Week 3
Linear Models

Nebius Academy

The objectives for today

1. Controlling LLM’s creativity vs reproducibility

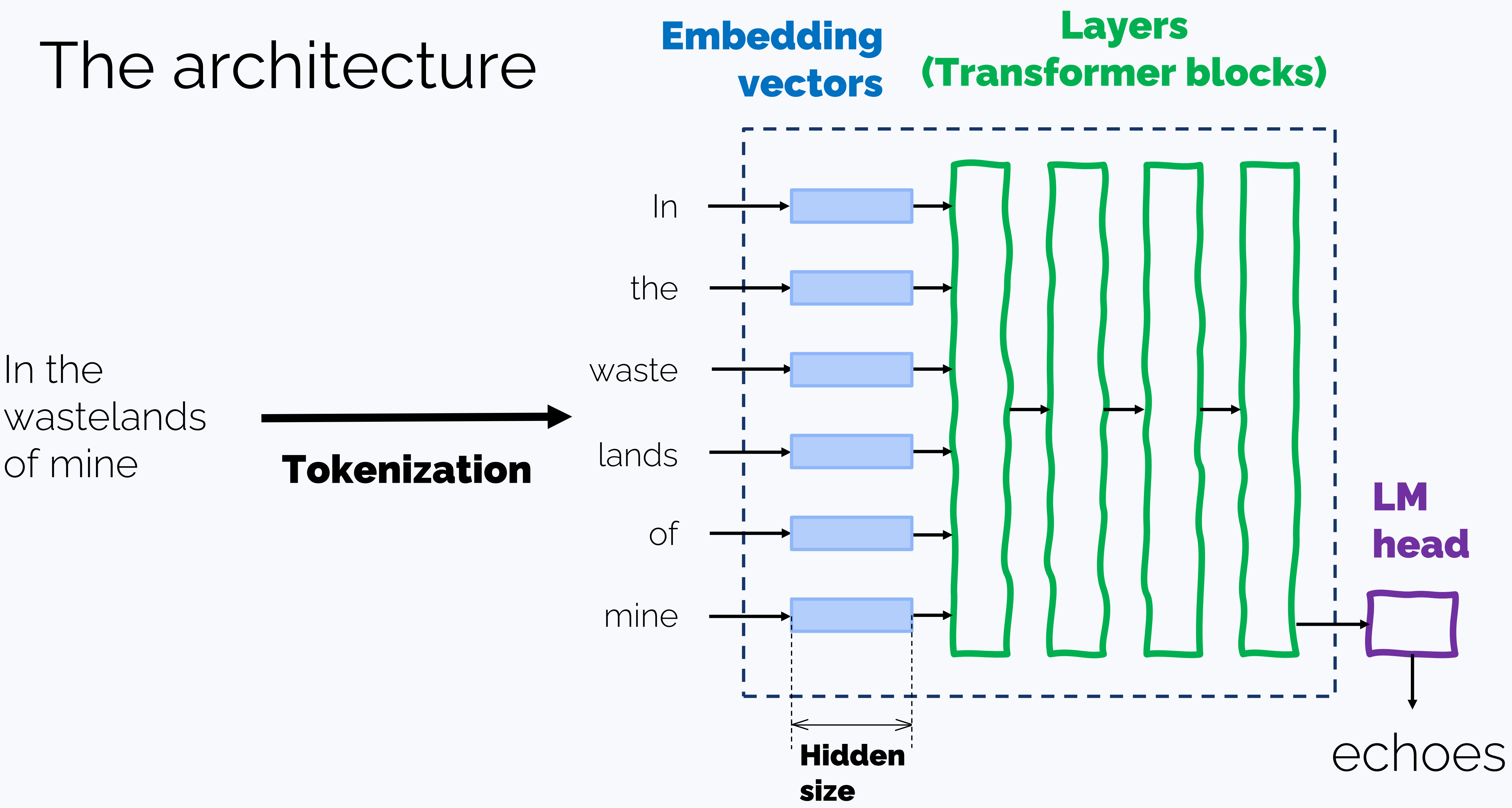
2. Close view on linear models

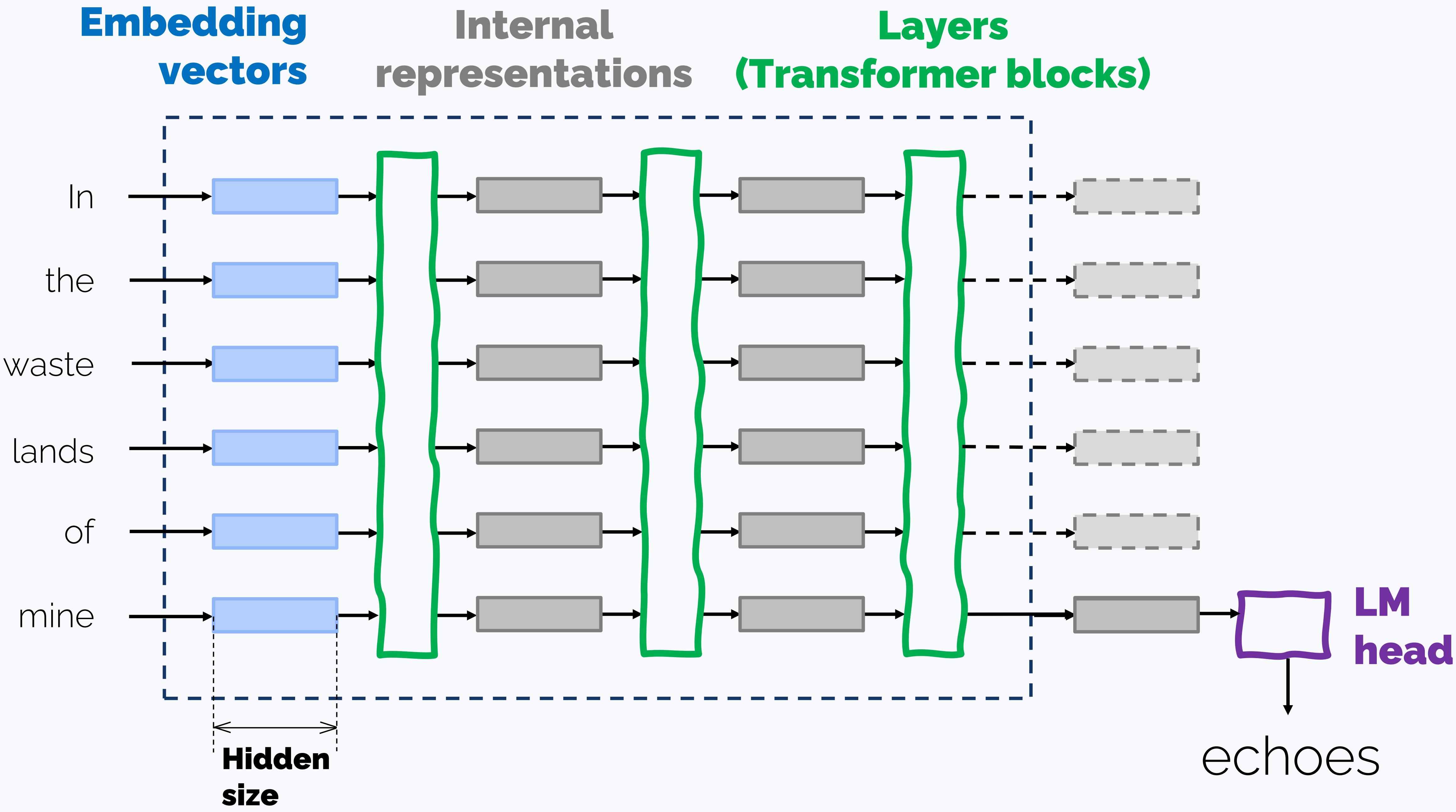
3. The concept of loss function + on which loss the LLMs are trained

4. Get our hands dirty with linear models

Bird's eye view of LLM architecture

The architecture



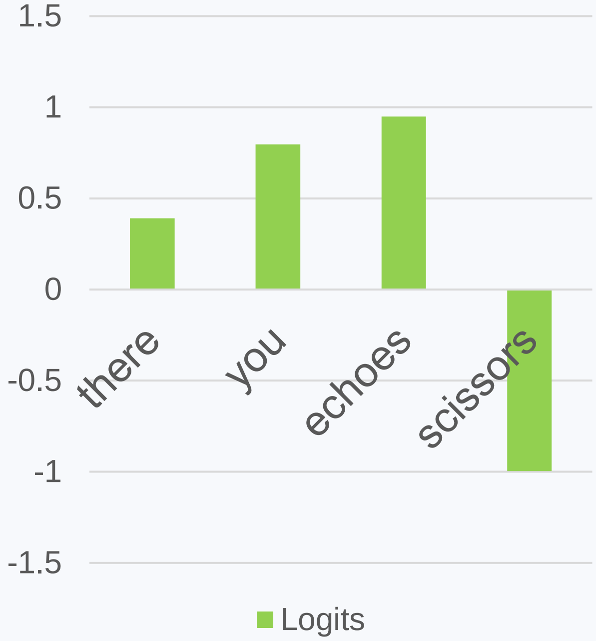

Embedding vectors
Internal representations
Layers (Transformer blocks)
In
the
waste
lands
of
mine
Hidden size
LM head
echoes

Several final steps

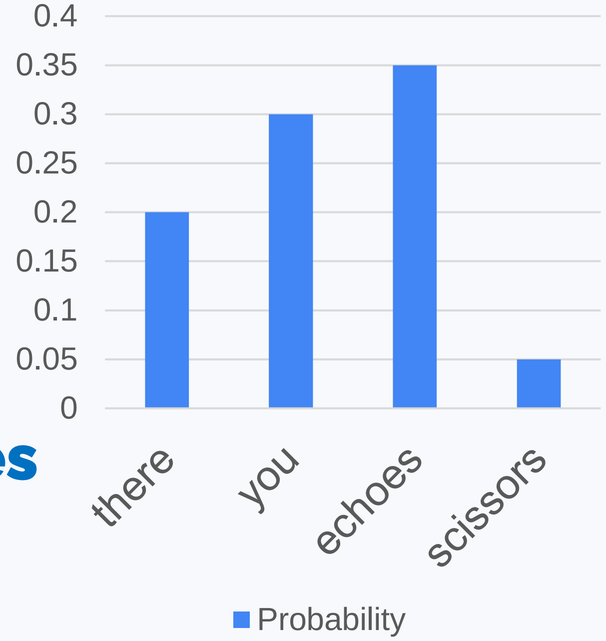
If the vocabulary size is V,
the LM head
produces V logits

In the wastelands of mine

LM head
produces logits



Softmax
produces
probabilities



Sampling
or
Argmax

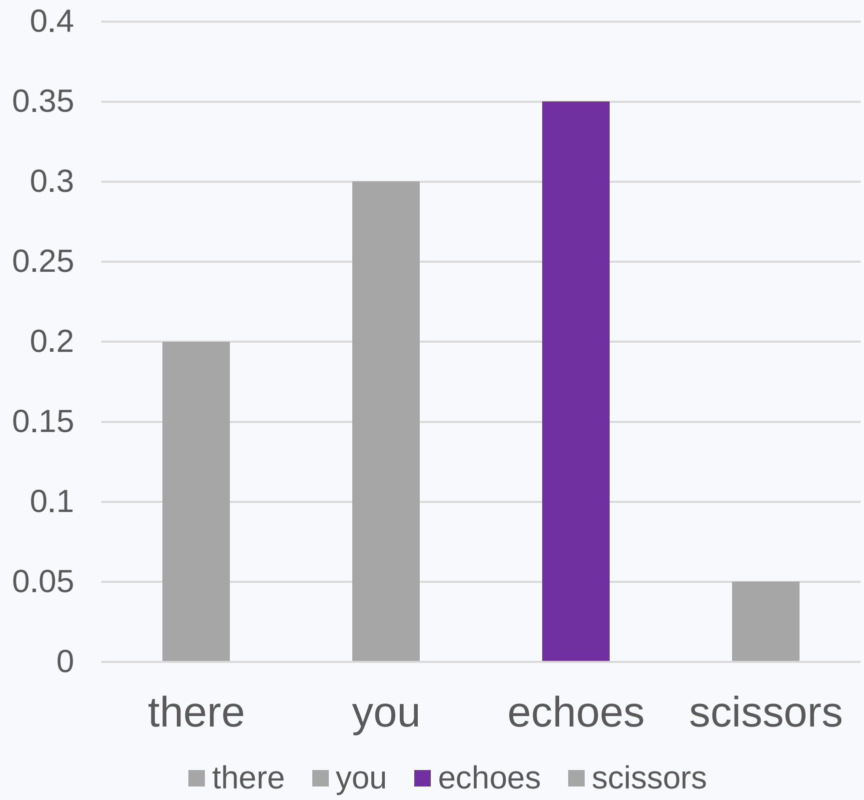
echoes

Sampling – Argmax

The simplest strategy is **Argmax**: always take the most probable token.

Pro:
- Reproducibility,
- Good for tasks with THE right answer.

Cons:
- Can be repetitive,
- Not suitable for “creative” tasks,
- Not for Self-Consistency



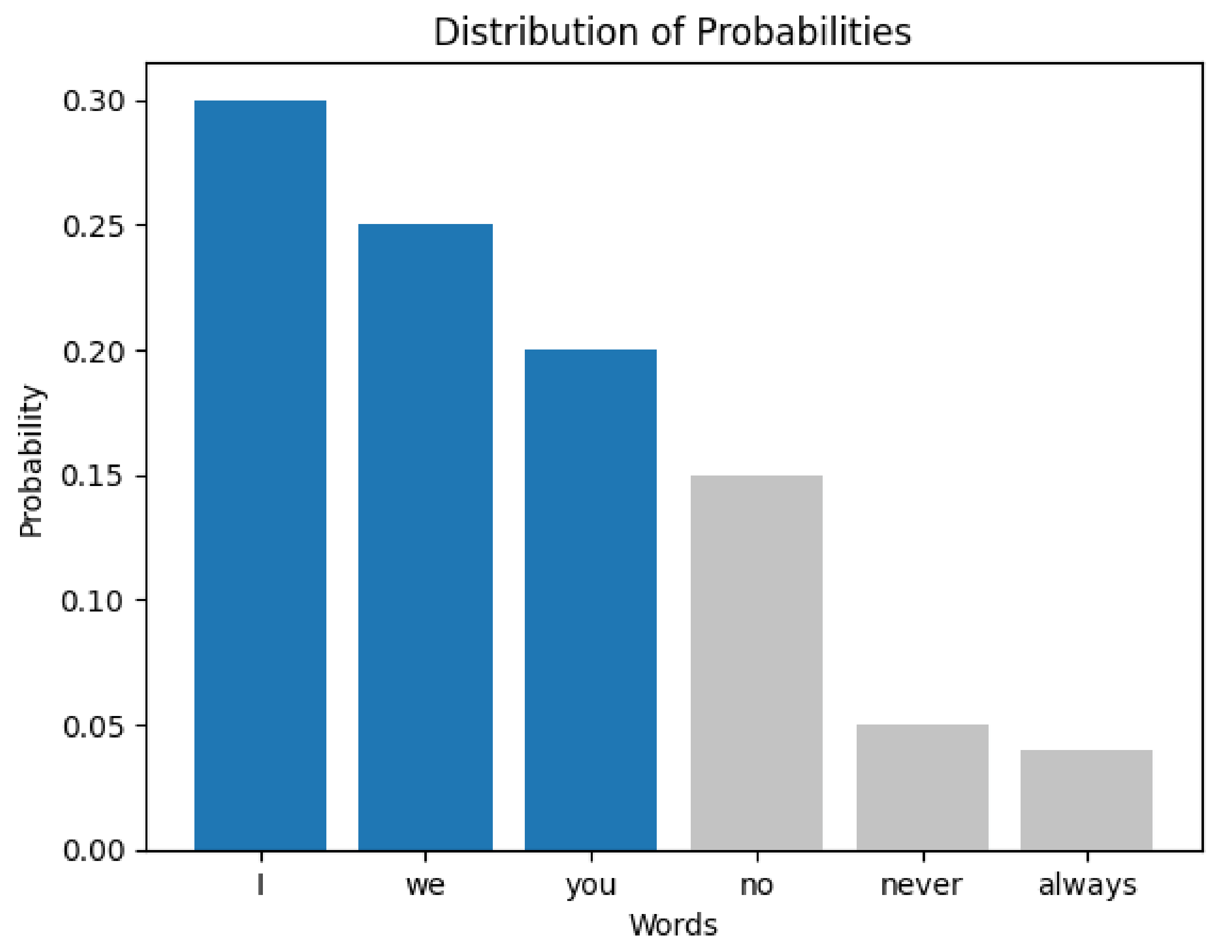
Sampling – Probabilistic

This way, any token may be generated.
But what if we don't want less-probable tokens?

Top-K
Take K most probable tokens,
Renormalize probabilities.

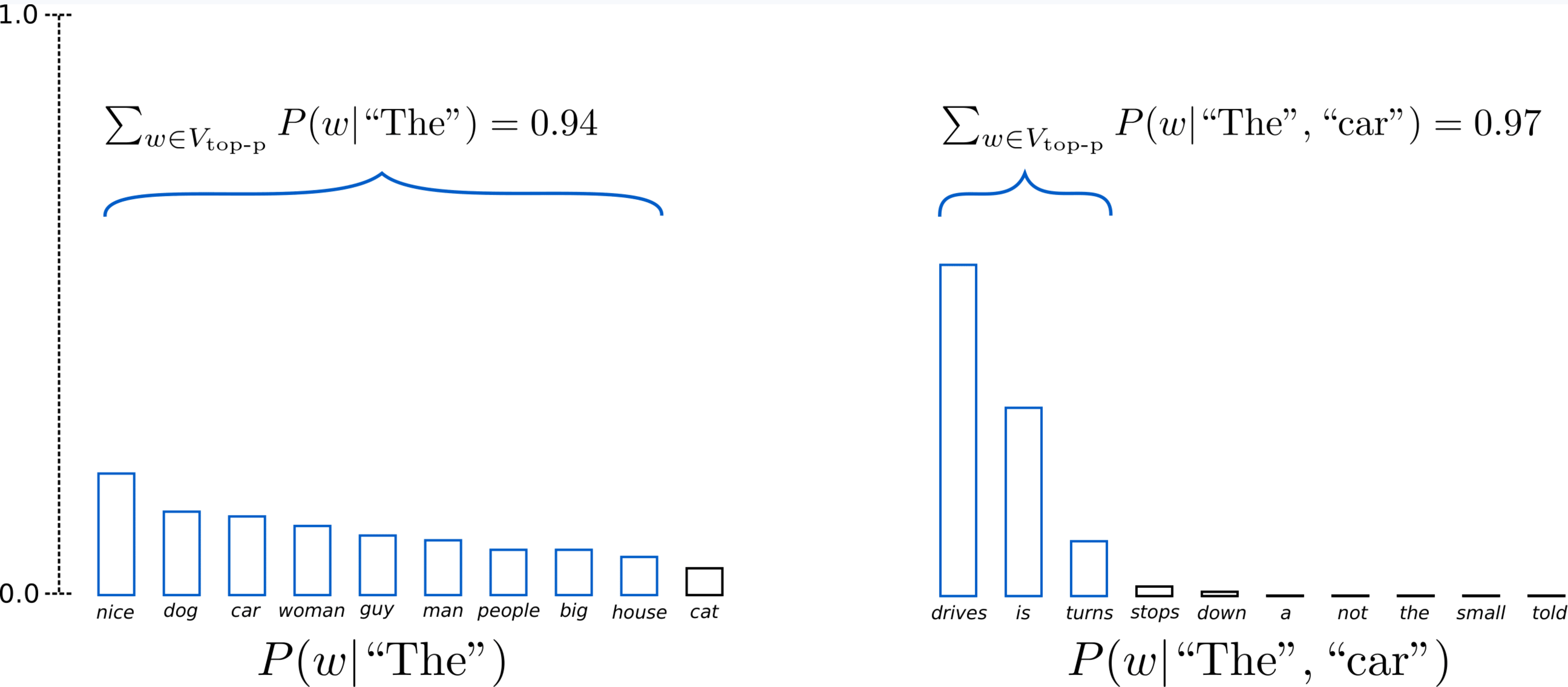
$$(0.3, 0.25, 0.2) \to \left(\frac{0.3}{0.75}, \frac{0.25}{0.75}, \frac{0.2}{0.75}\right) \to$$

$$\to \ (0.4, 0.333333, 0.266667)$$



https://huggingface.co/blog/how-to-generate

Top-p

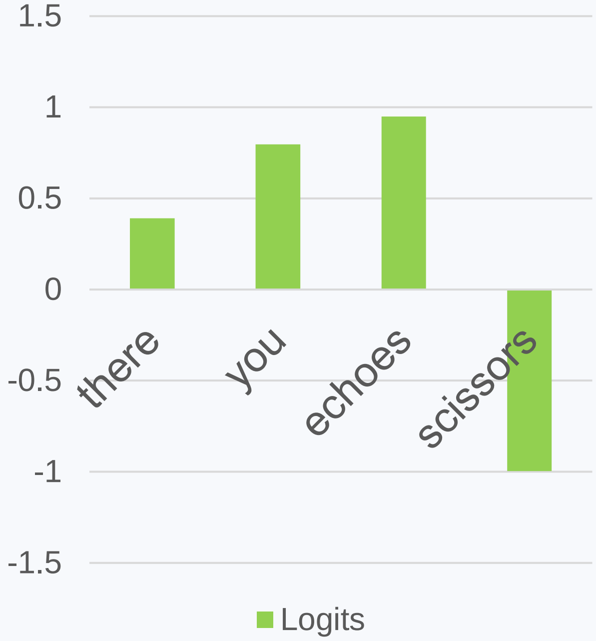


Several final steps

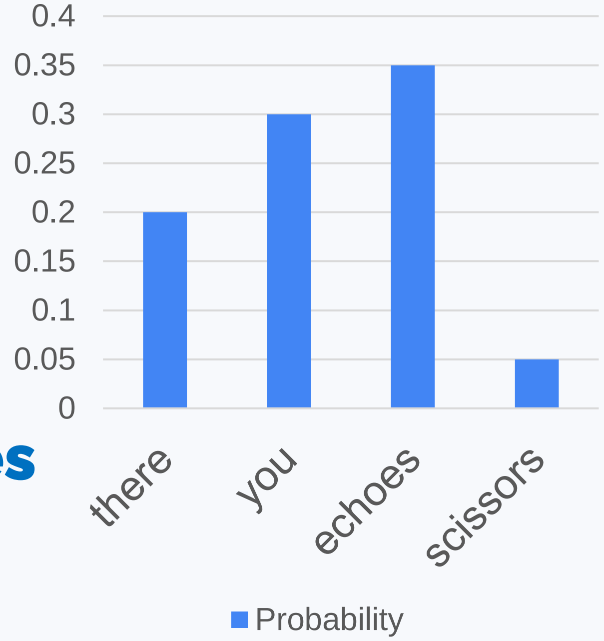
If the vocabulary size is V,
the LM head
produces V logits

In the wastelands of mine

LM head
produces logits



Softmax
produces
probabilities



Sampling
or
Argmax

echoes

Softmax

Turns logits (can be any numbers) into probabilities, that

- Are all non-negative,
- Sum to 1

$$(x_1, \dots, x_V) \rightarrow (e^{x_1}, \dots, e^{x_V}) \rightarrow \left(\frac{e^{x_1}}{\sum_t e^{x_t}}, \dots, \frac{e^{x_V}}{\sum_t e^{x_t}}\right)$$

Make non-negative

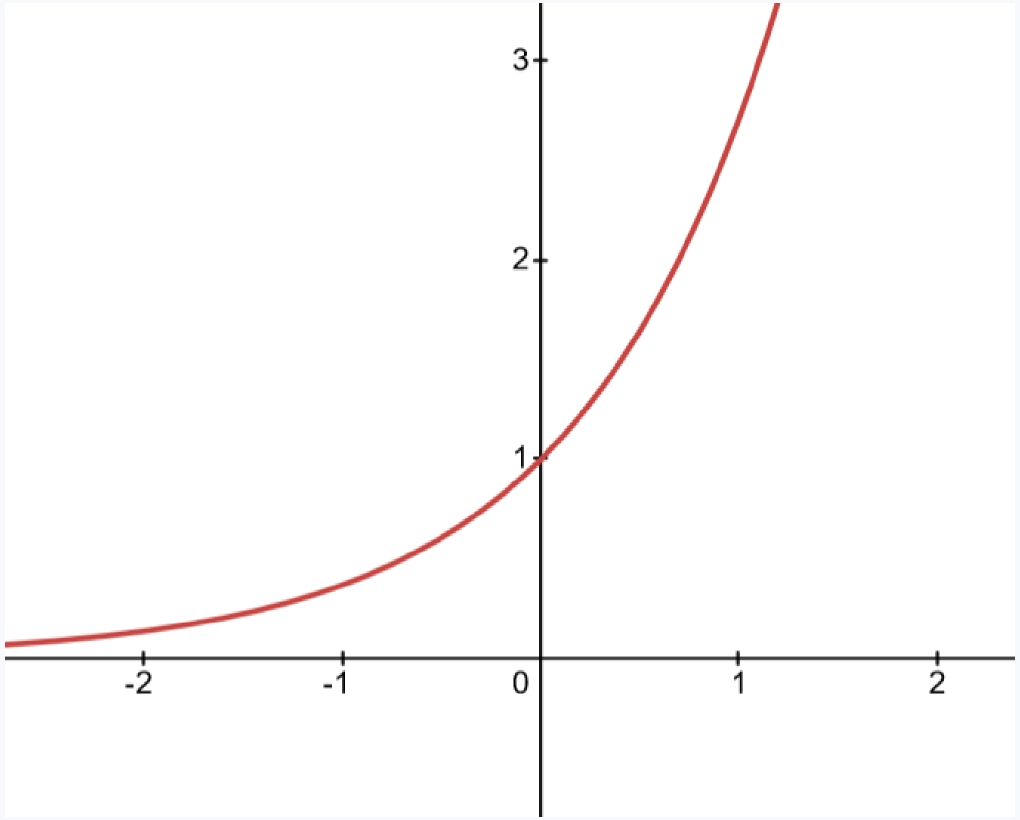
This is softmax

Why exponent and not square?

