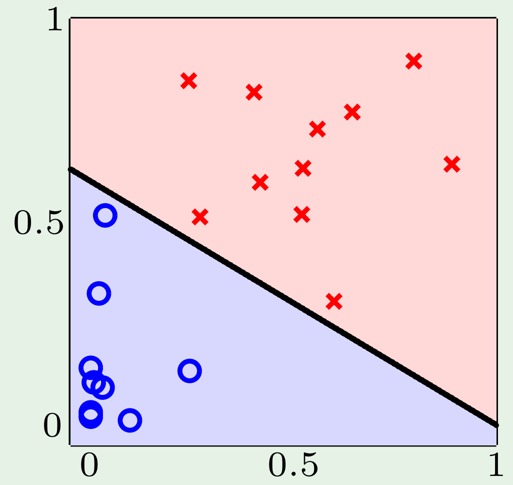
- Nonlinear transformation (continued)
- Error measures
- Noisy targets
- Preamble to the theory


1. Original data
$\mathbf{x}_n \in \mathcal{X}$

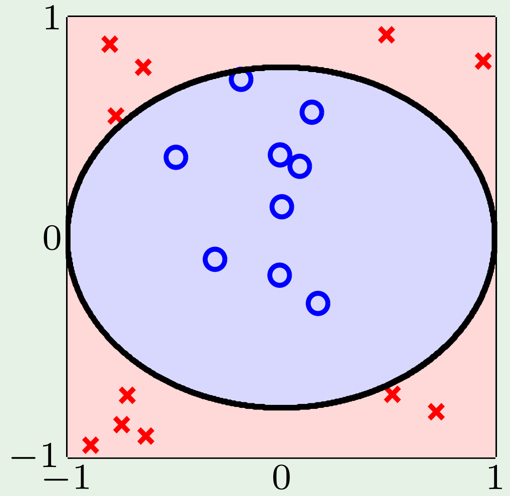
$\xrightarrow{\Phi}$


2. Transform the data
$\mathbf{z}_n = \Phi(\mathbf{x}_n) \in \mathcal{Z}$

$\downarrow$


3. Separate data in $\mathcal{Z}$-space
$\tilde{g}(\mathbf{z}) = \text{sign}(\tilde{\mathbf{w}}^{\mathsf{T}}\mathbf{z})$

$\xleftarrow{`\Phi^{-1}\text{'}}$


4. Classify in $\mathcal{X}$-space
$g(\mathbf{x}) = \tilde{g}(\Phi(\mathbf{x})) = \text{sign}(\tilde{\mathbf{w}}^{\mathsf{T}}\Phi(\mathbf{x}))$

Can transform from 2d (or however many you data has) to however many you require to separate - note however that the wrong transformation can lead to poor generalization

What transforms to what

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

$$\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N \xrightarrow{\Phi} \mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_N$$

$$y_1, y_2, \cdots, y_N \xrightarrow{\Phi} y_1, y_2, \cdots, y_N$$

e.g. in classification, each training example has the same class in Z space

No weights in $\mathcal{X}$ $\quad$ $\tilde{\mathbf{w}} = (w_0, w_1, \cdots\cdots\cdots, w_{\tilde{d}})$

