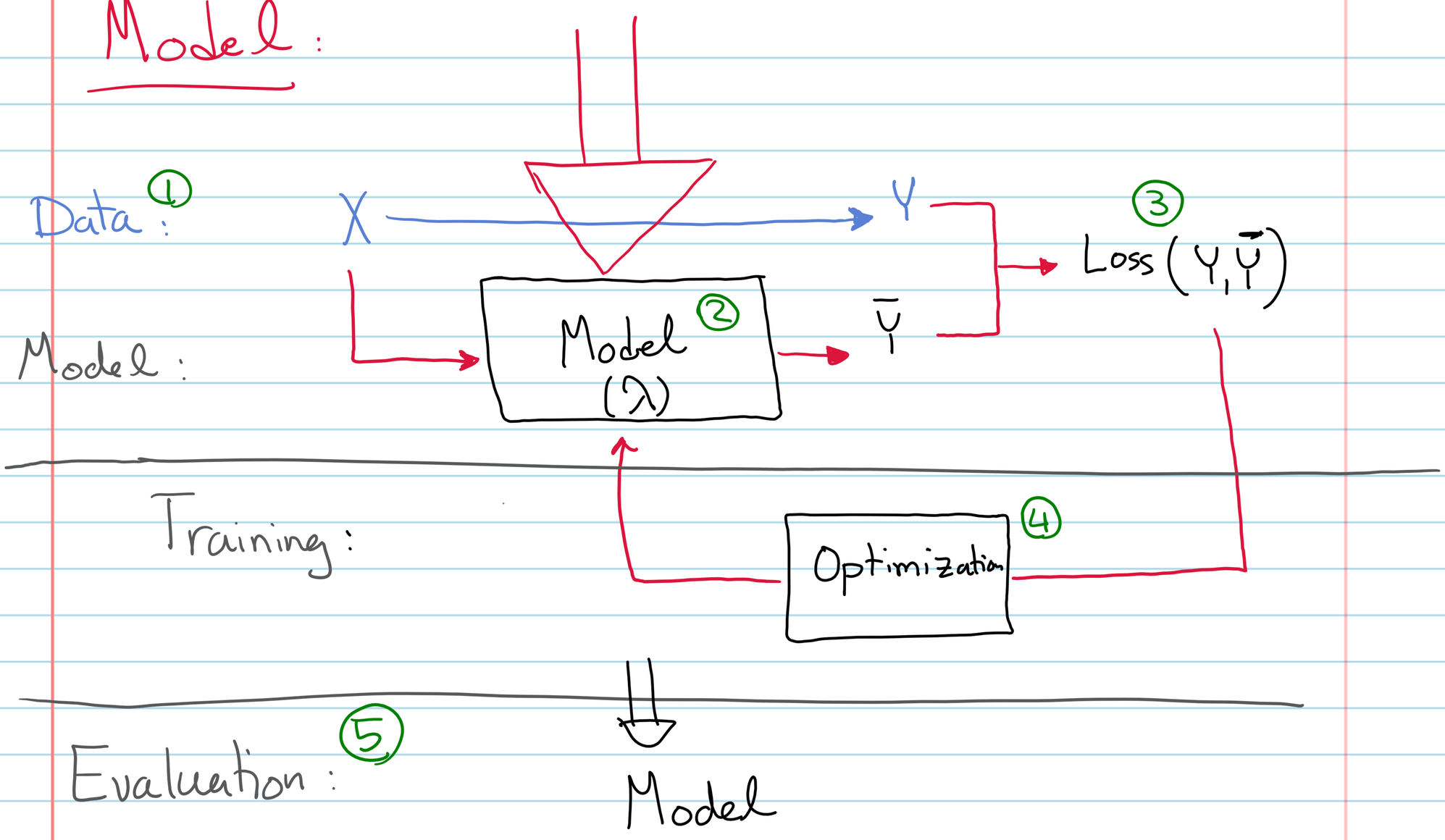
Model:


Data: 1
X
Y
3
Loss(Y, Ȳ)
Model: 2
Model (λ)
Ȳ
Training:
4
Optimization
Evaluation: 5
Model

Reg.