Exponent is monotonous.

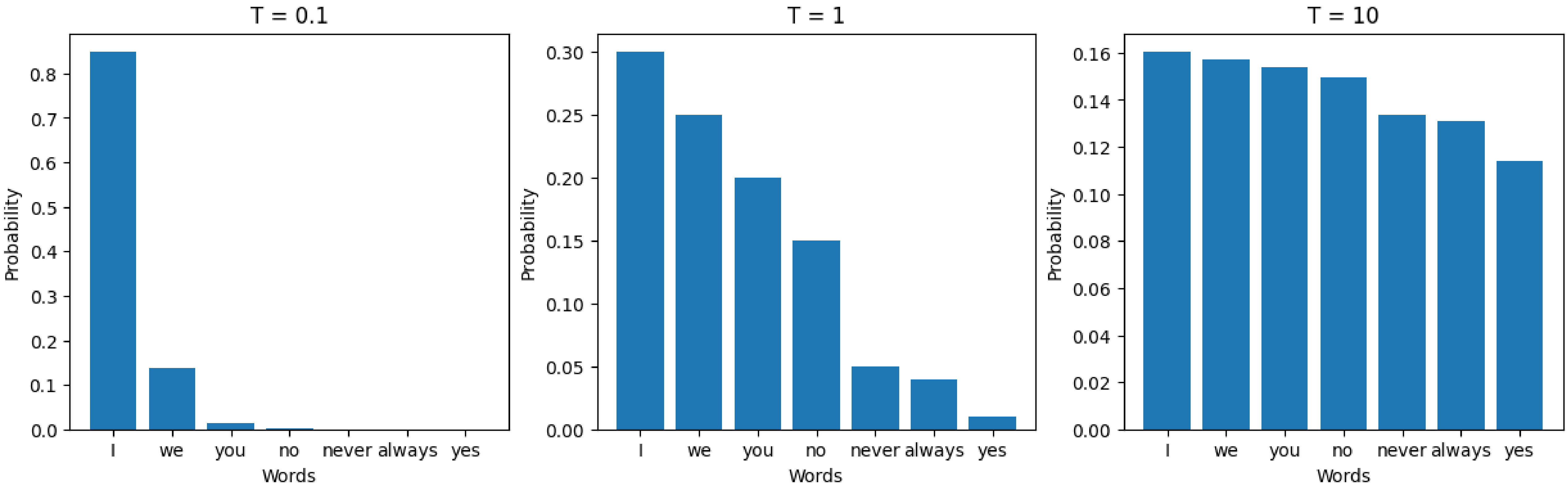
If $logit_u < logit_v$, then

$e^{logit_u} < e^{logit_v}$



Temperature

$$\left(\frac{e^{x_1/T}}{\sum_t e^{x_t/T}}, \ldots, \frac{e^{x_V/T}}{\sum_t e^{x_t/T}}\right)$$

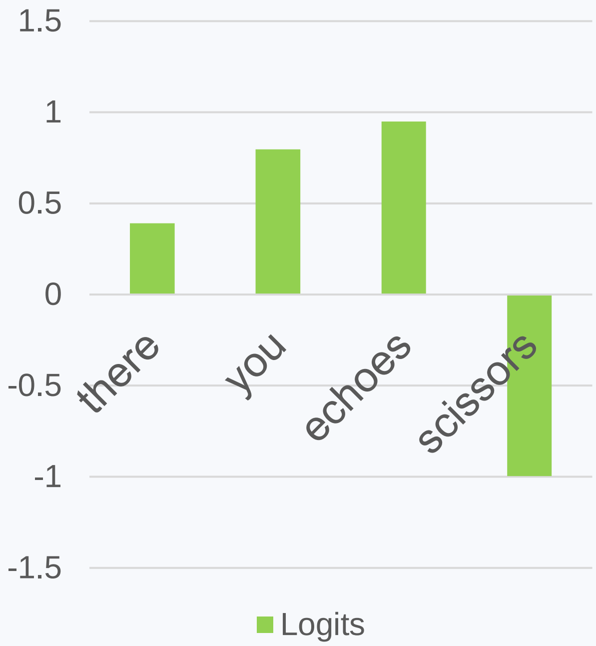


Several final steps

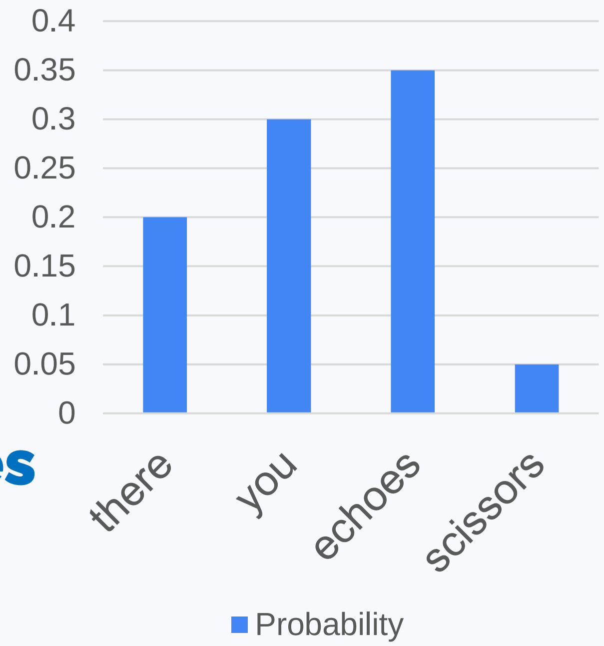
If the vocabulary size is V, the LM head produces V logits

In the wastelands of mine

LM head produces logits



Softmax produces probabilities



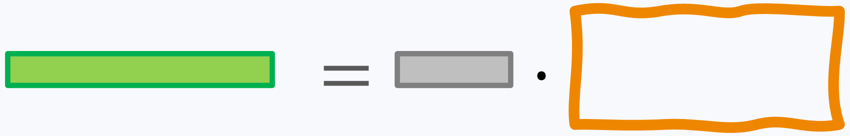
Sampling or Argmax

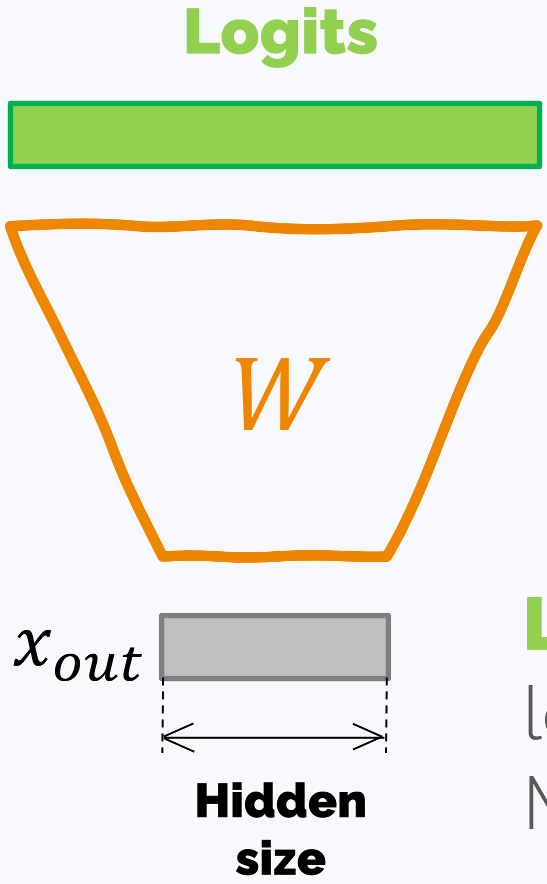
echoes

LM head

Basically, it is
a linear transformation:

$$\textbf{Logits} = x_{out} \cdot W$$

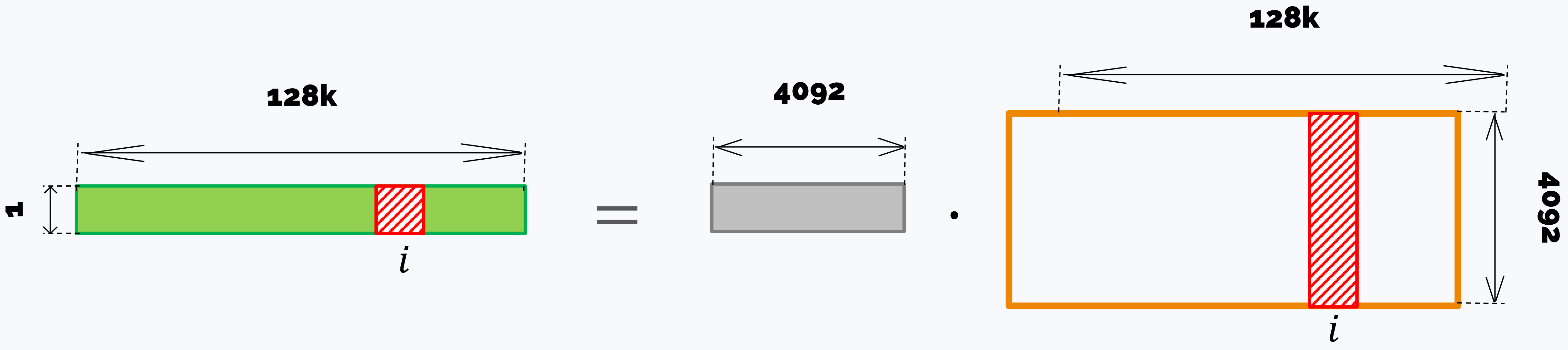




Logits is a vector of length = vocabulary size
May be ~32k (Mistral) or ~128k (Llama 3)

Hidden size is typically kind of 4k, 8k or 12k

A small reminder about matrix multiplication



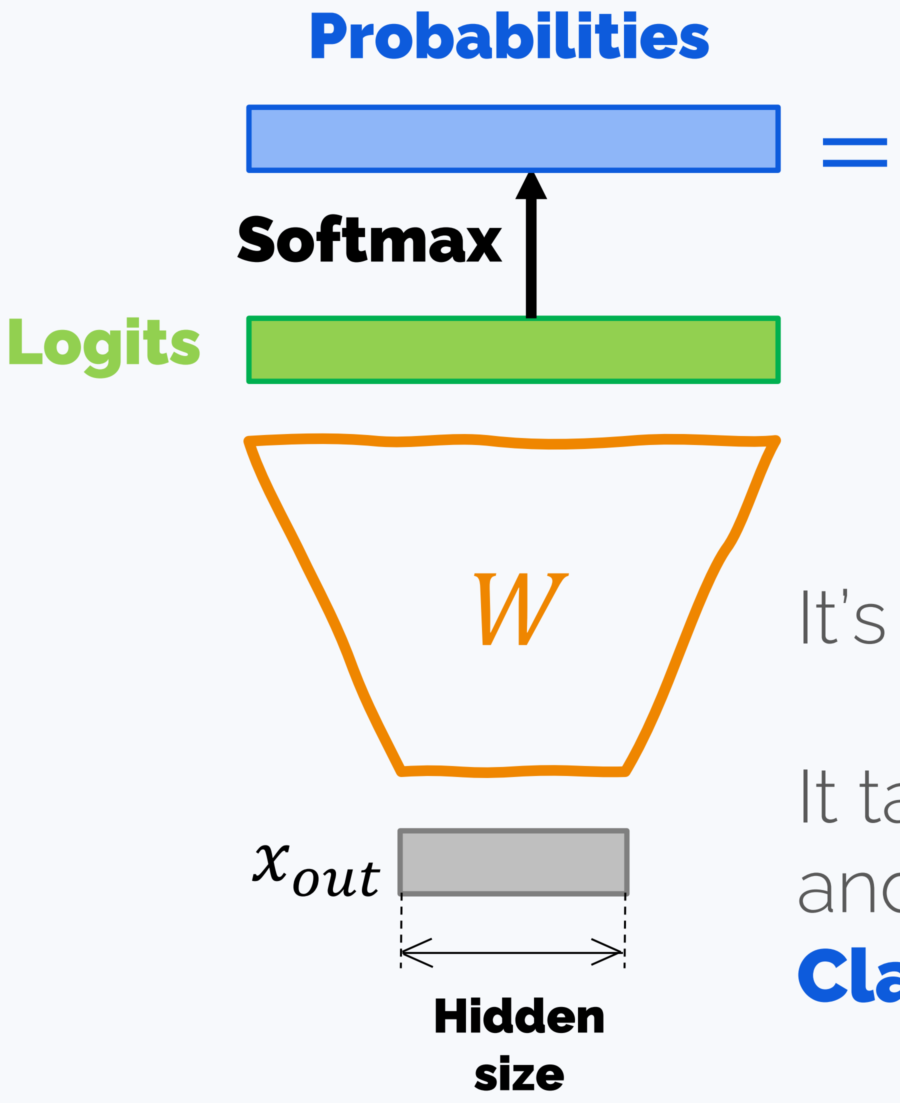
$$\text{Logits}_i = x_1 \cdot w_{1i} + x_2 \cdot w_{2i} + \cdots + x_H \cdot w_{Hi}$$

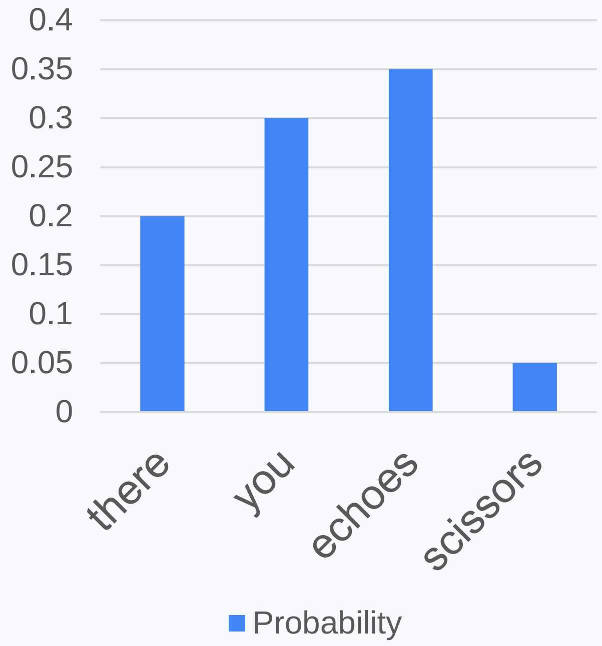
LM head + softmax

Basically, it is
a linear transformation:

$$\textbf{Logits} = x_{out} \cdot W$$

$$\textbf{Probabilities} = \textbf{Softmax}(x_{out} \cdot W)$$





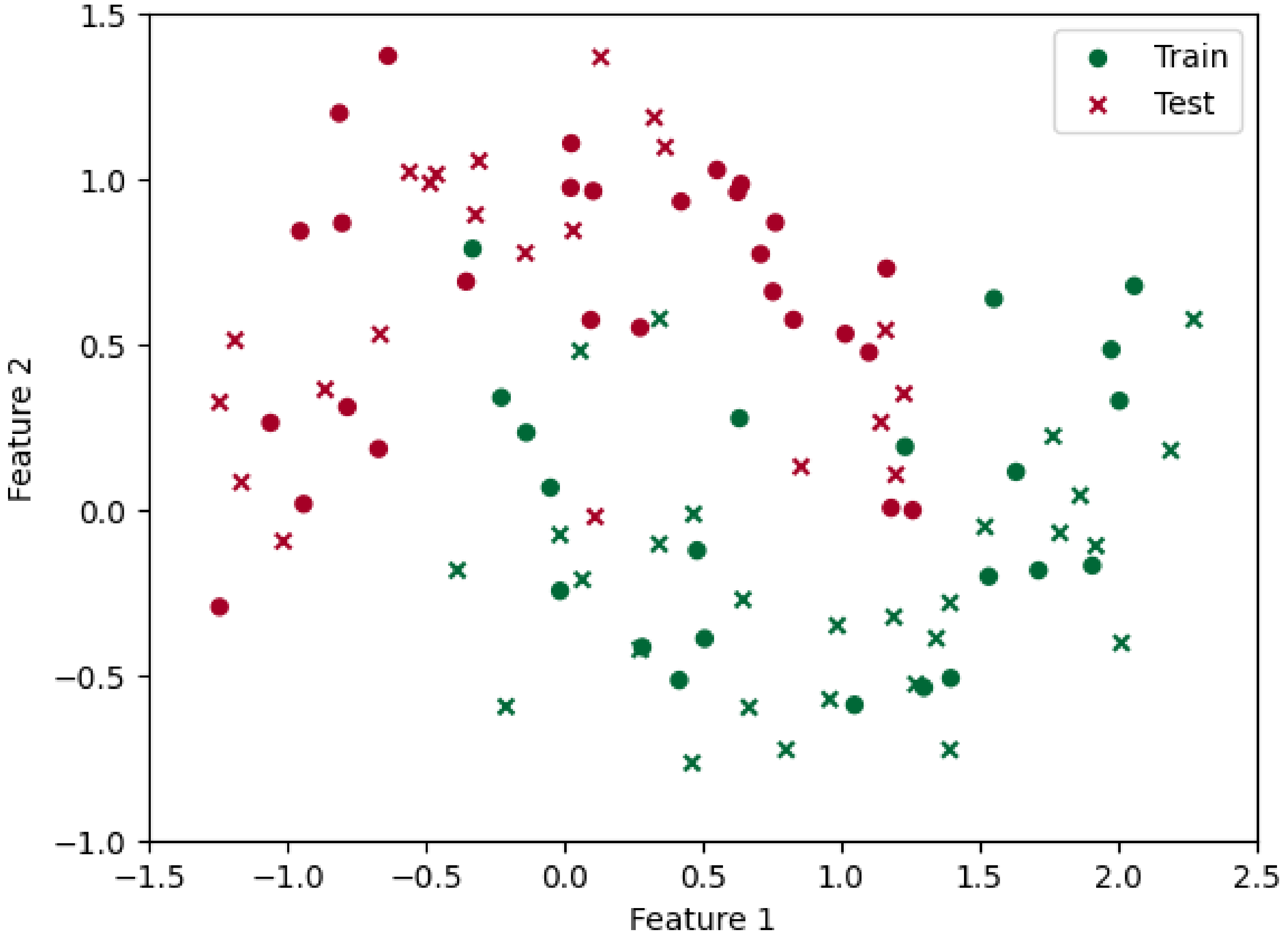
It's a **Linear Classifier**

It takes **Features**
and outputs
Class Probabilities

Binary linear classification

A motivational example

Let's classify 2d points:
red vs green

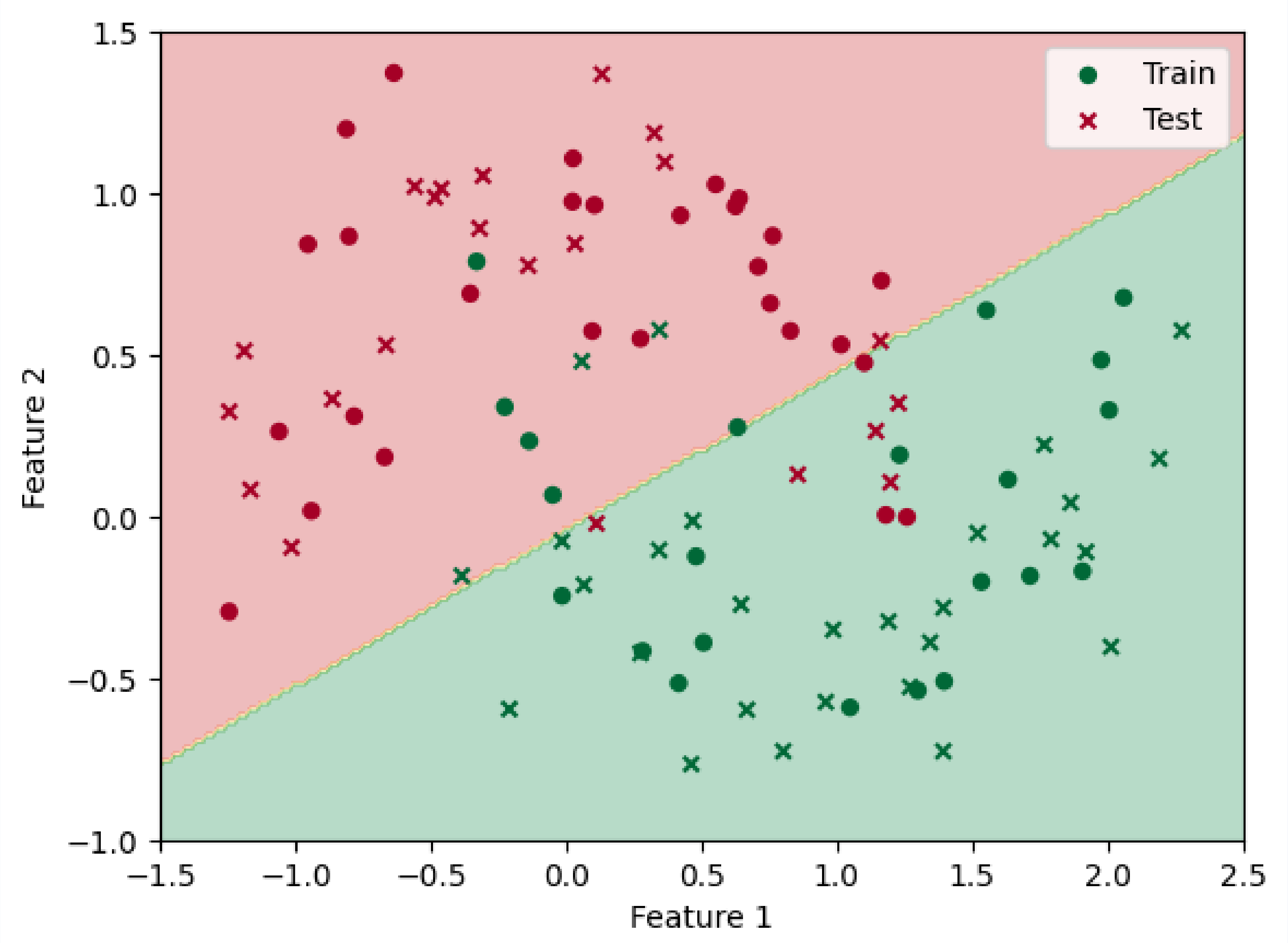


A motivational example

Let's classify 2d points:
red vs green

The simplest
classification rule
is a linear rule

So, let's just find a line
optimally
discriminating
between the two
classes!



The important assumption

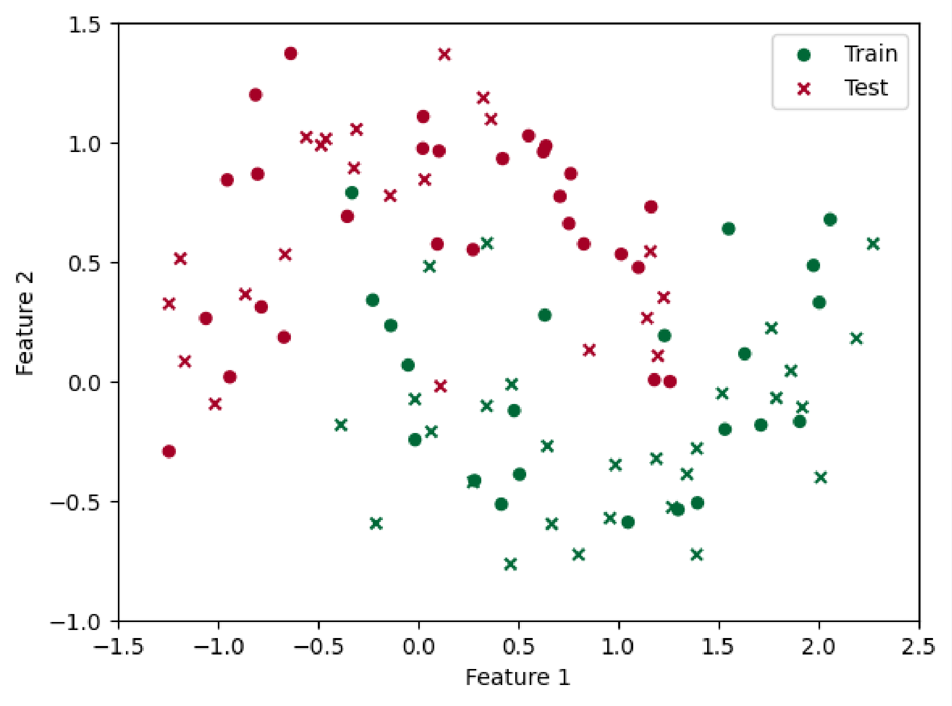
To consider linear models, we need to be sure that **our data is described by numerical features**.

Numerical	**Not numerical**
Income/loss	Job title
Age	Product category
Temperature	Full text review
Pixel intensity (R/G/B)	Is there “18+” in the text

We’ll learn how to deal with non-numerical features a bit later.

Let's add math

Class 0
Class 1



Feature description of an object:

$$x = (x_1, \dots, x_D)$$

x_i – i-th feature value.

