AI in Mathematics
Lecture 3
Classic ML. Part 2.

Bar-Ilan University
Nebius Academy | Stevens Institute of Technology
April 1, 2025

About This Course

~~1 week: Intro~~

2 weeks: Classic ML

2 weeks: Deep Learning in Mathematics

3 weeks: Math as an NLP problem (LLMs etc.)

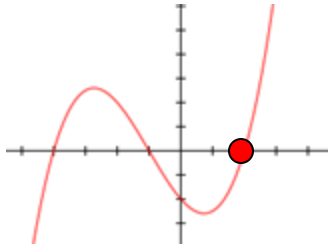
3 weeks: Reinforcement Learning (RL) in Math

1 week: Advanced AI topics

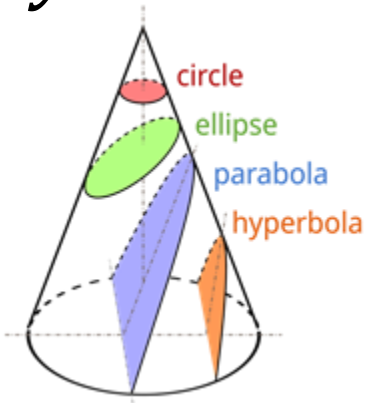
1 week: Project Presentations

Regression and Classification

Regression task: What is the largest root of polynomial $Ax^3 + Bx^2 + Cx + D = 0$?



Classification task: What type of quadratic curve is defined by equation $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$?



Regression

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n},$$

Each row is a data point, consisting of n features

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \in \mathbb{R}^{m}.$$

Each value is a label of a data point in X

We want to construct T: $\mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{m}$ such that T is taken from a **simple enough** class of functions and $T(X)$ approximates y **good enough**

Classification

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n},$$

Each row is a data point, consisting of n features

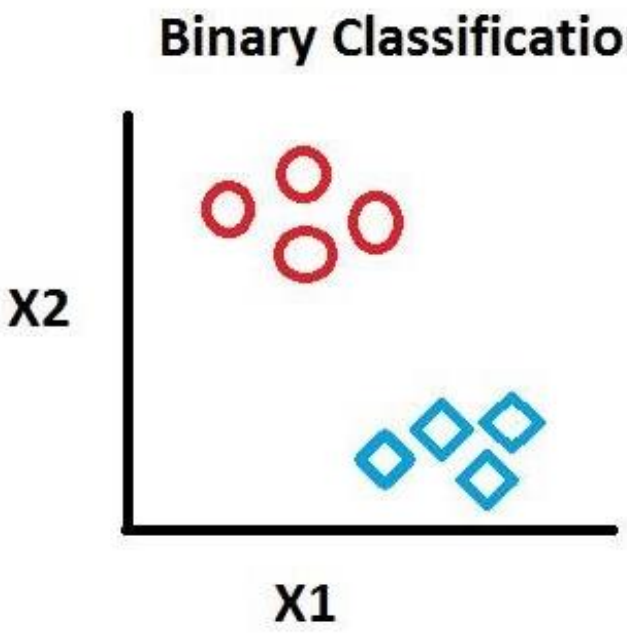
$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \in \mathbb{Y}^m.$$

Each value is from $\mathbb{Y}$ – class of abstract objects

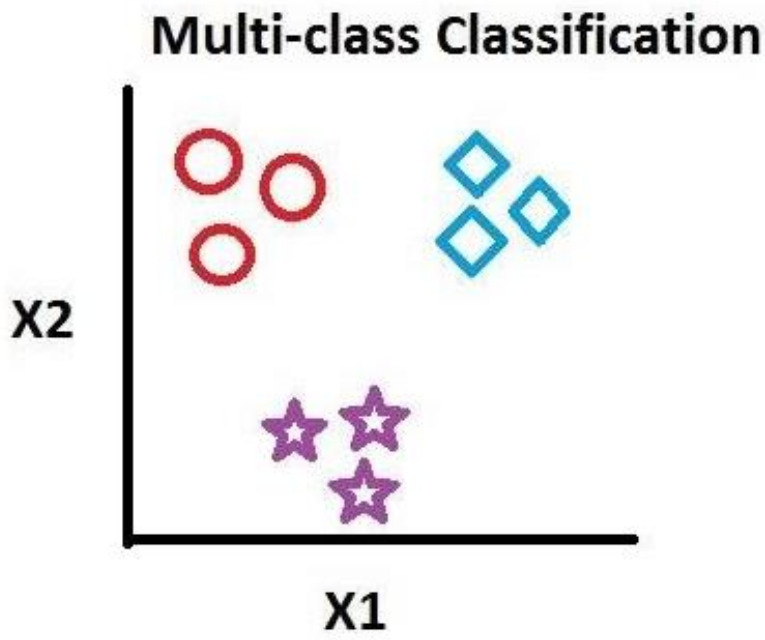
Class can be any set of objects, but for ML purposes we can assign numbers to them:

$$\mathbb{Y} = \{1, \dots, |\mathbb{Y}|\}.$$

Classification

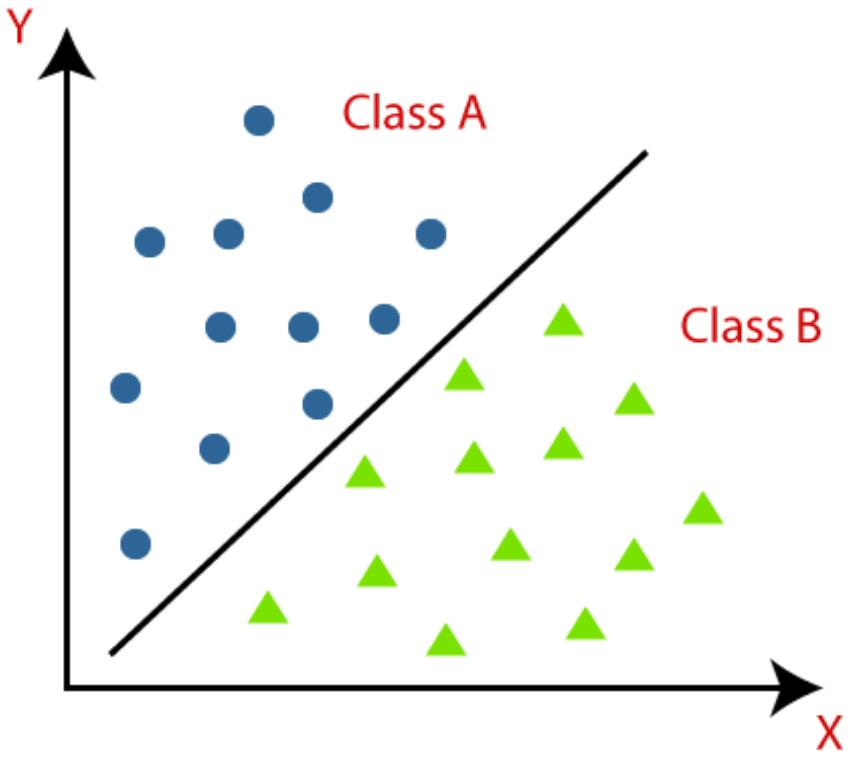


Two labels:
$\mathbb{Y} = \{0, 1\}$ or $\mathbb{Y} = \{-1, 1\}$.