$$\vec{X}^{(1)} = (x_1^1, x_2^1, \dots x_{n_f}^1) \longrightarrow Y^{(1)}$$

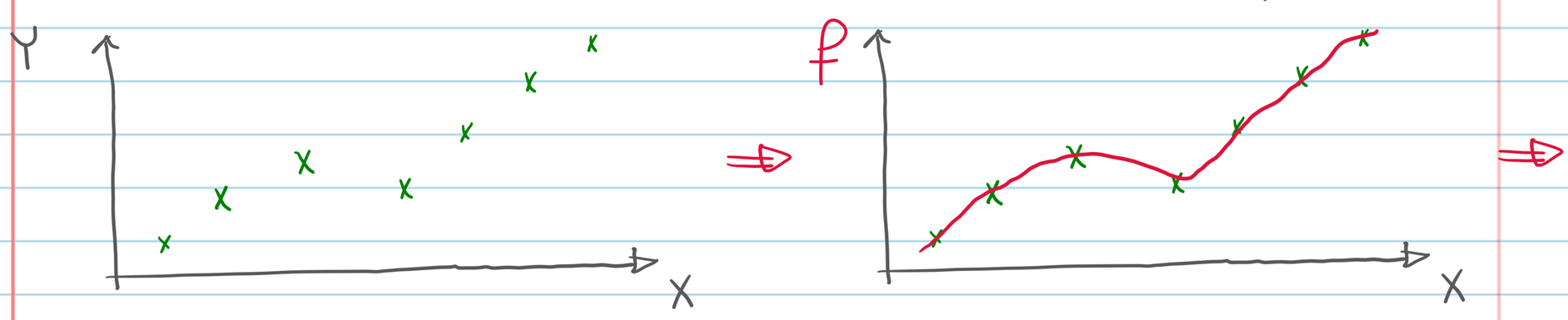
$$\vec{X}^{(2)} \longrightarrow Y^{(2)}$$

$$\vdots$$

$$\vec{X}^{(n_s)} \longrightarrow Y^{(n_s)}$$

$$\Rightarrow \text{Model} \equiv f : \quad \underline{Y^{(i)} \simeq f(\vec{X}^{(i)})}$$