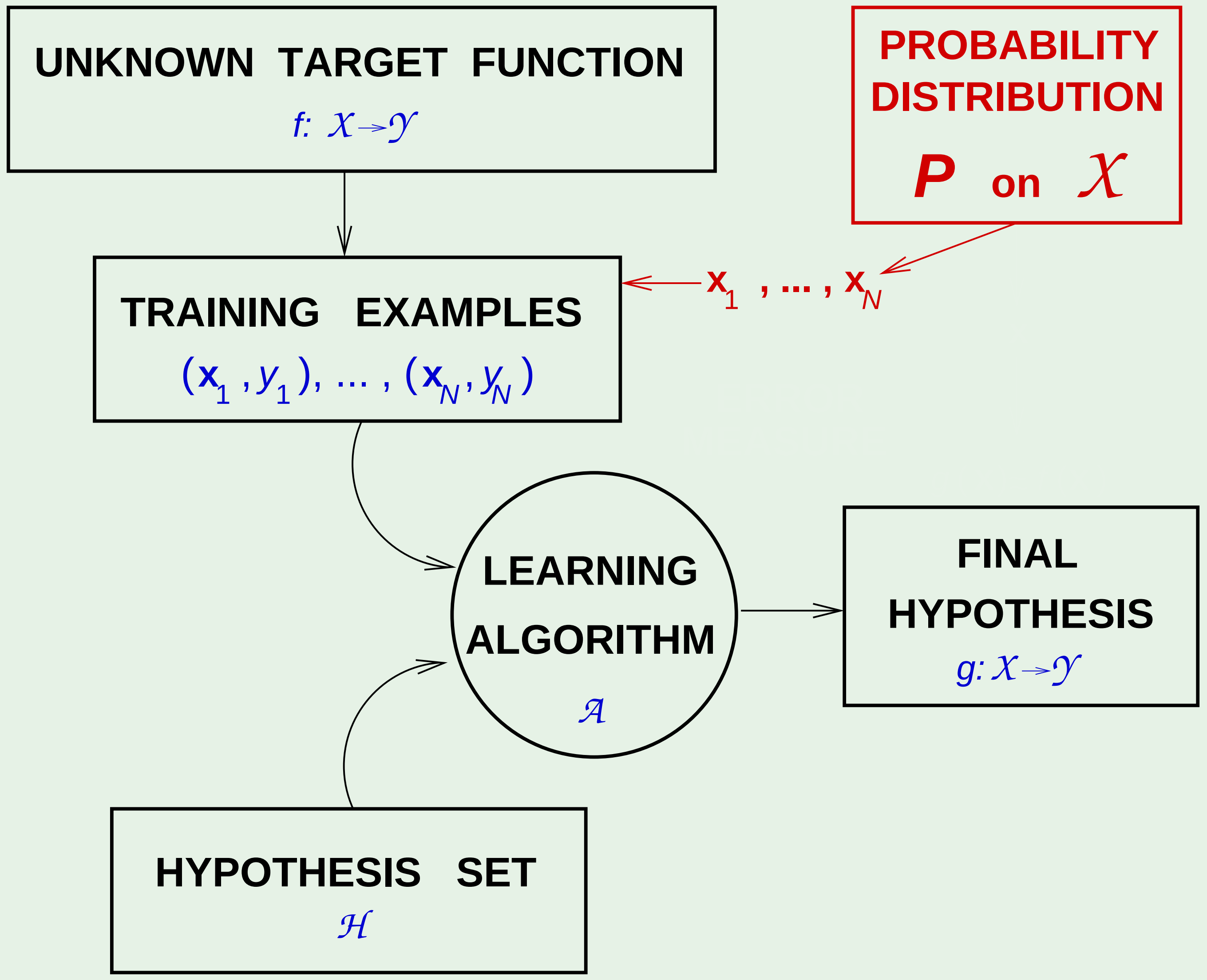
$$g(\mathbf{x}) = \text{sign}(\tilde{\mathbf{w}}^{\mathsf{T}}\Phi(\mathbf{x}))$$

As above, since the y in X and Z are the same (no transformation), points are classified the same in X and Z space

Outline

- Nonlinear transformation (continued)
- Error measures
- Noisy targets
- Preamble to the theory

The learning diagram - where we left it



Error measures

What does "$h \approx f$" mean?

Error measure: $E(h, f)$ technically a functional, rather than a function, since it returns a number based on two functions, rather than variables

Almost always *pointwise definition*: $\mathrm{e}\left(h(\mathbf{x}), f(\mathbf{x})\right)$

Examples:

Squared error: $\mathrm{e}\left(h(\mathbf{x}), f(\mathbf{x})\right) = \left(h(\mathbf{x}) - f(\mathbf{x})\right)^2$

Binary error: $\mathrm{e}\left(h(\mathbf{x}), f(\mathbf{x})\right) = [\![h(\mathbf{x}) \neq f(\mathbf{x})]\!]$ Again, [[...]] returns 1 if true, 0 if false

From pointwise to overall

Overall error $E(h, f)$ = average of pointwise errors $\mathrm{e}\left(h(\mathbf{x}), f(\mathbf{x})\right)$.

In-sample error:

$$E_{\text{in}}(h) = \frac{1}{N} \sum_{n=1}^{N} \mathrm{e}\left(h(\mathbf{x}_n), f(\mathbf{x}_n)\right)$$