More labels:
Generalization of Binary.

Linear Classification



Simple Classification Idea:

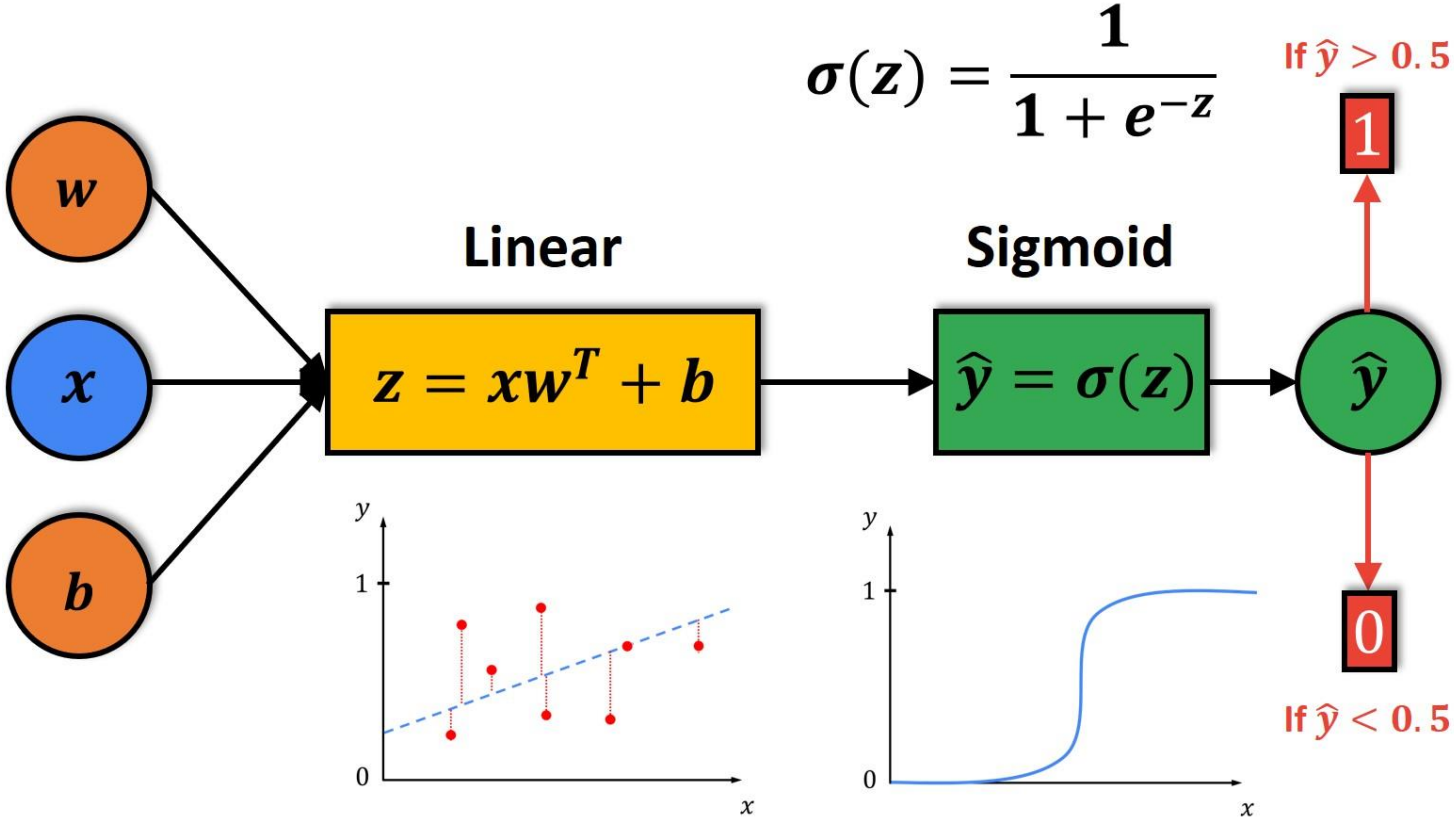
$$\hat{y}_i = \begin{cases} 0, & X_i w < \mathbf{0} \\ 1, & X_i w \geq \mathbf{0} \end{cases}$$

Why not solve a regression problem instead?

1. Different objective – separation.
2. For large $X_i w$ >> 1, the error becomes large.

Logistic Regression

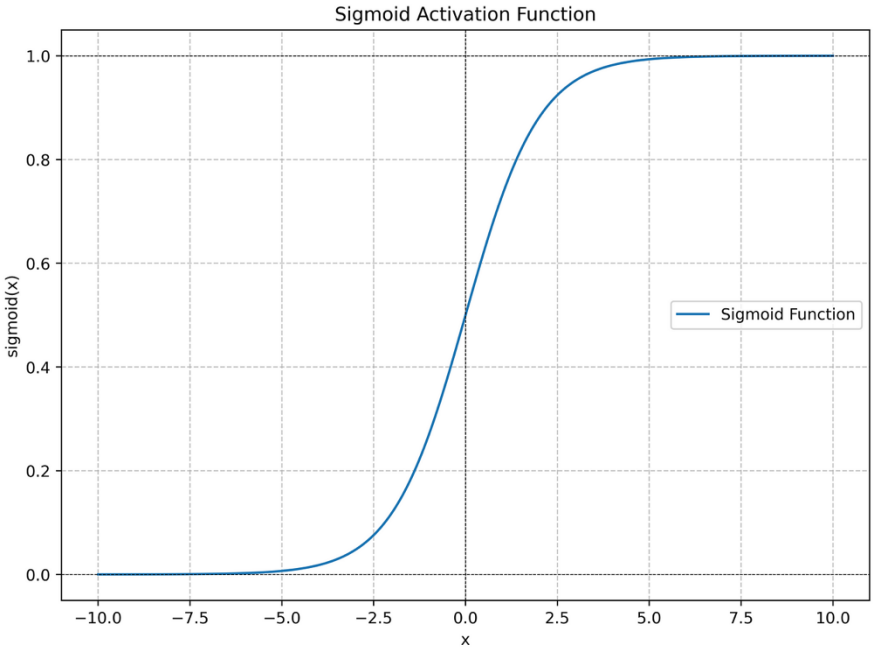
Wait, Regression again?