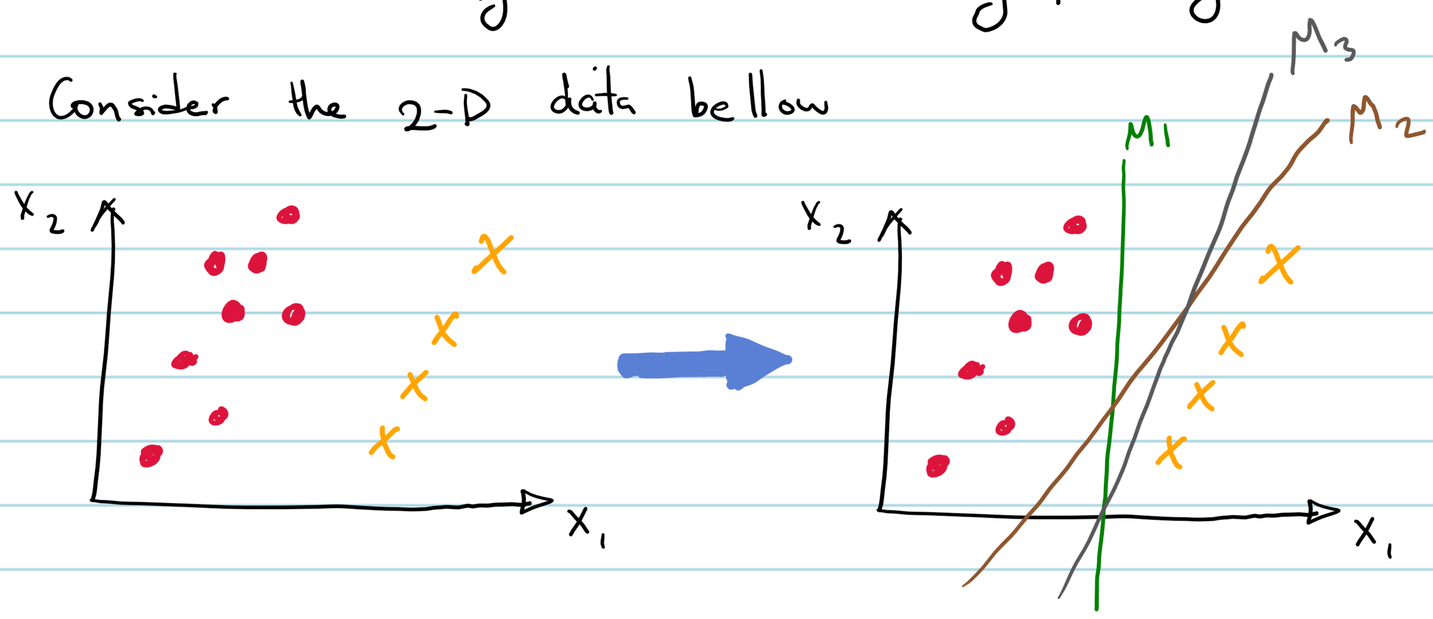
1-D example : $n_f = 1$



Classification (CLF)

This has similarity to both Clustering & Reg.

Consider the 2-D data bellow



$\vec{X} : (x_1, x_2 \ldots)$

The idea is to find a model that, based on the value of x, can predict whether $y = \times$ or $y = \bullet$.

Coming from Reg. one approach could be to find a function f such that:

$f(\vec{x}) > 0 \rightarrow \times$

$f(\vec{x}) < \rightarrow \bullet$

$f(x) = 0 \rightarrow$ <u>Decision Boundary</u>

There are other ways that we'll talk about.

Polynomial models

Probably the simplest models we can consider are Poly(n) functions.
We talked about this in the introduction.

For regression

$$f(\vec{x}) = \vec{X} \cdot \vec{W} = w_1 x_1 + w_2 x_2 + \dots$$

$$X : [n_s, n_f] \qquad \vec{W} : [n_f, 1]$$

$$\vec{X} \cdot \vec{W} = \vec{Y} \Rightarrow \underbrace{\vec{X}^T \cdot \vec{X}} \cdot \vec{W} = \vec{X}^T \cdot \vec{Y}$$

$$\underline{\vec{W} = (\vec{X}^T \cdot X)^{-1} \cdot \vec{X}^T \cdot Y} \rightarrow \underline{?}$$

For ploy(n) we can take $\vec{X}$ & generate

$$\tilde{X} = [\vec{X}^n, \vec{X}^{n-1}, \dots \vec{X}, 1].$$

Note that if $\underset{\text{(linear)}}{n_s < n_f}$ or $n_s < \underbrace{n_f \times n}_{\text{w/o cross terms}}$ this

would not give good results.

The solution
$\vec{W} = (\vec{X}^T \cdot X)^{-1} \cdot \vec{X}^T \cdot Y$, while simple has a big flaw.

Consider a noise data & a poly (n>>)

This model starts to fit the noise. See the nb.

But why is that a bad thing?

The trend is the general pattern & extends to other samples → Prediction power

but not the noise.

What are our expectations from a good model?

* To have accurate prediction
* To generalize well.
* Is easy to train.
* Is easy to use.

We will go through this list over the next few sessions and specify them.

To start with, we need to have a notion of accuracy.

i.e. $d(Y, \tilde{Y}) \approx 0$

Some examples: $d(Y, \tilde{Y}) = \frac{1}{n_s} \sum_i (Y_i - \tilde{Y}_i)^2$

$d(Y, \tilde{Y}) = \frac{1}{n_s} \sum_i |Y_i - \tilde{Y}_i|$

$d(Y, \tilde{Y}) = \cdots$

→ Model

$\underset{w}{\operatorname{argmin}} \; d(Y, f_w(X))$

→ w^*

$\boxed{f_{w^*}}$

Based on the first expectation, to train a model means to tune the parameters of the model such that it would minimizes $d(Y, \tilde{Y})$.

This applies to both Reg. & Clf.

Other techniques

We'll study some other techniques for classification.
But most of these technique can be used for reg. too.