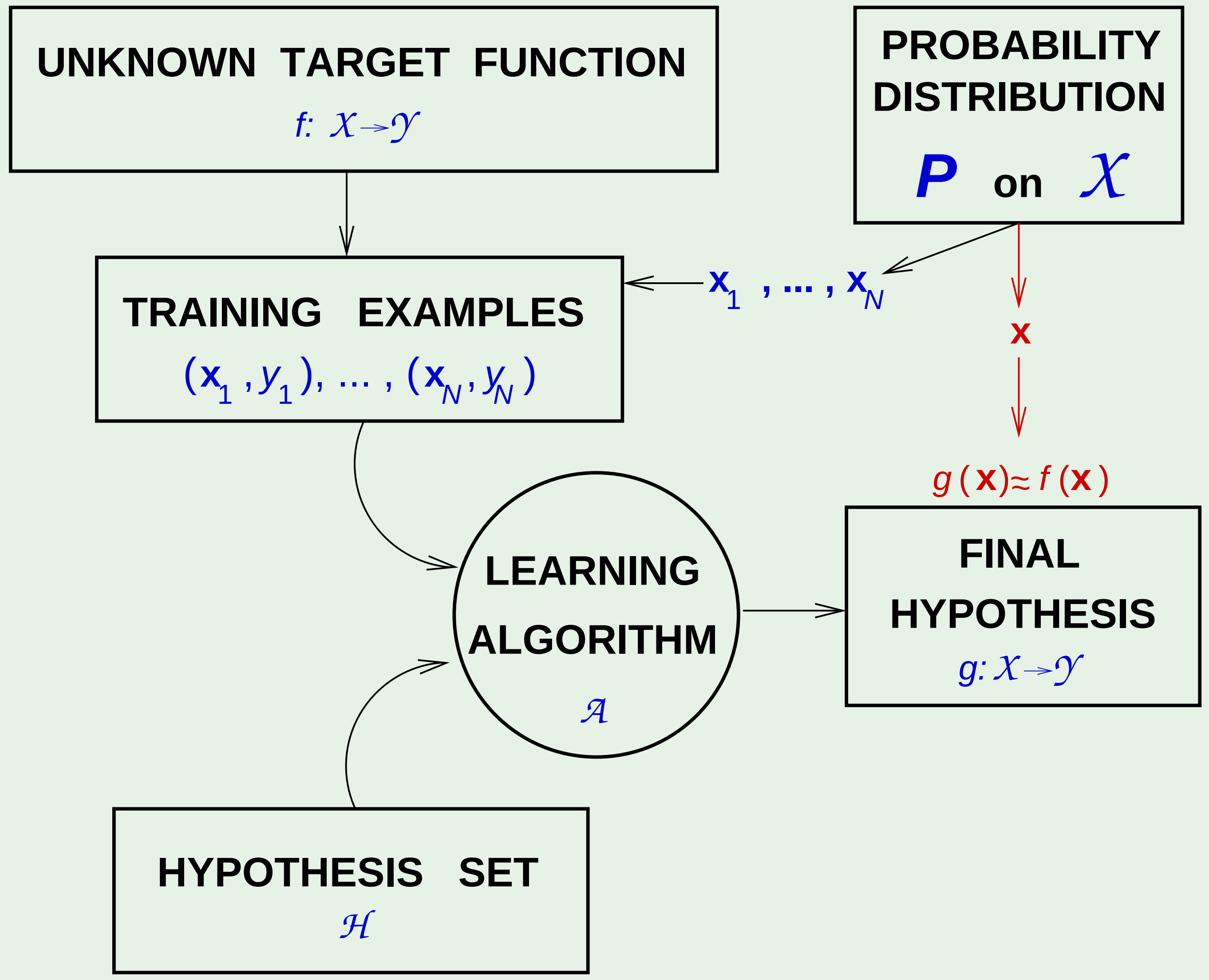
Out-of-sample error:

represents error over the whole space (hence expectation value)

$$E_{\text{out}}(h) = \mathbb{E}_{\mathbf{x}}\left[\mathrm{e}\left(h(\mathbf{x}), f(\mathbf{x})\right)\right]$$

expectation value with respect to x, where x is a general point in space - e.g. binary error in this formula gives the probability of error overall

The learning diagram - with pointwise error


UNKNOWN TARGET FUNCTION
$f: \mathcal{X} \to \mathcal{Y}$
PROBABILITY DISTRIBUTION
P on $\mathcal{X}$
$\mathbf{x}_1, \ldots, \mathbf{x}_N$
$\mathbf{x}$
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
$g(\mathbf{x}) \approx f(\mathbf{x})$
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
$g: \mathcal{X} \to \mathcal{Y}$
HYPOTHESIS SET
$\mathcal{H}$

So when you test the system you trained with a certain probability distribution (for the training data), you must test with points drawn from the same probability distribution (required to invoke Hoeffding (or the counterpart of Hoeffding for more elaborate functions) - so training and test data must be drawn from same P.