Logistic Regression

Function $\sigma(z) = \frac{1}{1+e^{-z}}$ is called **sigmoid**.
$\sigma(z)$: $\mathbb{R} \to [0,1]$.

It allows us to produce **"probability"** from regression output.



In classification task we need to prodice labels.
How can we get labels from output of sigmoid?

We apply $F_t(p) = \begin{cases} 1, & p < t \\ 0, & p \geq t \end{cases}$

Loss Function

Why not use MSE? $MSE = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$.

- If we set $\hat{y}_i\ (w) = \frac{1}{1+ e^{-X_i w}}$, then MSE is **non-convex**
- Also the MSE error will be bounded by 1

We maximize:

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

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

Loss Function

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

Natural way to measure the prediction quality: use Maximal likelihood:

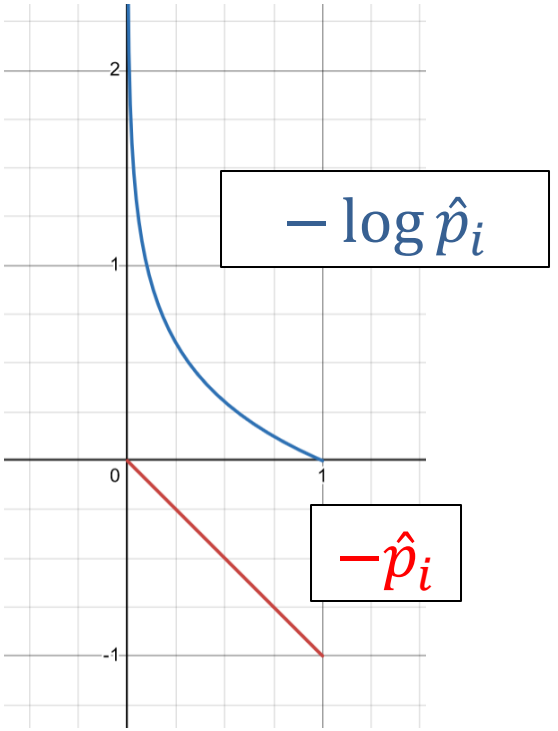
$$L(y, \hat{y}) = \prod_{i=1}^{m} \Pr(y_i | w, X_i) = \prod_{i=1}^{m} \hat{y}_i^{y_i} (1 - \hat{y}_i)^{1 - y_i} \rightarrow max$$

Loss Function

Using the logarithm can be beneficial:

$$-\log L(y_i, \hat{y}_i) = -\sum_{i=1}^{m} y_i \log(\hat{y}_i) + (1-y)\log(1-\hat{y}_i) \rightarrow min$$

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



Loss Function

$$L(w) = -\frac{1}{m}\sum_{i=1}^{m} y_i \log(\hat{y}_i) + (1-y)\log(1-\hat{y}_i).$$

Our goal is to find $\hat{w}$ that minimizes $L(w)$.

In regression we had an exact solution:

$$w = (X^T X)^{-1} X^T y.$$

Loss Function

In Logistic Regression, we do not have the **exact solution.**

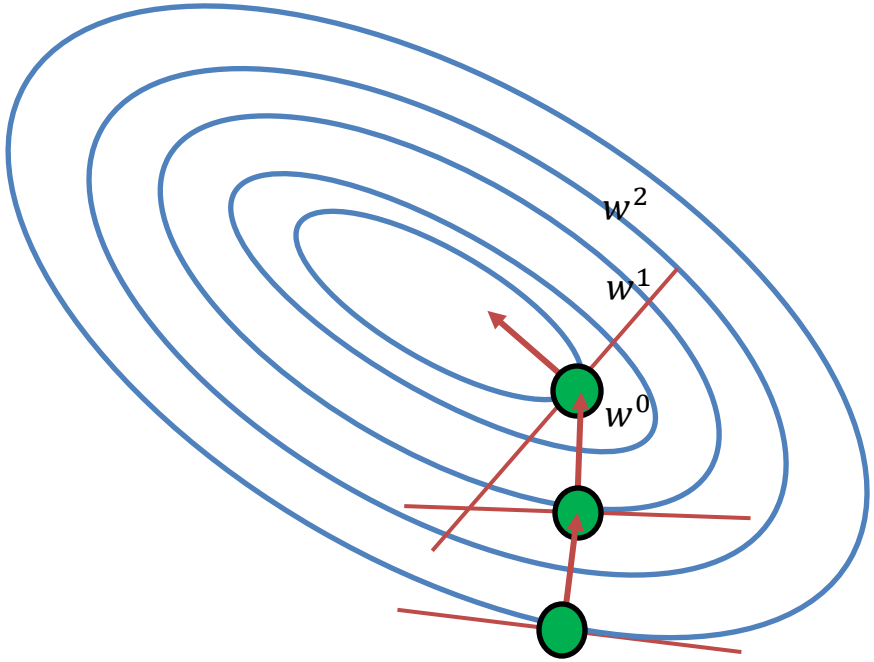
However…

The optimal $\hat{w}$ minimizes $L(w)$.

$$\nabla_w L(\hat{w}) = 0$$

L is **convex** $\Leftrightarrow \nabla_w L(w^*) = 0$ only for $w^* = \hat{w}$.

Gradient Descent



Iterative algorithm:

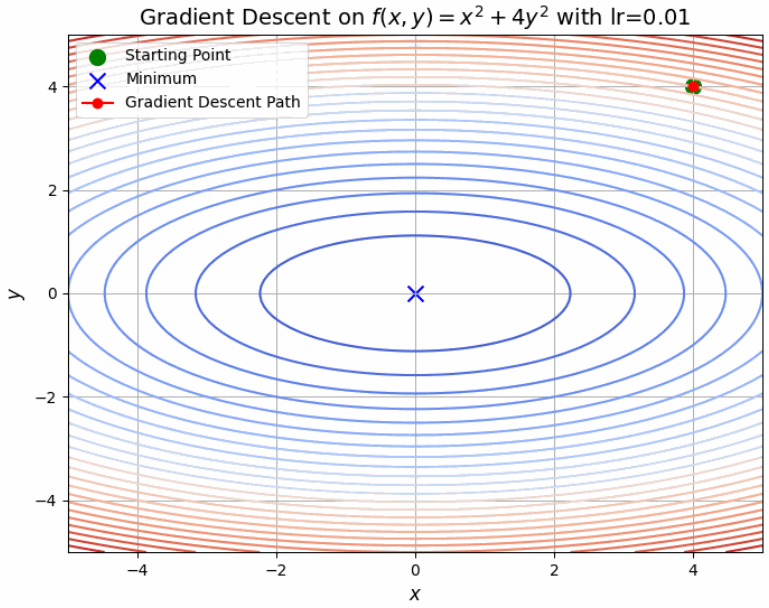
$$w^{k+1} = w^k - \alpha \nabla_w L\left(w^k\right)$$