In our case it's

$$x = (x_1, x_2)$$

Let's add math

Feature description of an object:

$$x = (x_1, x_2)$$

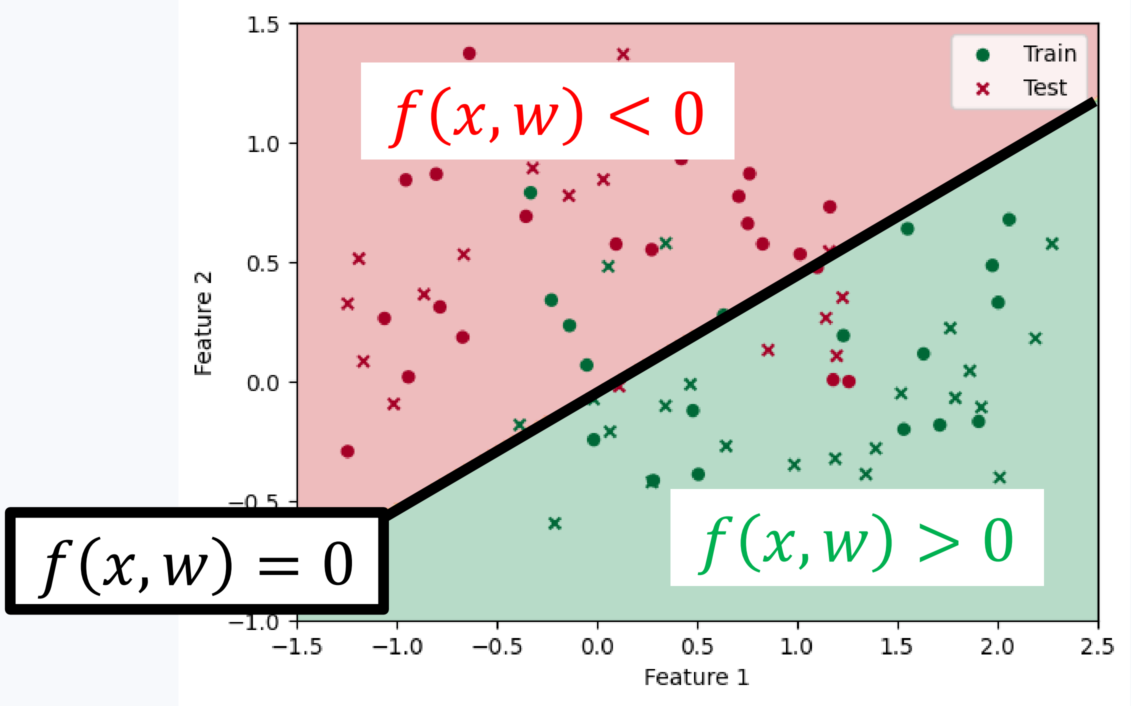
A **bias term** w_0, a **weight vector**:

$$w = (w_1, w_2)$$

A **linear model**:

$$f(x, w) = w_0 + x_1 w_1 + x_2 w_2$$

Class 0
Class 1



Let's add math

Class 0
Class 1

Feature description of an object:

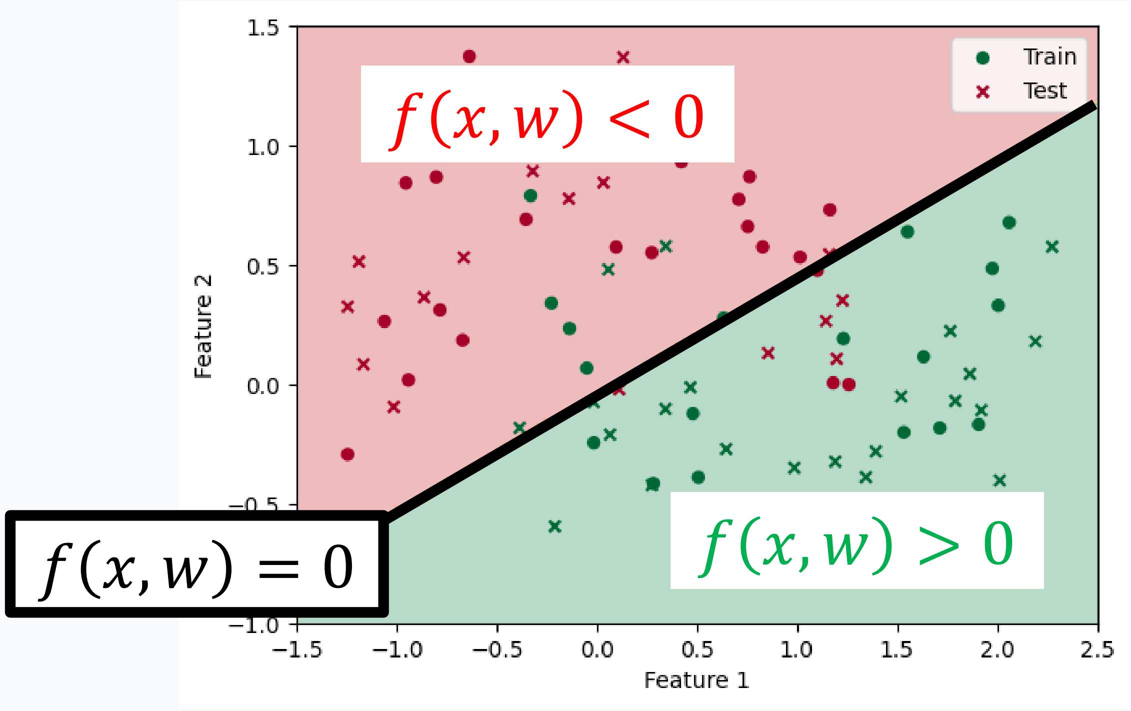
$$x = (x_1, \dots, x_D)$$

A **bias term** w_0, a **weight vector**:

$$w = (w_1, \dots, w_D)$$

A **linear model**:

$$f(x, w) = w_0 + x_1 w_1 + \dots + x_D w_D = w_0 + x w^T$$



$$\boxed{x w^T = (x_1, \dots, x_D) \begin{pmatrix} w_1 \\ \vdots \\ w_D \end{pmatrix}}$$

A word of caution about notation

$$x = (x_1, \ldots, x_D) \qquad w = \begin{pmatrix} w_1 \\ \vdots \\ w_D \end{pmatrix}$$

xw

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_D \end{pmatrix} \qquad w = \begin{pmatrix} w_1 \\ \vdots \\ w_D \end{pmatrix}$$

$x^T w$

$w^T x$

$$x_1 w_1 + \cdots + x_D w_D$$

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_D \end{pmatrix} \qquad w = (w_1, \ldots, w_D)$$

wx

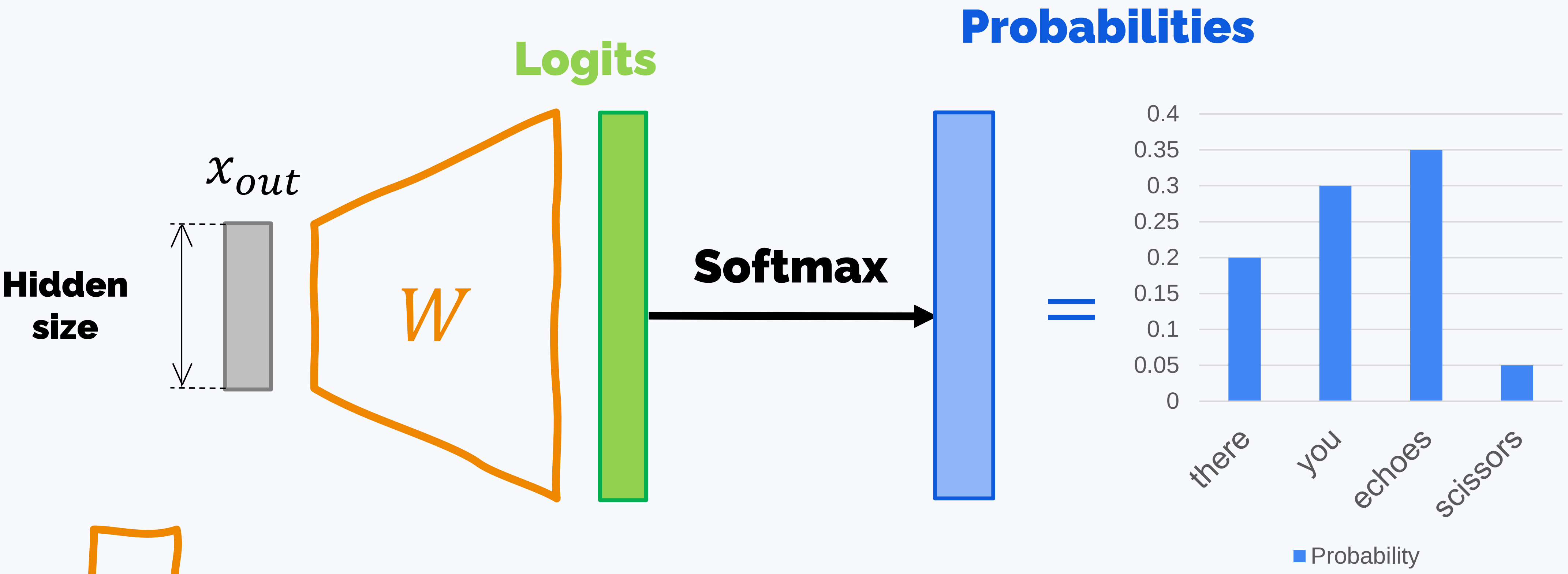
$$x = (x_1, \ldots, x_D) \qquad w = (x_1, \ldots, x_D)$$

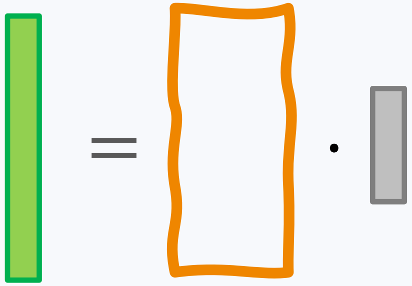
xw^T

$\langle w, x \rangle$

wx^T

In another world





$$\textbf{Probabilities} = \textbf{Softmax}(W \cdot x_{out})$$

My explanations

For x:

	Feature 1	...	Feature D
Object 1	.		
...			
Object N			

For w:

The usual weight shape in Pytorch is:

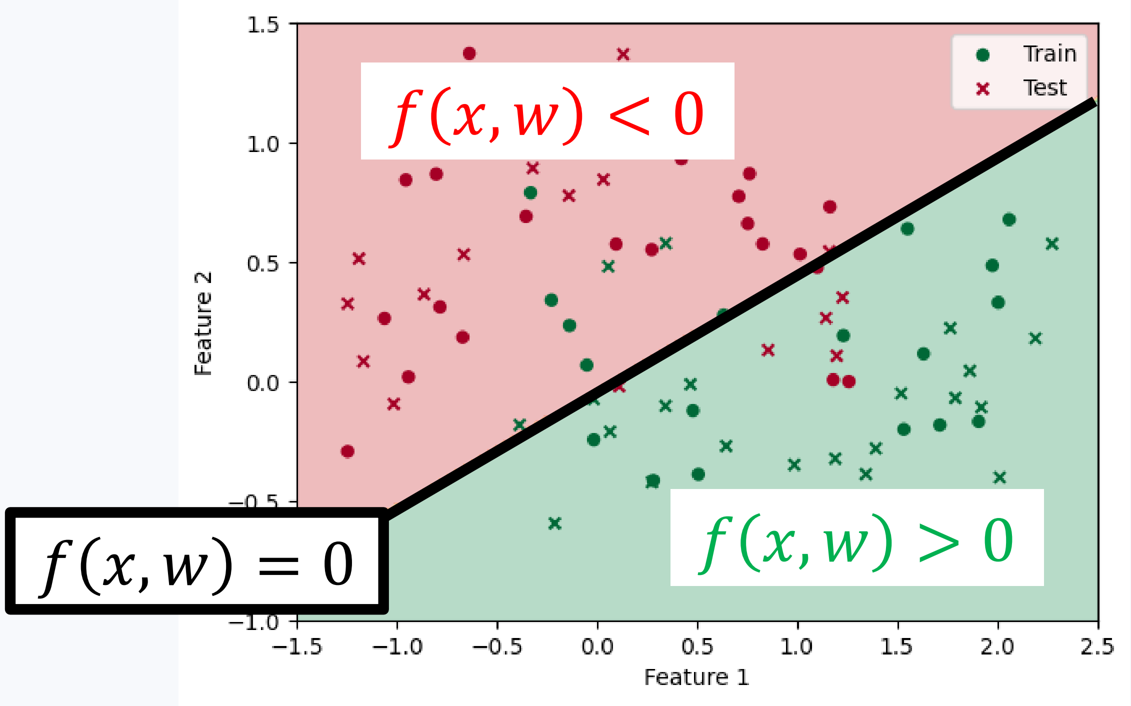
(out_features, in_features)

In our case: (1, D)

Let's add math

Class 0

Class 1



Feature description of an object:

$$x = (x_1, \dots, x_D)$$

A **bias term** w_0, a **weight vector**:

$$w = (w_1, \dots, w_D)$$

A **linear model**:

$$f(x, w) = w_0 + x_1 w_1 + \dots + x_D w_D = w_0 + x w^T$$

$$xw^T = (x_1, \dots, x_D) \begin{pmatrix} w_1 \\ \vdots \\ w_D \end{pmatrix}$$

Let’s get rid of the bias term

Feature description of an object:

$$\tilde{x} = (1, x_1, \dots, x_D)$$

A **bias term** w_0, a **weight vector**:

$$\widetilde{w} = (w_0, w_1, \dots, w_D)$$

A **linear model**:

$$f(x, w) = 1 \cdot w_0 + x_1 w_1 + \dots + x_D w_D = \tilde{x}\widetilde{w}^T$$

Working with features

What to do with non-numerical (categorical) features?

The simplest way is using dummy variables (one-hot encoding):

		Job title	
Atalya	...	Researcher	...
Ivan	...	Plumber	...
Praveen	...	Astronaut	...
Adriana	...	QA	...
Ye	...	Researcher	...

→

		Researcher	**Plumber**	**Astronaut**	
Atalya	...	**1**	0	0	...
Ivan	...	0	**1**	0	...
Praveen	...	0	0	**1**	...
Adriana	...	0	0	0	...
Ye	...	**1**	0	0	...

Take (n_values – 1) dummy variables

$$1 \cdot w_0 + x_r w_r + x_p w_p + x_a w_a + x_q w_q$$

		Researcher	Plumber	Astronaut	QA	
Atalya	...	**1**	0	0	0	...
Ivan	...	0	**1**	0	0	...
Praveen	...	0	0	**1**	0	...
Adriana	...	0	0	0	**1**	...
Ye	...	**1**	0	0	0	...

Sum to:

1
1
1
1
1

$1 \cdot w_0$	$x_r w_r$	$x_p w_p$	$x_a w_a$	$x_q w_q$	**SUM**
w_0	w_r	0	0	0	
w_0	0	w_p	0	0	
w_0	0	0	w_a	0	
w_0	0	0	0	w_q	
w_0	w_r	0	0	0	

Take (n_values – 1) dummy variables

$$1 \cdot (w_0 + a) + x_r(w_r - a) + x_p(w_p - a)$$
$$+ x_a(w_a - a) + x_q(w_q - a)$$

		Researcher	Plumber	Astronaut	QA	
Atalya	...	**1**	0	0	0	...
Ivan	...	0	**1**	0	0	...
Praveen	...	0	0	**1**	0	...
Adriana	...	0	0	0	**1**	...
Ye	...	**1**	0	0	0	...

Sum to:

1
1
1
1
1

$1 \cdot w_0'$	$x_r w_r'$	$x_p w_p'$	$x_a w_a'$	$x_q w_q'$	**SUM**
$w_0 + a$	$w_r - a$	0	0	0	$w_0 + w_r$
$w_0 + a$	0	$w_p - a$	0	0	$w_0 + w_p$
$w_0 + a$	0	0	$w_a - a$	0	$w_0 + w_a$
$w_0 + a$	0	0	0	$w_q - a$	$w_0 + w_q$
$w_0 + a$	$w_r - a$	0	0	0	$w_0 + w_r$

Feature engineering

Two purposes:

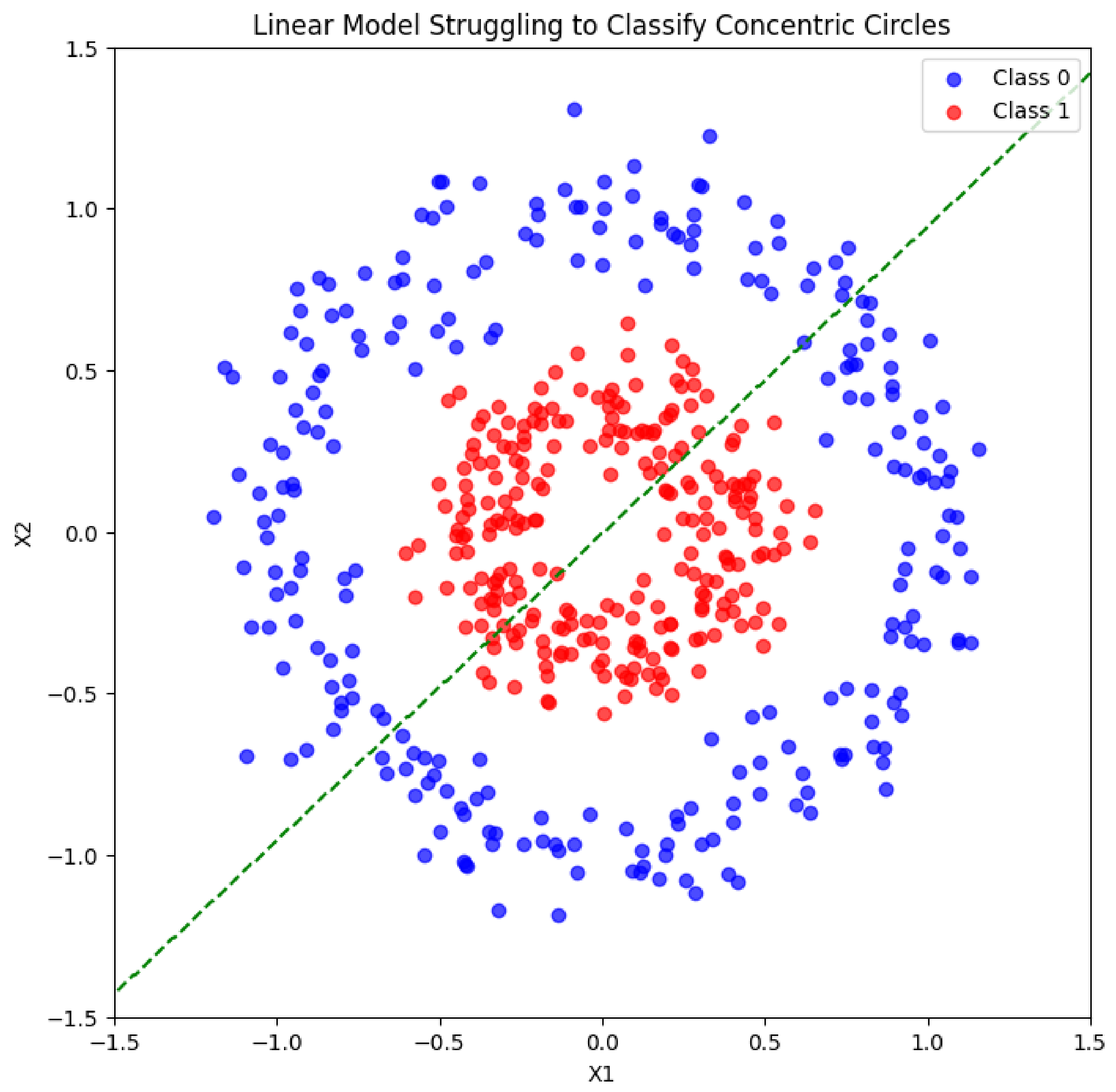
- Convert non-numerical features into smth that can be consumed by an ML model.
  Example: dummy features

- Make feature description of the objects more expressive

Feature expressivity matters

A linear model can't solve this classification task. At the same time, it can be solved by a linear model on advanced feature $x_1^2 + x_2^2$:

$$class = sign(x_1^2 + x_2^2 - 0.6^2)$$

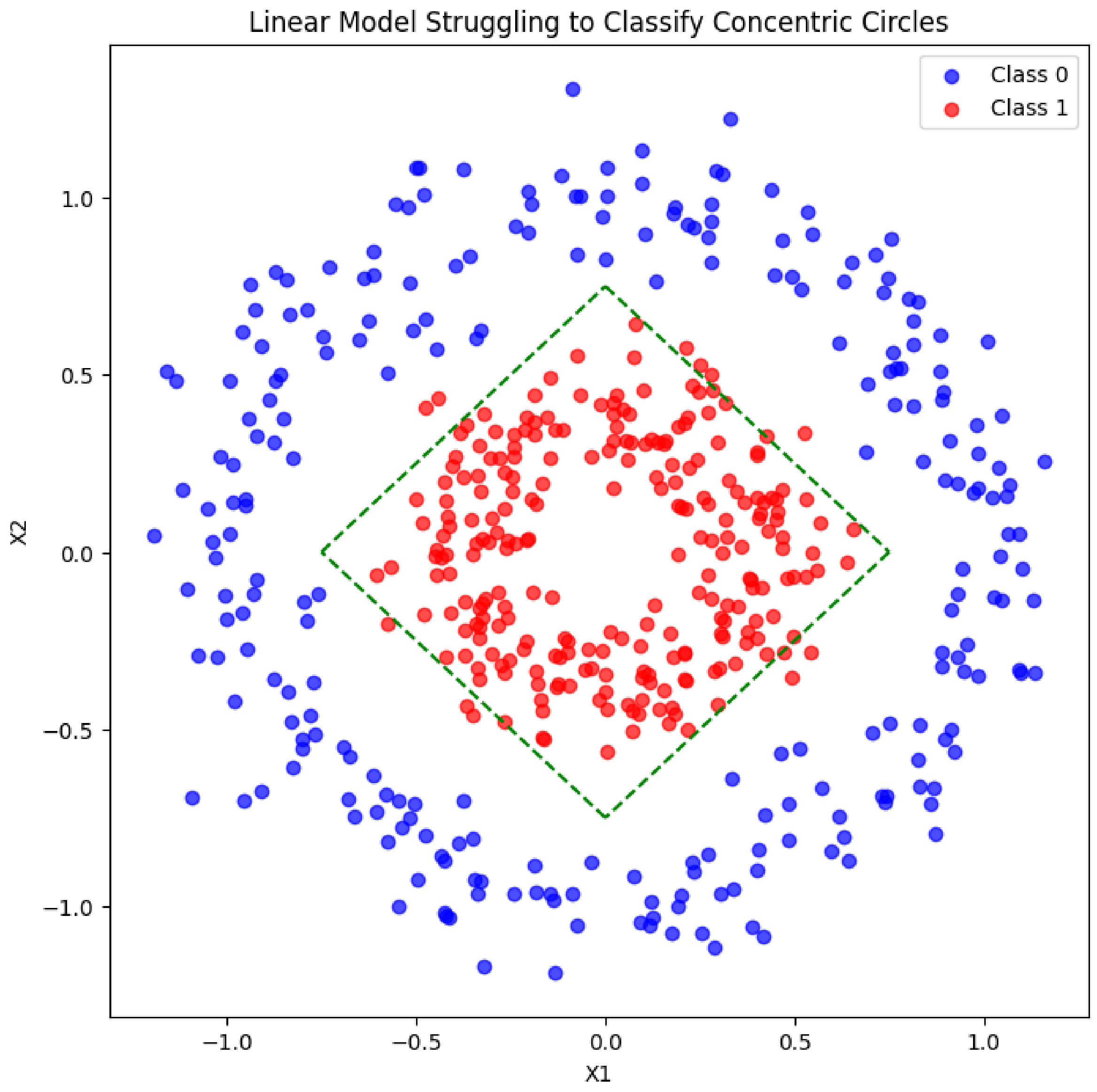


Feature expressivity matters

Or like this

$$\begin{aligned} class = sign(&\max(0, x_1 + x_2 - 0.75) \\ &+ \max(0, x_1 - x_2 - 0.75) \\ &+ \max(0, -x_1 - x_2 - 0.75) \\ &+ \max(0, -x_1 + x_2 - 0.75)) \end{aligned}$$

Classic ML had a whole universe of **feature engineering** methods.



