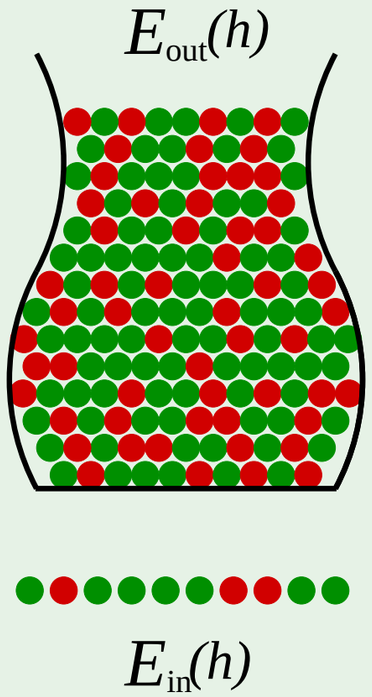
Review of Lecture 2

Is Learning feasible?

Yes, in a **probabilistic** sense.



$$\mathbb{P}\left[\ |E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\ \right] \ \leq\ 2e^{-2\epsilon^2 N}$$

Since g has to be one of $h_1, h_2, \cdots, h_M$, we conclude that

If:

$$|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon$$

Then:

$$|E_{\text{in}}(h_1) - E_{\text{out}}(h_1)| > \epsilon \quad \text{or}$$
$$|E_{\text{in}}(h_2) - E_{\text{out}}(h_2)| > \epsilon \quad \text{or}$$
$$\cdots$$
$$|E_{\text{in}}(h_M) - E_{\text{out}}(h_M)| > \epsilon$$

The problem with having multiple hypotheses is that there is a possible scenario where the probability of a hypothesis generalising badly (0.5% for h1, 0.5% for h2, etc.) accumulates and we end up with a significant probability that one of the hypotheses is bad. If we are unlucky that our final chosen hypothesis is this hypothesis, Ein will not track Eout for this chosen hypothesis. We use this to say that if g generalises badly, atleast one of the hypotheses we tested must also be bad, hence the if-then statement. Applying the union bound we can simplify this inequality - eventually ending up with the M factor.

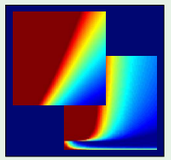
This gives us an added M factor.

Learning From Data

Yaser S. Abu-Mostafa
California Institute of Technology

Lecture 3: **Linear Models I**

Sponsored by Caltech's Provost Office, E&AS Division, and IST • Tuesday, April 10, 2012



Outline

- Input representation
- Linear Classification
- Linear Regression
- Nonlinear Transformation

A real data set

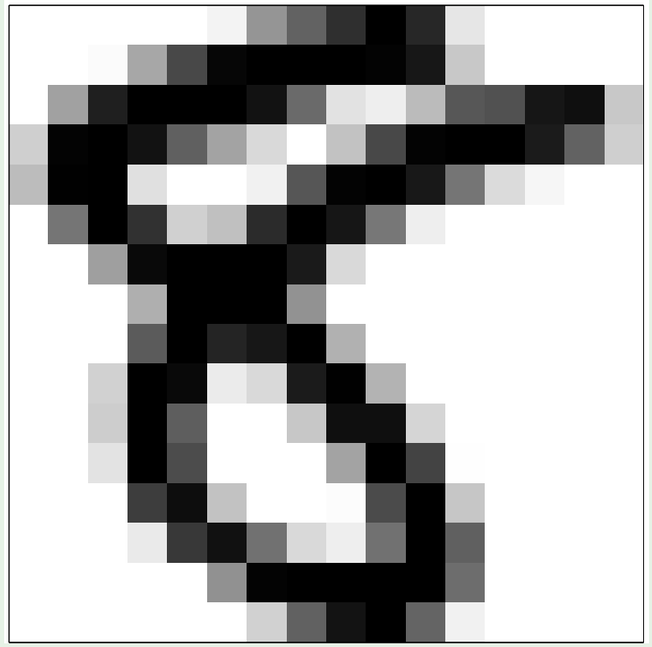
Input representation

'raw' input $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_{256})$

linear model: $(w_0, w_1, w_2, \cdots, w_{256})$

If there are too many parameters, the model is unlikely to generalize well, so need to extract useful information (ideally when we do feature extraction, we atleast remove as much unnecessary information as useful information.)

