MLP 2020
Sadegh Raeisi
Lec3 - P1

Predictive Machine Learning

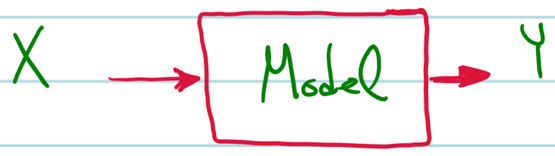
Classification & Regression

These types of algorithms take two inputs

X → The features of different samples

Y → The value of a target quantity for each sample

The goal is to output a **Model** that can predict the target quantity from features of the sample.



In this section, we will briefly talk about some of the well-know algorithms for this type of machine learning.

But before that, let's talk about Regression & classification and their differences.

For reg. Y is a (bunch of) continous varible : $Y \in \mathbb{R}$ or $Y \in \mathbb{R}^n$.

But for classification Y is discrete and categorical.

It means that we want the model to be able to determine the type of something.

Examples

Reg.

- Prediction of temperature from Blacke-Body radiation.
- Prediction of correlation in a state.

Clf.

- Is a state entangled?
- Is an object a conductor?
- Is a news fake?

MLP 2020
Sadegh Raeisi
Lec3 - P2

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

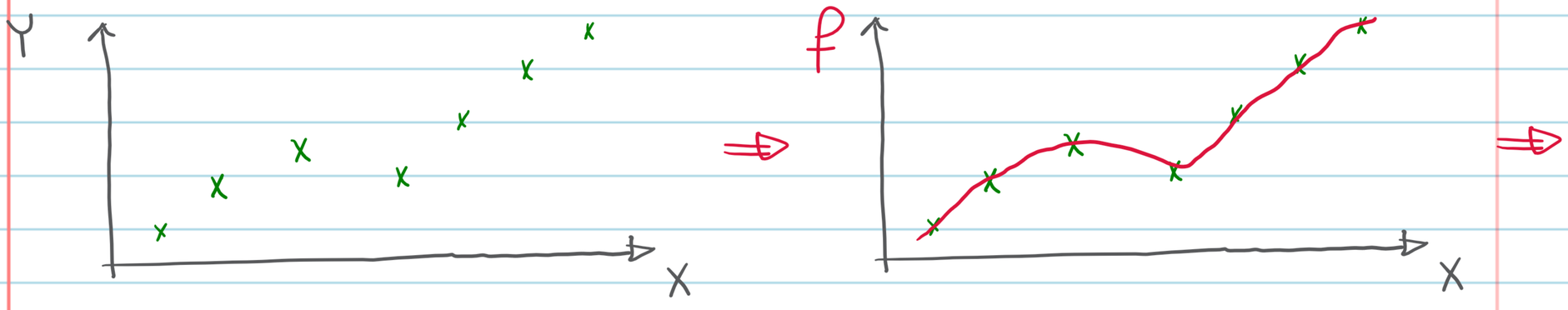
$\vec{X}^{(2)} \longrightarrow y^{(2)}$

$\vdots$

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

$\Rightarrow$ Model $= f$: $\underline{y^{(i)} \simeq f(\vec{X}^{(i)})}$

1-D example : $n_f = 1$

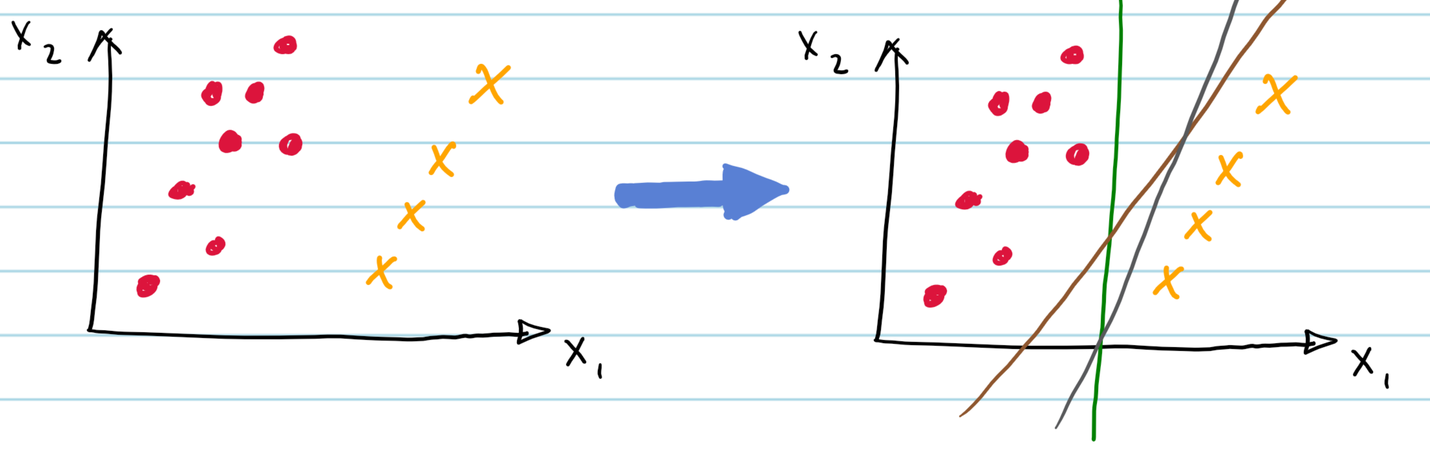


MLP 2020
Sadegh Raeisi
Lec3 - P3

Classification (CLF)

This has similarity to both Clustering & Reg.

Consider the 2-D data bellow