Automating feature engineering

Combinations of linear transformations and nonlinearity can do much:

$$(x_1, x_2) \rightarrow (x_1 + x_2 - 0.75, x_1 - x_2 - 0.75, -x_1 - x_2 - 0.75, -x_1 + x_2 - 0.75)$$

Linear

$$\rightarrow (\max(0, x_1 + x_2 - 0.75), \max(0, x_1 - x_2 - 0.75), \max(0, -x_1 - x_2 - 0.75), \max(0, -x_1 + x_2 - 0.75))$$

Nonlinearity

$$\rightarrow \max(0, x_1 + x_2 - 0.75) + \max(0, x_1 - x_2 - 0.75) + \max(0, -x_1 - x_2 - 0.75) + \max(0, -x_1 + x_2 - 0.75)$$

Linear

Automating feature engineering

Combinations of linear transformations and nonlinearity can do much:

$$(x, y) \rightarrow (w_{10} + w_{11}x_1 + w_{12}x_2, w_{20} + w_{21}x_1 + w_{22}x_2, w_{30} + w_{31}x_1 + w_{32}x_2, w_{40} + w_{41}x_1 + w_{42}x_2)$$

Linear

$$\rightarrow (\max(0, x_1'), \max(0, x_2'), \max(0, x_3'), \max(0, x_4'))$$

Nonlinearity

$$\rightarrow u_1 \max(0, x_1') + u_2 \max(0, x_2') + u_3 \max(0, x_3') + u_4 \max(0, x_4')$$

Linear

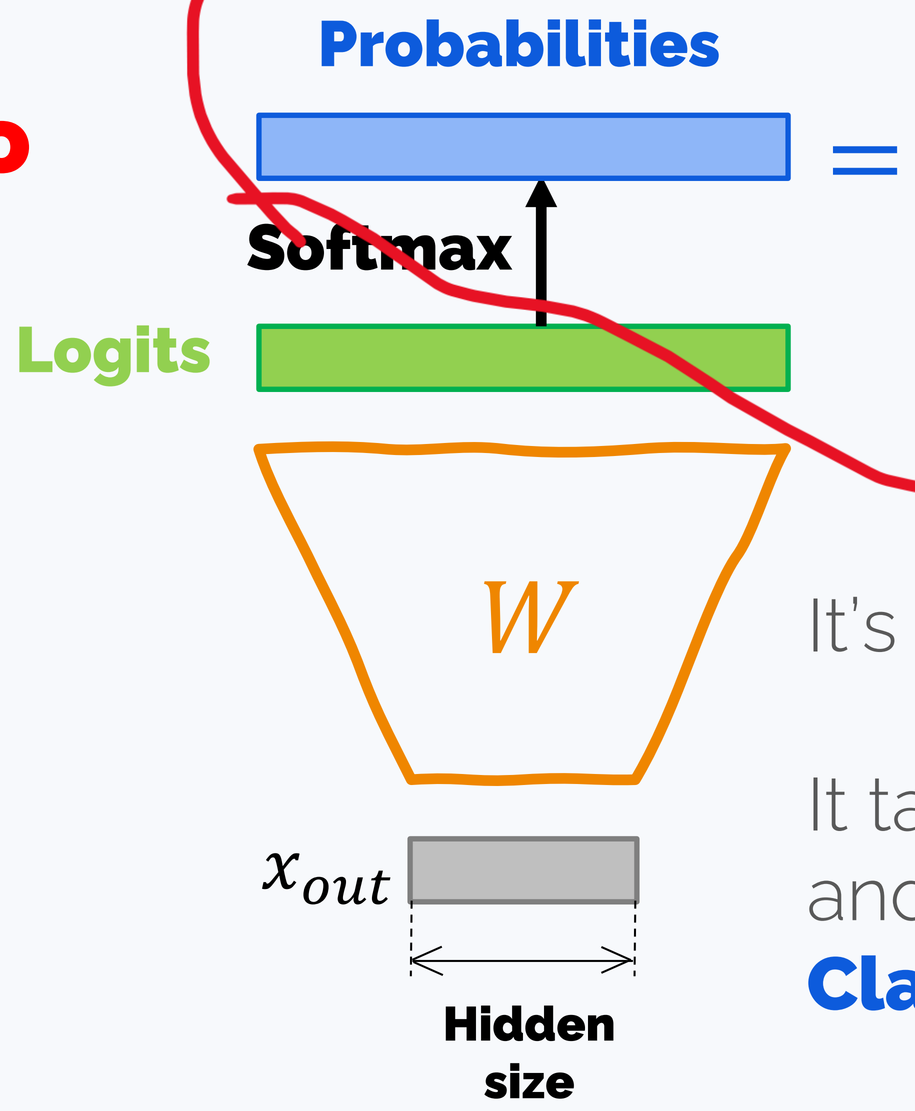
Class probabilities in binary classification and where to get them

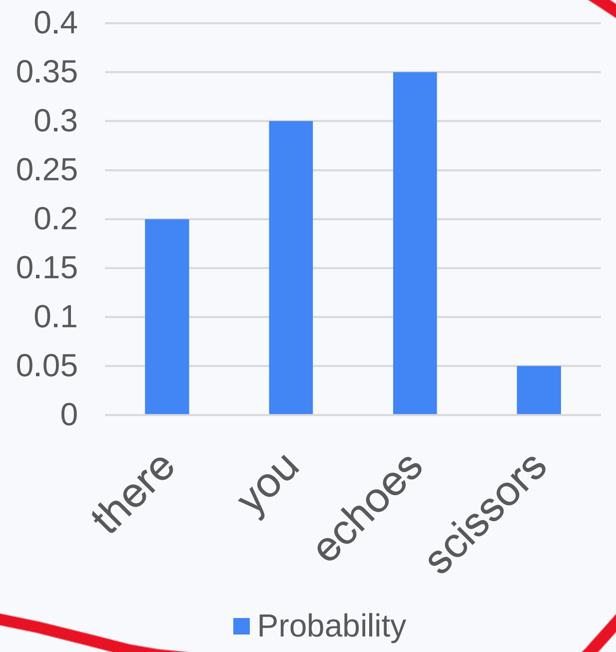
Class probabilities

We need them to sample the next token!

$$\text{Logits} = x_{out} \cdot W$$

$$\text{Probabilities} = \text{Softmax}(x_{out} \cdot W)$$





It's a **Linear Classifier**

It takes **Features** and outputs **Class Probabilities**

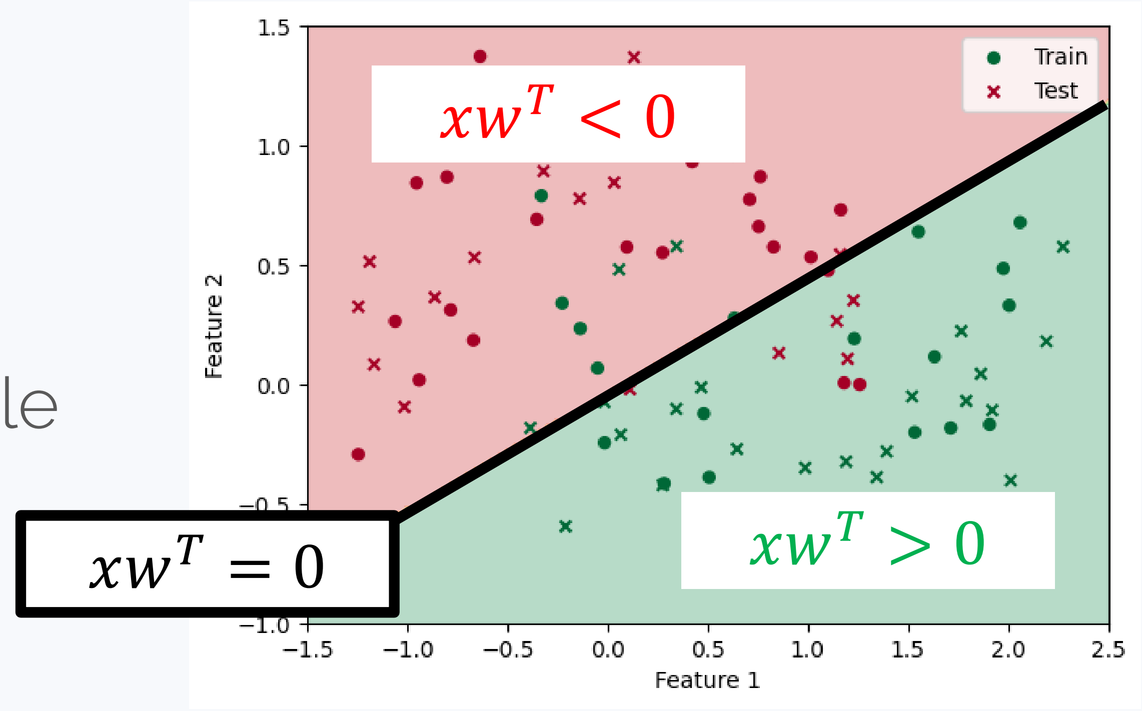
Class 0
Class 1

Logits in binary case

A **logit** is

$$f(x, w) = xw^T$$

This is like a "class 1 affiliation score", while "class 0 score is" $-xw^T$.



$$xw^T = (x_1, \dots, x_D)\begin{pmatrix} w_1 \\ \vdots \\ w_D \end{pmatrix}$$

Let's use softmax

Softmax turns logits xw^T into probabilities. In binary case:

$$x \to (-xw^T, xw^T) \to \left(\frac{e^{-xw^T}}{e^{-xw^T}+e^{xw^T}}, \frac{e^{xw^T}}{e^{-xw^T}+e^{xw^T}}\right) =$$

$$= \left(whatever, \frac{1}{e^{-xw^T-xw^T}+1}\right)$$

This is class 1 probability

Class 1 probability in binary case

$$x \to \left(whatever, \frac{1}{e^{-xw^T - xw^T} + 1} \right) =$$

$$= \left(whatever, \frac{1}{1 + e^{-2xw^T}} \right)$$

This is class 1 probability

Class 1 probability in binary case

$$\left(whatever, \frac{1}{1+e^{-xw^T}}\right)$$

This is class 1 probability

$$\| $$

$$\sigma(x)$$

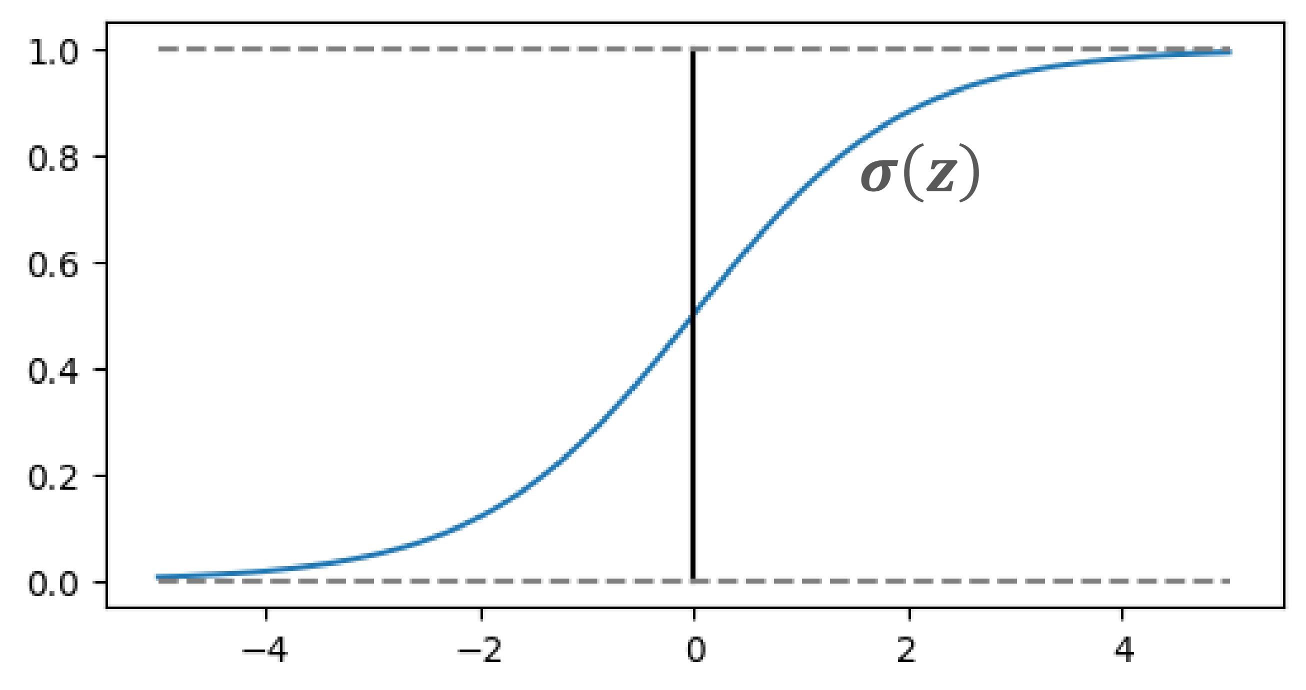
sigmoid function

Logistic regression

Logistic regression is a particular type of linear classifier:

$$Class\ 1\ probability = \sigma(w_0 + xw^T)$$

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



Logistic regression

Logistic regression is a particular type of linear classifier:

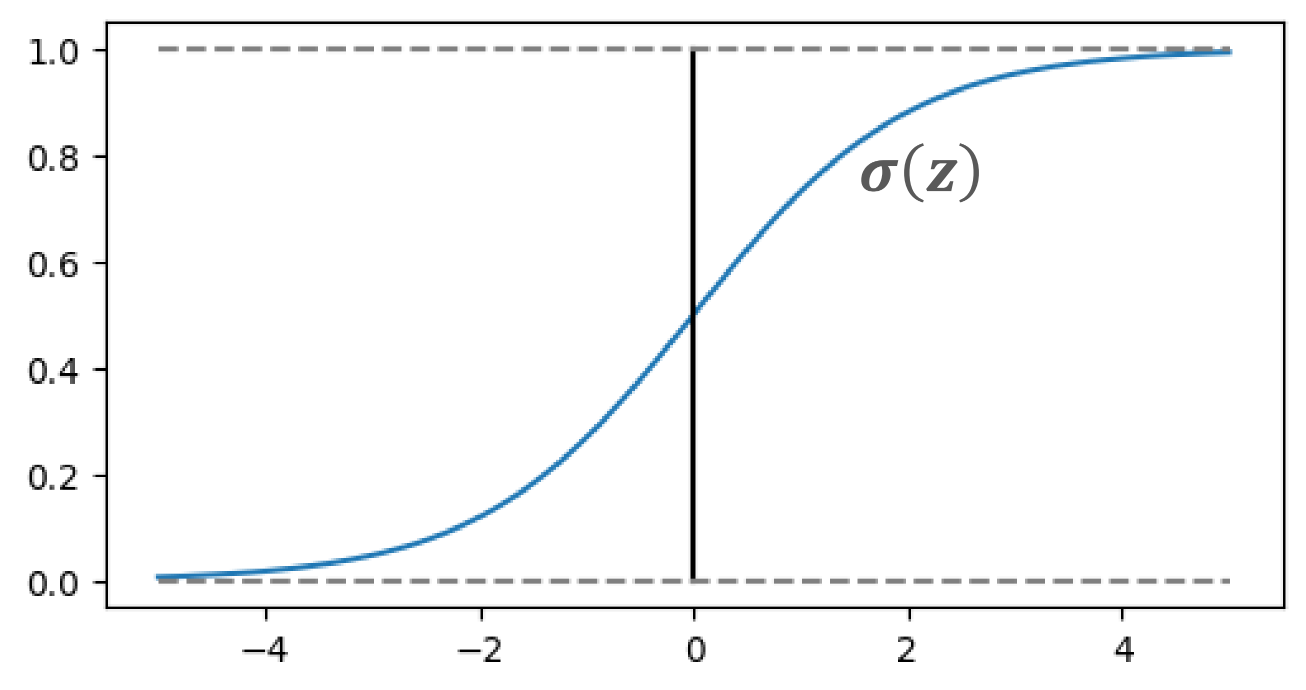
$$Class\ 1\ probability = \sigma(w_0 + xw^T)$$

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

An important property:

$$1 - \sigma(z) = \sigma(-z)$$

It's class 0 probability



But what does class probability mean?

Today I woke up…

Token	**p_pred**
tired	0.3
happy	0.2
at	0.4
not	0.05
…	

But what does class probability mean?

A data point is either class 0 or class 1, how can we say that it has probability 0.3?

A naïve version:

Take many data points with the same feature description as x. The ratio of class 1 among them is class 1 probability for x.

No, that doesn't work.

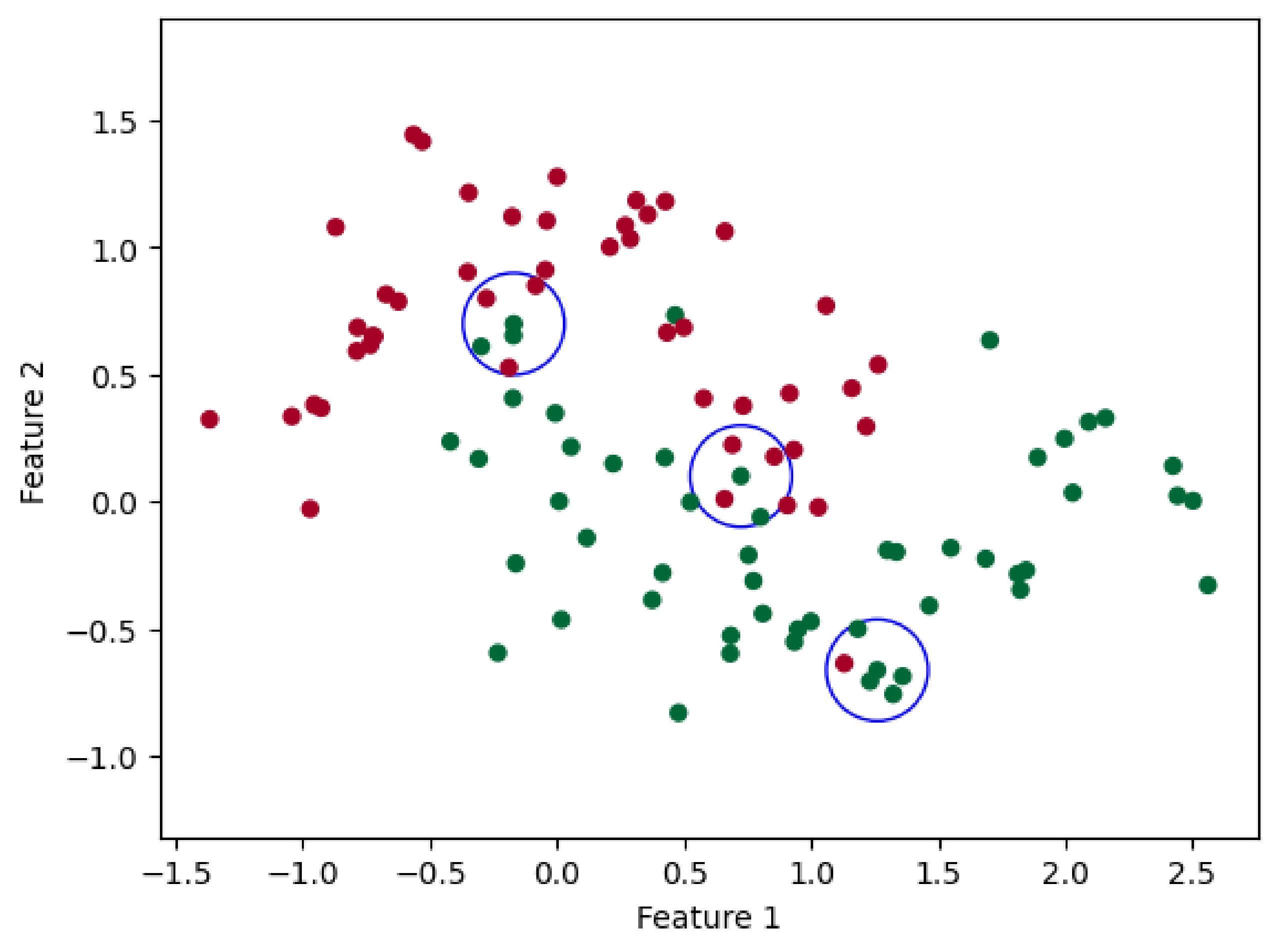
But what does class probability mean?

A data point is either class 0 or class 1, how can we say that it has probability 0.3?

These three points are both class 1.

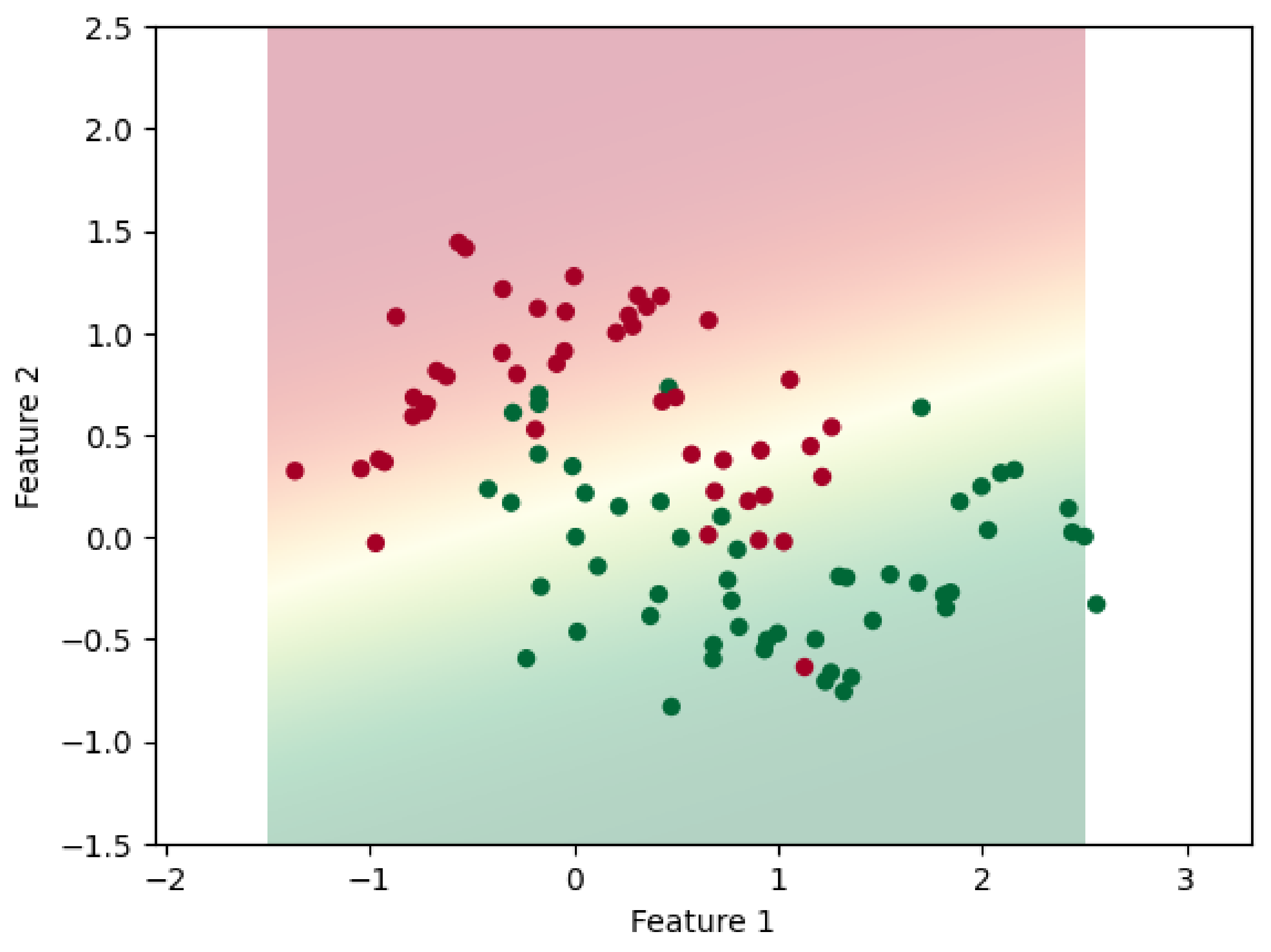
But in their neighborhoods, we have different probabilities of class 1:

Top: 0.5
Middle: 0.4
Bottom: 5/6

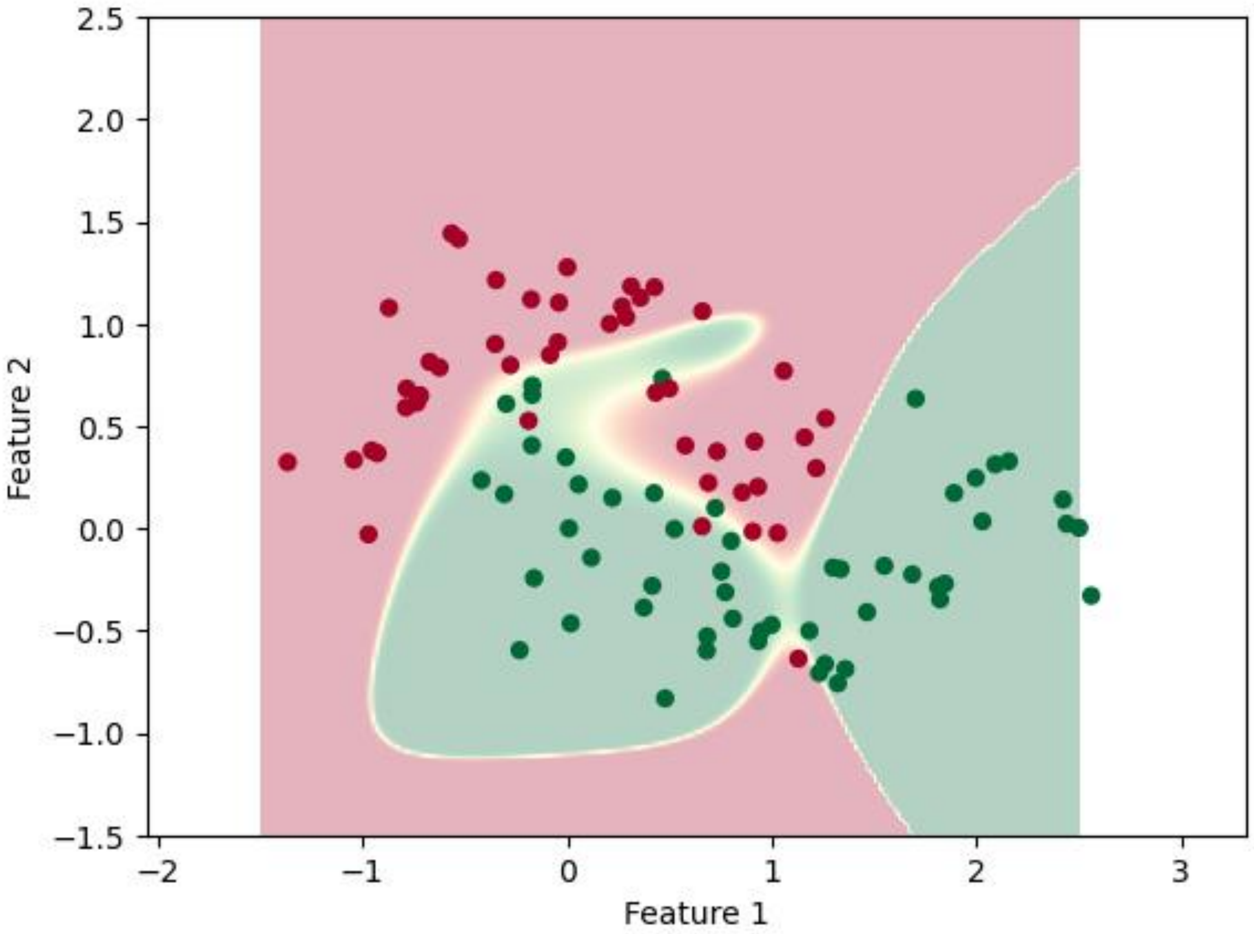


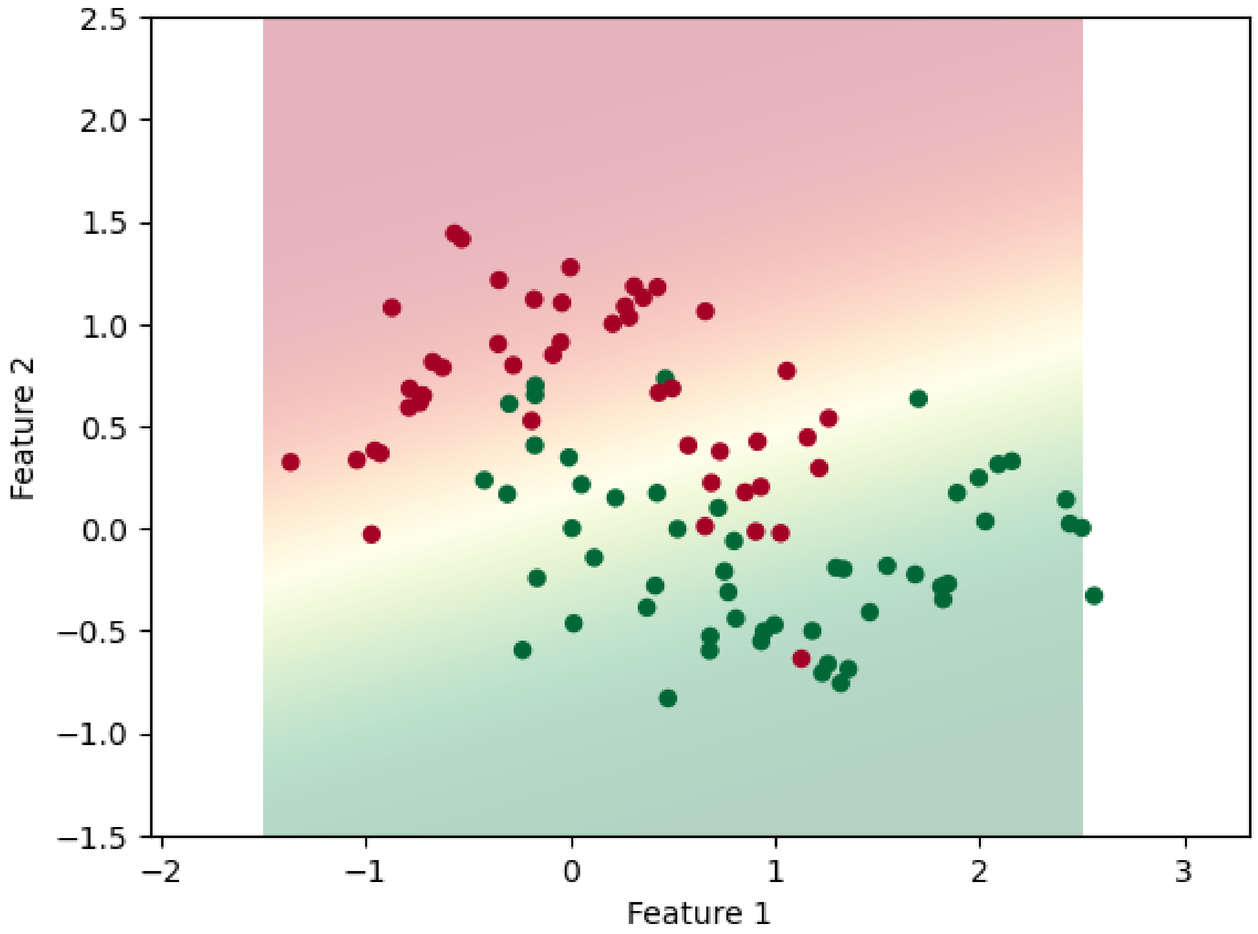
Does logistic regression predict correct probabilities?