How to choose the error measure

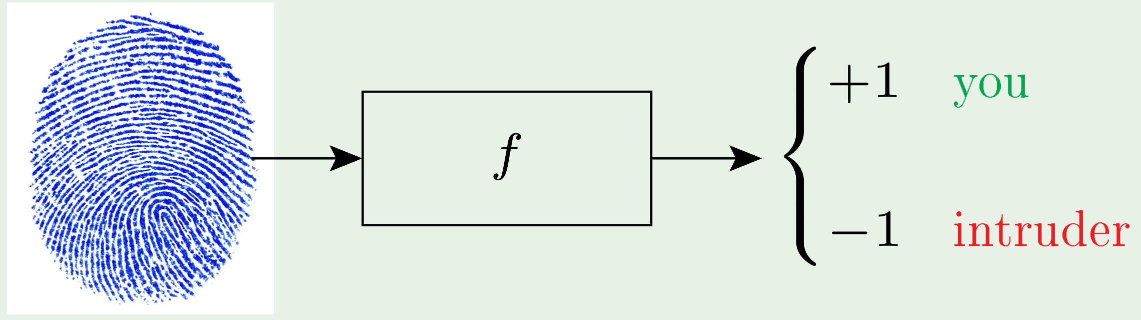
Fingerprint verification:

Two types of error:

false accept and *false reject*

How do we penalize each type?

		f	
		$+1$	-1
h	$+1$	no error	*false accept*
	-1	*false reject*	no error

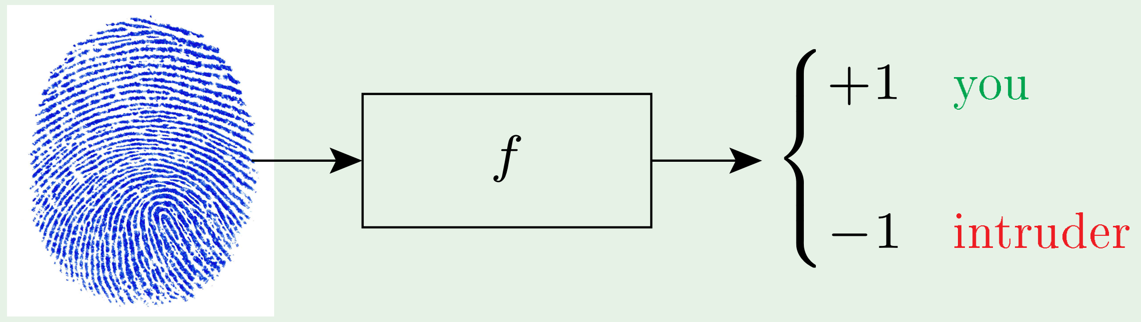


The error measure - for supermarkets

Supermarket verifies fingerprint for discounts

False reject is costly; customer gets annoyed!

False accept is minor; gave away a discount and intruder left their fingerprint ☺



		f	
		$+1$	-1
h	$+1$	0	1
	-1	10	0

The error measure - for the CIA

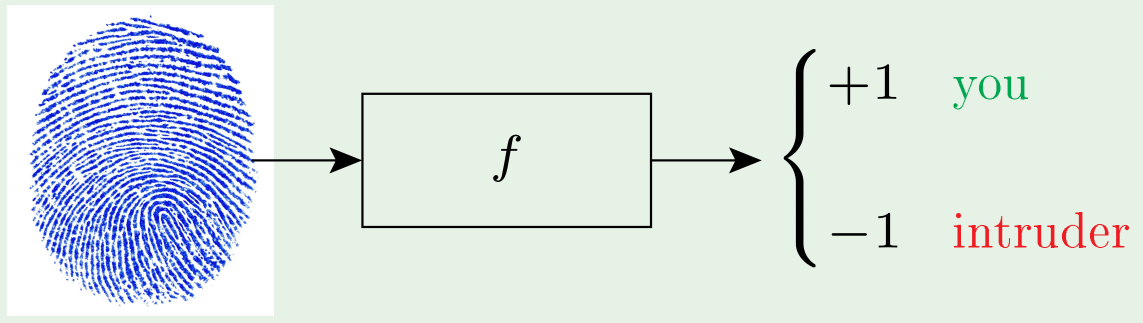
CIA verifies fingerprint for security

False accept is a disaster!

False reject can be tolerated

Try again; you are an employee ☺

$$\begin{array}{cc|cc} & & \multicolumn{2}{c}{f} \\ & & +1 & -1 \\ \hline h & +1 & 0 & 1000 \\ & -1 & 1 & 0 \end{array}$$



Take-home lesson

From the previous two examples, we see that the error measure is different between two application domains for exactly the same machine learning system (same training data, same target function) - the actual values involved could be determined by assessing the cost of a false accept and false reject to the application domain/user.

The error measure should be <u>specified by the user</u>.