Features: Extract useful information, e.g.,

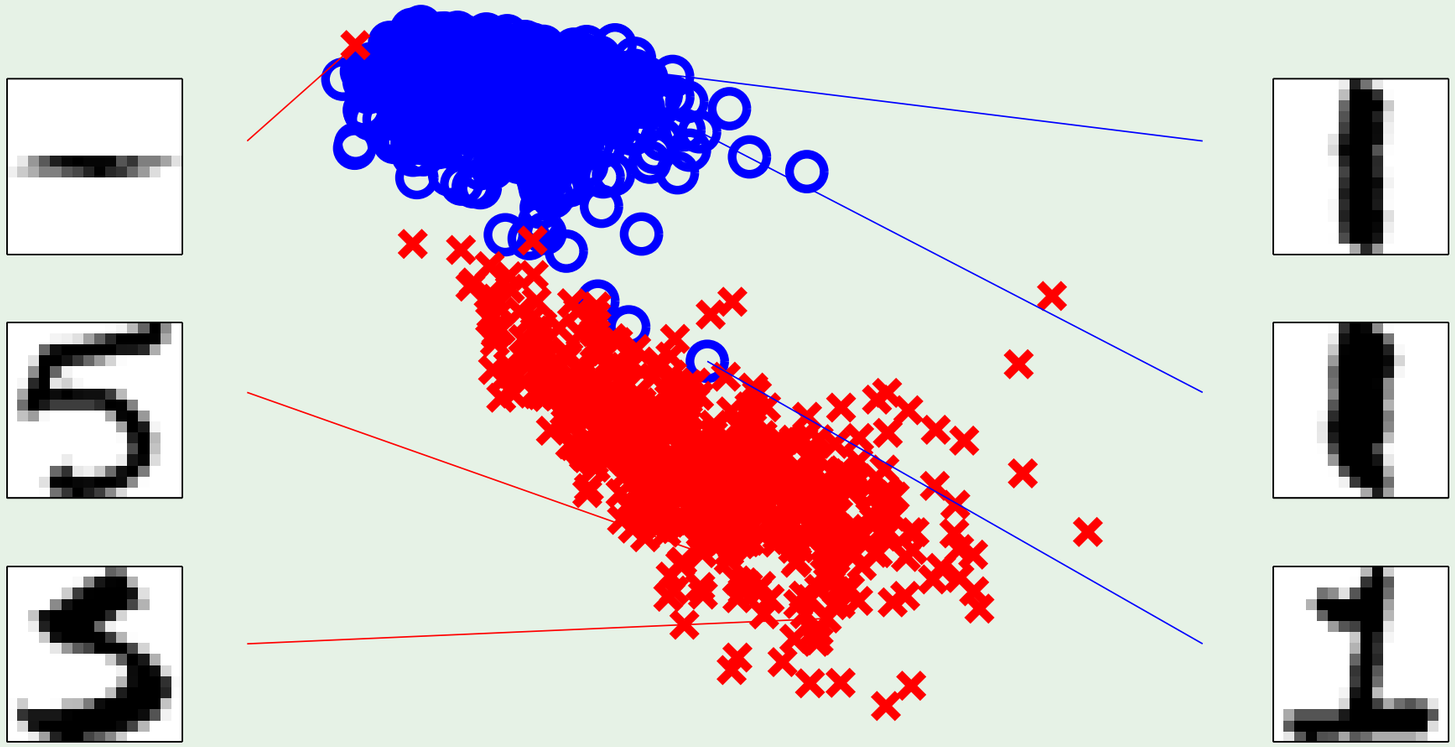


intensity and symmetry $\mathbf{x} = (x_0, x_1, x_2)$

linear model: (w_0, w_1, w_2)