Well, it tries, but it is not guaranteed to succeed.

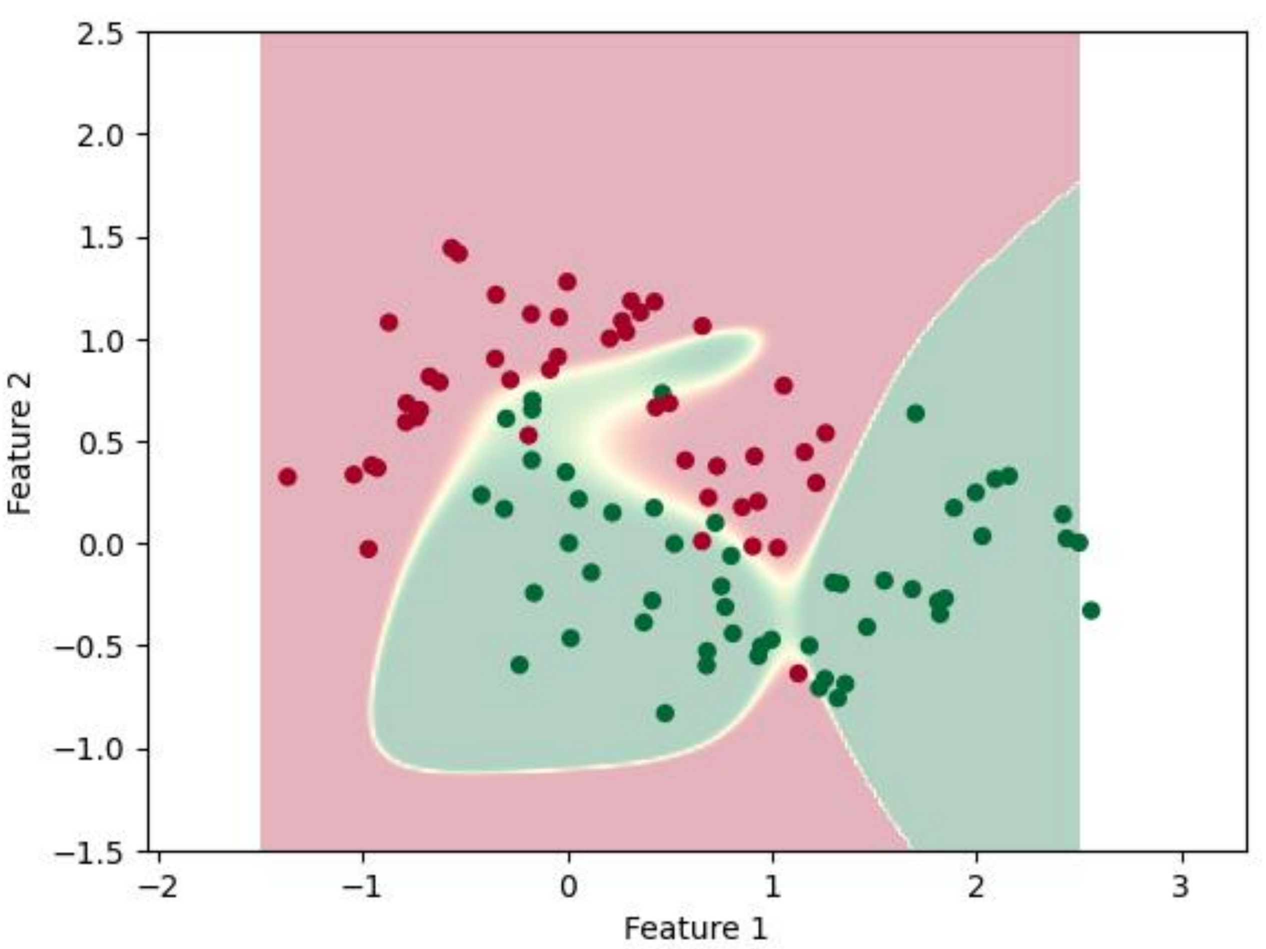


Does logistic regression predict correct probabilities?





Does logistic regression predict correct probabilities?



This illustrates **overfitting**:

the model does a great job on training data, but the rule is unnatural and wouldn't generalize to test data.

Moreover, the model is **overconfident**

Choosing the right threshold

We classify x as class 1 if

$$f(x, w) \geq 0$$

Choosing the right threshold

We classify x as class 1 if

$$f(x, w) \geq \theta$$

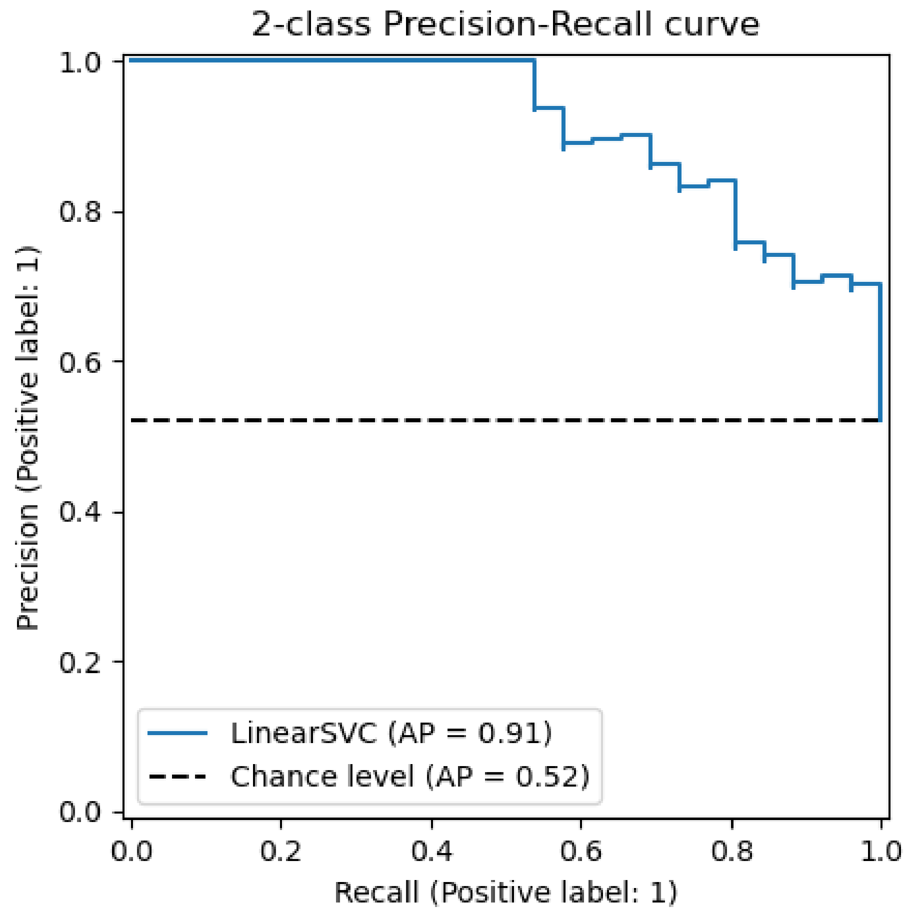
Choosing the right threshold

We classify x as class 1 if

$$f(x, w) \geq \theta$$

We can use Precision-Recall curve to choose the right θ.

But we should do it on a separate **validation dataset**, not on the test data.



The magic square

Notation:

Class 1 = Nasty (**positive**)

Class 0 = Common cold (**negative**)

Classified by the model as…

And it is…

	Classified as: Class 1	Classified as: Class 0
Class 1	**True Positive (TP)**	**False Negative (FN)**
Class 0	**False Positive (FP)**	**True Negative (TN)**

Recall

Class 1 = Nasty virus (**positive**)

Class 0 = Common cold (**negative**)

$$\frac{TP}{TP + FN}$$

This metric is called **recall**

	Classified as: Class 1	Classified as: Class 0
Class 1	**True Positive (TP)**	**False Negative (FN)**
Class 0	**False Positive (FP)**	**True Negative (TN)**

Precision

Class 1 = Nasty virus (**positive**)

Class 0 = Common cold (**negative**)

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

This metric is called **precision**

`	Classified as: Class 1	Classified as: Class 0
Class 1	**True Positive (TP)**	**False Negative (FN)**
Class 0	**False Positive (FP)**	**True Negative (TN)**

AUC ROC

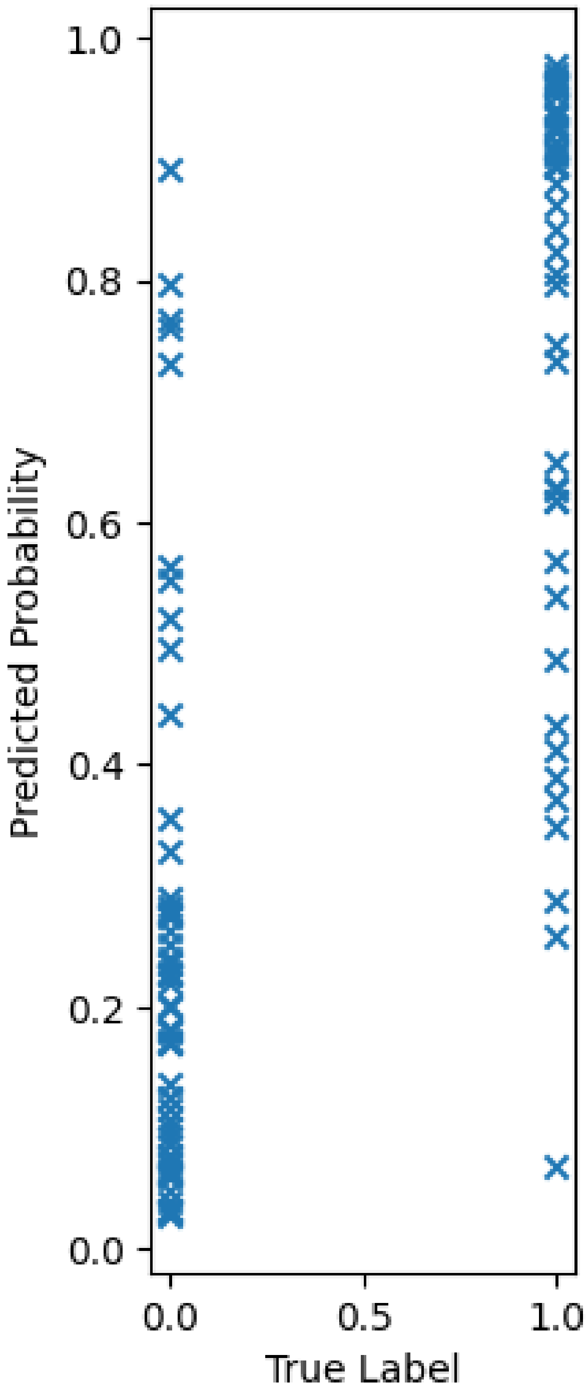
Can we score a classifier before we actually choose a threshold?

AUC ROC

Can we score a classifier before we actually choose a threshold?

$$AUC\ ROC = \frac{\#Pairs\ with\ \hat{p}(x_{class\ 0}) < \hat{p}(x_{class\ 1})}{\#All\ pairs\ (x_{class\ 0}, x_{class\ 1})}$$

*Pairs with $\hat{p}(x_{class\ 0}) = \hat{p}(x_{class\ 1})$ are counted as half a pair each.



How to train a model

Two parts of the answer

We need to somehow get w:

$$f(x, w) = xw^T$$

Part 1: Loss function. We're searching for an optimal w, and the loss function tells us **what** we want to optimize.

$$\mathcal{L}(X_{train}, y_{train}, w_{better}) > \mathcal{L}(X_{train}, y_{train}, w_{worse})$$

Part 2: Optimization method. Tells us **how** do we find the optimal w. [next week]

Loss functions for linear models

What is a loss function

y_i – true answers, $\hat{y}_i$ – predicted answers

A loss function

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, \hat{y}_i)$$

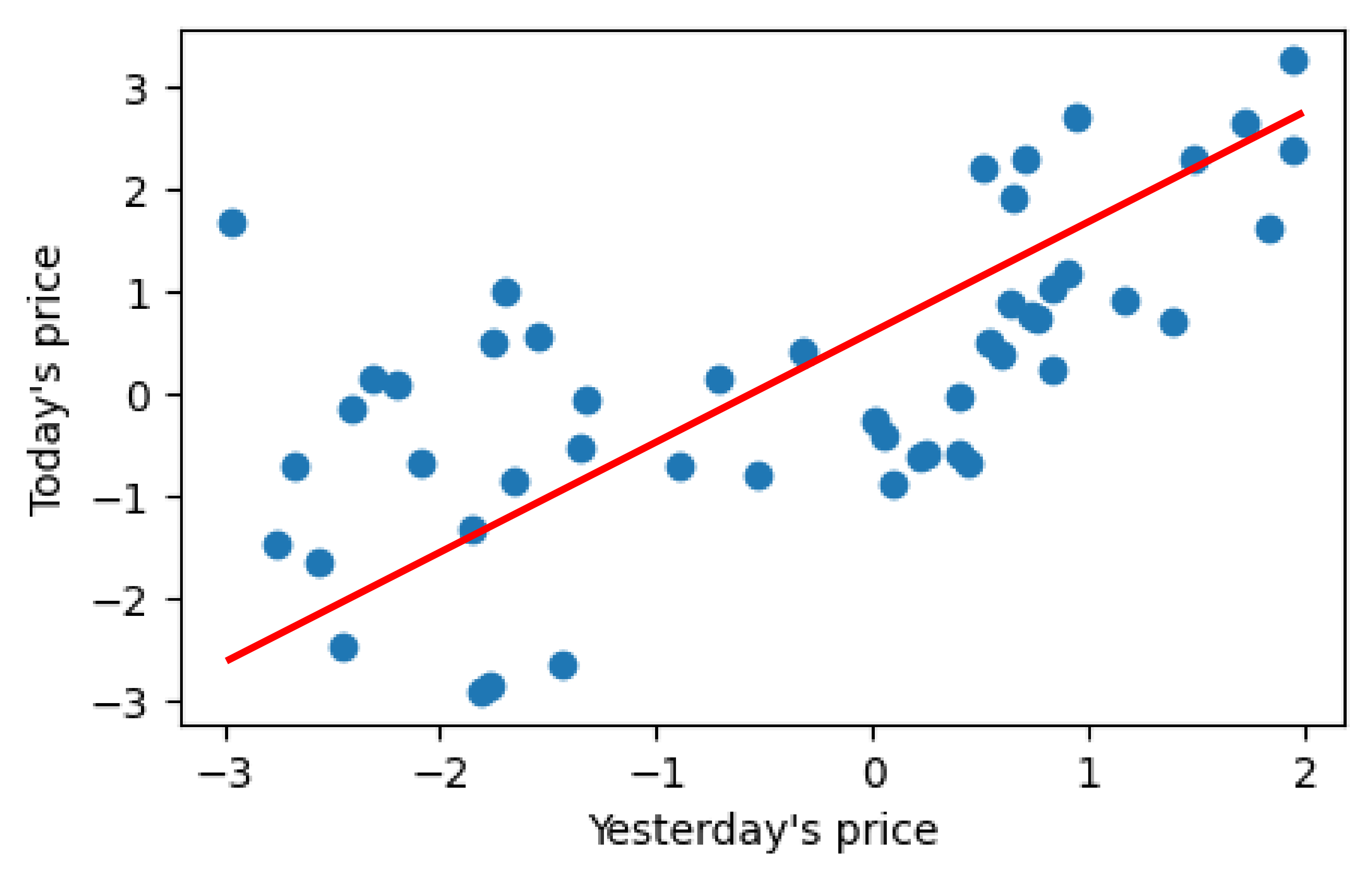
shows how much we deviate from the truth. And it should be **optimizable**.

Choosing the loss = choosing which types of mistakes are worse
Usually, lower loss ~ better model

Example: linear regression

Predicting the value with a linear function:

$$\hat{y}_i = w_0 + xw^T$$

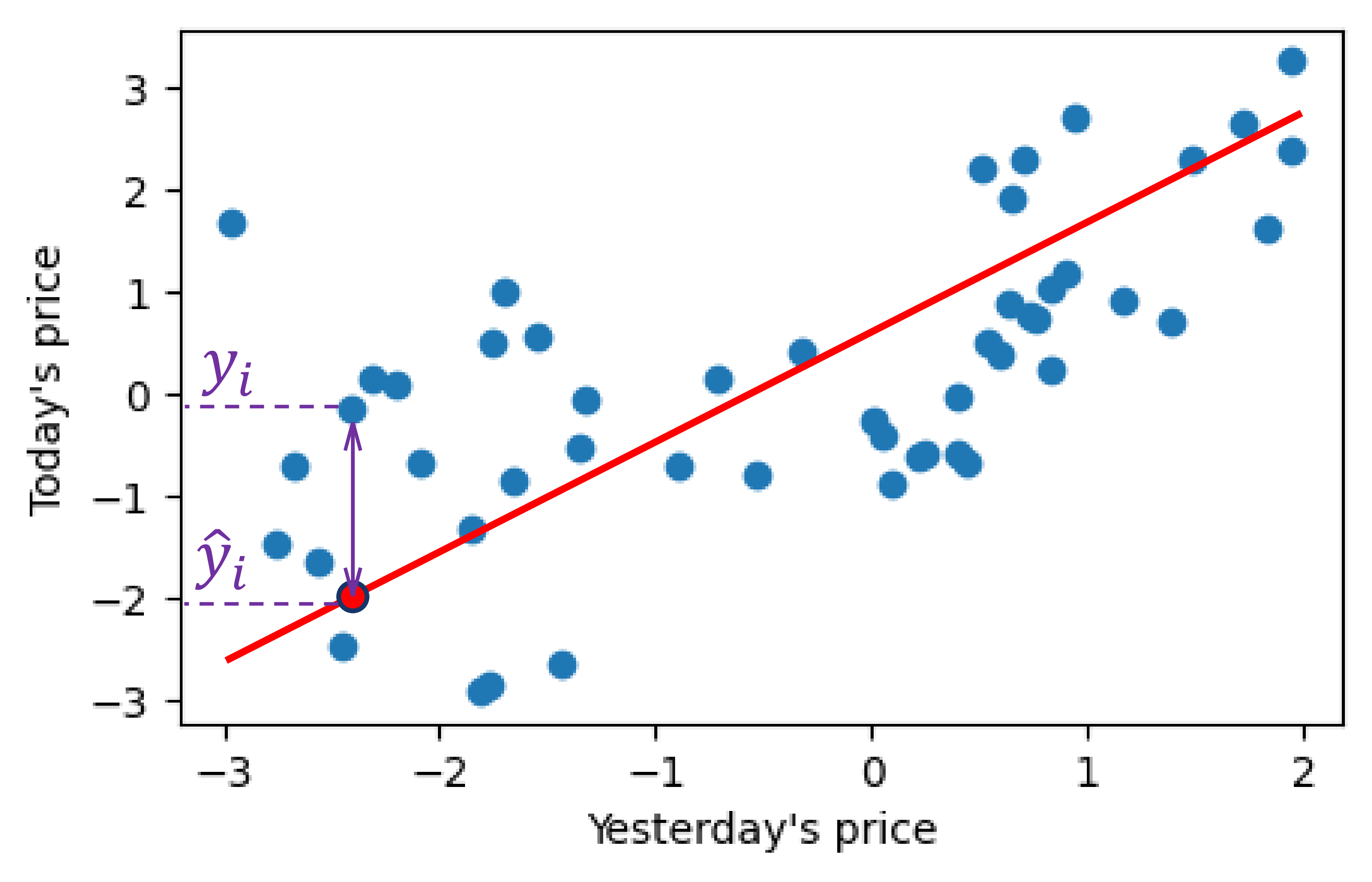


Example: linear regression

Loss

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, \hat{y}_i)$$

y_i – true value
$\hat{y}_i$ – predicted value



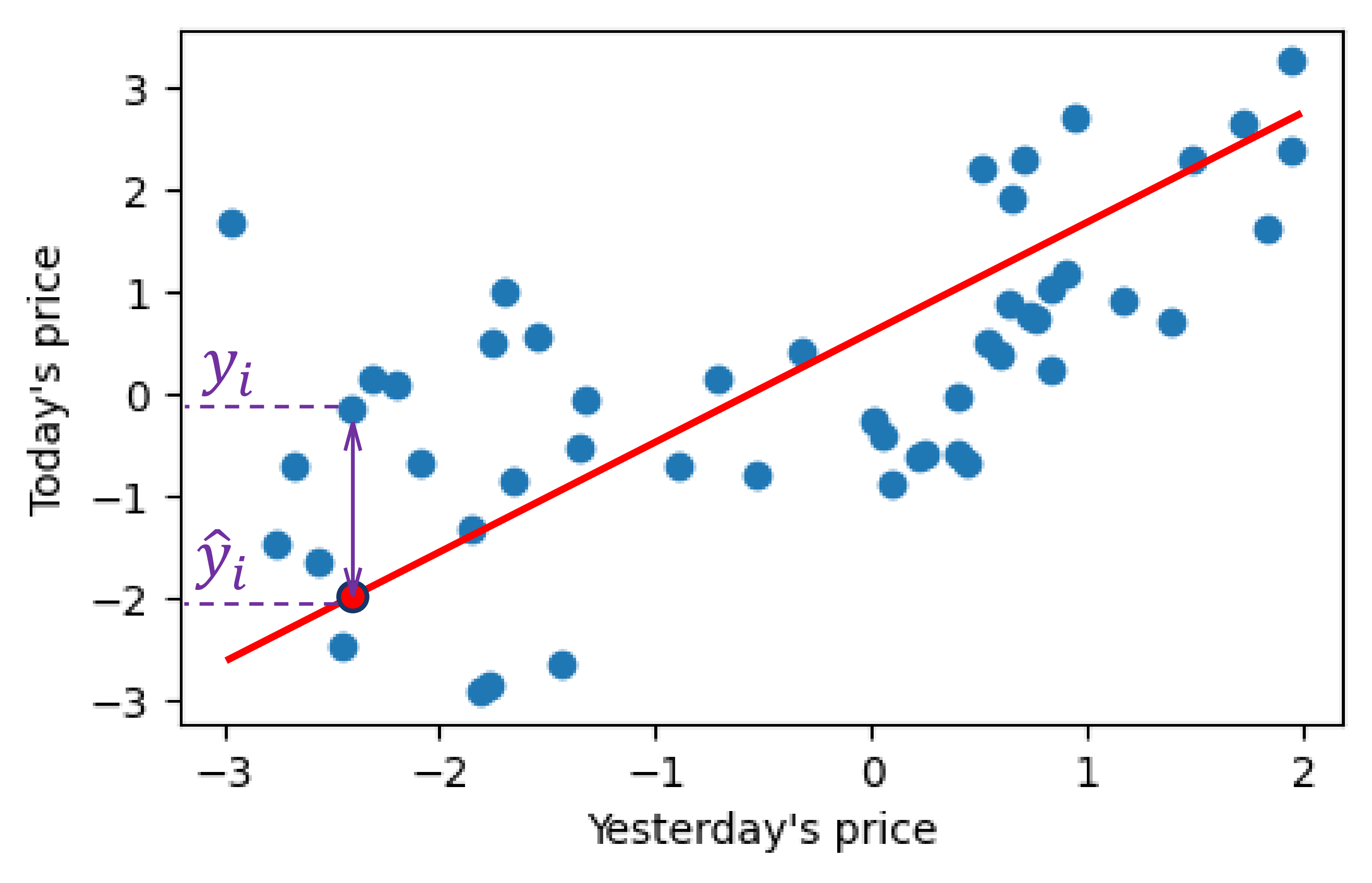
Example: linear regression

Loss

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(y_i, \hat{y}_i)$$

y_i – true value
$\hat{y}_i$ – predicted value

Sanity check: why not
$\mathcal{L}(y_i, \hat{y}_i) = (y_i - \hat{y}_i)$?