$\boxed{P_1(x) = 1/2}$

Logistic Reg.

$$f(\vec{x}) = Sig(\vec{x} \cdot \vec{w}) = \underline{Sig}(w_0 + w_1 x_1 + w_2 x_2 + \dots)$$

— $E=x$

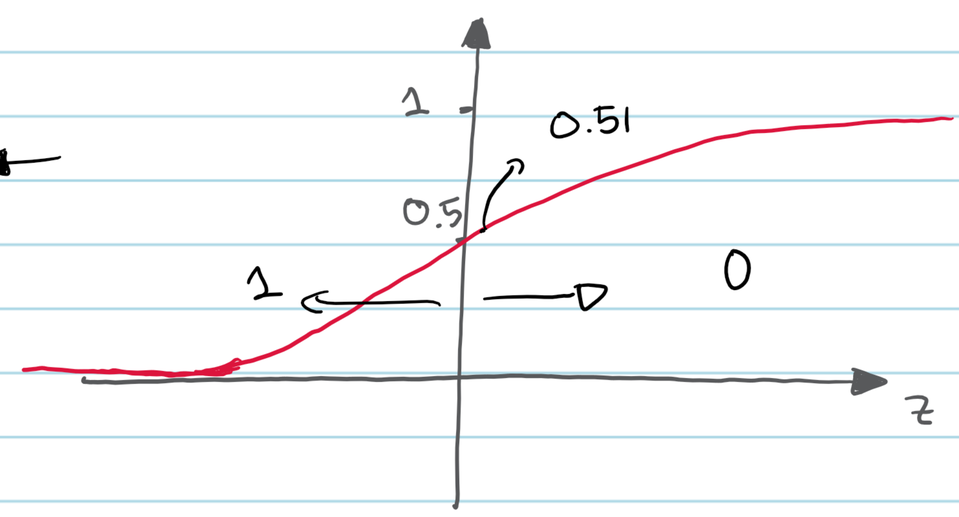
— $E=0$

Sigmoid

$$\sigma(z) = \frac{e^x}{1+e^x}$$

↓

Gibbs distribution
probability of
activation / excitation



$f \sim 0.999$

This is giving a probability for the outcome.

If $Pr \geq 1/2 \rightarrow$ Class 1

$Pr < 1/2 \rightarrow$ Class 0

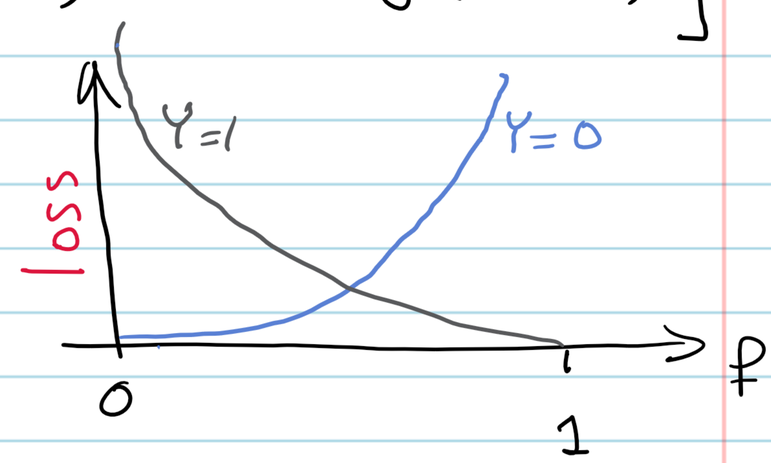
For LR we usually use cross-entropy for the loss function:

$$\ell(X,Y) = \frac{-1}{n_s} \sum_i \left[Y_i \log(f(x_i)) + (1-Y_i)\log(1-f(x_i)) \right]$$

$\boxed{0.1}$

$Y_i = 1 \rightarrow \frac{-1}{n_s} \log(f(x_i))$

$Y_i = 0 \rightarrow \frac{-1}{n_s} \log(1-f(x_i))$



Question: What's the advantage of using the cross-entropy as a lost for LR?