$\vec{X} : (X_1, X_2 \cdots)$

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:

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

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

$f(x) = 0 \rightarrow$ Decision Boundary

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

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

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

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

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

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 $n_s < n_f$ (linear) 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. $d(Y, \tilde{Y}) \simeq 0$

Some examples:

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

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

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

→ Model

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

$$w \rightarrow w^*$$

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

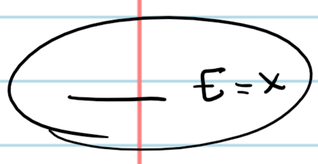
We'll study some other techniques for classification.

But most of these technique can be used for reg. too.

$P_1(x) = 1/2$

Logistic Reg.

$$f(\vec{X}) = Sig(\vec{X}.\vec{W}) = \underline{Sig}(W_0 + W_1 X_1 + W_2 X_2 + \dots)$$



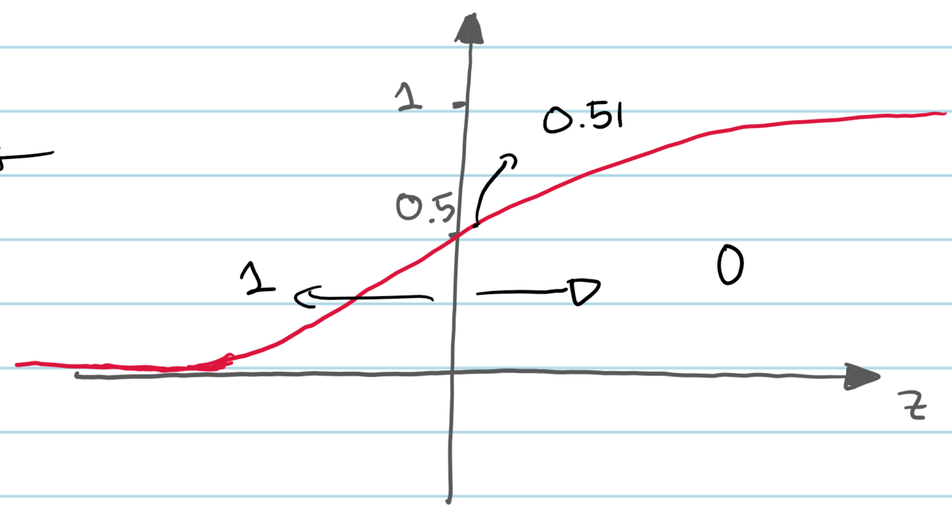
Sigmoid

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

Gibbs distribution

probability of

activation / excitation



$Pr \sim 0.999$

This is giving a probability for the outcome.