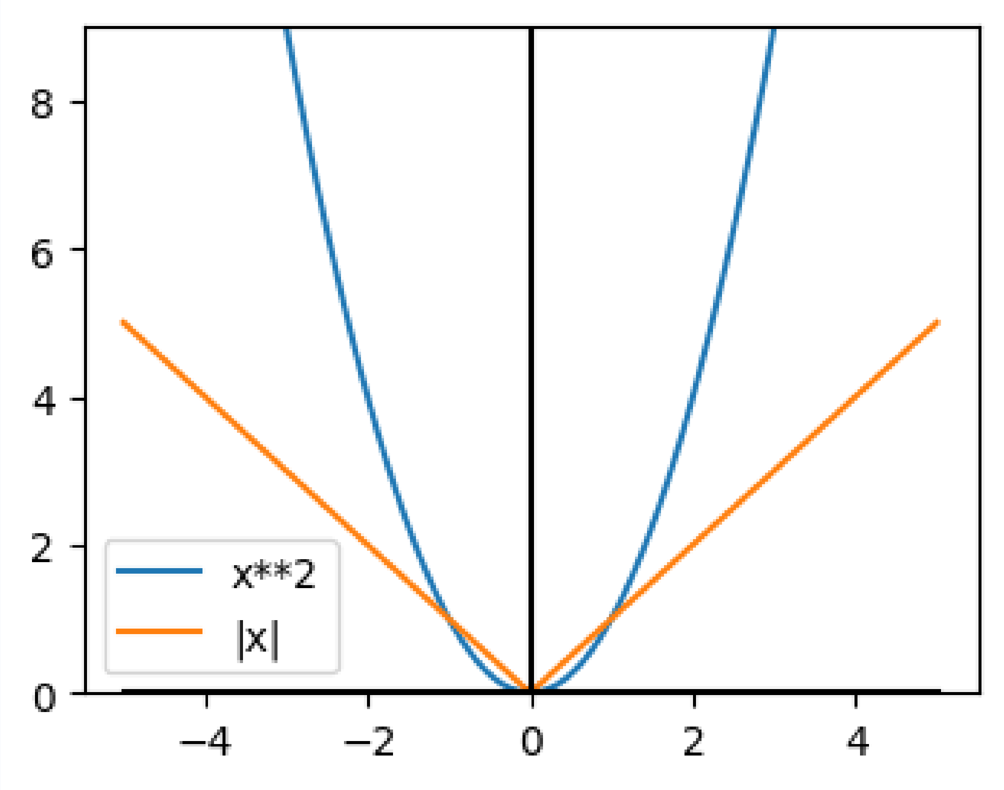


Example: linear regression

$\mathcal{L}(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$ is penalizing more harshly for large errors than

$\mathcal{L}(y_i, \hat{y}_i) = |y_i - \hat{y}_i|$

Why is it important?



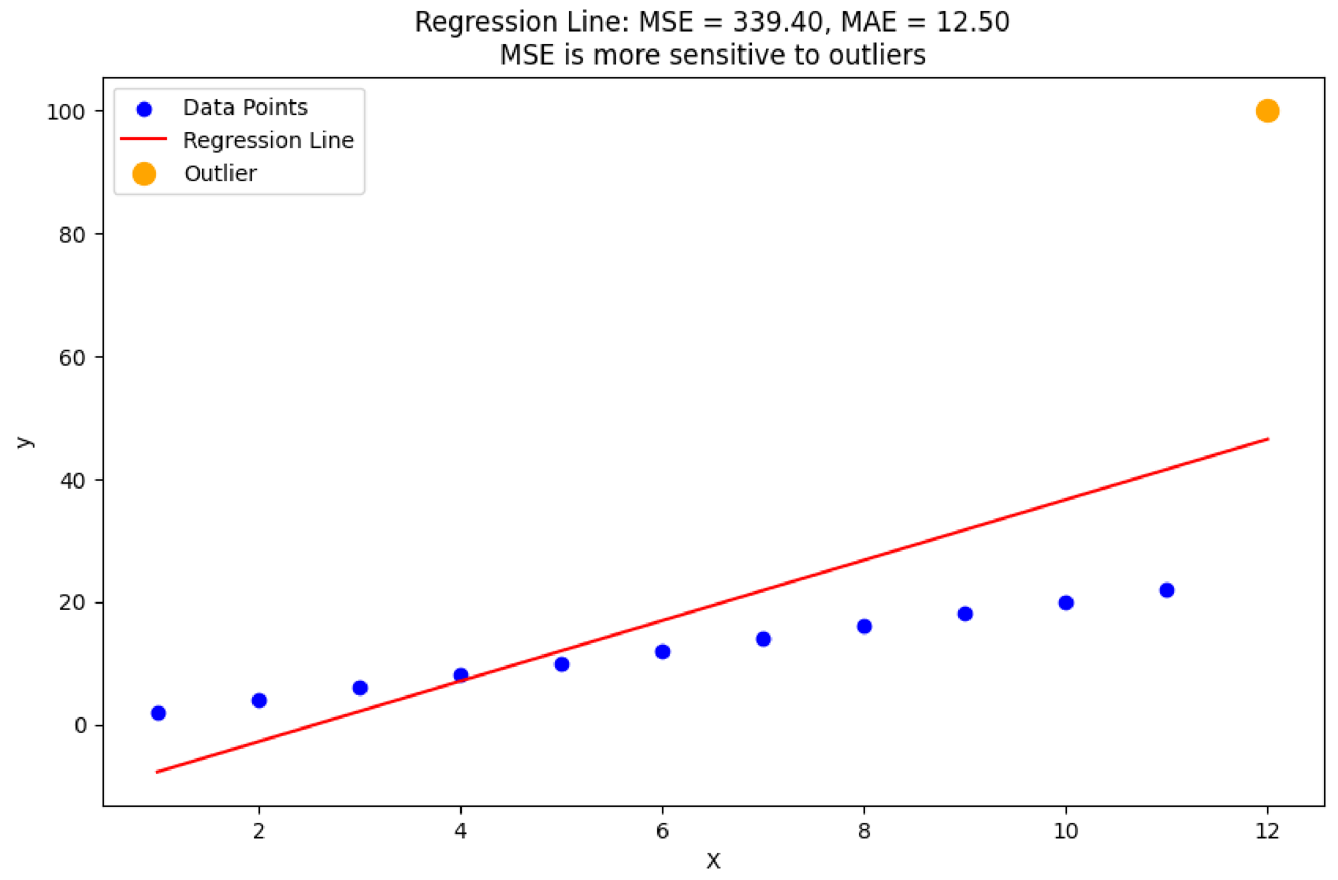
Example: linear regression

$\mathcal{L}(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$ is penalizing more harshly for large errors than

$\mathcal{L}(y_i, \hat{y}_i) = |y_i - \hat{y}_i|$

Why is it important?

Less problems from outliers.



Regression losses

y_i – true answers, $\hat{y}_i$ – predicted answers

MSE (Mean Squared Error)

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

MAE (Mean Absolute Error)

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|^2$$

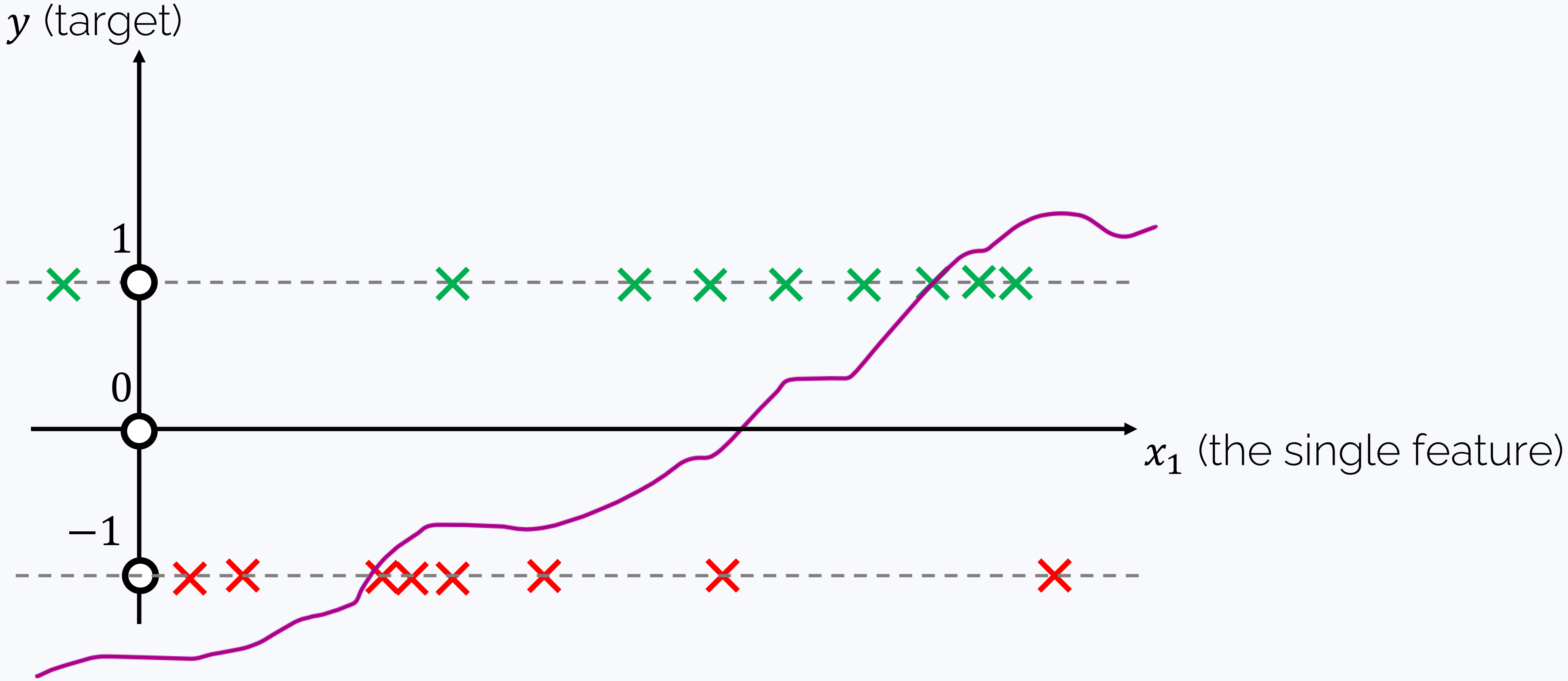
Linear classification is… regression!

A linear model outputs not a class index, but a real number, a "class 1 score". We may suppose that $> \mathbf{0}$ means "class 1".

We don't have a regression target though…

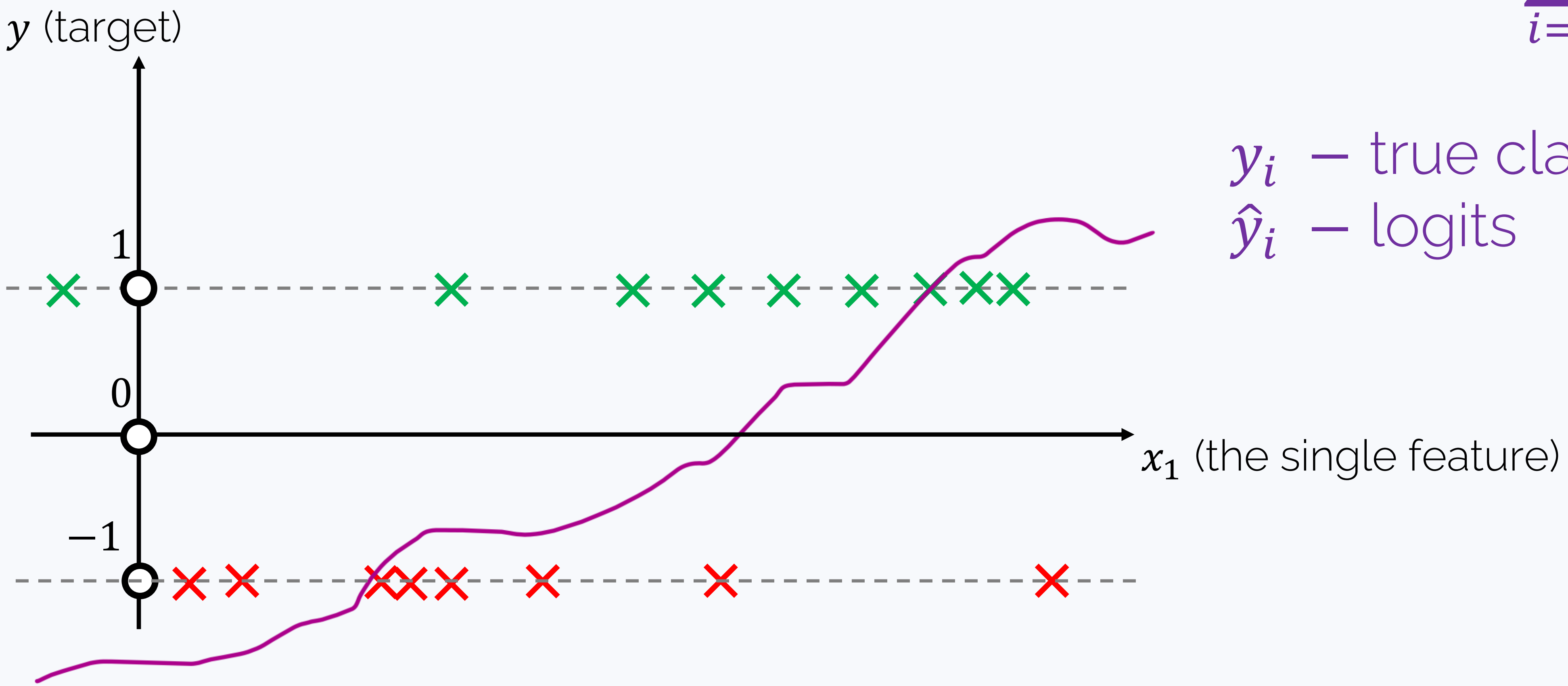
Attempt no. 1

Let class labels -1 and 1 be our target



Attempt no. 1

Let class labels -1 and 1 be our target



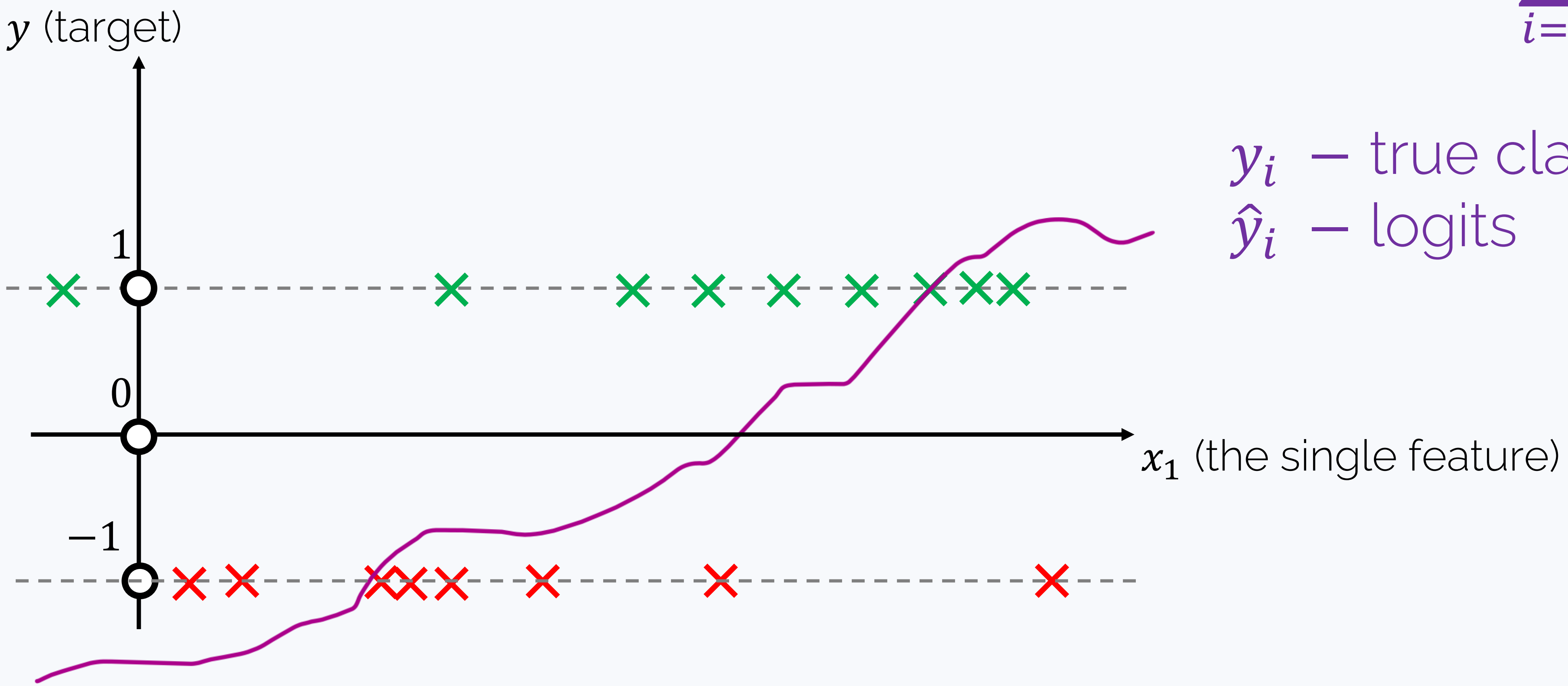
Loss

$$\mathcal{L}(y, \hat{y}) = \sum_{i=1}^{N} \mathcal{L}(y_i, \hat{y}_i)$$

y_i – true class labels
$\hat{y}_i$ – logits

Attempt no. 1. MSE loss

Let class labels -1 and 1 be our target



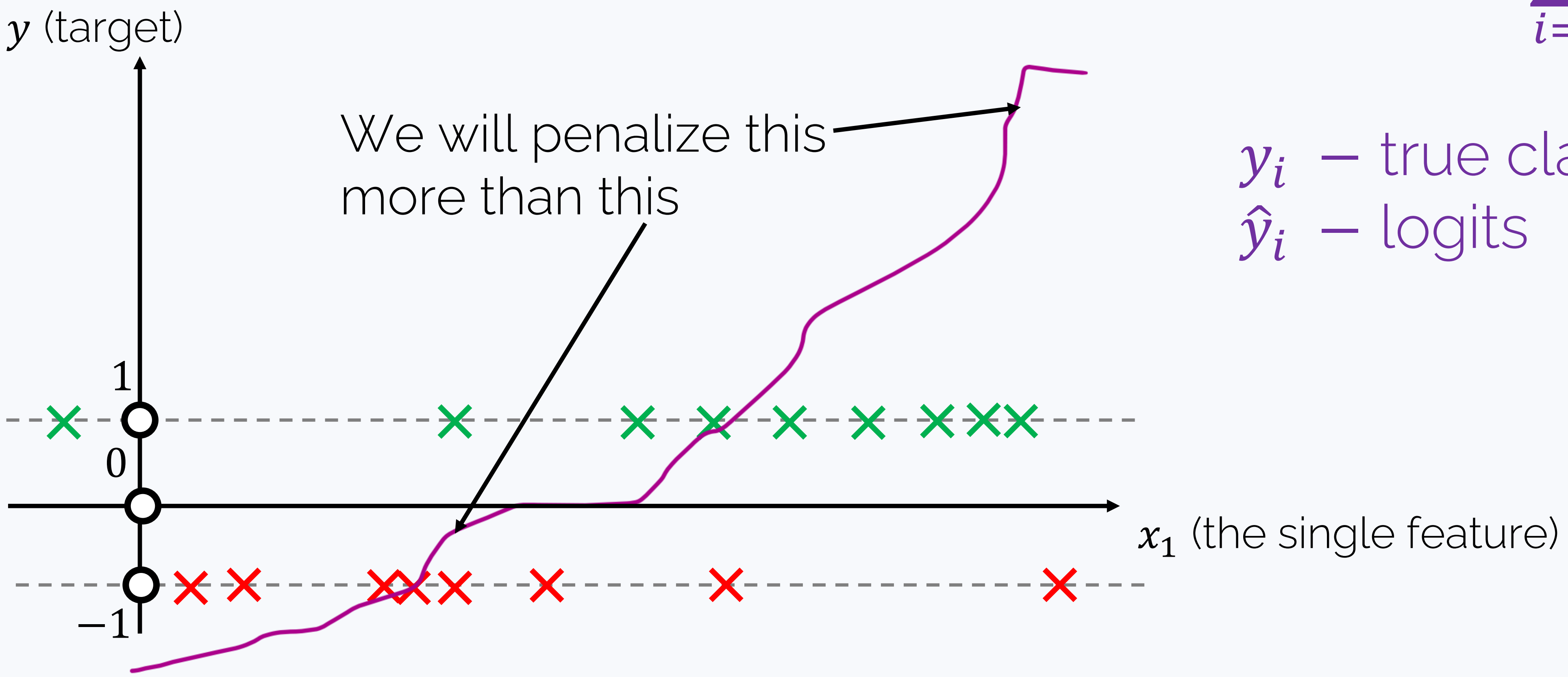
MSE Loss

$$\mathcal{L}(y, \hat{y}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

y_i – true class labels
$\hat{y}_i$ – logits

Attempt no. 1

Let class labels -1 and 1 be our target

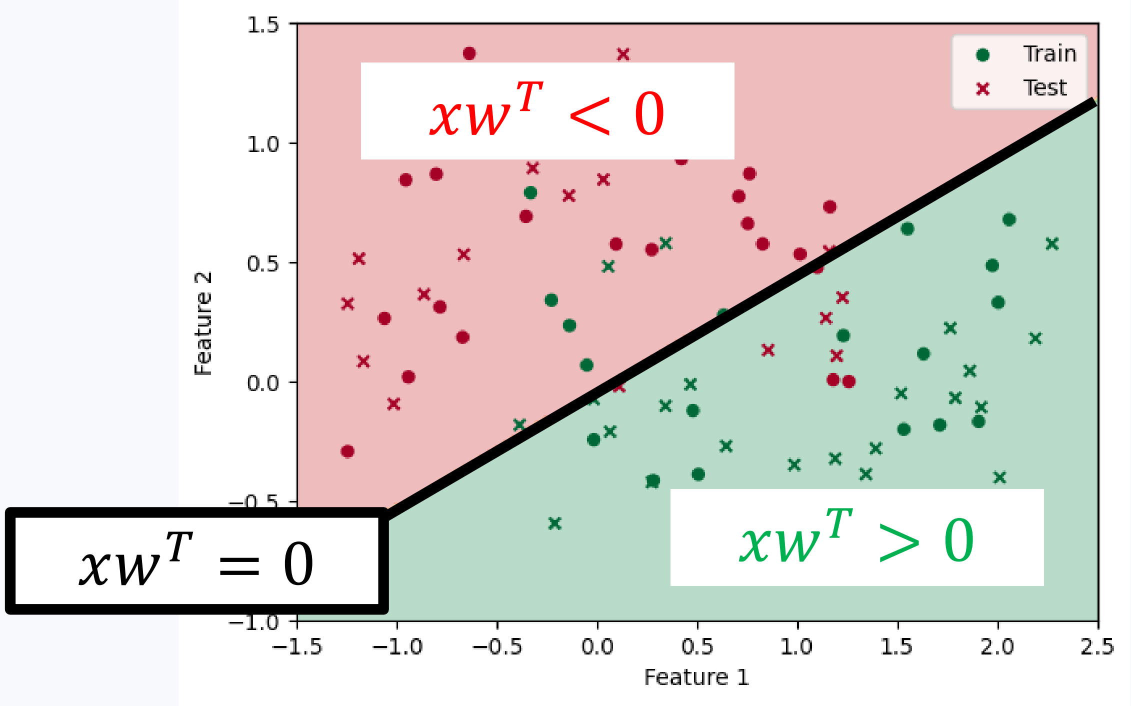


MSE Loss

$$\mathcal{L}(y, \hat{y}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

y_i – true class labels
$\hat{y}_i$ – logits

Attempt no. 2: accuracy



Accuracy

$$\mathcal{L}(y, \hat{y}) = \sum_{i=1}^{N} \mathbb{I}[\hat{y}_i = y_i]$$

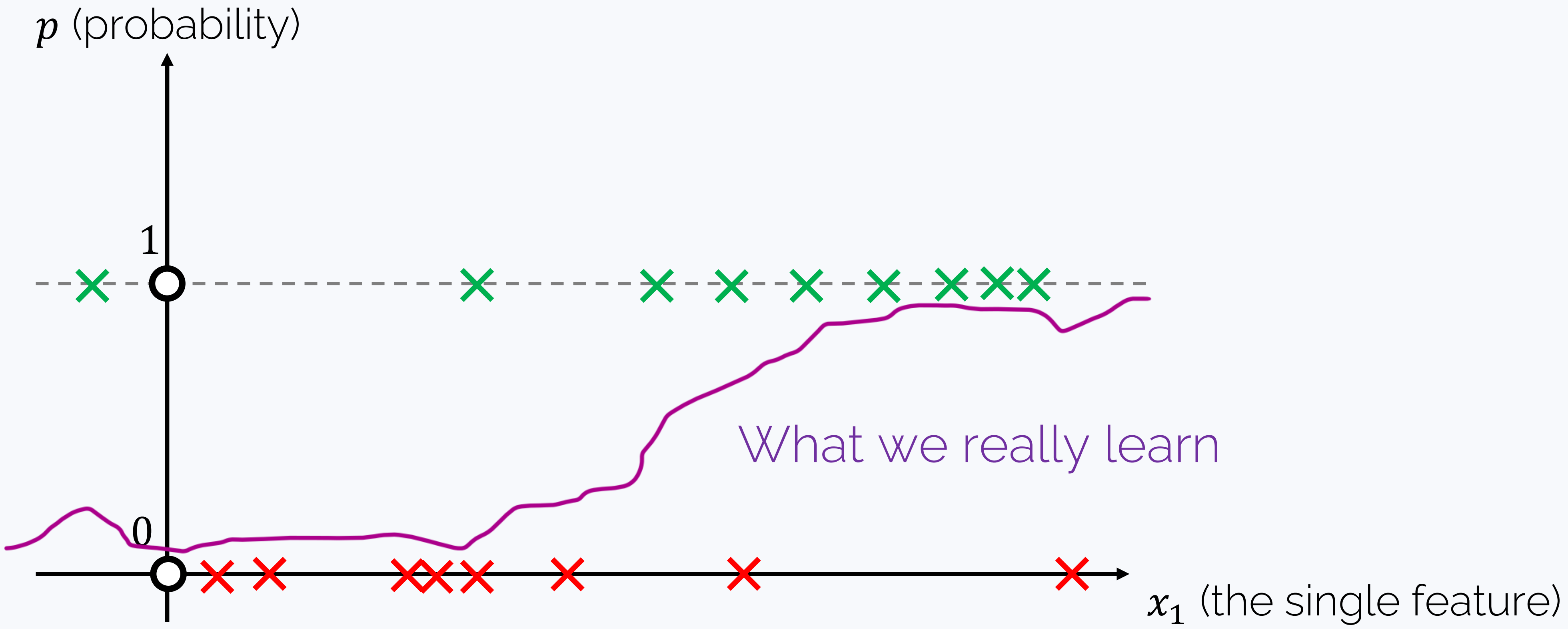
y_i – true class labels (1 or -1)
$\hat{y}_i = xw^T$ – logits

Metric vs loss

- Metric is our proxy of success
- If metric is optimizable, it's the loss
- Otherwise, loss is a proxy of metric

Attempt no. 3: let's use probabilities!