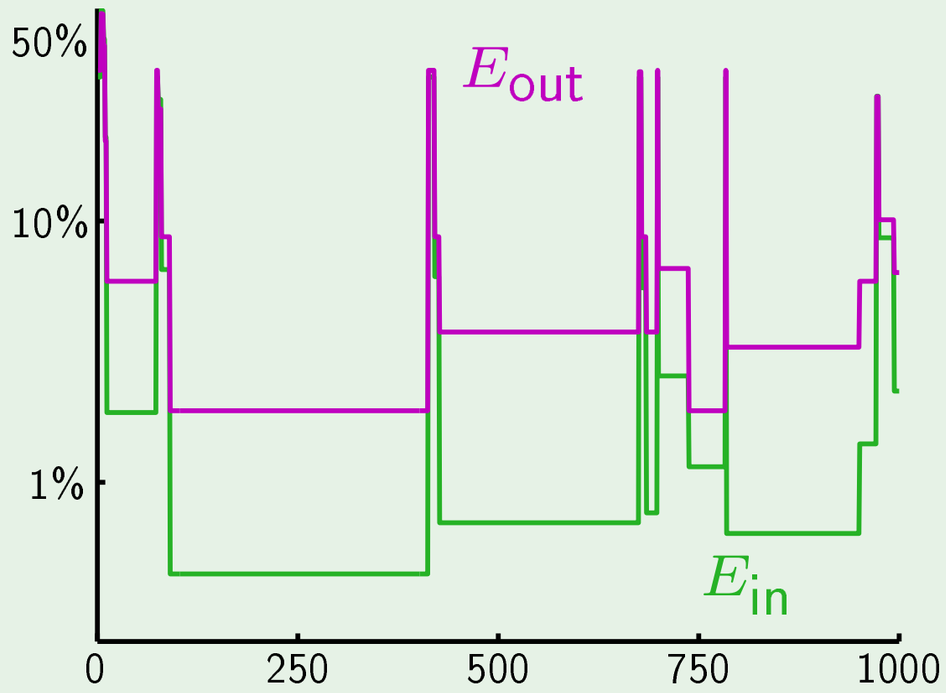
Illustration of features

$\mathbf{x} = (x_0, x_1, x_2)$ x_1: intensity x_2: symmetry

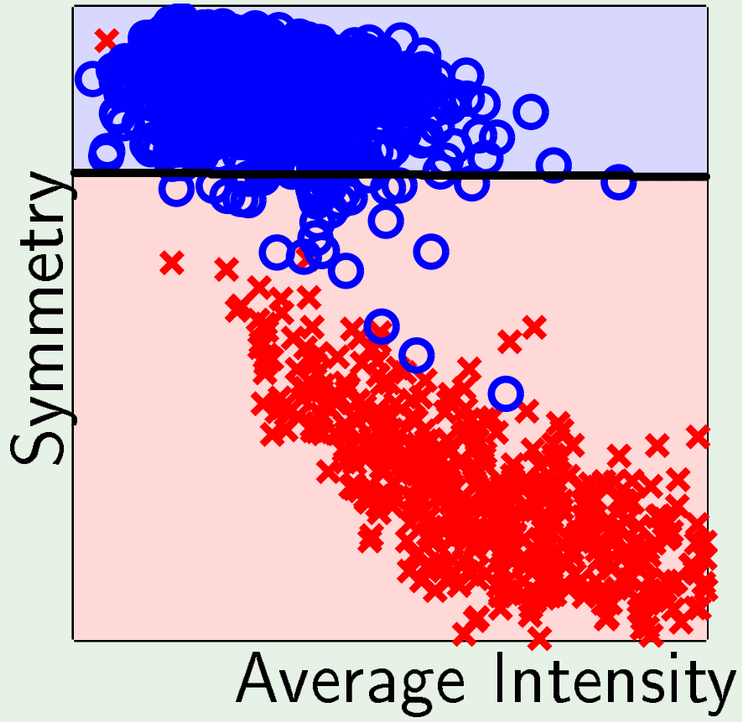


What PLA does

Evolution of E_{in} and E_{out}



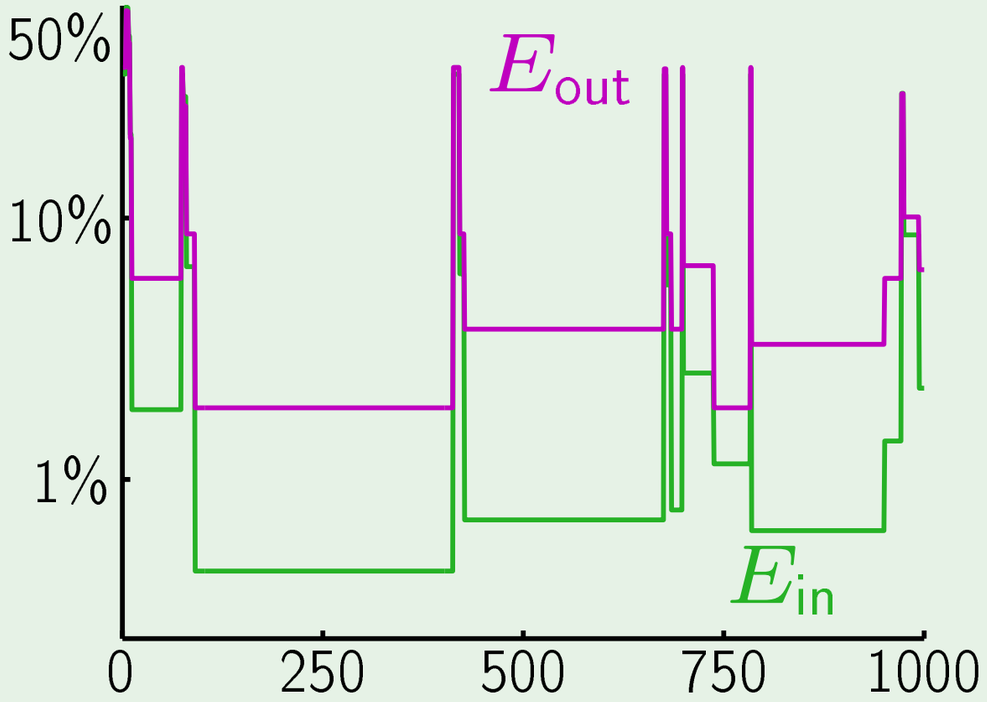
Final perceptron boundary



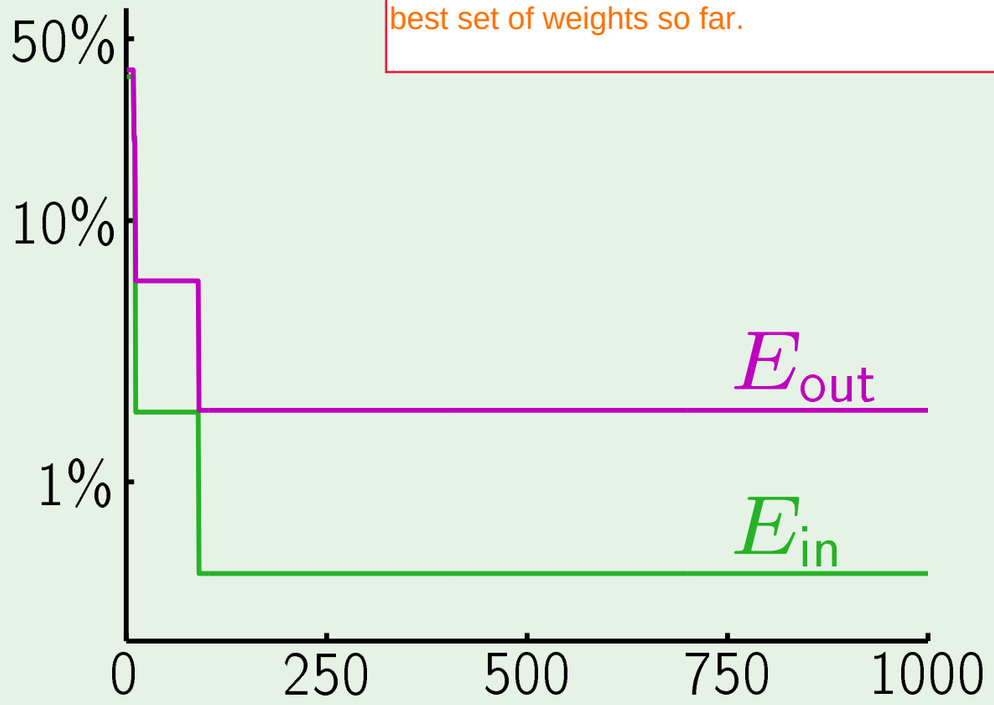
Typical behaviour of PLA since data not linearly separable - also it will not converge, so forced to stop at 1000. Note that Eout tracks Ein