Not always possible. Alternatives:

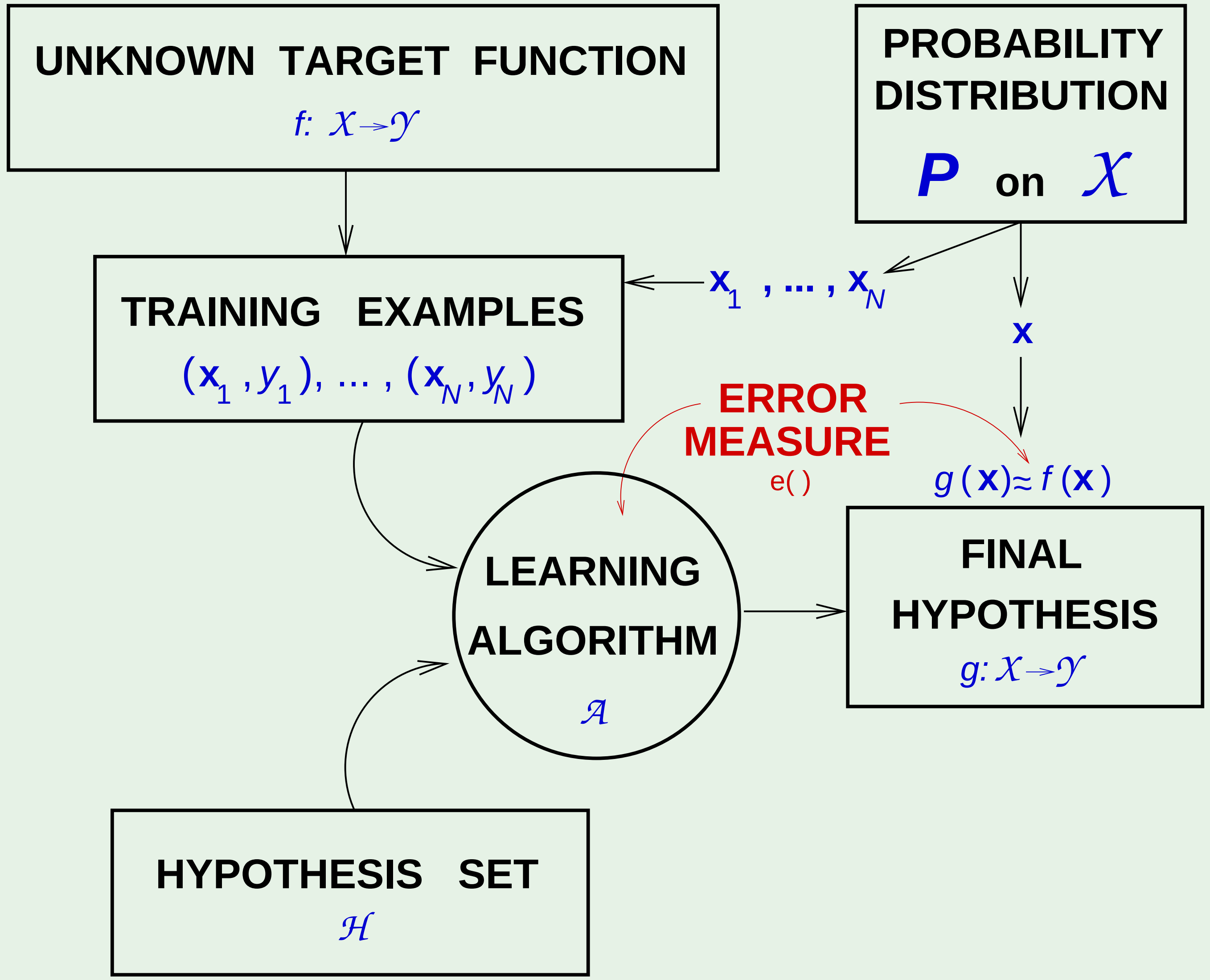
Plausible measures: squared error $\equiv$ Gaussian noise

based on conceptual appeal/merit - e.g. squared error is preferred in general over absolute difference since squared error is optimized over a smooth parabola (has preferable properties) while absolute error has a vertex/discontinuity so becomes a combinatiroail optimization instead of a smooth function). Note that if the user specifies that you must minimize the absolute function, it can be worked with. If you are making an analytic choice though, you should pick the friendlier option (in terms of the concept or the optimization).

Friendly measures: closed-form solution, convex optimization

practical appeal/easy to use - e.g. we used the least-squared error measure in linear regression, which allowed us to derive a closed-form solution for the calculating the weights in the one-step linear regression learning algorithm, similarly those that allow convex optimization techniques in the determination of weights can be relatively easily solved.