Predicted class 1 probability $\hat{p} = \sigma(xw^T)$ are between 0 and 1.



Attempt no. 3: maximize the right probability

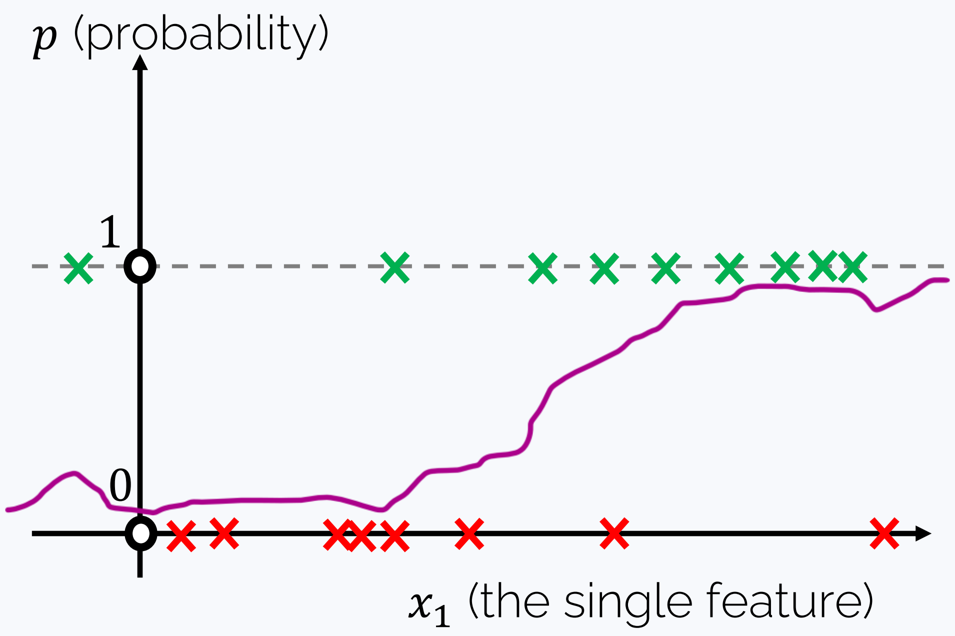
We maximize:

$$l(y,\hat{p}) = \begin{cases} \hat{p}_i, & if\ y_i = 1 \\ 1-\hat{p}_i, & if\ y_i = 0 \end{cases}$$

y_i – true class labels
$\hat{p}_i$ – predicted probabilities

If the true class is 1, we want $\mathrm{P(class\ 1)} = \hat{p}_i$ as large as possible.

If the true class is 0, we want $\mathrm{P(class\ 0)} = 1 - \hat{p}_i$ as large as possible.



Attempt no. 3: logarithm is better

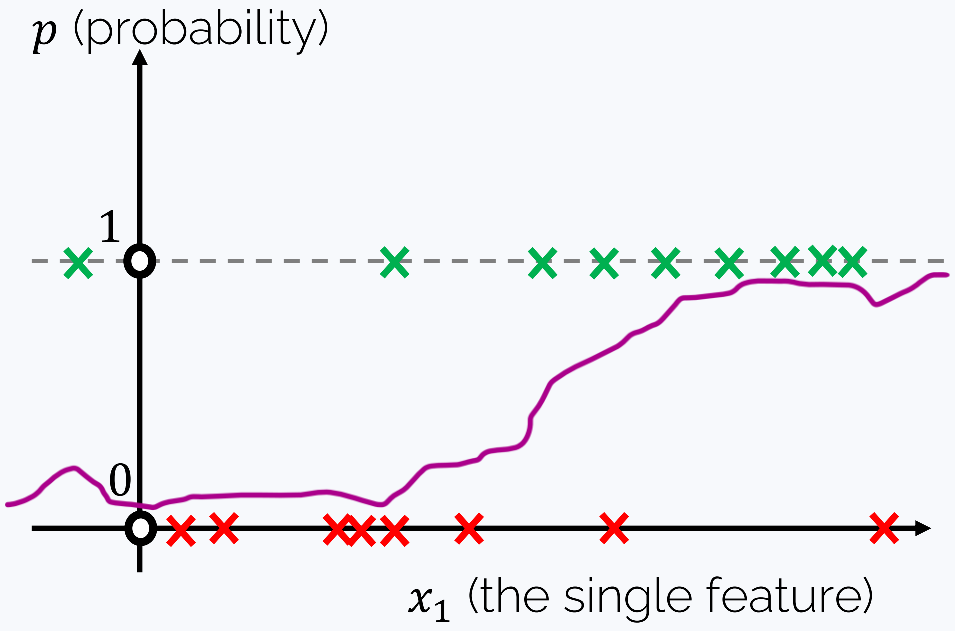
We maximize:

$$l(y_i, \hat{p}_i) = \begin{cases} \log \hat{p}_i\,, & if\ y_i = 1 \\ \log(1 - \hat{p}_i), & if\ y_i = 0 \end{cases}$$

y_i – true class labels
$\hat{p}_i$ – predicted probabilities

If the true class is 1, we want $\mathrm{P(class\ 1)} = \hat{p}_i$ as large as possible.

If the true class is 0, we want $\mathrm{P(class\ 0)} = 1 - \hat{p}_i$ as large as possible.



Attempt no. 3: logarithm is better

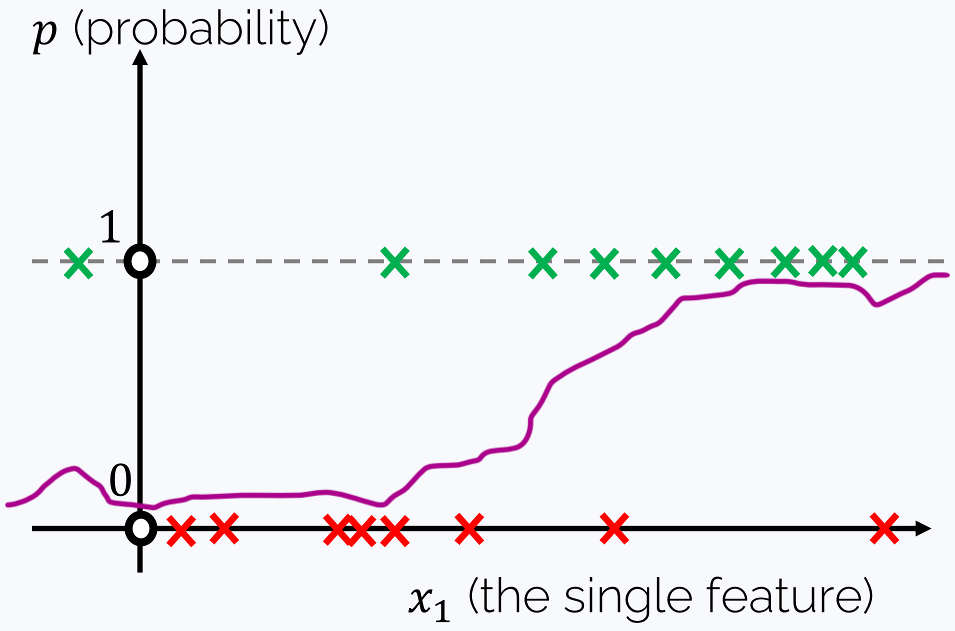
We **minimize**:

$$l(y_i, \hat{p}_i) = \begin{cases} -\log \hat{p}_i\,, & if\ y_i = 1 \\ -\log(1-\hat{p}_i)\,, & if\ y_i = 0 \end{cases}$$

y_i – true class labels
$\hat{p}_i$ – predicted probabilities

If the true class is 1, we want $\mathrm{P(class\ 1)} = \hat{p}_i$ as large as possible.

If the true class is 0, we want $\mathrm{P(class\ 0)} = 1 - \hat{p}_i$ as large as possible.

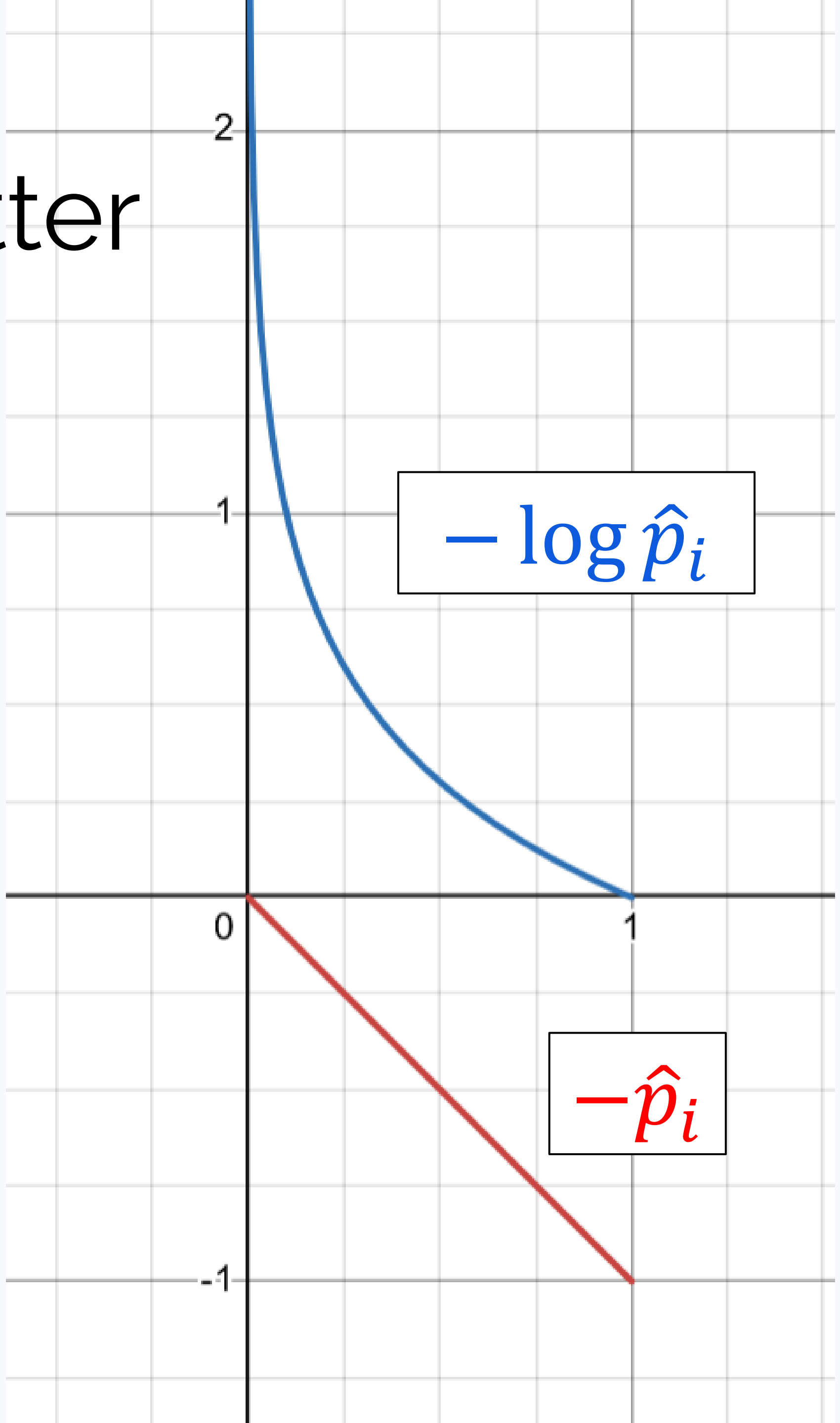


Attempt no. 3: logarithm is better

We maximize:

$$l(y_i, \hat{p}_i) = \begin{cases} \log \hat{p}_i\,, & if\ y_i = 1 \\ \log(1 - \hat{p}_i), & if\ y_i = 0 \end{cases}$$

y_i – true class labels
$\hat{p}_i$ – predicted probabilities



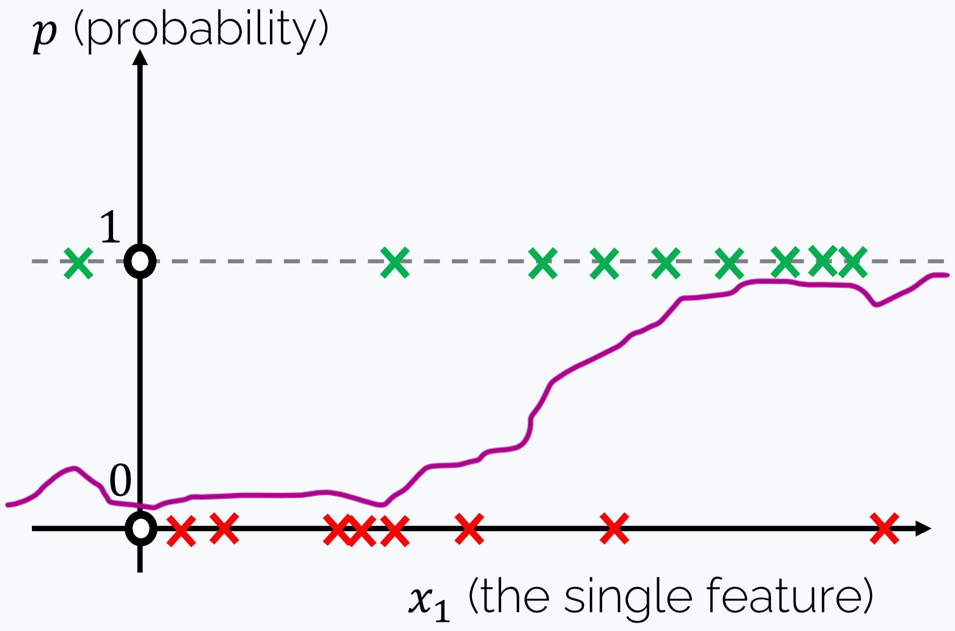
Attempt no. 3: Cross-entropy loss

We **minimize**:

$$l(y_i, \hat{p}_i) = \begin{cases} -\log \hat{p}_i\,, & if\ y_i = 1 \\ -\log(1-\hat{p}_i)\,, & if\ y_i = 0 \end{cases}$$

$$\boxed{\mathcal{L}(y_i, \hat{p}_i) = -y_i \log \hat{p}_i - (1-y_i)\log(1-\hat{p}_i)}$$

y_i – true class labels
$\hat{p}_i$ – predicted probabilities



Dancing with probabilities

Why log is good: probabilistic answer

$$\mathcal{L}(y, \hat{p}) = -y_i \log \hat{p}_i - (1 - y_i) \log(1 - \hat{p}_i)$$

Why log is good: probabilistic answer

$$\mathcal{L}(y_i, \hat{p}_i) = -(1 - y_i)\log(1 - \hat{p}_i) - y_i \log \hat{p}_i =$$

$$= -\mathbb{I}[y_i = 0]\log \hat{p}_{i0} - \mathbb{I}[y_i = 1]\log \hat{p}_{i1} =$$

$$= -\log\ \hat{p}_{i0}{}^{\mathbb{I}[y_i=0]} - \log\ \hat{p}_{i1}{}^{\mathbb{I}[y_i=1]} =$$

$$= -\log\ \hat{p}_{i0}^{\mathbb{I}[y_i=0]}\hat{p}_{i1}^{\mathbb{I}[y_i=1]} = -\log(\textit{predicted prob of true class of } x_i)$$

$\hat{p}_{i0}$ – predicted probability of class 0
$\hat{p}_{i1}$ – predicted probability of class 1

Why log is good: probabilistic answer

$$\mathcal{L}(y,\hat{p}) = \sum_{i=1}^{N} \mathcal{L}(y_i,\hat{p}_i) = -\sum_{i=1}^{N} (1-y_i)\log(1-\hat{p}_i) + y_i \log \hat{p}_i =$$

$$= -\sum_{i=1}^{N} \log \hat{p}_{i0}^{\mathbb{I}[y_i=0]} \hat{p}_{i1}^{\mathbb{I}[y_i=1]} = -\log \prod_{i=1}^{N} \hat{p}_{i0}^{\mathbb{I}[y_i=0]} \hat{p}_{i1}^{\mathbb{I}[y_i=1]} =$$

$$= -\log \prod_{i=1}^{N} \textit{predicted prob of true class of } x_i$$

Why log is good: probabilistic answer

$$\mathcal{L}(y,\hat{p}) = \sum_{i=1}^{N} \mathcal{L}(y_i,\hat{p}_i) = -\sum_{i=1}^{N} (1-y_i)\log(1-\hat{p}_i) + y_i \log \hat{p}_i =$$

$$= -\sum_{i=1}^{N} \log \hat{p}_{i0}^{\mathbb{I}[y_i=0]} \hat{p}_{i1}^{\mathbb{I}[y_i=1]} = -\log \prod_{i=1}^{N} \hat{p}_{i0}^{\mathbb{I}[y_i=0]} \hat{p}_{i1}^{\mathbb{I}[y_i=1]} =$$

$$= -\log \prod_{i=1}^{N} \mathbb{P}_{pred}\{x_i \ has \ class \ y_i\}$$

Why log is good: probabilistic answer

$$\mathcal{L}(y, \hat{p}) = -\log \prod_{i=1}^{N} \mathbb{P}_{pred}\{x_i \ has \ class \ y_i\} =$$

$$= -\log\left(\mathbb{P}_{pred}\{(x_1 \ has \ class \ y_1) \ \& \dots \& \ (x_N \ has \ class \ y_N)\}\right)$$

An important assumption: **different** (x_i, y_i) **are independent**.

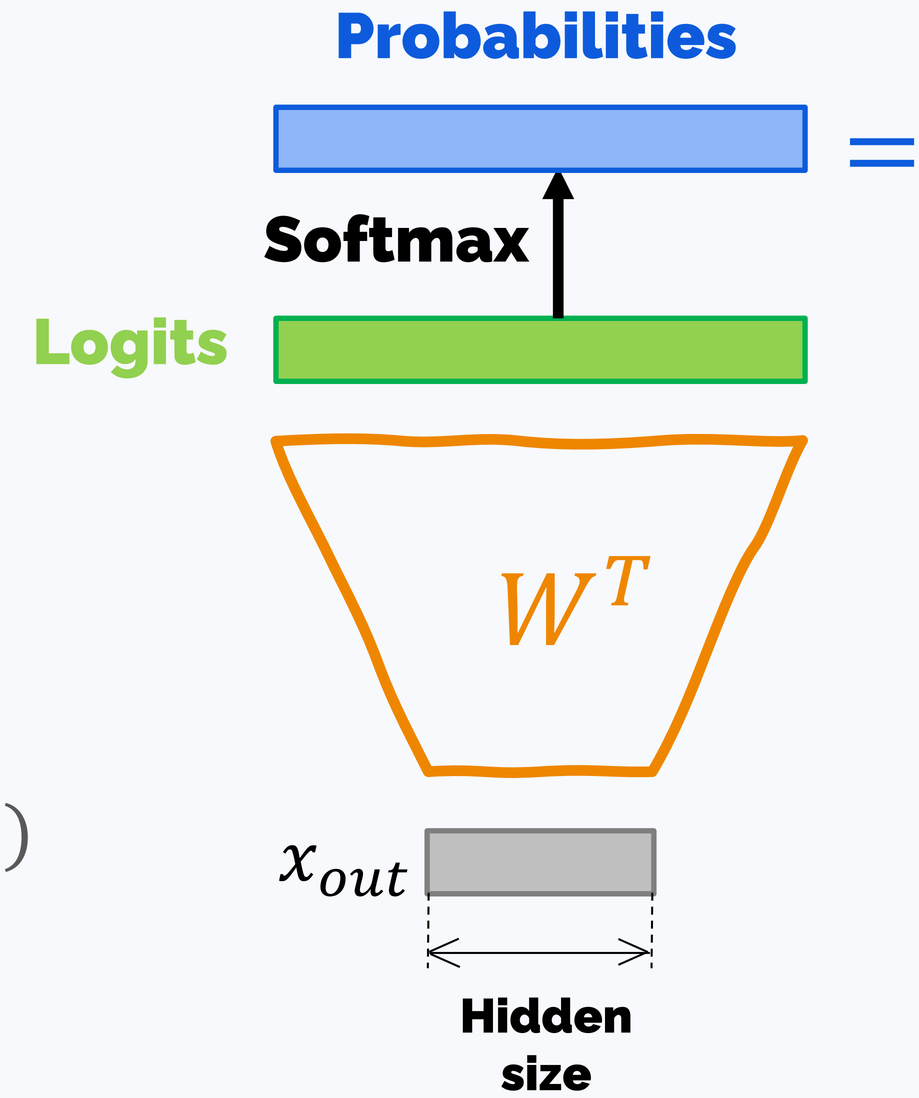
Loss functions for multiclass classification: an informal introduction

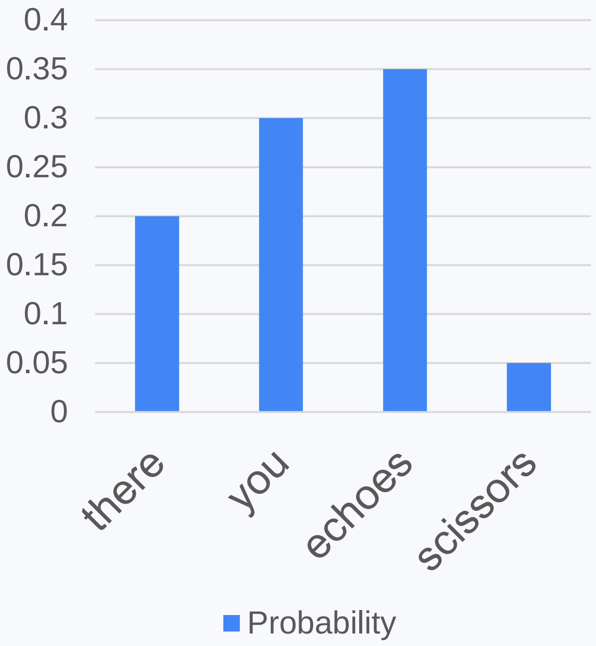
Multiclass classification reminder (with a twist)

Mind the transpose

$$\textbf{Logits} = x_{out} \cdot W^T$$

$$\textbf{Probabilities} = \textbf{Softmax}(x_{out} \cdot W^T)$$





Multiclass classification reminder (with a twist)

$$\textbf{Logits} = x_{out} \cdot W^T$$

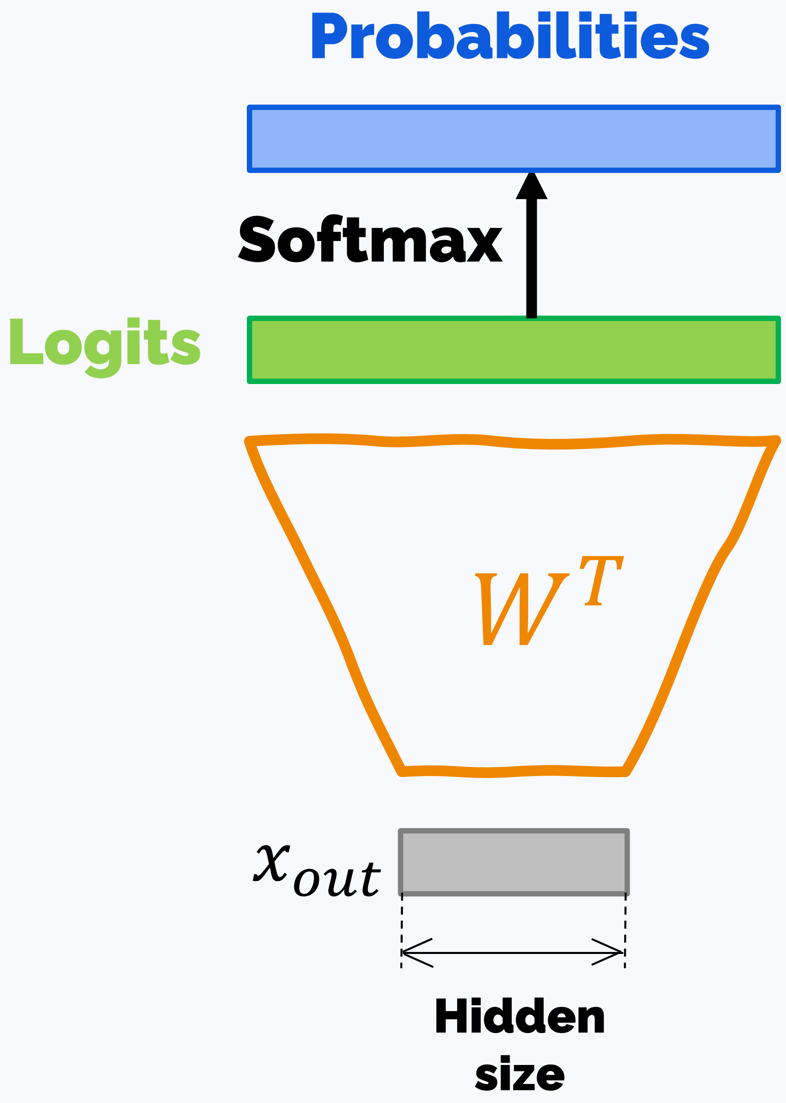
$$\textbf{Probabilities} = \textbf{Softmax}(x_{out} \cdot W^T)$$