well since there are only 3 parameters (very simple model) so will generalize well

The 'pocket' algorithm

PLA:



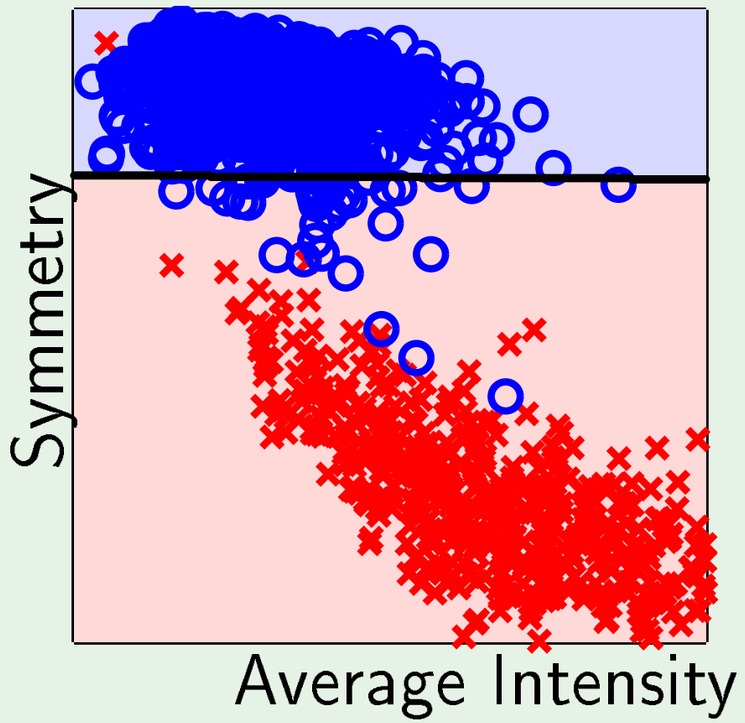
Pocket:



Simple modification where we keep track of the in-sample error Ein and keep track of it throughout if it is not an improvement, skip and carry on with the algorithm. essentially, keep the best set of weights so far.

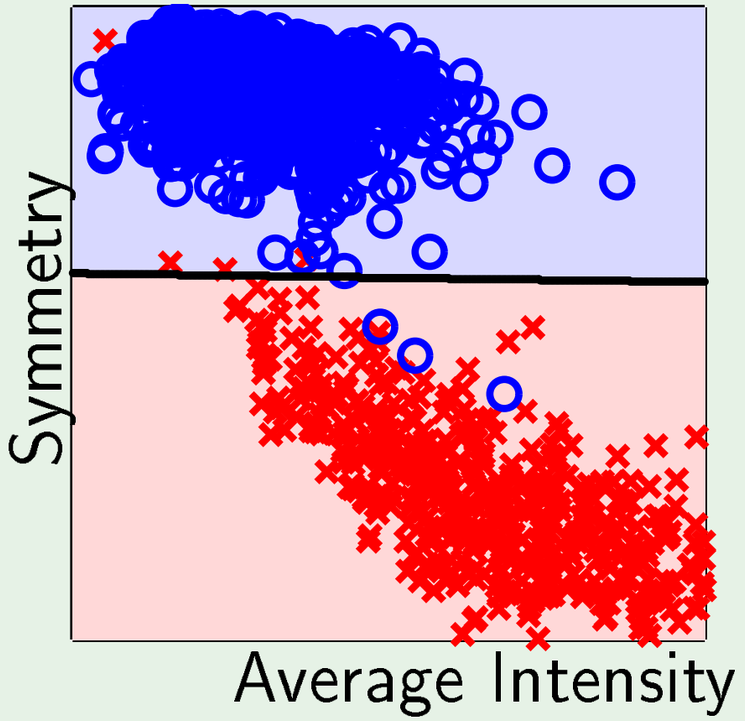
Classification boundary - PLA versus Pocket