SVM

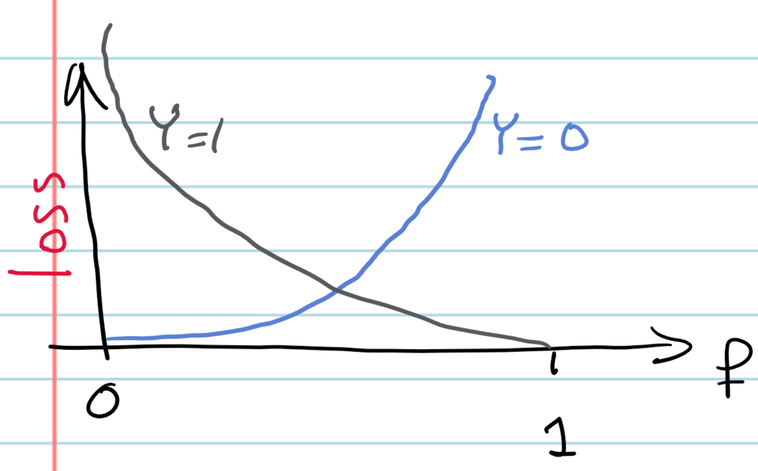
Support Vector machines (SVM) have some similarities to LR. It is a linear model, i.e.

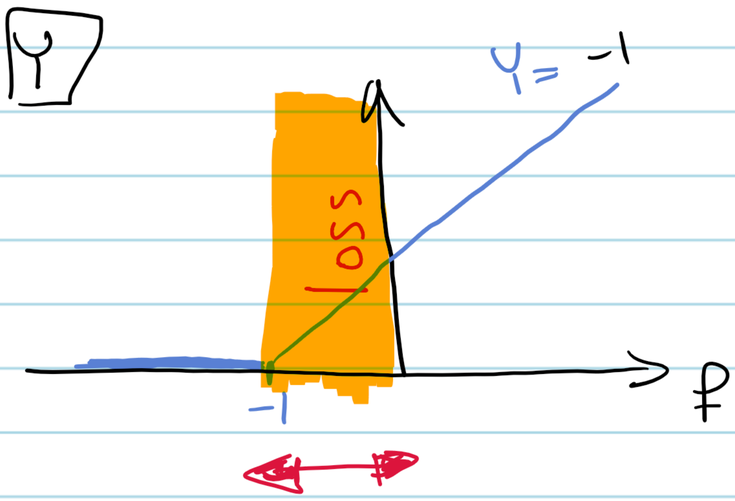
$$f(x) = \vec{x}.\vec{w} = w_0 + w_1 x_1 + w_2 x_2 + \dots$$

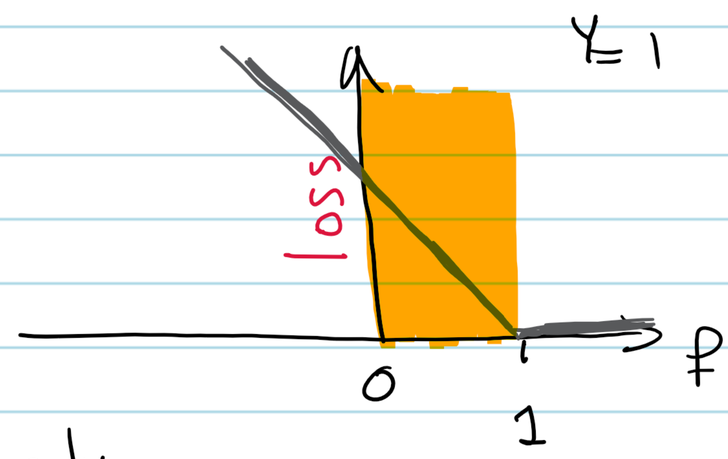
but for the loss, we use sth similar to the cross-entropy:

$$\begin{cases} f(x) \geq 0 \rightarrow \text{class } 1 \\ f(x) < 0 \rightarrow \text{class } -1 \end{cases}$$

$\boxed{Y}$







$\boxed{Y=1}$ ✓

$\boxed{f(x)=0}$ ✗ $l=1$

$f(x)=1$ ✓ $l=0$

This is to make sure that not only $f(x) > 0$ gives the class, but it is the most confident model that does this. → Max margin.