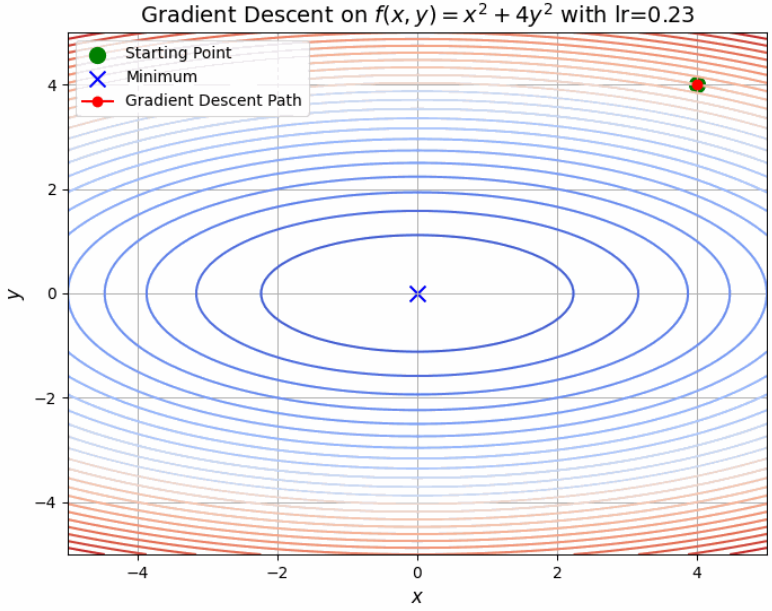
α – learning rate (LR).