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

$Pr < 1/2 \rightarrow$ Class 0

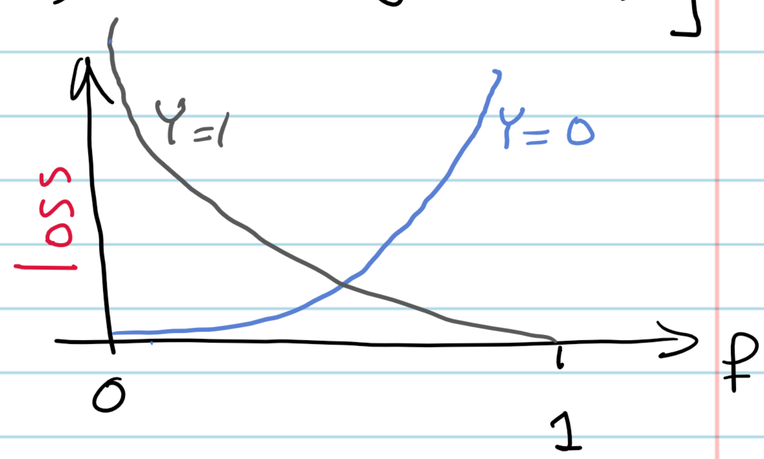
For LR we usually use cross-entropy for the loss function:

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

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

(0.1)

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



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

SVM

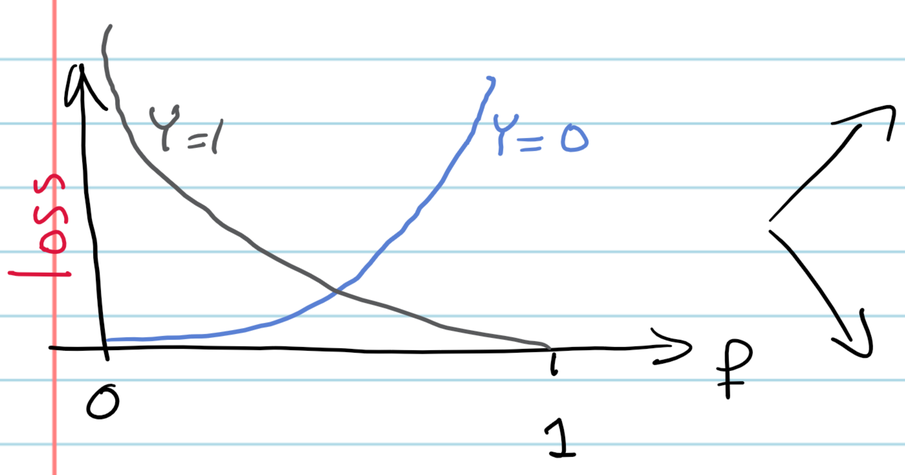
MLP 2020
Sadegh Raeisi
Lec3 - P8

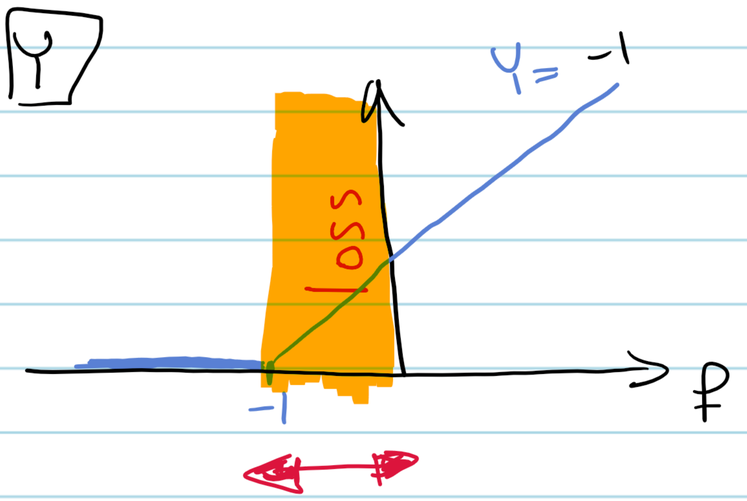
Support Vector machines (SVM) have some similarities to LR. It is a linear model, i.e.