This is known as the "Hinge" function:

$$l_{Hinge}(x, y) = \max(0, 1 - \underbrace{\vec{Y}.\vec{f}(x)}_{\downarrow})$$

$Y=1$

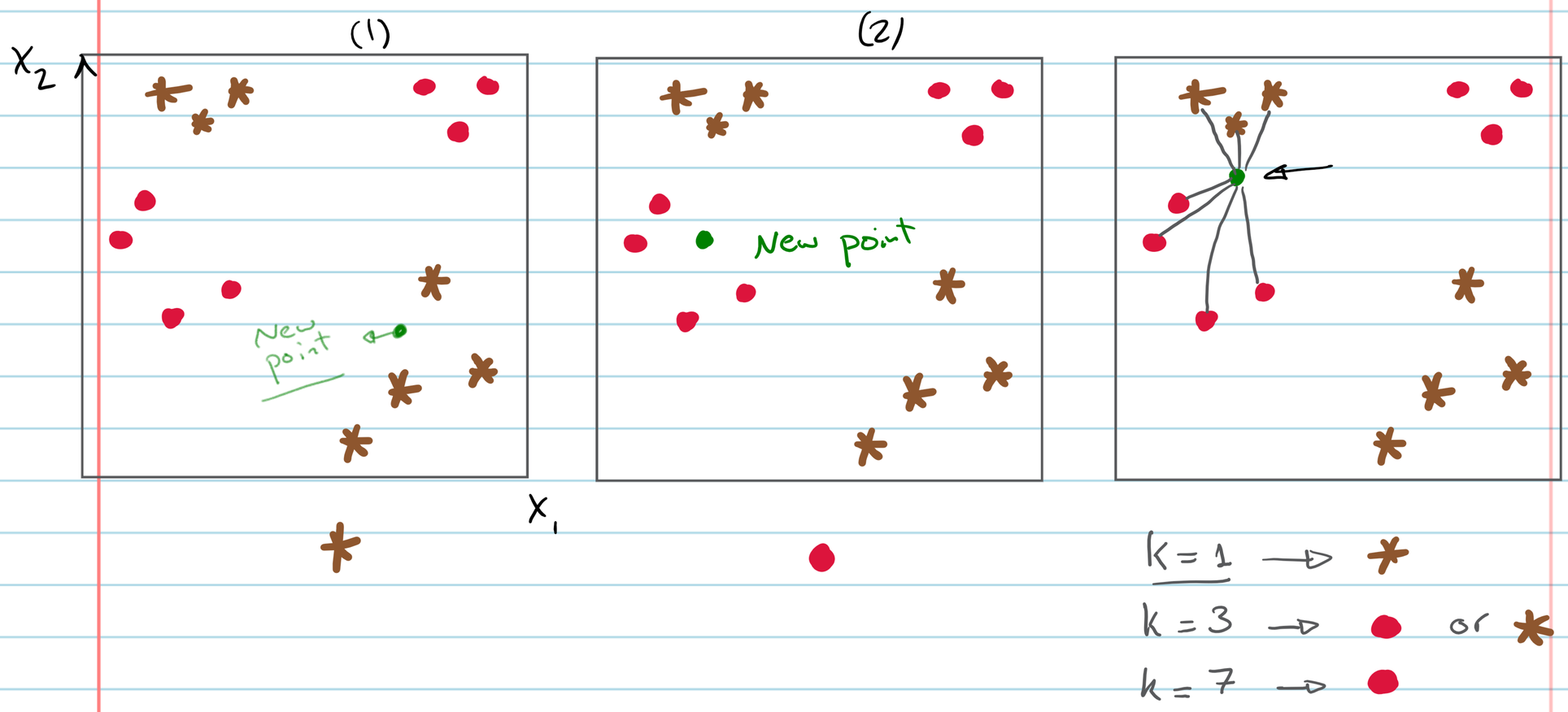
$Y=-1$

$$d(x^1, x^2) \rightarrow d(\Phi(x^1), \Phi(x^2))$$

KNN

This is an "instance-based" algorithm.

Basically, it memorizes all the samples & given a new sample, it looks at the K closest neighbors of the new sample.