The learning diagram - with error measure


UNKNOWN TARGET FUNCTION
$f: \mathcal{X} \rightarrow \mathcal{Y}$
PROBABILITY DISTRIBUTION
P on $\mathcal{X}$
$\mathbf{x}_1, \ldots, \mathbf{x}_N$
$\mathbf{x}$
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
ERROR MEASURE
e( )
$g(\mathbf{x}) \approx f(\mathbf{x})$
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
$g: \mathcal{X} \rightarrow \mathcal{Y}$
HYPOTHESIS SET
$\mathcal{H}$

Noisy targets

The 'target function' is not always a *function*

Consider the credit-card approval:

age	23 years
annual salary	\$30,000
years in residence	1 year
years in job	1 year
current debt	\$15,000
$\cdots$	$\cdots$

can end up exhibiting

two 'identical' customers $\longrightarrow$ two different behaviors

- so essentially we have one data point mapping to different outputs, so not a function (def: returns a unique value for all points in the domain)

Target 'distribution'

Instead of $y = f(\mathbf{x})$, we use target *distribution*:

$$P(y \mid \mathbf{x})$$

- some y's are more likely for a given x, instead of there being only one y for a given x (in a function)

$(\mathbf{x}, y)$ is now generated by the joint distribution:

$$P(\mathbf{x})P(y \mid \mathbf{x})$$

x used to be generated by the input probability distribution, but now instead of y being deterministic after you have generated x, y is also probabilistic generated by the above distribution. Now (x,y) can be treated as a pair generated by the joint distribution, P(x)P(y | x) (assuming independence).

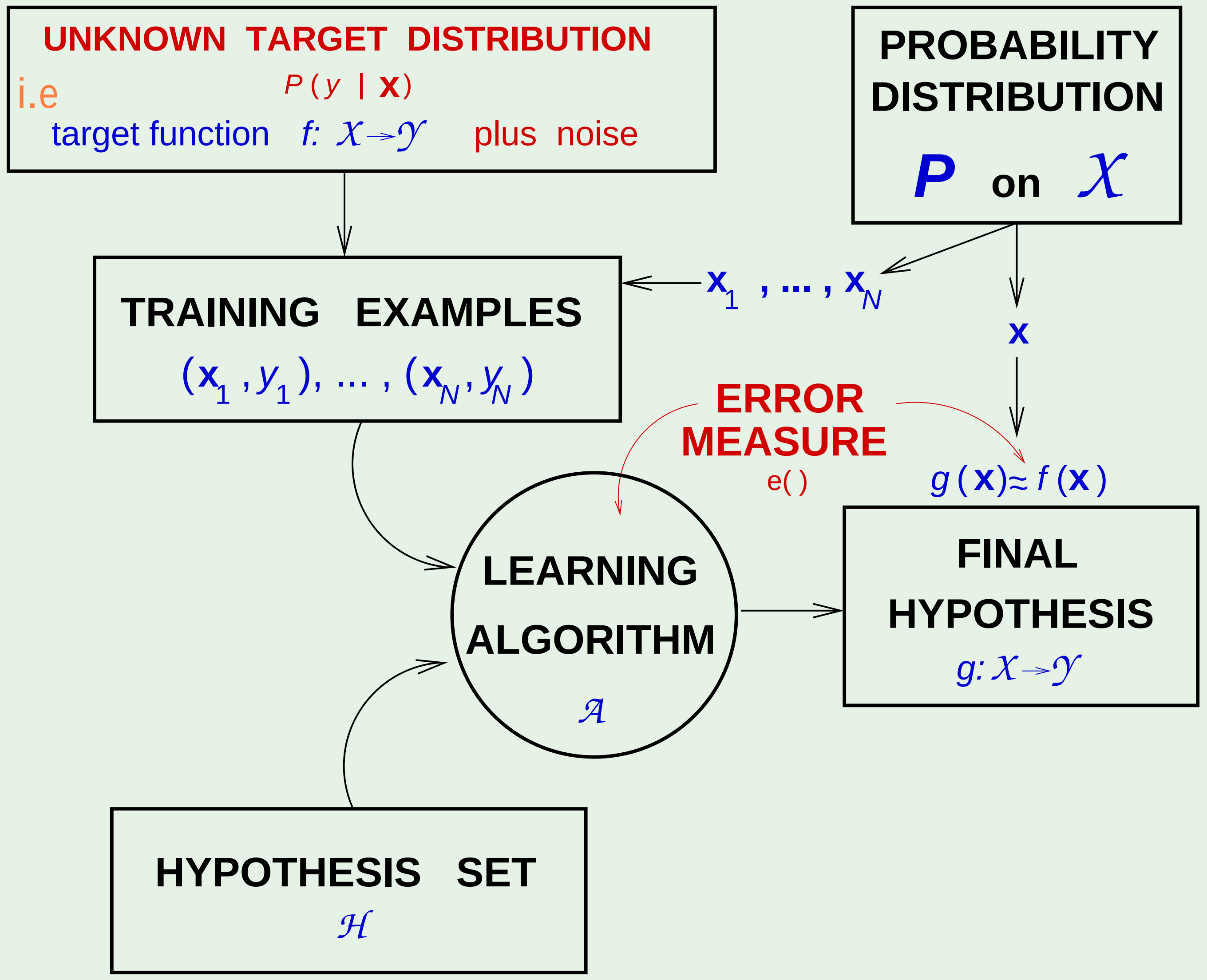
Noisy target = deterministic target $f(\mathbf{x}) = \mathbb{E}(y|\mathbf{x})$ plus noise $y - f(\mathbf{x})$