PLA:



Pocket algorithm allows us to separate linearly separable data (it is basically separable but there are a few outliers which make it difficult to separate) - does a reasonable job at separating nonetheless

Pocket:



Outline

- Input representation
- Linear Classification
- Linear Regression **regression $\equiv$ real-valued output**
- Nonlinear Transformation

Credit again

Classification: Credit approval (yes/no)

Regression: Credit line (dollar amount)

Input: $\mathbf{x} =$

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

Linear regression output: $h(\mathbf{x}) = \sum_{i=0}^{d} w_i \, x_i = \mathbf{w}^\mathsf{T}\mathbf{x}$

The data set

Credit officers decide on credit lines:

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$$

$y_n \in \mathbb{R}$ is the credit line for customer $\mathbf{x}_n$.

Linear regression tries to replicate that.

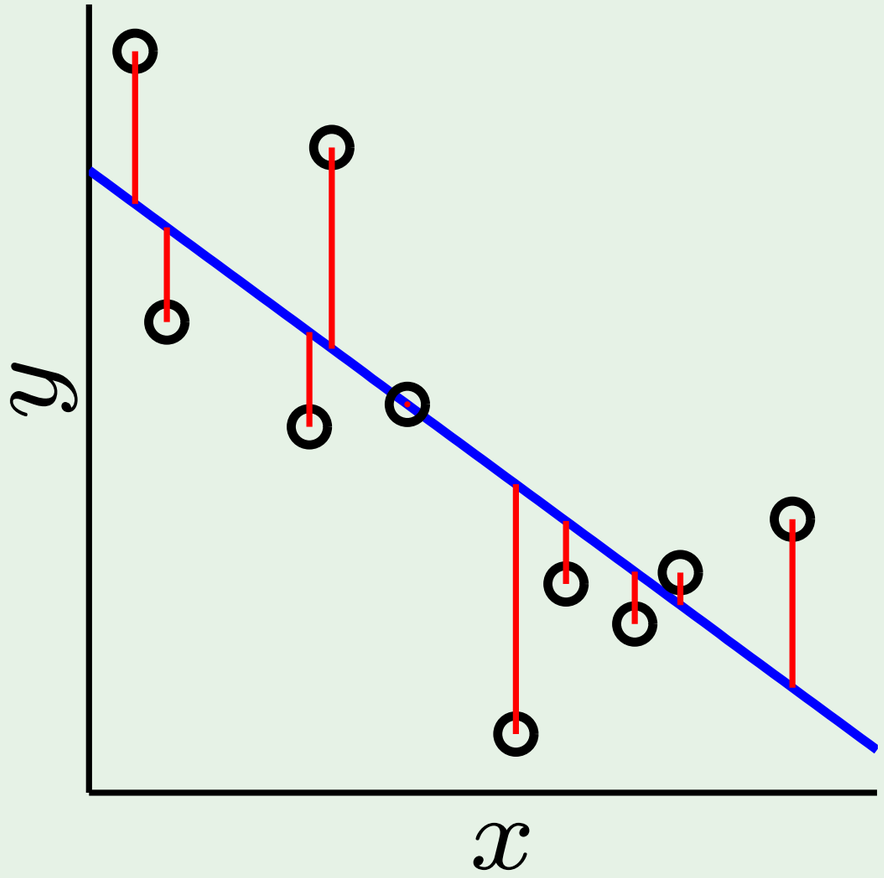
How to measure the error

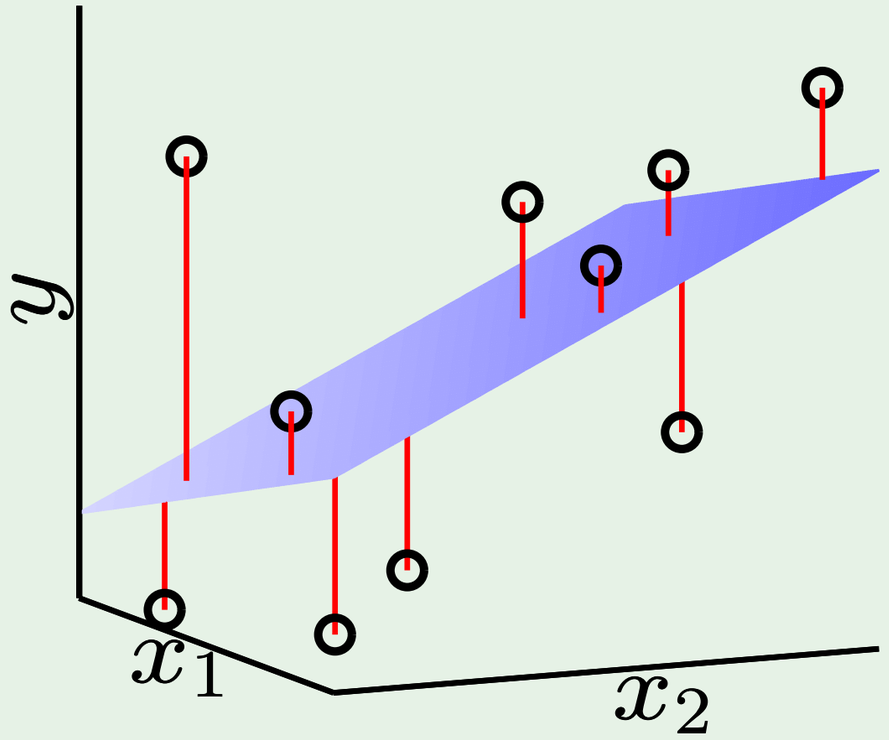
How well does $h(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x}$ approximate $f(\mathbf{x})$?

