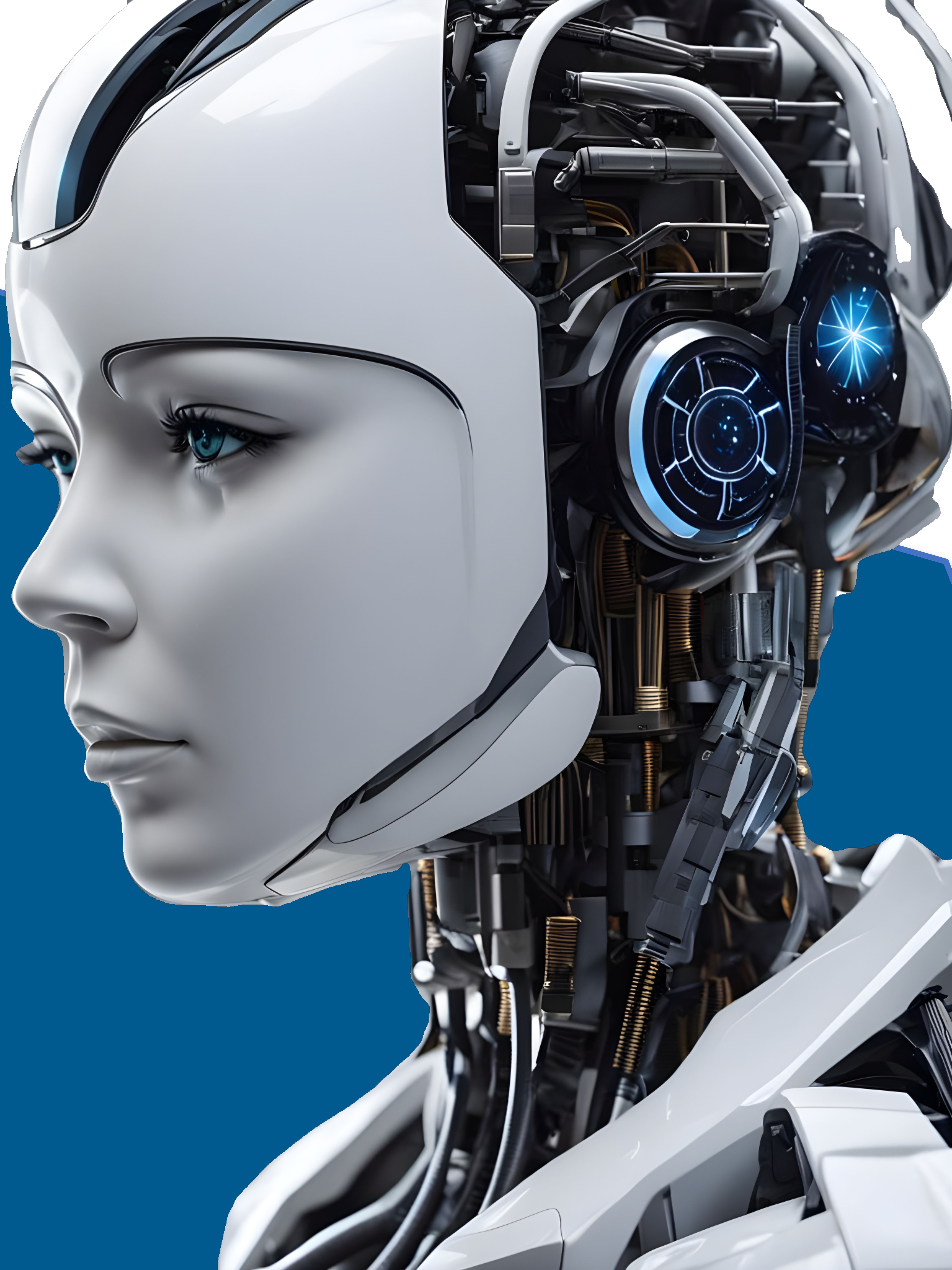
Tishk International University
IT Department
Course Code: IT-344/A