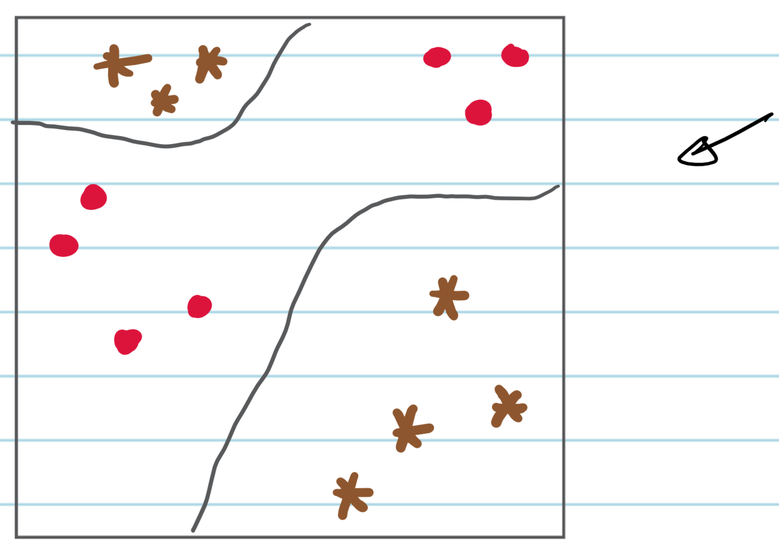


$k = 1 \rightarrow$ *

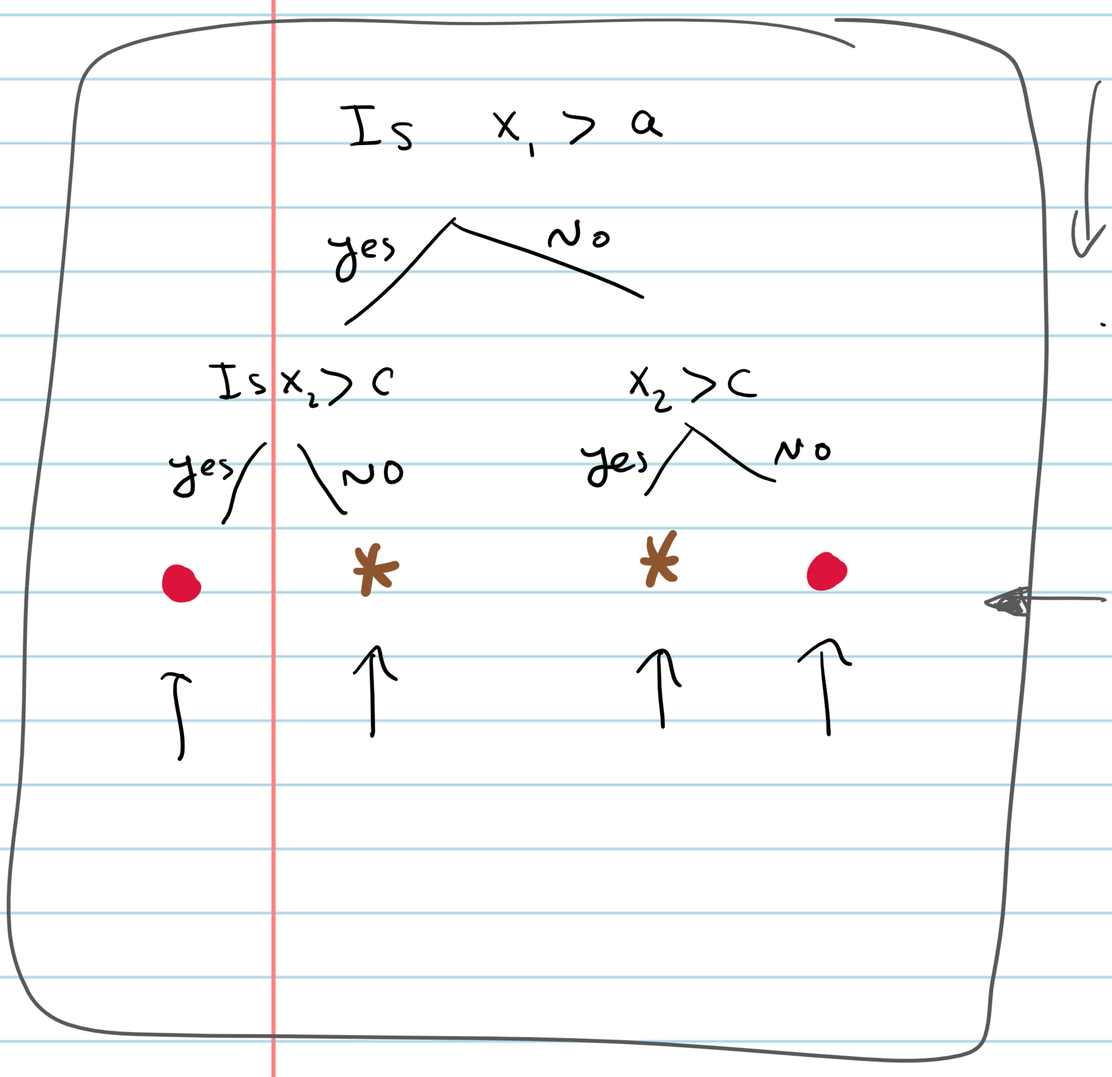
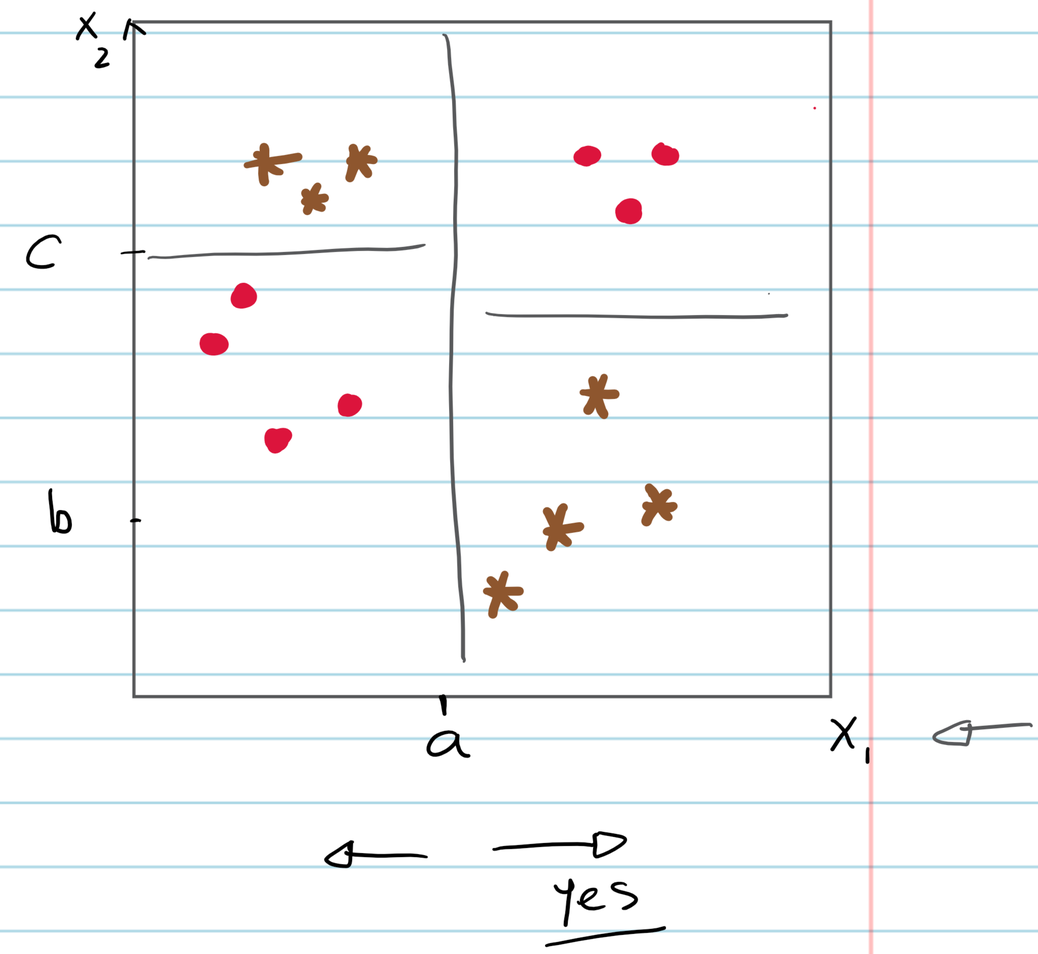
$k = 3 \rightarrow$ ● or *

$k = 7 \rightarrow$ ●

Remark: The decision-boundary would depend on k and the algorithm for voting.



* Fast to train $(X \rightarrow Y)$
* Slower to predict $\rightarrow$ X_{new} $O(n)$
* Requires a metric $d(\vec{x}_1, \vec{x}_2)$

Question: Can you think of a situation where majority voting is not good?

Decision tree