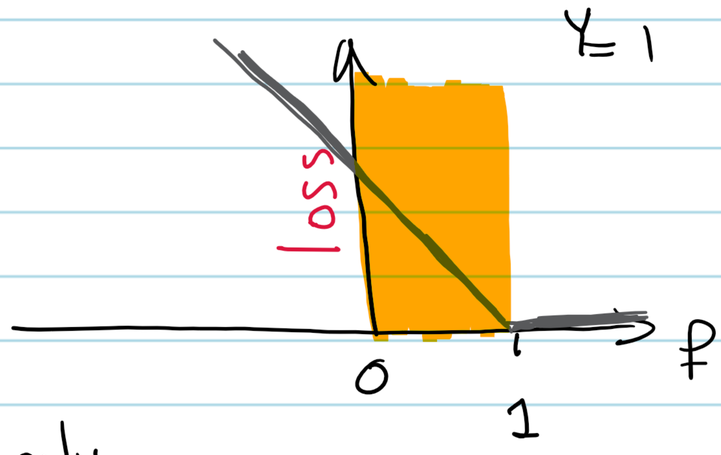
$$f(X) = \vec{X}.\vec{W} = w_0 + w_1 x_1 + w_2 x_2 + \dots$$

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

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







Y=1
f(x)=0 ✗
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:

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

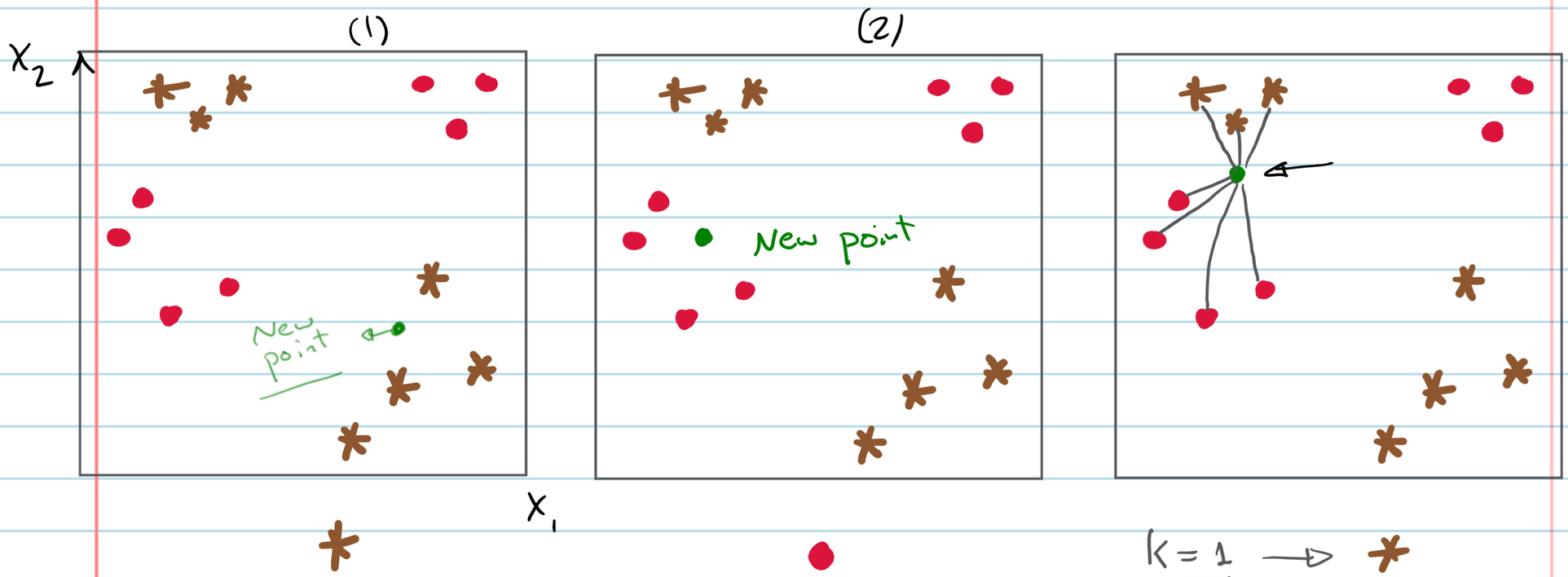
Y=1
Y=-1

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

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.