In linear regression, we use squared error $(h(\mathbf{x}) - f(\mathbf{x}))^2$

$$\textbf{in-sample error: } E_{\text{in}}(h) = \frac{1}{N}\sum_{n=1}^{N}\left(h(\mathbf{x}_n) - y_n\right)^2$$

Illustration of linear regression





The expression for E_{in}

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

$$= \frac{1}{N}\|X\mathbf{w} - \mathbf{y}\|^2$$

where $X = \begin{bmatrix} —\mathbf{x}_1{}^{\mathsf{T}}— \\ —\mathbf{x}_2{}^{\mathsf{T}}— \\ \vdots \\ —\mathbf{x}_N{}^{\mathsf{T}}— \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$

Minimizing E_{in}

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2$$

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{2}{N}\mathrm{X}^{\mathsf{T}}(\mathrm{X}\mathbf{w} - \mathbf{y}) = \mathbf{0}$$

$$\mathrm{X}^{\mathsf{T}}\mathrm{X}\mathbf{w} = \mathrm{X}^{\mathsf{T}}\mathbf{y}$$

$$\mathbf{w} = \mathrm{X}^{\dagger}\mathbf{y} \quad \text{where} \quad \mathrm{X}^{\dagger} = (\mathrm{X}^{\mathsf{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathsf{T}}$$

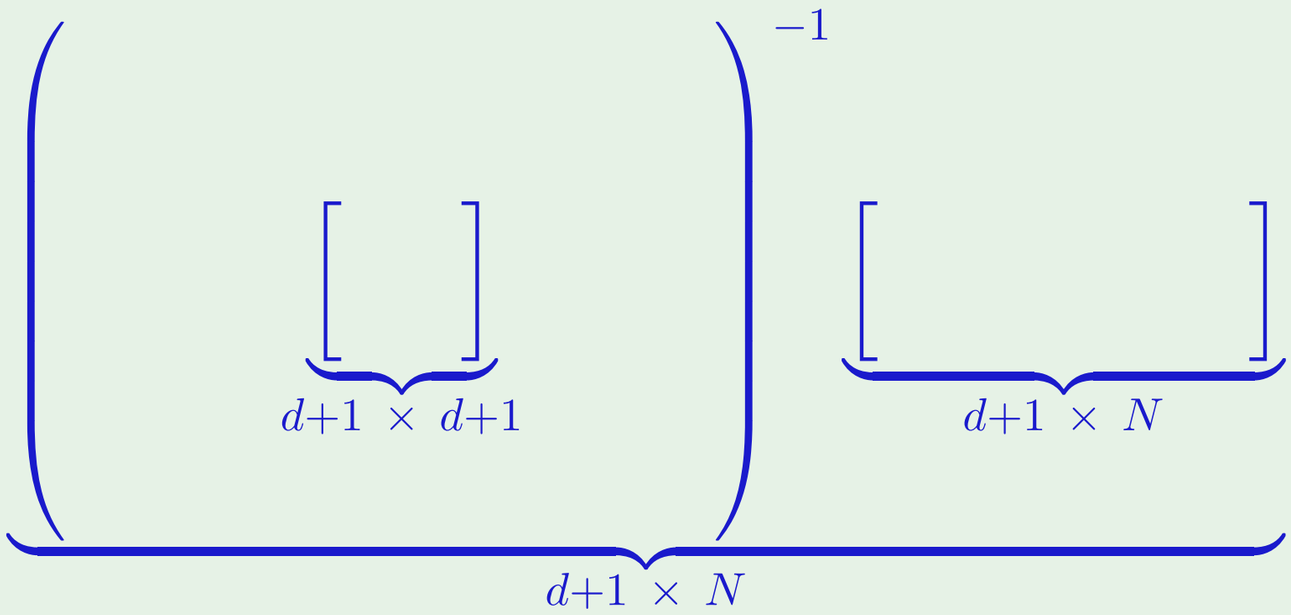
$\mathrm{X}^{\dagger}$ is the '**pseudo-inverse**' of X

since Xdagger*X = Identity

The pseudo-inverse

d = number of parameters (d+1 due to bias or x0)
N = number of training examples

$$\mathrm{X}^\dagger = (\mathrm{X}^\mathsf{T}\mathrm{X})^{-1}\mathrm{X}^\mathsf{T}$$