Learning Rate

$$w^{k+1} = w^k - \alpha \nabla_w L\left(w^k, X_k, y_k\right)$$

It is important to select a good Learning rate

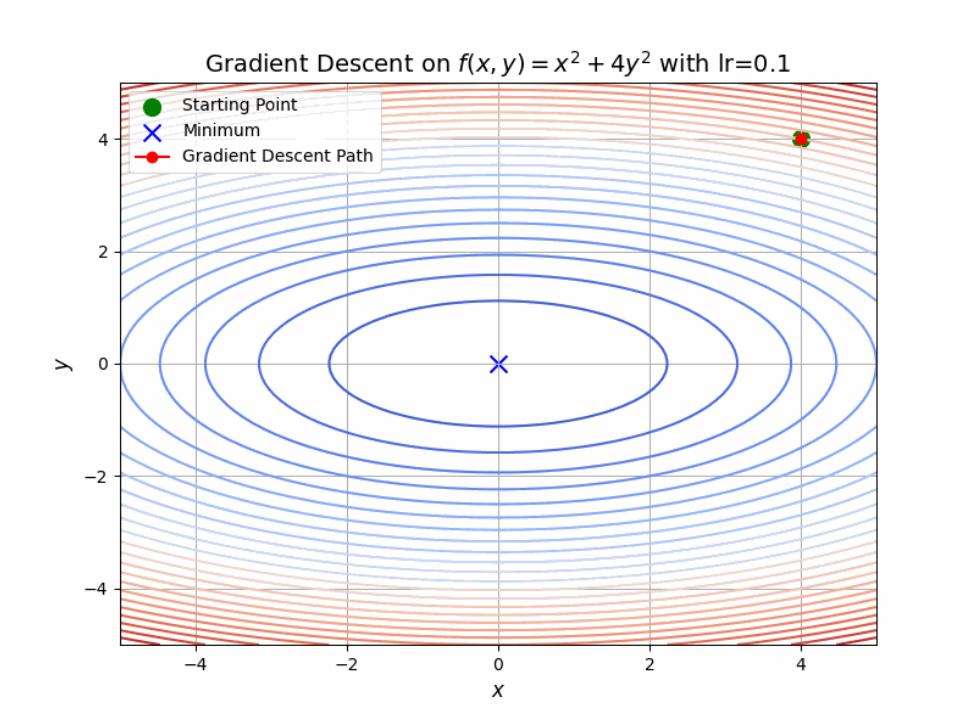




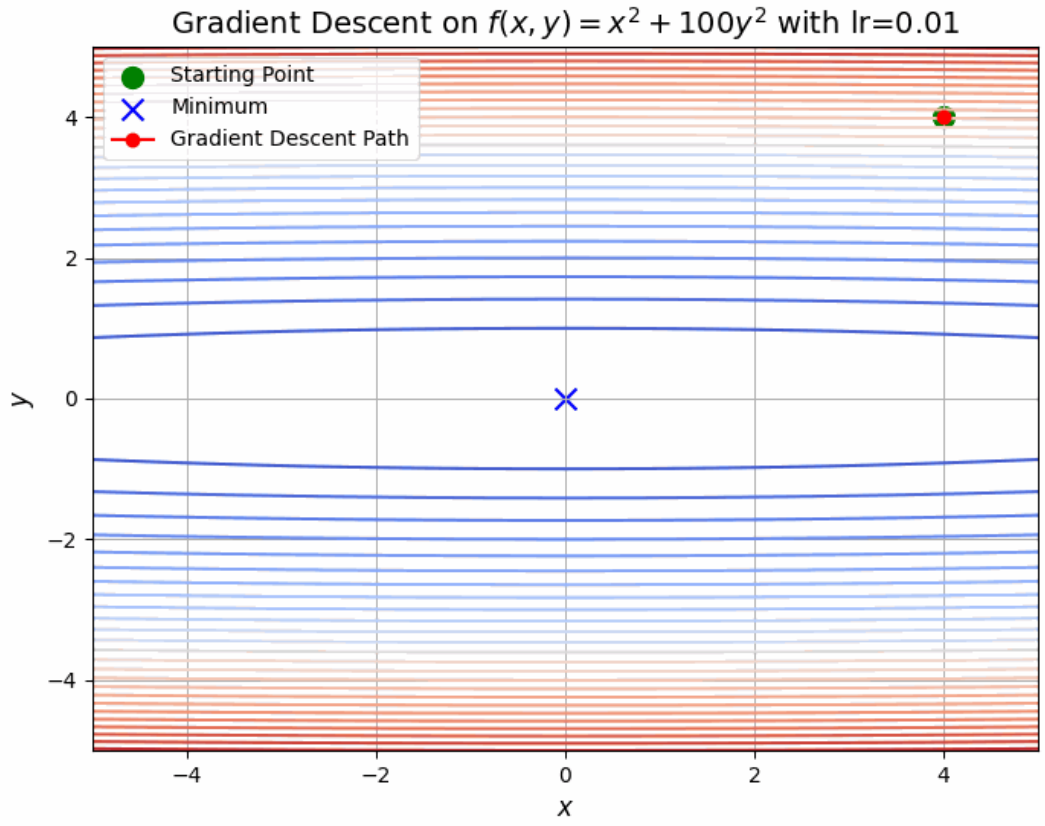
Learning Rate

$$w^{k+1} = w^k - \alpha \nabla_w L\left(w^k, X_k, y_k\right)$$

It is important to select a good Learning rate

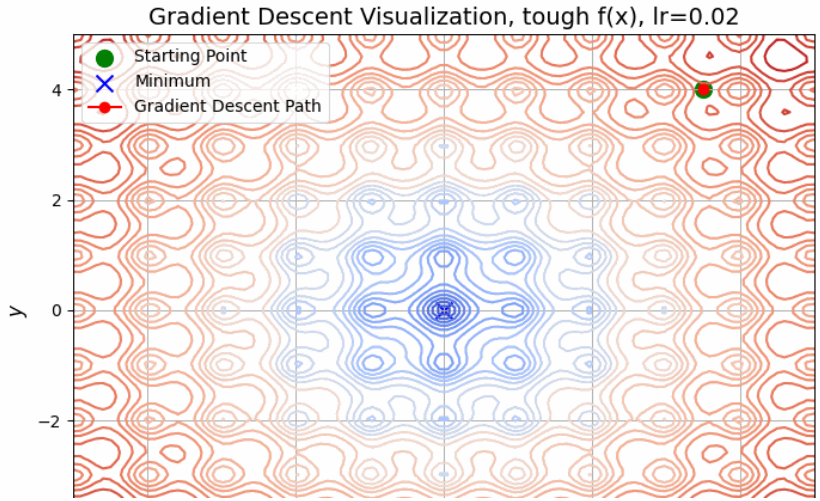


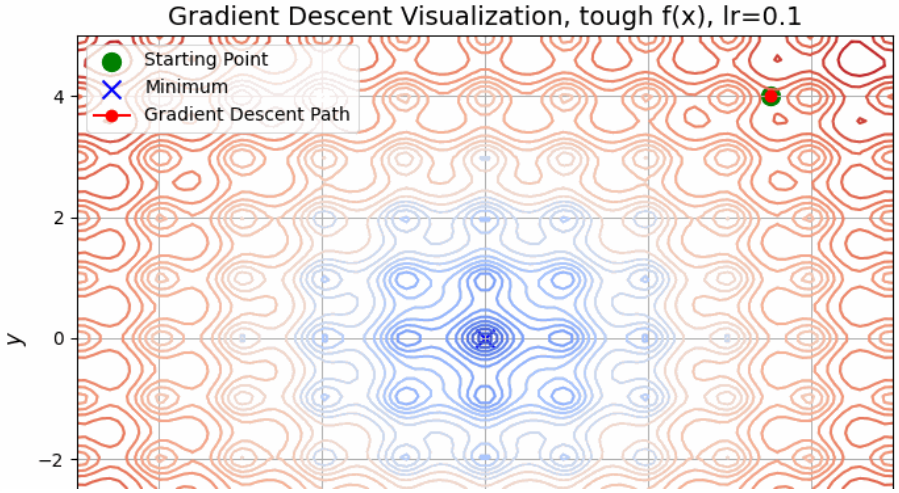
Badly conditioned problems



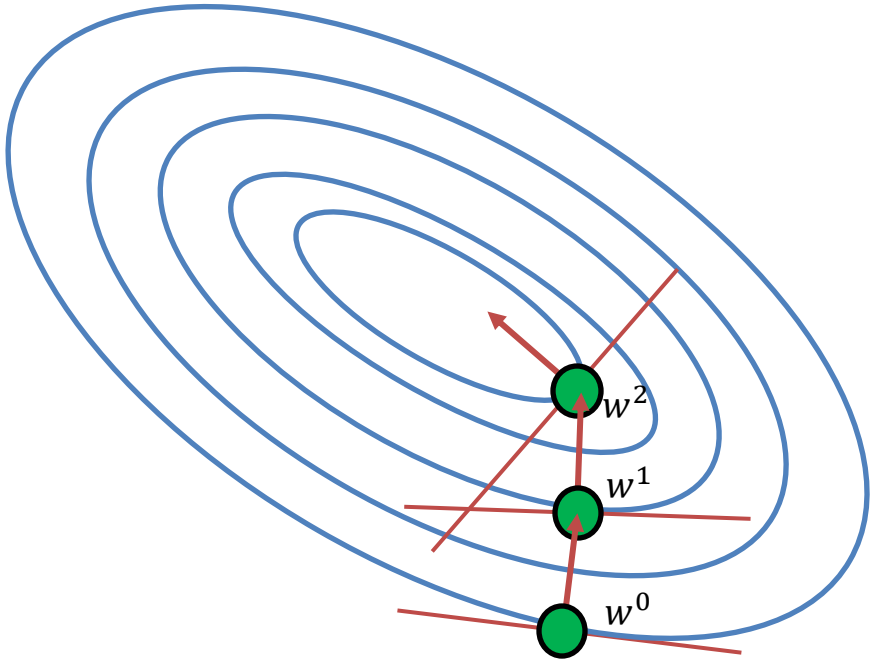
Learning Rate

Gradient descent may get stuck in local optima





Gradient Descent



Iterative algorithm:

$$w^{k+1} = w^k - \alpha \nabla_w L\left(w^k\right)$$

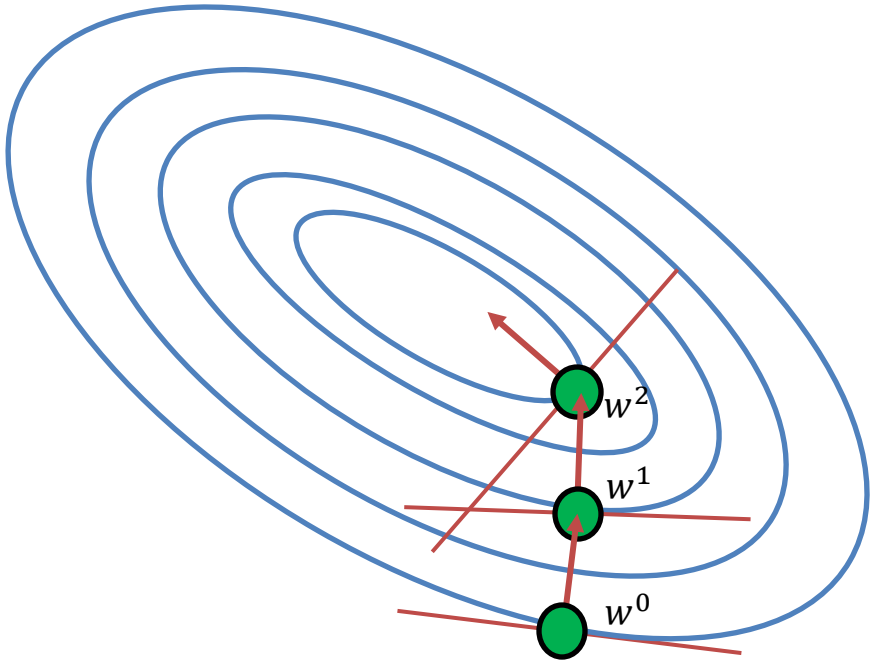
α – learning rate (LR).

In case of Logistic Regression, the gradient:

$$\nabla_w L\left(w^k\right) = -\frac{1}{m}\sum_{i=1}^{m} y_i \nabla_w \log\left(\sigma\left(X_i w^k\right)\right) + (1-y)\nabla_w \log\left(1 - \sigma\left(X_i w^k\right)\right)$$

$$= \frac{\mathbf{1}}{\boldsymbol{m}} \boldsymbol{X^T}\left(\boldsymbol{\sigma}\left(\boldsymbol{X_i w^k}\right) - \boldsymbol{y}\right).$$

Time Complexity



Iterative algorithm:

$$w^{k+1} = w^k - \alpha \nabla_w L(w^k)$$

What is the **time complexity** of one step of Gradient Descent?

Answer: $\boldsymbol{O(nm)}$, where m is the number of samples and n is the number of features. Since

$$\boldsymbol{\nabla_w L(w^k) = \frac{1}{m} X^T(\sigma(X_i w^k) - y)}.$$

Stochastic Gradient Descent

But if m(the number of samples) **is very large**:

- Computing the full gradient can take a lot of time, spent without any intermediate feedback.
- We also want to avoid storing huge matrices in memory.

Idea: compute the gradient w.r.t. the loss on a smaller sub-dataset

$$L(w, \boldsymbol{X}', \boldsymbol{y}') = -\frac{1}{m'}\sum_{i=1}^{m'} y_i \log(\widehat{y_i}) + (1-y)\log(1 - \widehat{y_i}).$$

(Mini-Batch) Stochastic Gradient Descent

$$L(w, \boldsymbol{X}', \boldsymbol{y}') = -\frac{1}{m'}\sum_{i=1}^{m'} y_i \log(\widehat{y_i}) + (1-y)\log(1 - \widehat{y_i}).$$

Time complexity now: $\boldsymbol{O(nm')}$.

If on each iteration we use the batch of size m', then this approach is called **Stochastic Gradient Descent with batch size $\boldsymbol{m'}$.**

Stochastic Gradient Descent

The extreme case is when $\boldsymbol{m' = 1}$. We take a gradient by only one sample:

$$L(w, \boldsymbol{X_i}, \boldsymbol{y_i}) = -y_i \log(\hat{y}_i) - (1 - y)\log(1 - \hat{y}_i).$$

If we select the same $(\boldsymbol{X_i}, \boldsymbol{y_i})$ on each iteration of Gradient Descent, we will get optimal w for $(\boldsymbol{X_i}, \boldsymbol{y_i})$.

It's not what we wanted☹.

The solution is to select $(\boldsymbol{X_i}, \boldsymbol{y_i})$ randomly on each iteration.

Stochastic Gradient Descent

SGD Algorithm with batchsize B:

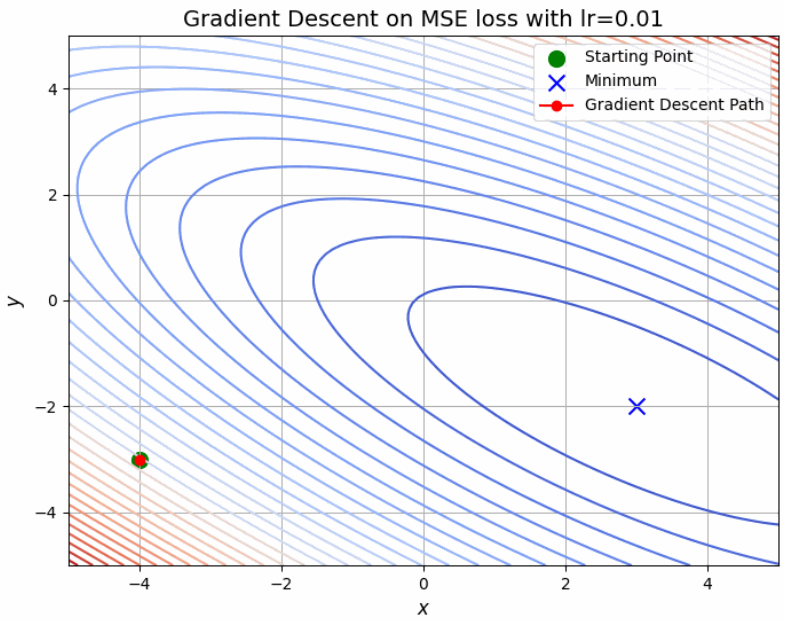
- In each epoch (set of $\frac{m}{B}$ iterations) we **randomly** reorder datapoints. It is important to shuffle every epoch! (Why?)
- Iteratively calculate: $w^{k+1} = w^k - \alpha \nabla_w L\left(w^k, X', y'\right)$, where (X', y') in each epoch come in order defined on a previous step.