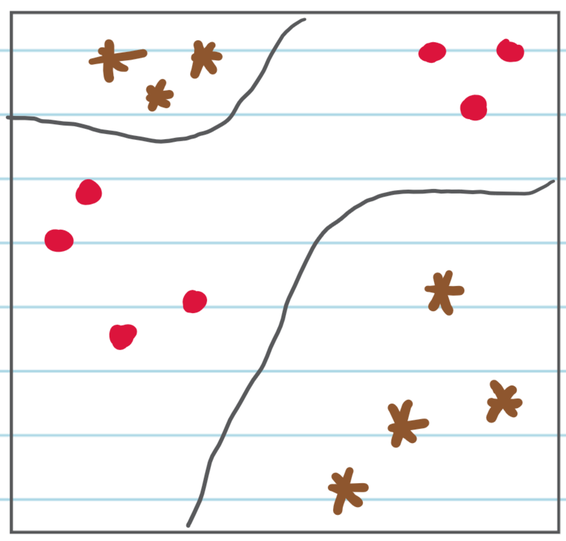


k = 1 → ✱

k = 3 → ● or ✱

k = 7 → ●

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



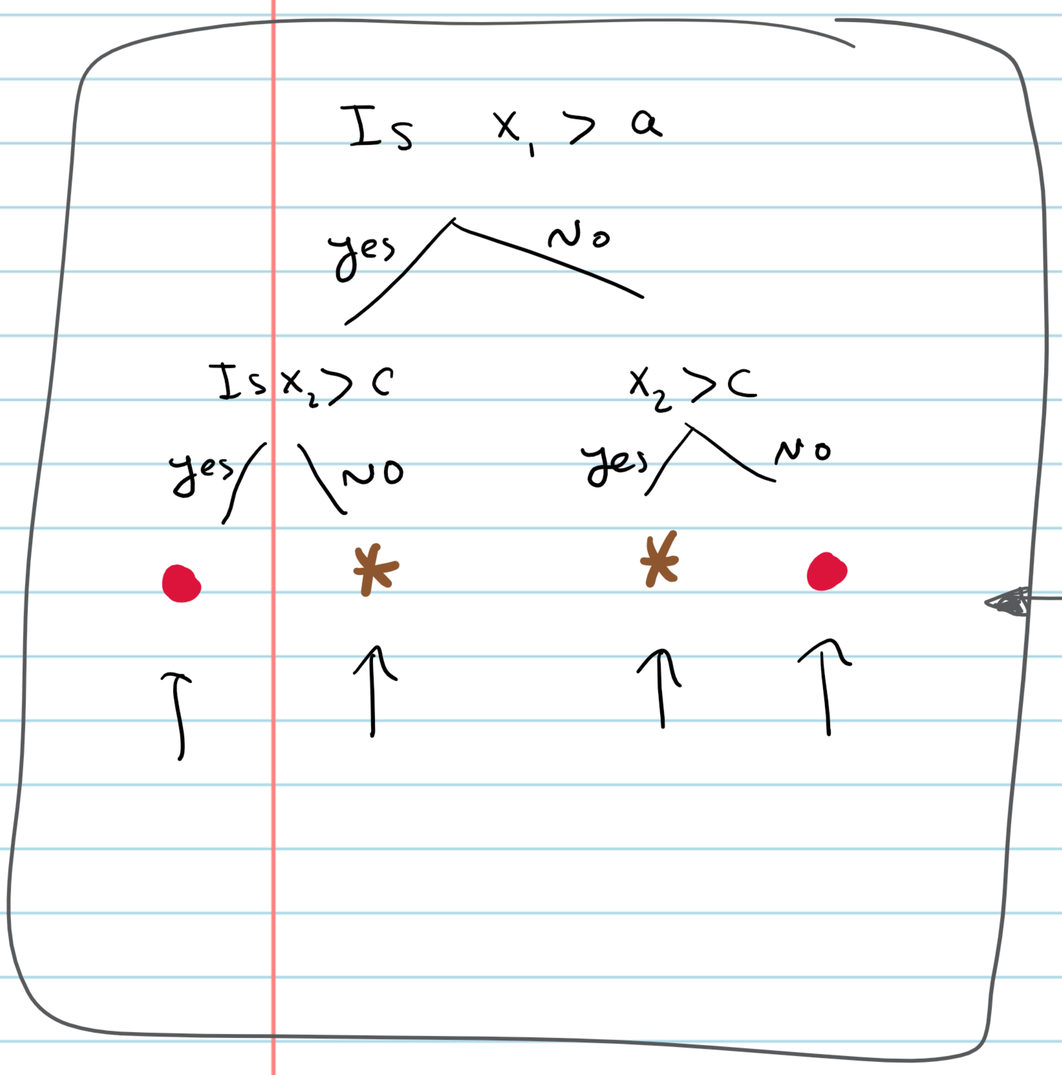
* Fast to train ↙ (X, Y)