The linear regression algorithm

1: Construct the matrix X and the vector $\mathbf{y}$ from the data set $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$ as follows

$$\underbrace{\mathrm{X} = \begin{bmatrix} \text{---} \mathbf{x}_1^\mathsf{T} \text{---} \\ \text{---} \mathbf{x}_2^\mathsf{T} \text{---} \\ \vdots \\ \text{---} \mathbf{x}_N^\mathsf{T} \text{---} \end{bmatrix}}_{\text{input data matrix}}, \qquad \underbrace{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}.$$

2: Compute the pseudo-inverse $\mathrm{X}^\dagger = (\mathrm{X}^\mathsf{T}\mathrm{X})^{-1}\mathrm{X}^\mathsf{T}$.

3: Return $\mathbf{w} = \mathrm{X}^\dagger \mathbf{y}$.

Chances in a real problem are overwhelming that X will be an invertible matrix since the number of parameters/columns is usually small, with many rows (training examples).

Linear regression for classification

Linear regression learns a real-valued function $y = f(\mathbf{x}) \in \mathbb{R}$

Binary-valued functions are also real-valued! $\pm 1 \in \mathbb{R}$

Use linear regression to get $\mathbf{w}$ where $\mathbf{w}^{\mathsf{T}}\mathbf{x}_n \approx y_n = \pm 1$

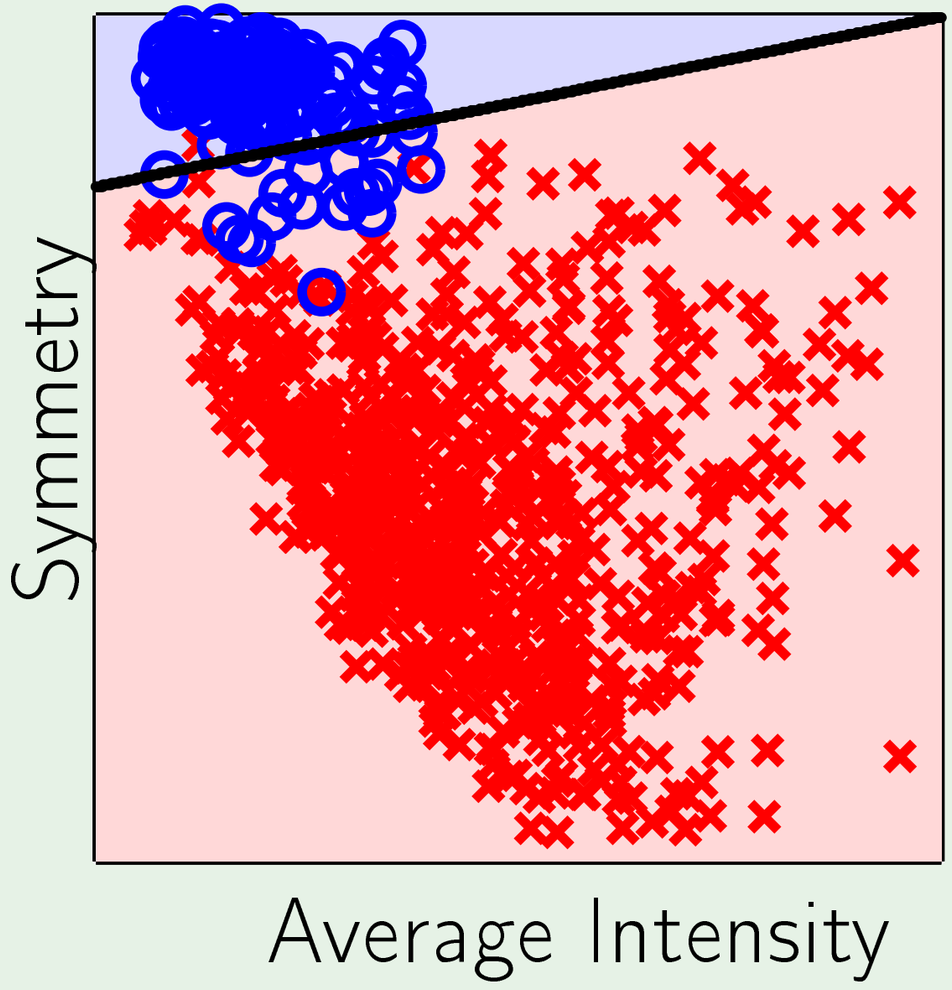
In this case, $\text{sign}\left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n\right)$ is likely to agree with $y_n = \pm 1$

Provide

Good initial weights for classification

which can then be fed to a classification algorithm (e.g. the 'pocket' algorithm) to get a better result. Not a good idea to only use linear regression since it will try to make all of the red points (see next slide) the value of -1, even the ones very far away from the theoretical perfect boundary between the groups - in doing so it is likely to move the line further downwards, perhaps meaning more blue points are incorrectly classified for the sake of already-red points (attempting to fit points irrelevant to classification, it can worsen the classification). Hence, better to use this as an initial weight for a proper classification algorithm.