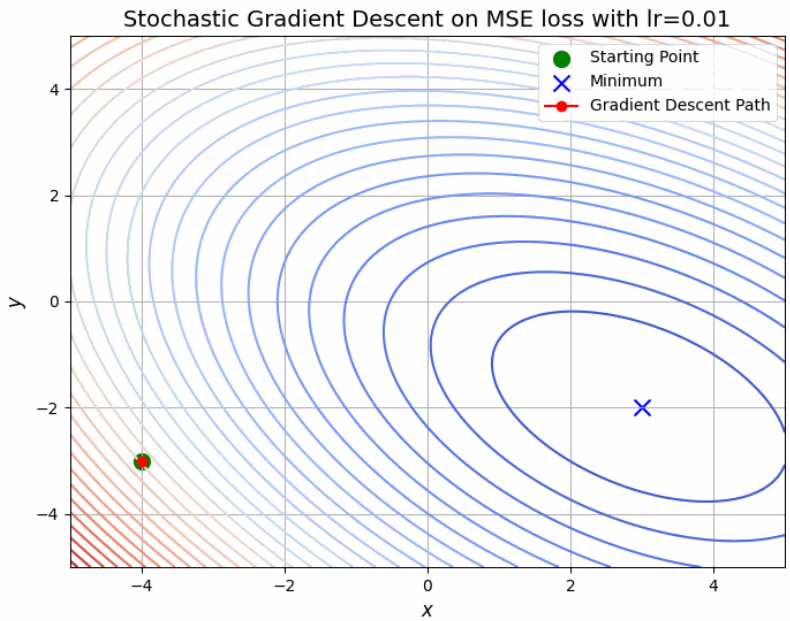
But why does SGD works?

$$\mathbb{E}[\nabla L(w, X_i, y_i)] = \nabla L(w, X, y).$$

Important intuition:
On a non-convex loss, SGD can more easily escape local minima!

Stochastic Gradient Descent





SGD is erratic; Smaller batches make it worse.

Use SGD only when full GD is impractical.

Vanilla SGD is rarely used nowadays, people prefer modified versions. We will discuss them next time.

A tricky question

GD	**SGD with batch size B**
$w_{k+1} = w_k - \frac{\alpha}{N}\sum_{i=1}^{N} \nabla_w L(y_i, x_i, w_k)$	Pick B random data points $(x_{i_s}, y_{i_s})_{s=1}^{B}$, $w_{k+1} = w_k - \frac{\alpha}{???}\sum_{s=1}^{B} \nabla_w L(y_{i_s}, x_{i_s}, w_k)$
$\nabla_w L(y, X, w) = \frac{1}{N}\sum_{i=1}^{N} \nabla_w L(y_i, x_i, w_k)$	$\nabla_w L(y, X, w) \approx \frac{1}{B}\sum_{s=1}^{B} \nabla_w L(y_{i_s}, x_{i_s}, w_k)$ A Monte Carlo estimate

A tricky question

GD	**SGD with batch size B**
$w_{k+1} = w_k - \frac{\alpha}{N}\sum_{i=1}^{N} \nabla_w L(y_i, x_i, w_k)$	Pick B random data points $(x_{i_s}, y_{i_s})_{s=1}^{B}$, $w_{k+1} = w_k - \frac{\alpha}{B}\sum_{s=1}^{B} \nabla_w L(y_{i_s}, x_{i_s}, w_k)$
$\nabla_w L(y, X, w) = \frac{1}{N}\sum_{i=1}^{N} \nabla_w L(y_i, x_i, w_k)$	$\nabla_w L(y, X, w) \approx \frac{1}{B}\sum_{s=1}^{B} \nabla_w L(y_{i_s}, x_{i_s}, w_k)$ A Monte Carlo estimate

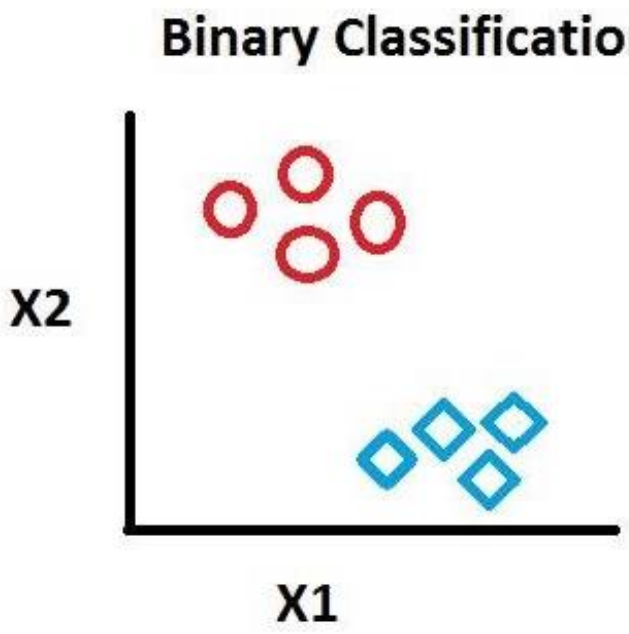
Gradient Descent Stopping

We can iterate over possible w^k indefenetly.
So **when do we stop?**

Possible stopping criterea:

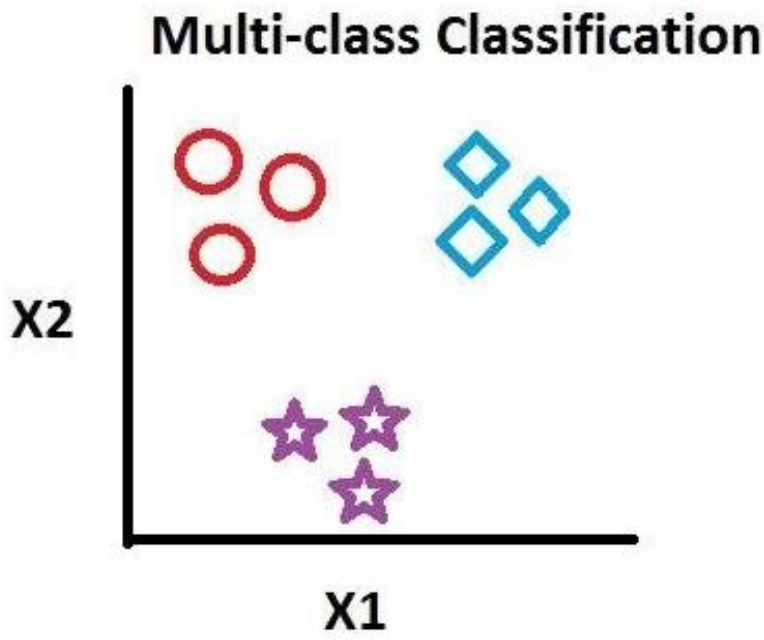
- Gradient is sufficiently small
- Parameters increment is small
- Loss function change is small
- **Maximum number of iterations reached**

Classification



Two labels:

$\mathbb{Y} = \{0, 1\}$ or $\mathbb{Y} = \{-1, 1\}$.