Introduction to Machine Learning

Linear Classifier SVM



Lecture 6

Spring 2024

Hemin Ibrahim, PhD
hemin.ibrahim@tiu.edu.iq

Outline

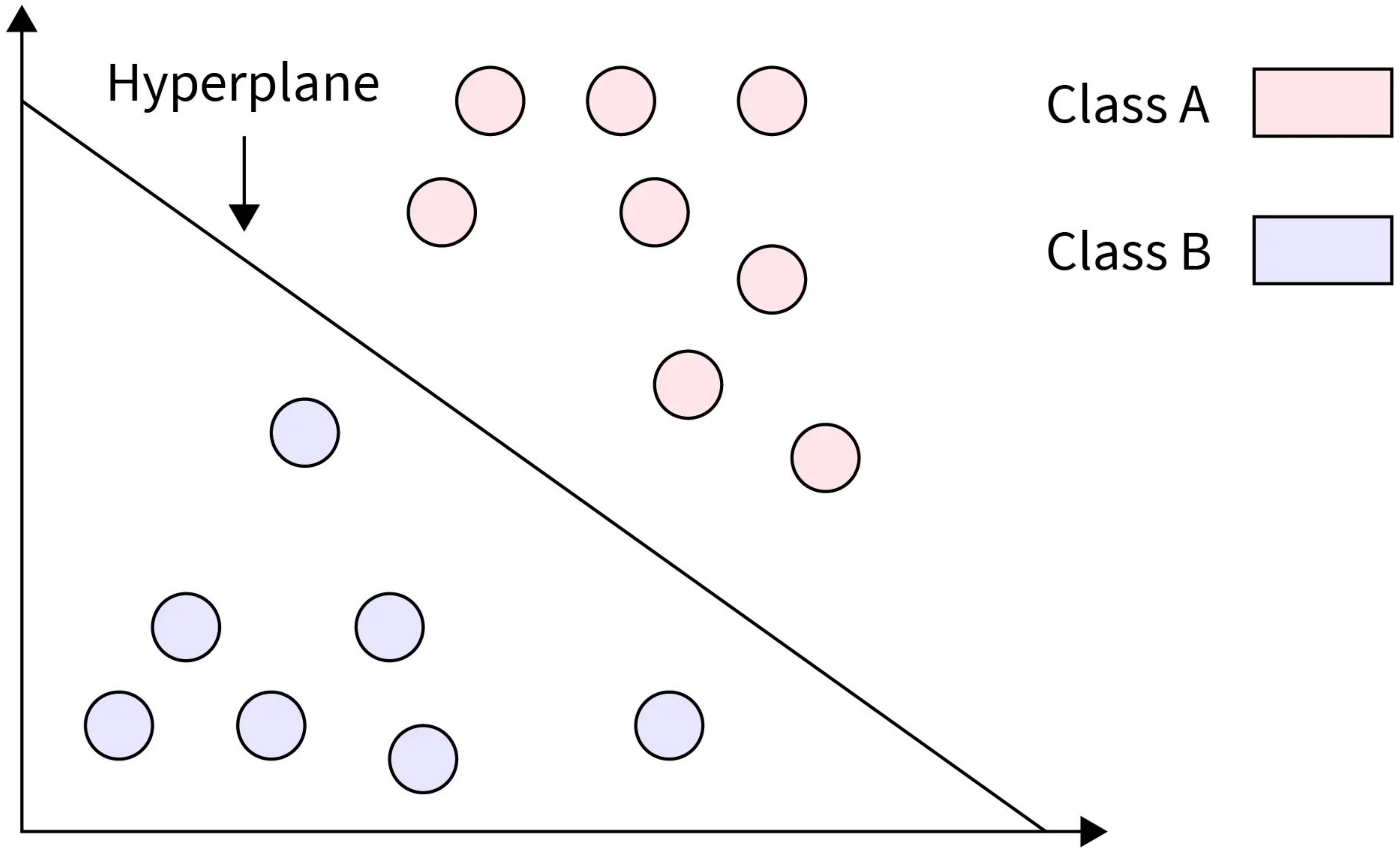
- Linear Classifiers
- Perceptron
- Support Vector Machines
- Classifier Margin
- Non-Linear
- Kernels

Objectives

- Understand the basic concept of linear classifiers and their role in machine learning.
- Learn about the architecture and components of a perceptron, including input neurons, weights, and bias.
- Understand the basic concepts of support vector machines and the principles of margin maximization.
- Learn about non-linear relationships in data and how they can be captured using non-linear models.
- Understand the concept of kernels in machine learning and its types.

Linear classifiers

- Linear classifiers are foundational models in machine learning used for classification tasks.
- They are widely used in various areas due to their simplicity and effectiveness.
- Despite their simplicity, they form the basis for more complex models and techniques in machine learning
- Linear classifiers establish decision boundaries in the feature space that separate different classes.



Linear classifiers - Equation

- The output of a linear classifier is computed using the equation:

$$h(x) = \text{sign}\left(\sum_i (x_i w_i) + b\right)$$

$h(x)$: Predicted class label.

x_i : Input feature.

w_i : Weight associated with the feature.

b: Bias term.

$$\text{sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Linear classifiers - Training

- Training involves optimizing weights and bias to minimize a loss function.
  - **Optimizing**: The goal of training is to find the best values for the weights and bias that minimize the loss function.
  - **Loss function:** The training process defines the loss function which quantifies the error between the predicted class labels and the true class labels.

- Objective: Find weights and bias that minimize classification error or maximize margin.