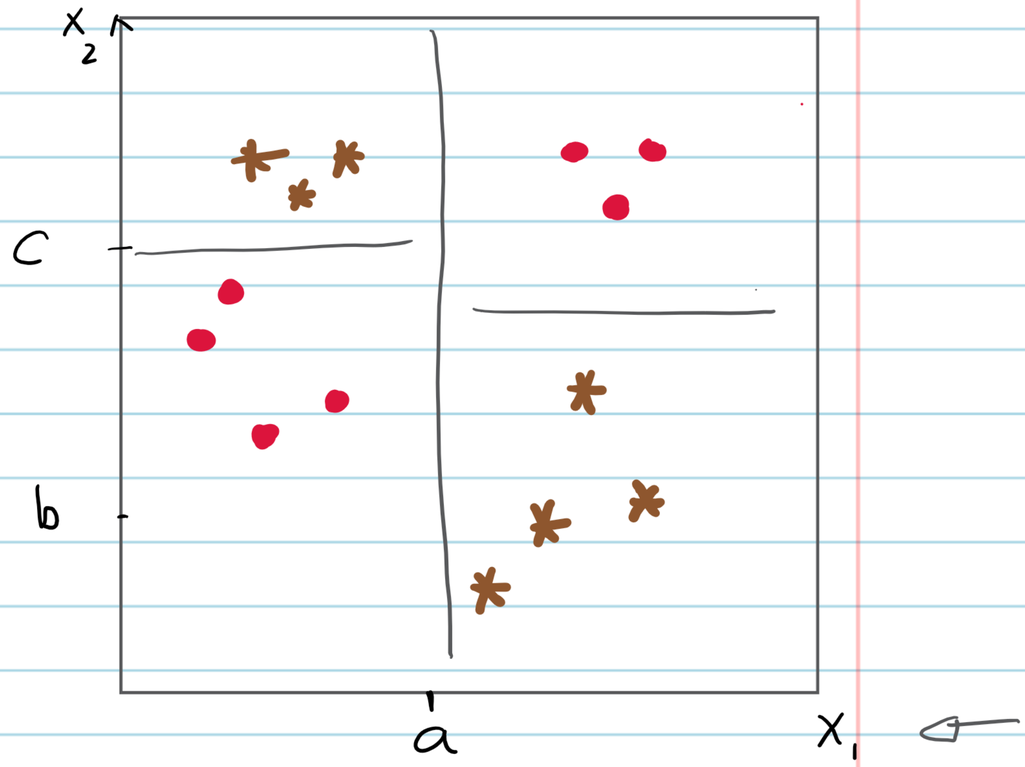
* Slower to predict → X_{new} $O(n)$

* 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





yes