pure noise, so the average should be around zero

Deterministic target is a special case of noisy target:

$$P(y \mid \mathbf{x}) \text{ is zero except for } y = f(\mathbf{x})$$

The learning diagram - including noisy target


UNKNOWN TARGET DISTRIBUTION
$P(y \mid \mathbf{x})$
i.e
target function $f: \mathcal{X} \to \mathcal{Y}$ plus noise
PROBABILITY DISTRIBUTION
P on $\mathcal{X}$
$\mathbf{x}_1, \ldots, \mathbf{x}_N$
$\mathbf{x}$
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
ERROR MEASURE
e( )
$g(\mathbf{x}) \approx f(\mathbf{x})$
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
$g: \mathcal{X} \to \mathcal{Y}$
HYPOTHESIS SET
$\mathcal{H}$

Distinction between $P(y|\mathbf{x})$ and $P(\mathbf{x})$

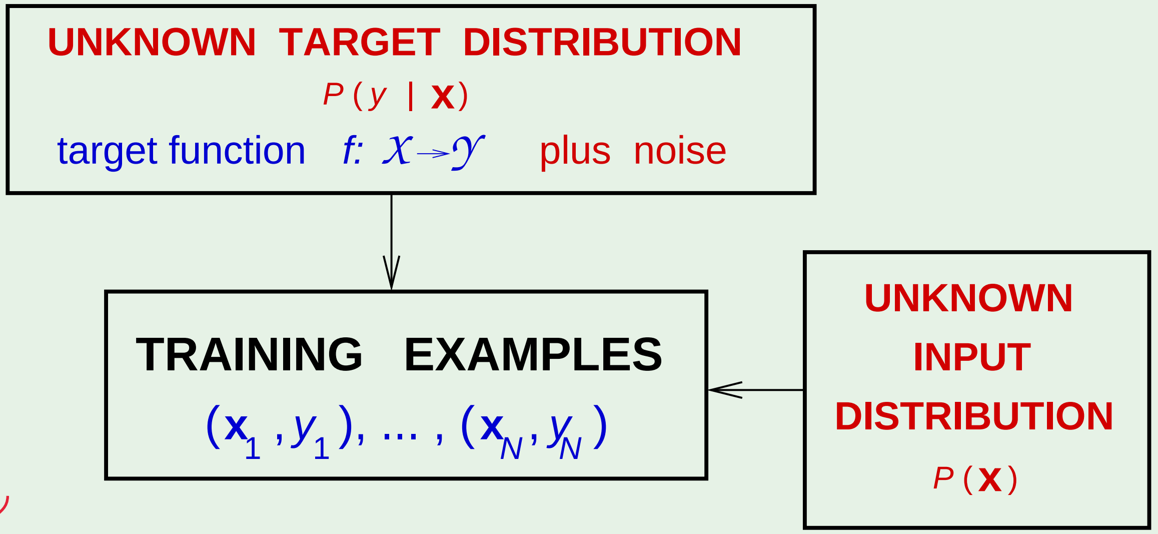
Both convey probabilistic aspects of $\mathbf{x}$ and y

The target distribution $P(y \mid \mathbf{x})$

is what we are trying to learn

The input distribution $P(\mathbf{x})$

quantifies relative importance of $\mathbf{x}$


UNKNOWN TARGET DISTRIBUTION
$P(y \mid \mathbf{x})$
target function $f: \mathcal{X} \to \mathcal{Y}$ plus noise
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
UNKNOWN INPUT DISTRIBUTION
$P(\mathbf{x})$

In the credit approval example: the target distribution is the probability of credit worthiness given the input (e.g. salary) - decide on the risk of defaulting and assign output +1 (approve credit) with probability 0.9 and reject with 0.1 - this is what we are trying to learn. The input distribution only tells us the distribution of solaries in the general population. So even though P(x) matters, in the instance that P(x) is skewed to the right (higher salaries) and high salaries lead to good credit worthiness. This model will be tested largely in the comfortable region of high salaries, so alot of approved credit with little error. However, if the mass of probability is around the borderline cases, the system will perform worse categorizing these borderline cases. So P(x) does give the weights that will finally grade the hypothesis but we are not trying to learn P(x)

target

used to generate training+test samples

Merging $P(\mathbf{x})P(y|\mathbf{x})$ as $P(\mathbf{x}, y)$

mixes the two concepts

Outline

- Nonlinear transformation (continued)
- Error measures
- Noisy targets
- Preamble to the theory

What we know so far

by relying on the fact that

Learning is feasible. It is likely that

$$E_{\text{out}}(g) \approx E_{\text{in}}(g)$$

(which represents generalization - Ein is a proxy for Eout which we do not know)

Is this learning?