Linear classifiers - Example

And

X1	X2	Output
0	0	0
0	1	0
1	0	0
1	1	1

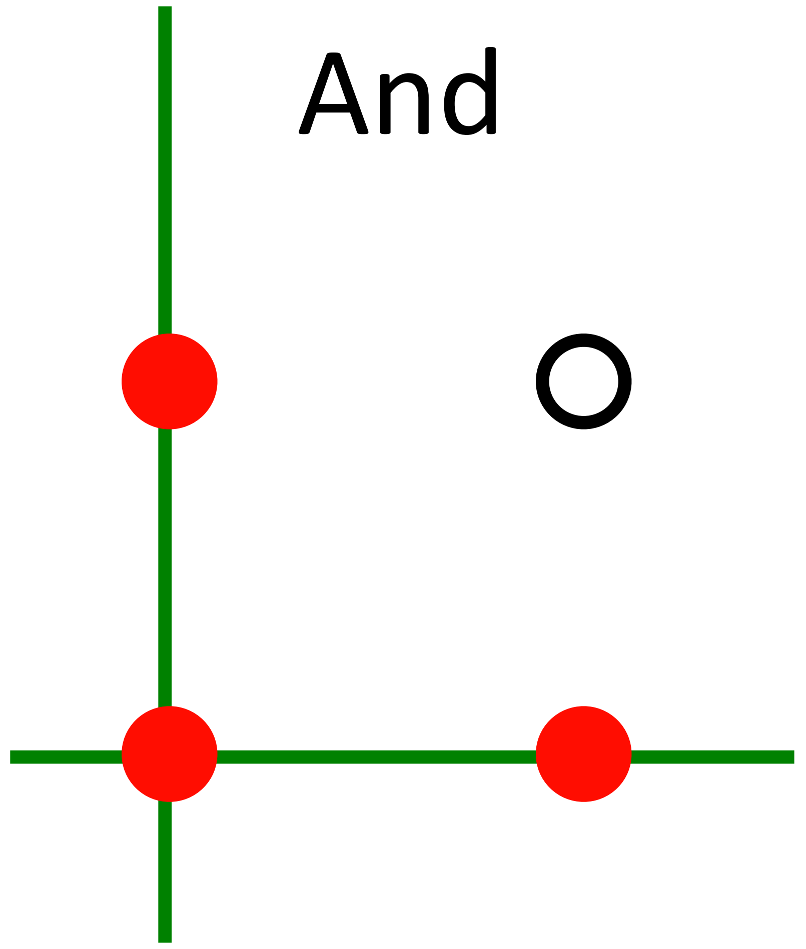
OR

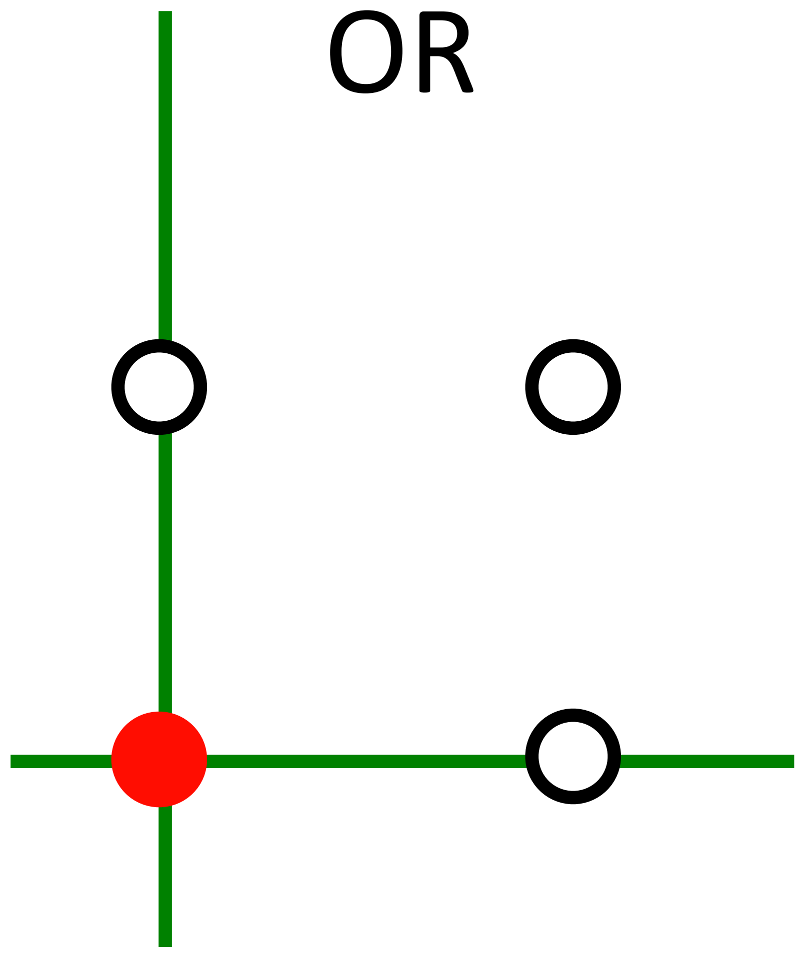
X1	X2	Output
0	0	0
0	1	1
1	0	1
1	1	1

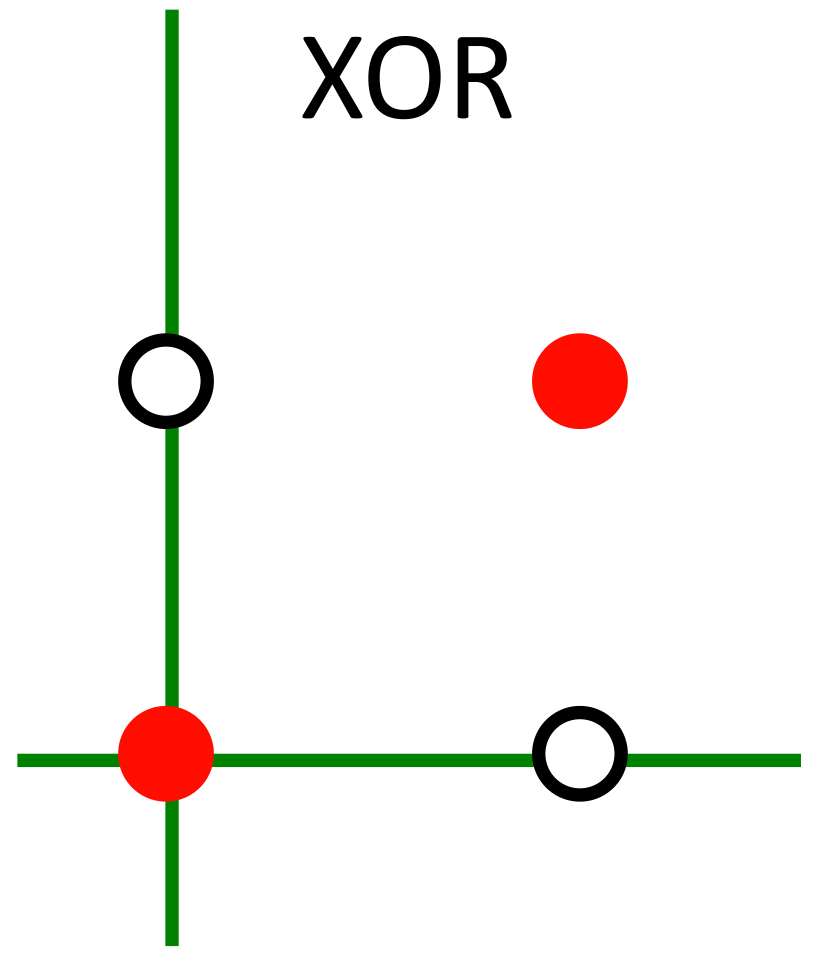
XOR

X1	X2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Linear classifiers - Example







X1	X2	Output
0	0	0
0	1	0
1	0	0
1	1	1

X1	X2	Output
0	0	0
0	1	1
1	0	1
1	1	1

X1	X2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Linear classifiers - Example

AND:

Let's initialize w1=1 and w2=1, and b=−1.5

$$h(x) = \text{sign}\left(\sum_i (x_i w_i) + b\right)$$

1. For $x_1 = 0$ **and** $x_2 = 0$

$$h(x) = sign((0 \times 1) + (0 \times 1) - 1.5) = sign(-1.5) = -1$$

2. For $x_1 = 0$ **and** $x_2 = 1$

$$h(x) = sign((0 \times 1) + (1 \times 1) - 1.5) = sign(-0.5) = -1$$

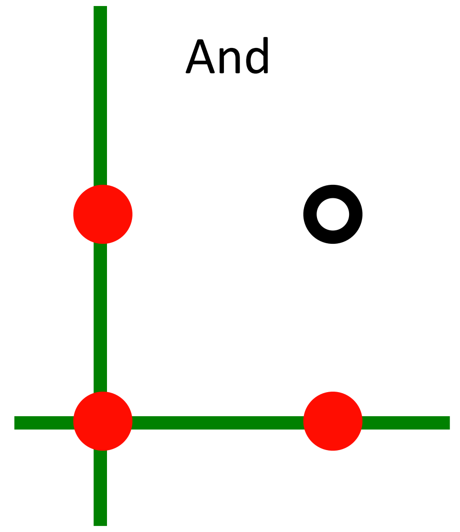
3. For $x_1 = 1$ **and** $x_2 = 0$

$$h(x) = sign((1 \times 1) + (0 \times 1) - 1.5) = sign(-0.5) = -1$$

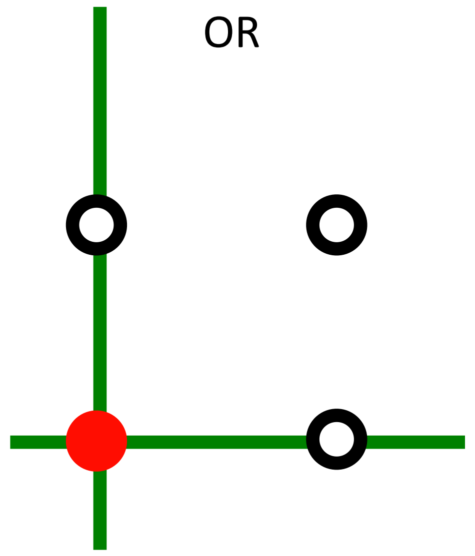
4. For $x_1 = 1$ **and** $x_2 = 1$