Binary:

$$\log \hat{p}_{true\ class} \rightarrow max$$

Multiclass:

$$\log \hat{p}_{true\ class} \rightarrow max$$



Multiclass classification reminder (with a twist)

$$\textbf{Logits} = x_{out} \cdot W^T$$

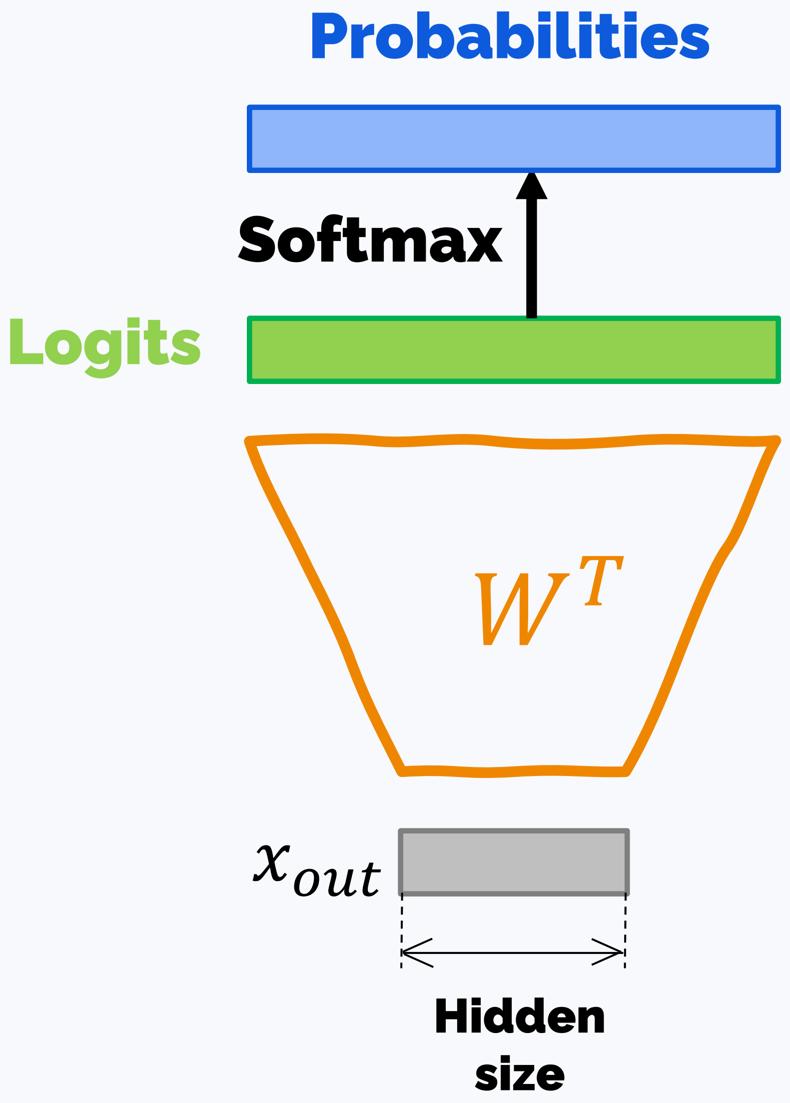
$$\textbf{Probabilities} = \textbf{Softmax}(x_{out} \cdot W^T)$$

Binary:

$$y_i \log \hat{p}_i + (1 - y_i) \log(1 - \hat{p}_i)$$

Multiclass:

$$y_{i1} \log \hat{p}_{i1} + y_{i2} \log \hat{p}_{i2} + \cdots + y_{iC} \log \hat{p}_{iC}$$
$$y_{ij} = \mathbb{I}[y_i = j]$$



Log-loss for two classes

$$\mathcal{L}(y, \hat{p}) = \sum_{i=1}^{N} \mathcal{L}(y_i, \hat{p}_i) = -\sum_{i=1}^{N} (1 - y_i)\log(1 - \hat{p}_i) + y_i \log \hat{p}_i =$$

$$= -\sum_{i=1}^{N} \log \hat{p}_{i0}^{\mathbb{I}[y_i=0]} \hat{p}_{i1}^{\mathbb{I}[y_i=1]} = -\log \prod_{i=1}^{N} \hat{p}_{i0}^{\mathbb{I}[y_i=0]} \hat{p}_{i1}^{\mathbb{I}[y_i=1]} =$$

$$= -\log \prod_{i=1}^{N} \mathbb{P}_{pred}\{x_i \; has \; class \; y_i\}$$

Cross-entropy loss for >2 classes

$$\mathcal{L}(y, \hat{p}) = \sum_{i=1}^{N} \mathcal{L}(y_i, \hat{p}_i) = -\sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log \hat{p}_{ij} =$$

$$= -\sum_{i=1}^{N} \sum_{j=1}^{C} \log \hat{p}_{ij}^{\mathbb{I}[y_i = j]} = -\log \prod_{i=1}^{N} \prod_{j=1}^{C} \hat{p}_{ij}^{\mathbb{I}[y_i = j]} =$$

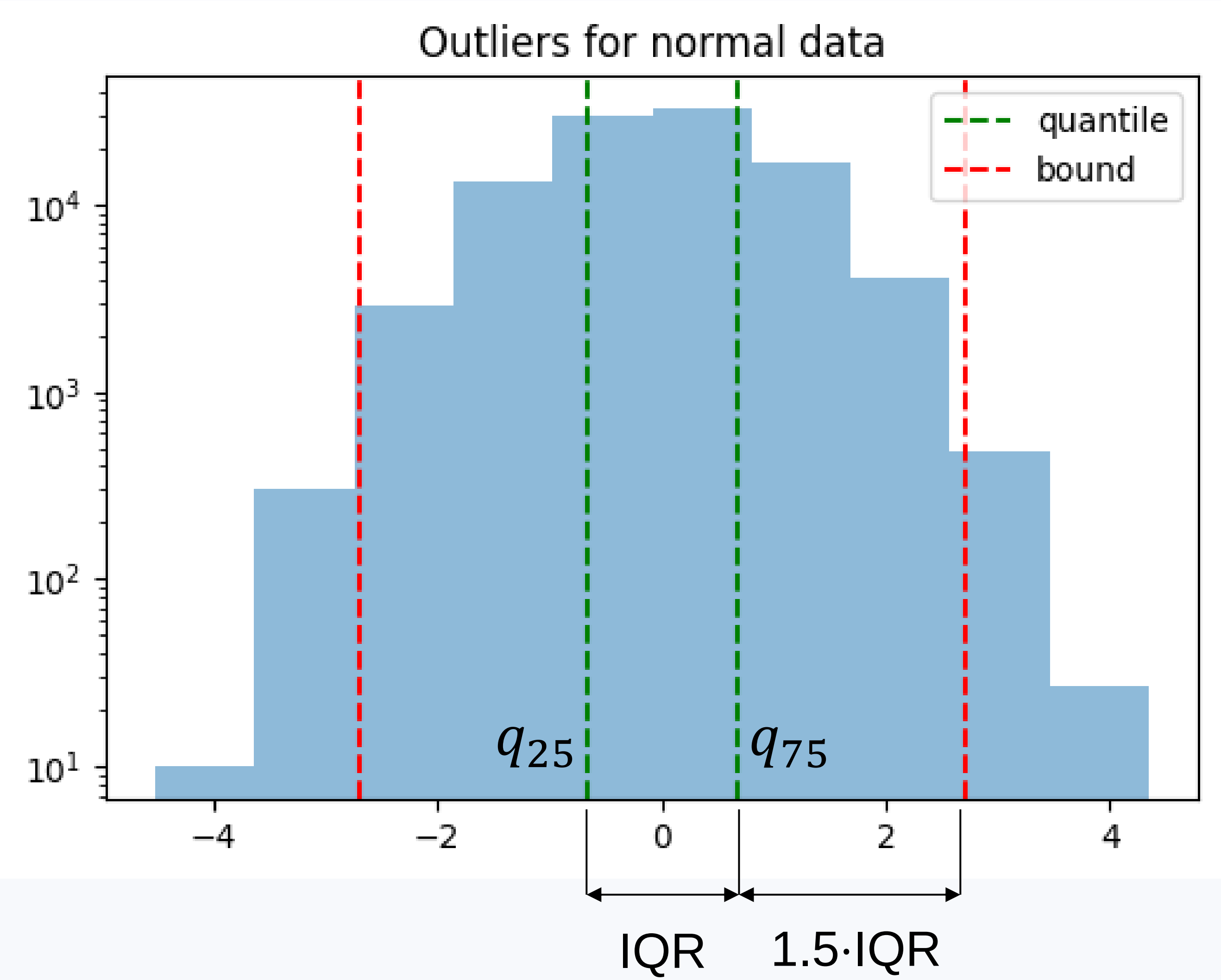
$$= -\log\left(\mathbb{P}_{pred}\{(x_1 \text{ } has \text{ } class \text{ } y_1) \: \& \ldots \& \: (x_N \text{ } has \text{ } class \text{ } y_N)\}\right)$$

Practice time!

Missing value imputation

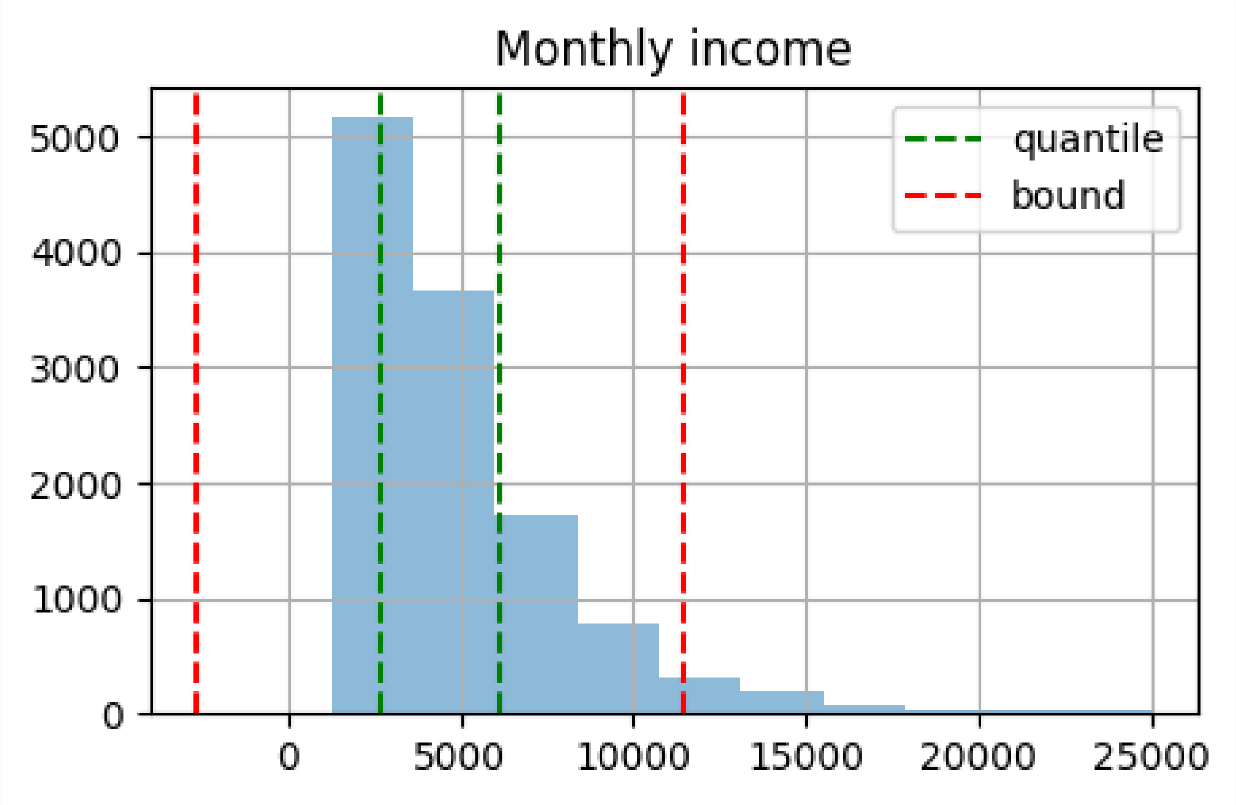
- **Numerical features**: mean, median
- **Categorical features**: most frequent value, additional "missing" category
- **With any features**:
  - Create a new "this feature missing" indicator feature
  - Predict from other features. Be extremely careful with this! At the very least, train your classifier on separate data

Outliers



Outliers for normal data
quantile
bound
10^4
10^3
10^2
10^1
q_{25}
q_{75}
−4
−2
0
2
4
IQR
1.5·IQR

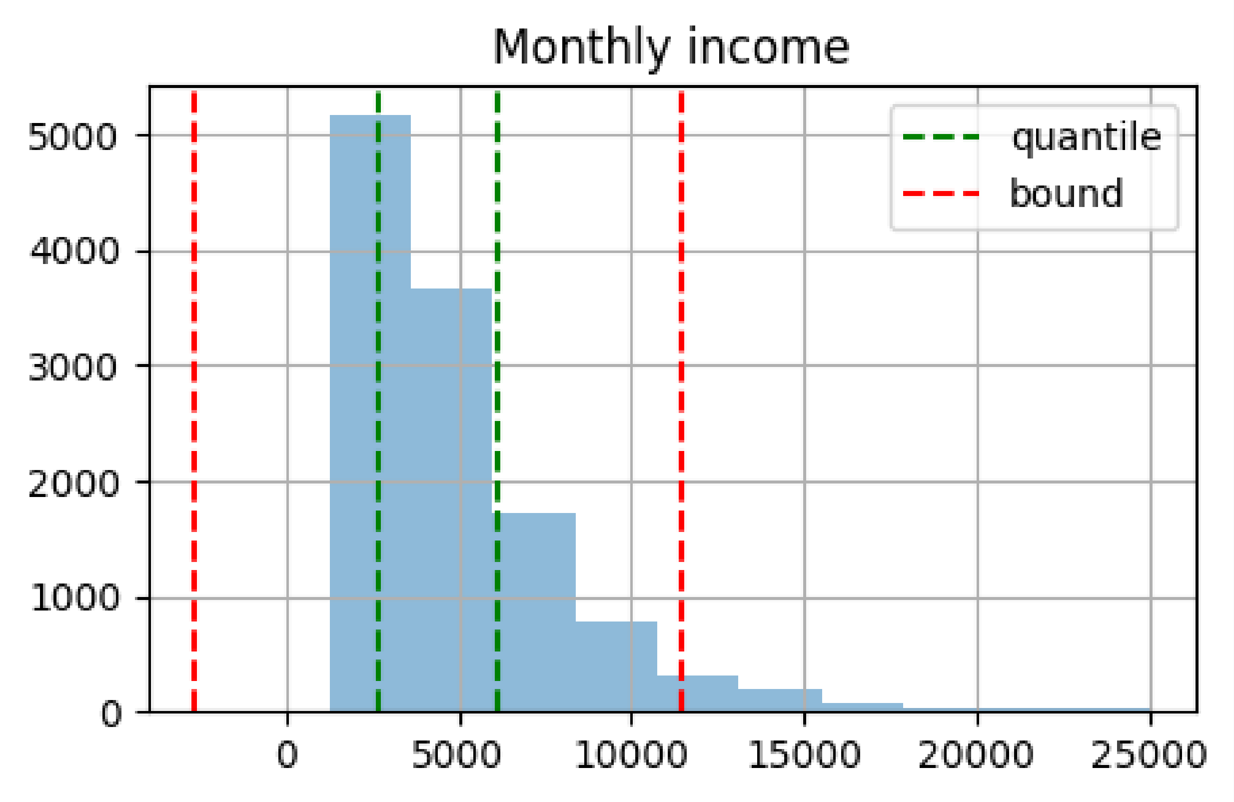
Outliers

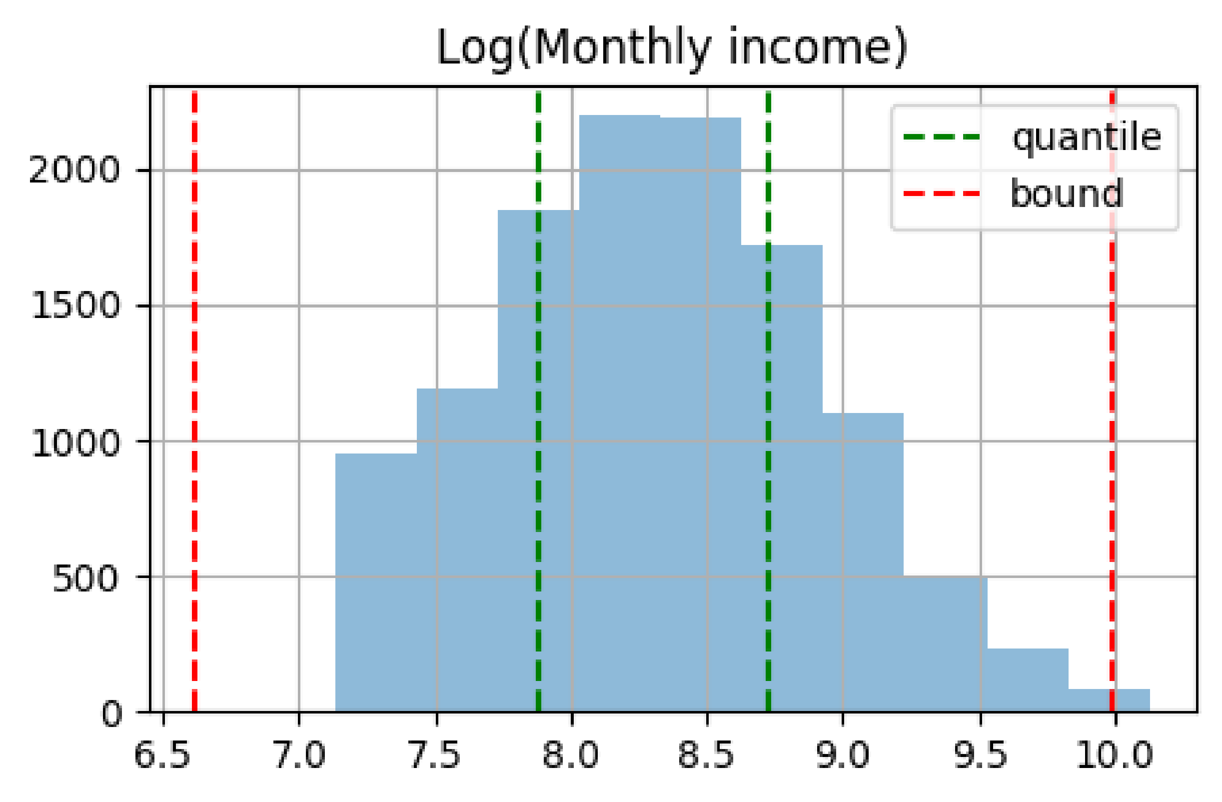


Monthly income
quantile
bound
5000
4000
3000
2000
1000
0
0
5000
10000
15000
20000
25000

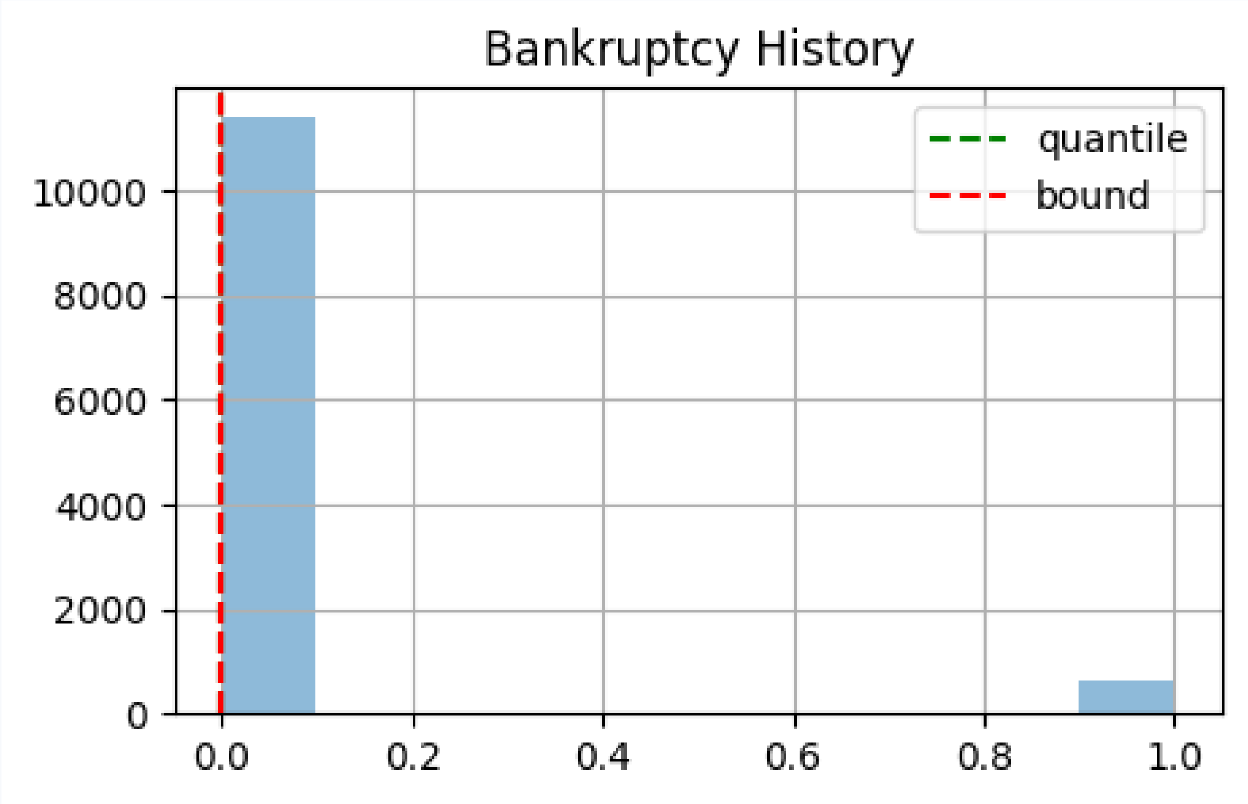
Outliers

Use log of feature or create additional indicator that the income is very large

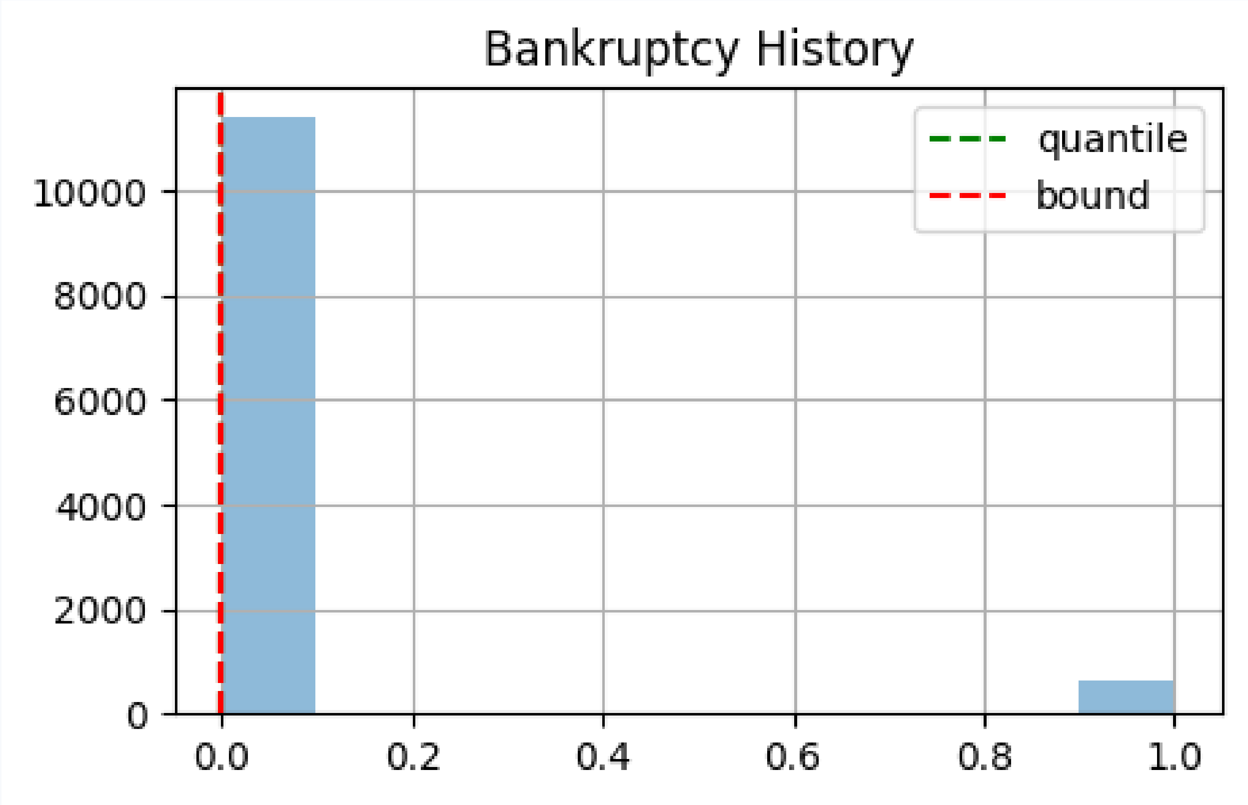




Outliers


Bankruptcy History
quantile
bound
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0

Outliers



It's actually a categorical variable