We need $g \approx f$, which means

$$E_{\text{out}}(g) \approx 0$$

(which means we learnt well)

The 2 questions of learning

$E_{\text{out}}(g) \approx 0$ is achieved through:

$$\underbrace{E_{\text{out}}(g) \approx E_{\text{in}}(g)}_{\text{Lecture 2}} \qquad \text{and} \qquad \underbrace{E_{\text{in}}(g) \approx 0}_{\text{Lecture 3}}$$

by Hoeffding

We know Ein and can use a learning algorithm (like linear regression) to make it as small as possible

Learning is thus split into 2 questions:

1. Can we make sure that $E_{\text{out}}(g)$ is close enough to $E_{\text{in}}(g)$?

2. Can we make $E_{\text{in}}(g)$ small enough?

Theoretical question - over next 4 lectures

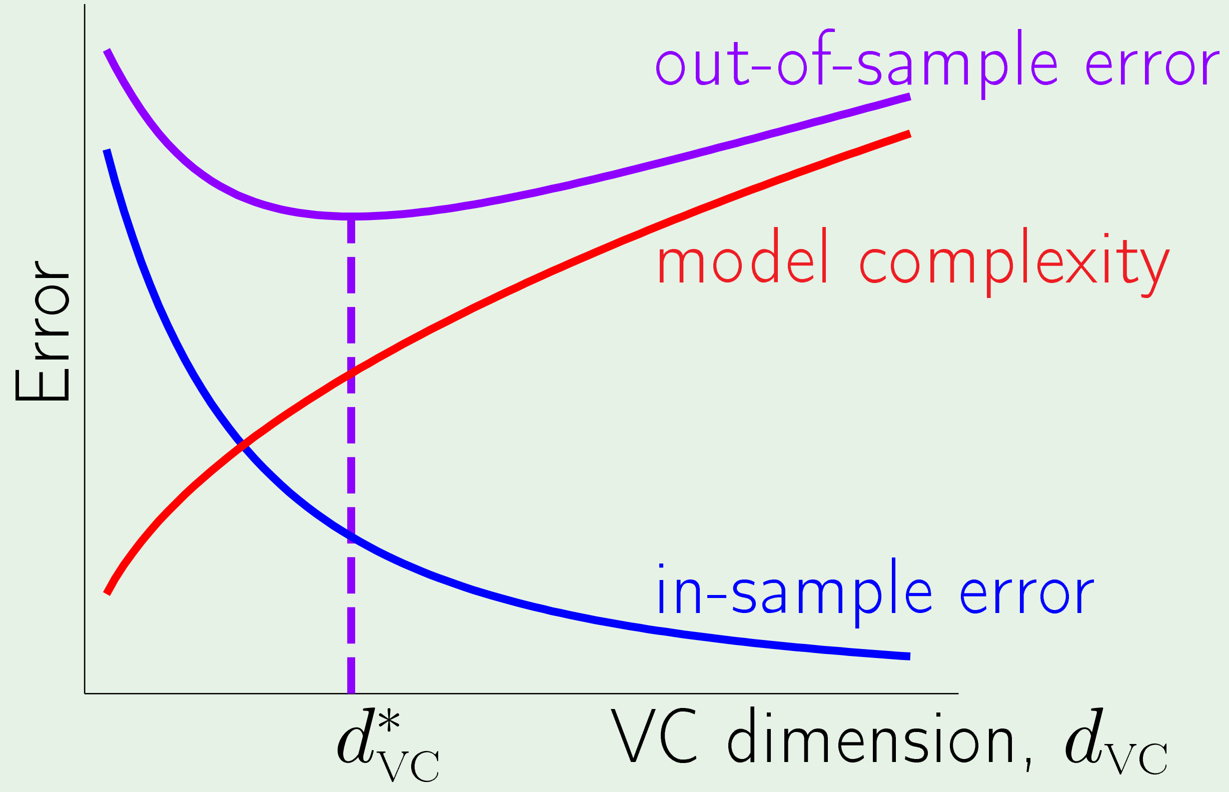
Practical question - over 8 lectures after Q1

What the theory will achieve

Characterizing the feasibility of learning for infinite M

Characterizing the tradeoff:

Model complexity	$\uparrow$	E_{in}	$\downarrow$
Model complexity	$\uparrow$	$E_{\text{out}} - E_{\text{in}}$	$\uparrow$


out-of-sample error
model complexity
in-sample error
Error
d^*_{VC}
VC dimension, d_{VC}