$$h(x) = sign((1 \times 1) + (1 \times 1) - 1.5) = sign(0.5) = 1$$

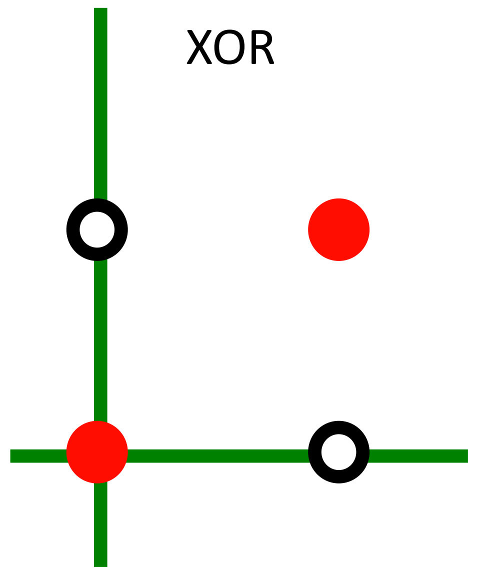
X1	X2	Output
0	0	0
0	1	0
1	0	0
1	1	1



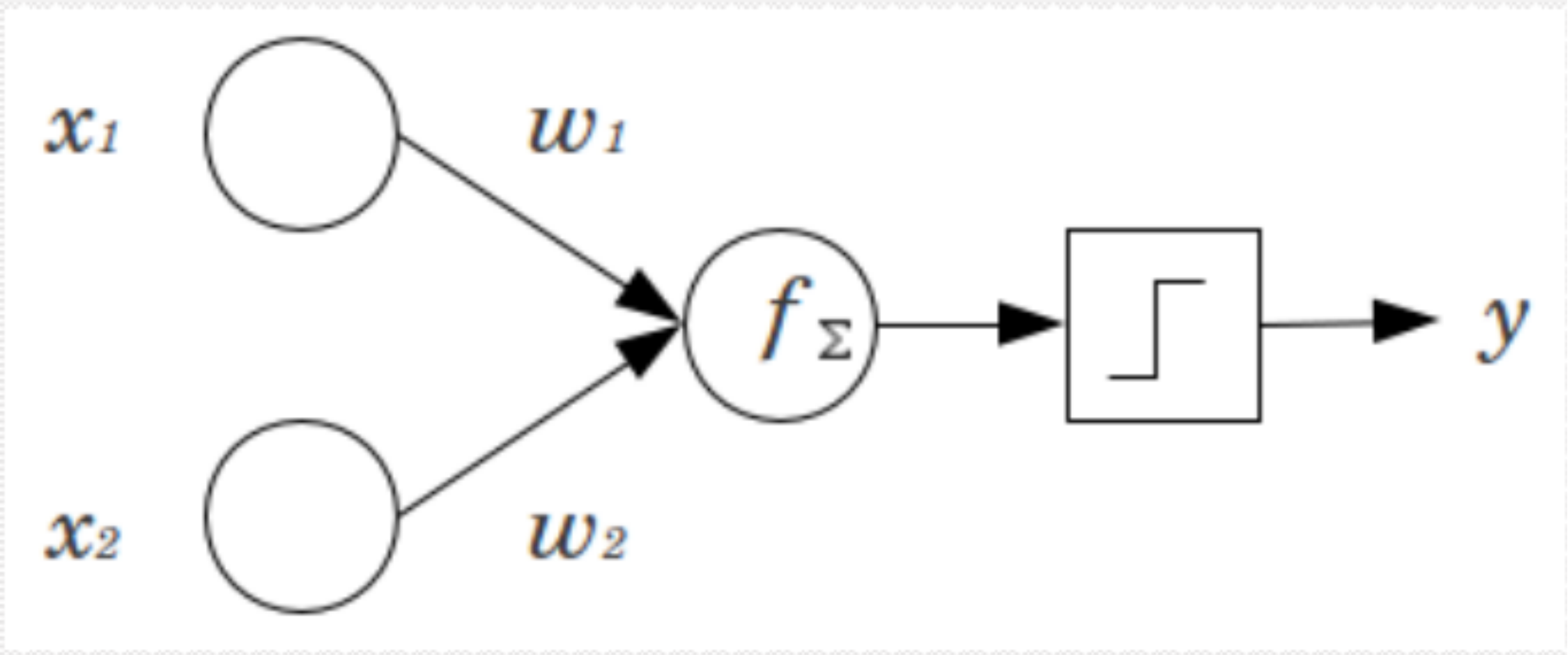
X1	X2	Output
0	0	0
0	1	1
1	0	1
1	1	1



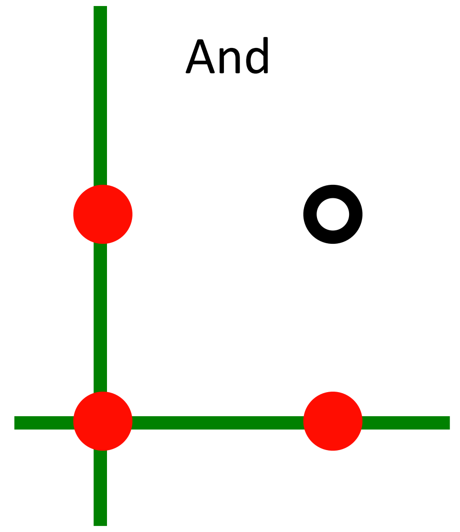
X1	X2	Output
0	0	0
0	1	1
1	0	1
1	1	0



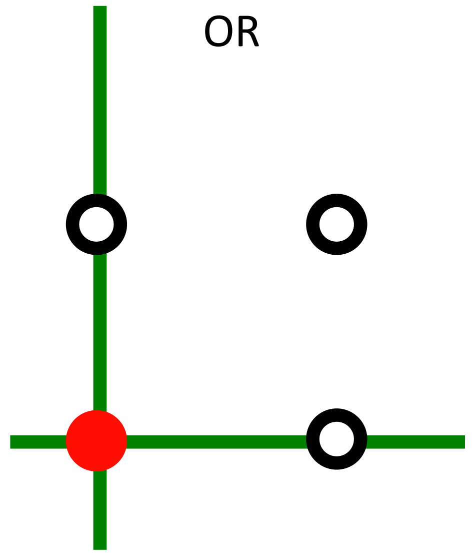
Linear classifiers - Example



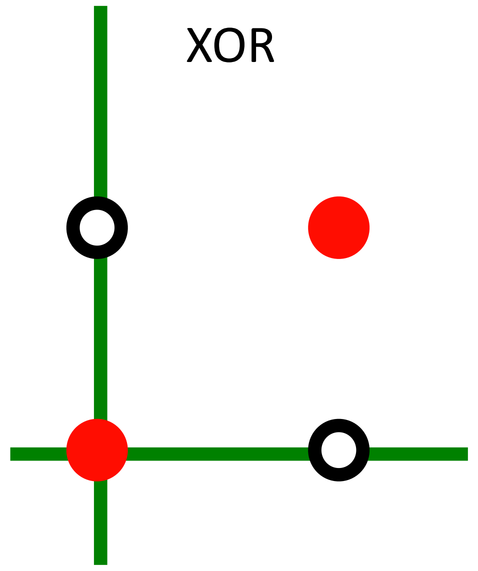
X1	X2	Output
0	0	0
0	1	0
1	0	0
1	1	1



X1	X2	Output
0	0	0
0	1	1
1	0	1
1	1	1



X1	X2	Output
0	0	0
0	1	1
1	0	1
1	1	0



Linear classifiers - Example

OR:

Let's initialize w1=1 and w2=1, and b=−0.5

$$h(x) = \text{sign}\left(\sum_i (x_i w_i) + b\right)$$

1. For $x_1 = 0$ **and** $x_2 = 0$

$$h(x) = sign((0 \times 1) + (0 \times 1) - 0.5) = sign(-1.5) = -1$$

2. For $x_1 = 0$ **and** $x_2 = 1$

$$h(x) = sign((0 \times 1) + (1 \times 1) - 0.5) = sign(0.5) = 1$$

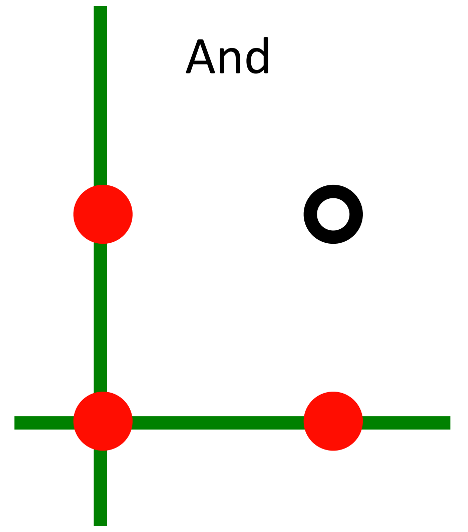
3. For $x_1 = 1$ **and** $x_2 = 0$

$$h(x) = sign((1 \times 1) + (0 \times 1) - 0.5) = sign(0.5) = 1$$

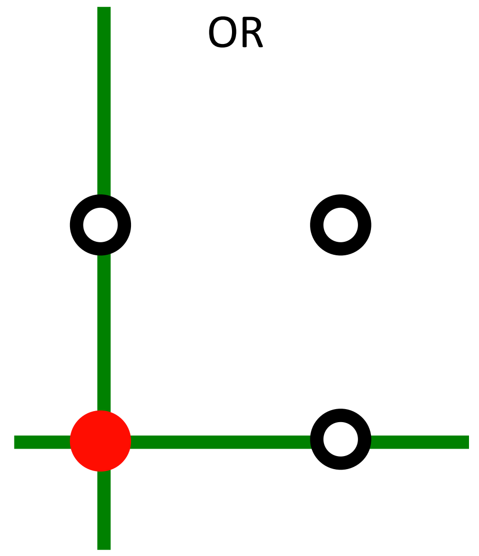
4. For $x_1 = 1$ **and** $x_2 = 1$

$$h(x) = sign((1 \times 1) + (1 \times 1) - 0.5) = sign(1.5) = 1$$

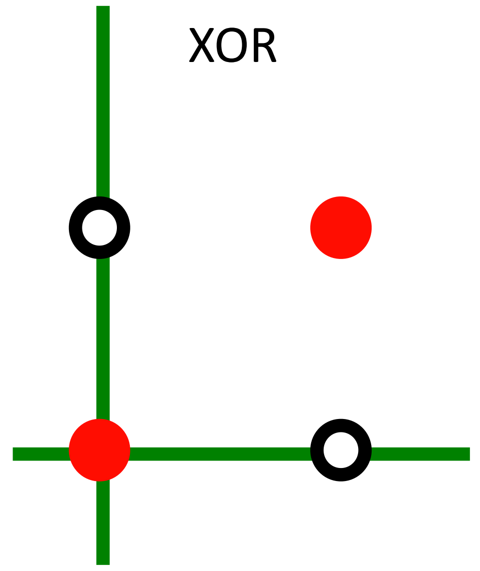
X1	X2	Output
0	0	0
0	1	0
1	0	0
1	1	1



X1	X2	Output
0	0	0
0	1	1
1	0	1
1	1	1



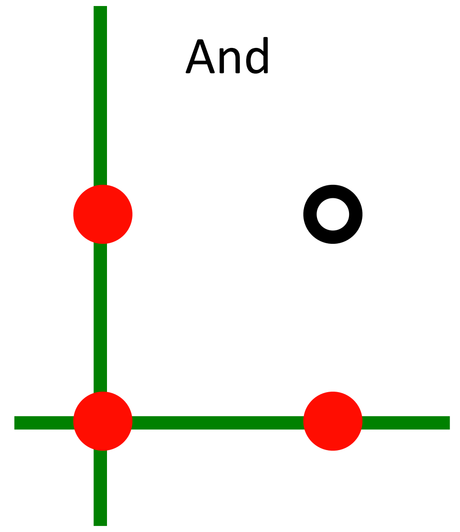
X1	X2	Output
0	0	0
0	1	1
1	0	1
1	1	0



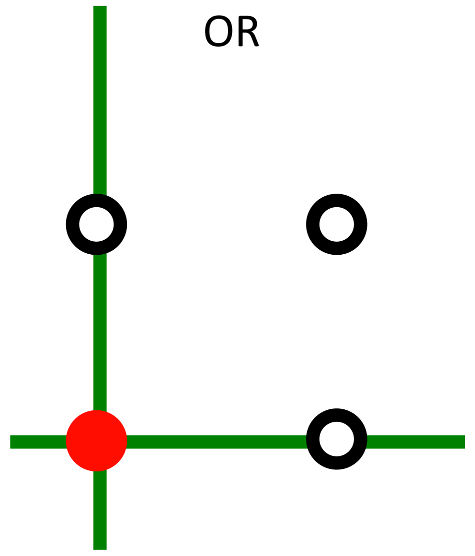
Linear classifiers - Example

What about XOR?