More labels:

Generalization of Binary.

Multiclass Classification

$$\mathbb{Y} = \{1, \ldots, k\}.$$

Idea 1 **(one-vs-all)** :

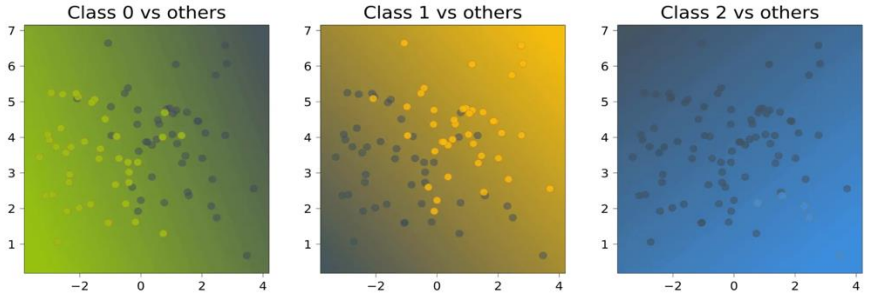
Train k binary classifiers, each separating one class from all others.

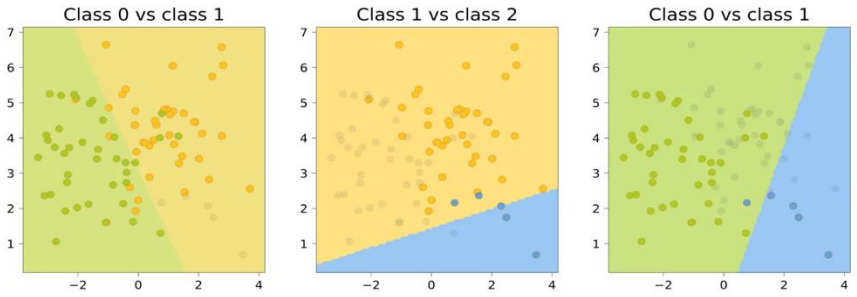
Then choose the label that yields the strongest signal.

Idea 2 **(all-vs-all)**:

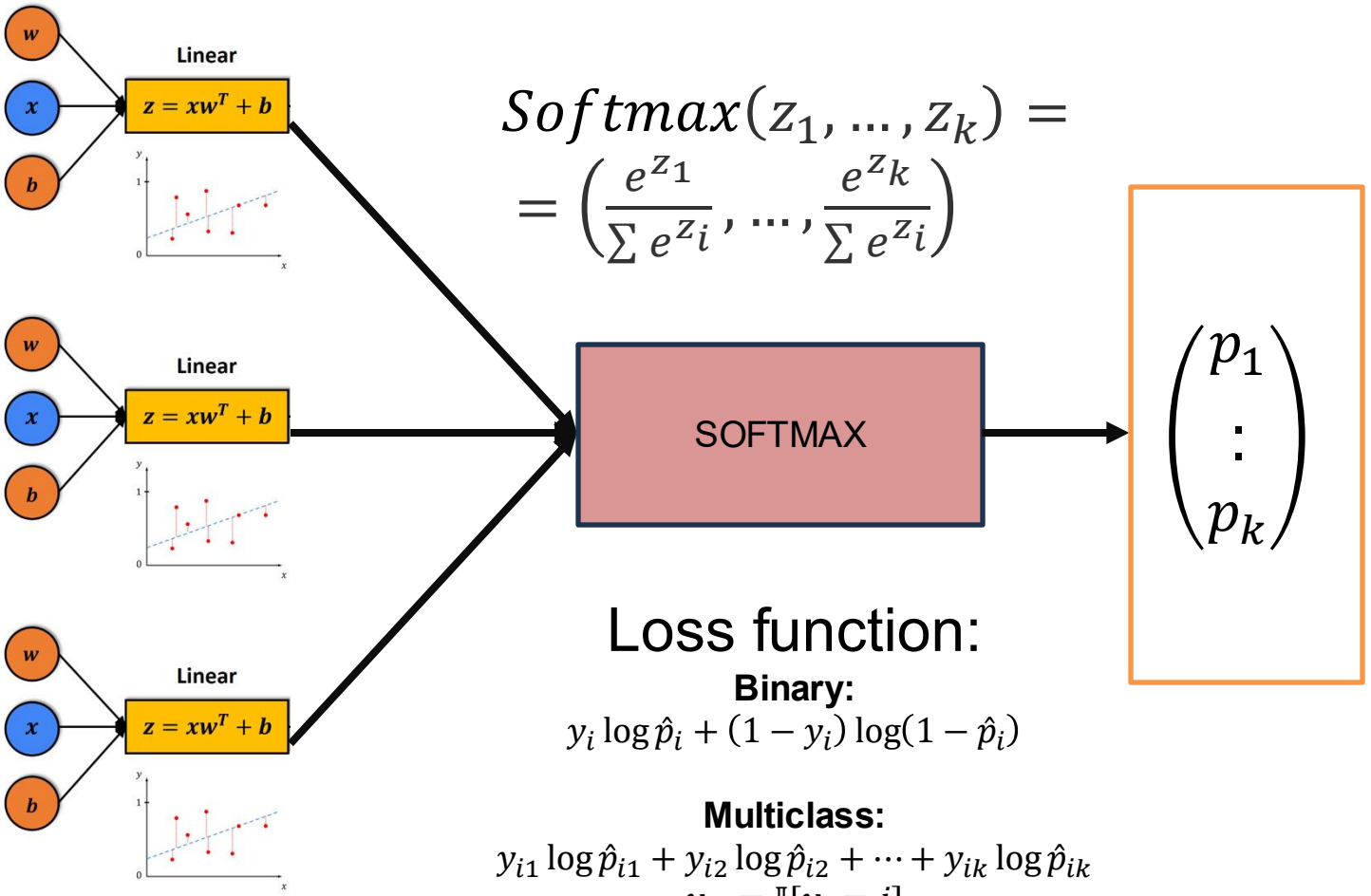
Train $\binom{k}{2}$ binary classifiers, each separating two classes.

Then pick the label that gets the most votes.





Multiclass Logistic Regression



$$Softmax(z_1, \ldots, z_k) = = \left(\frac{e^{z_1}}{\sum e^{z_i}}, \ldots, \frac{e^{z_k}}{\sum e^{z_i}}\right)$$

Loss function:

Binary:

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

Multiclass:

$$y_{i1} \log \hat{p}_{i1} + y_{i2} \log \hat{p}_{i2} + \cdots + y_{ik} \log \hat{p}_{ik}$$

$$y_{ij} = \mathbb{I}[y_i = j]$$