Linear regression boundary

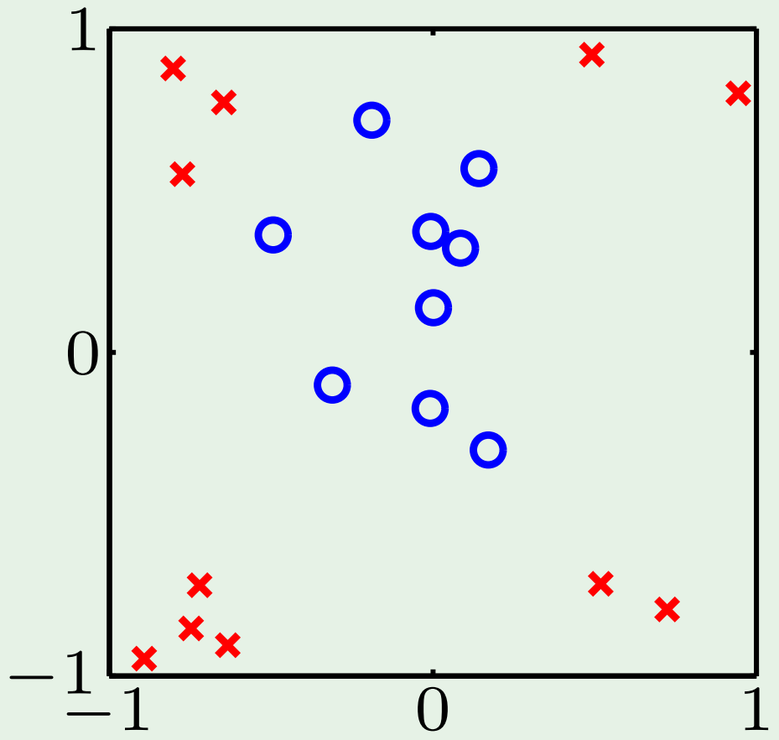


Outline

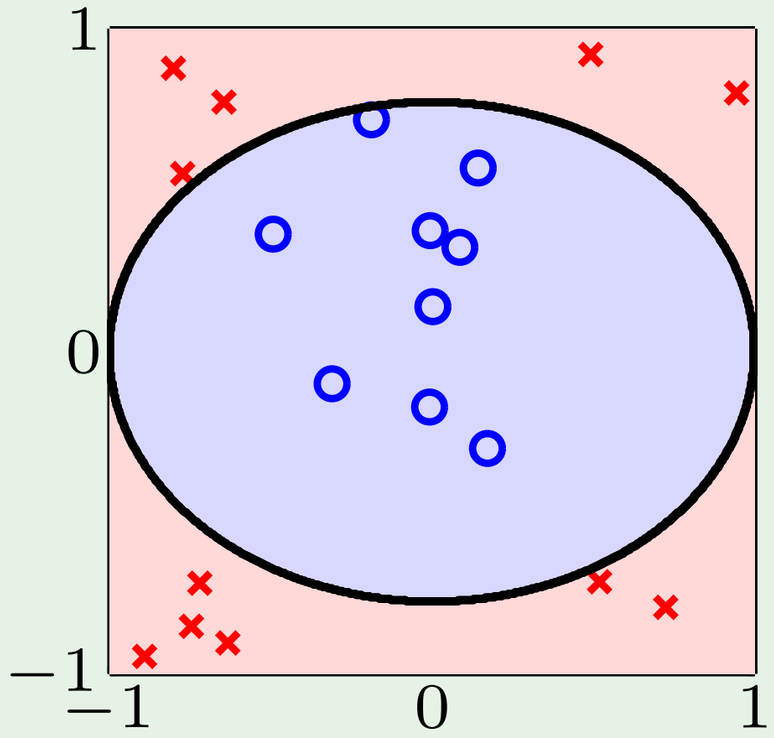
- Input representation
- Linear Classification
- Linear Regression
- Nonlinear Transformation

Linear is limited

Data:



Hypothesis:



Another example

Credit line is affected by 'years in residence'

but **not** in a linear way!

Nonlinear $[[x_i < 1]]$ and $[[x_i > 5]]$ are better.

[[...]] indicates that these return 1 if the condition is true and 0 if it is false

Can we do that with linear models?

Note the 'feature extraction/creation' seen here - a feature is made when you take the input and transform it into something else (a higher level representation of a raw input)

Linear in what?

Linear regression implements

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

Linear classification implements

$$\text{sign}\left(\sum_{i=0}^{d} w_i \, x_i\right)$$

Algorithms work because of **linearity in the weights**

These algorithms are linear because they are linear in the weights - the inputs (x) are constants and the parameters/weights (w) are the variables that we care about

Transform the data nonlinearly

$$(x_1, x_2) \xrightarrow{\Phi} (x_1^2, x_2^2)$$