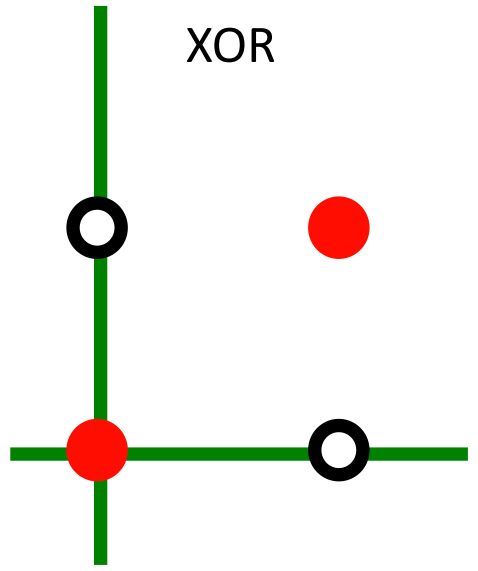
X1	X2	Output
0	0	0
0	1	0
1	0	0
1	1	1



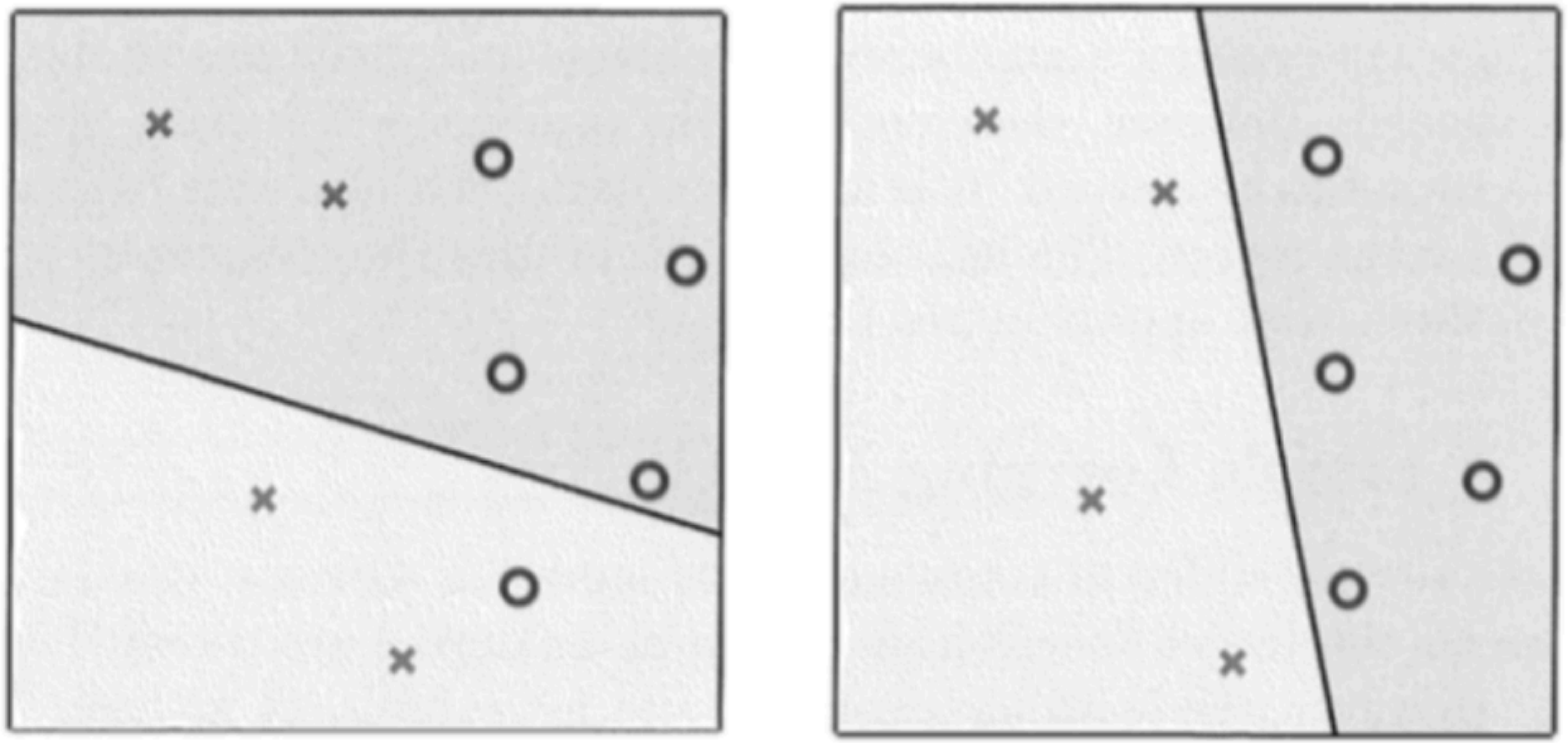
X1	X2	Output
0	0	0
0	1	1
1	0	1
1	1	1



X1	X2	Output
0	0	0
0	1	1
1	0	1
1	1	0



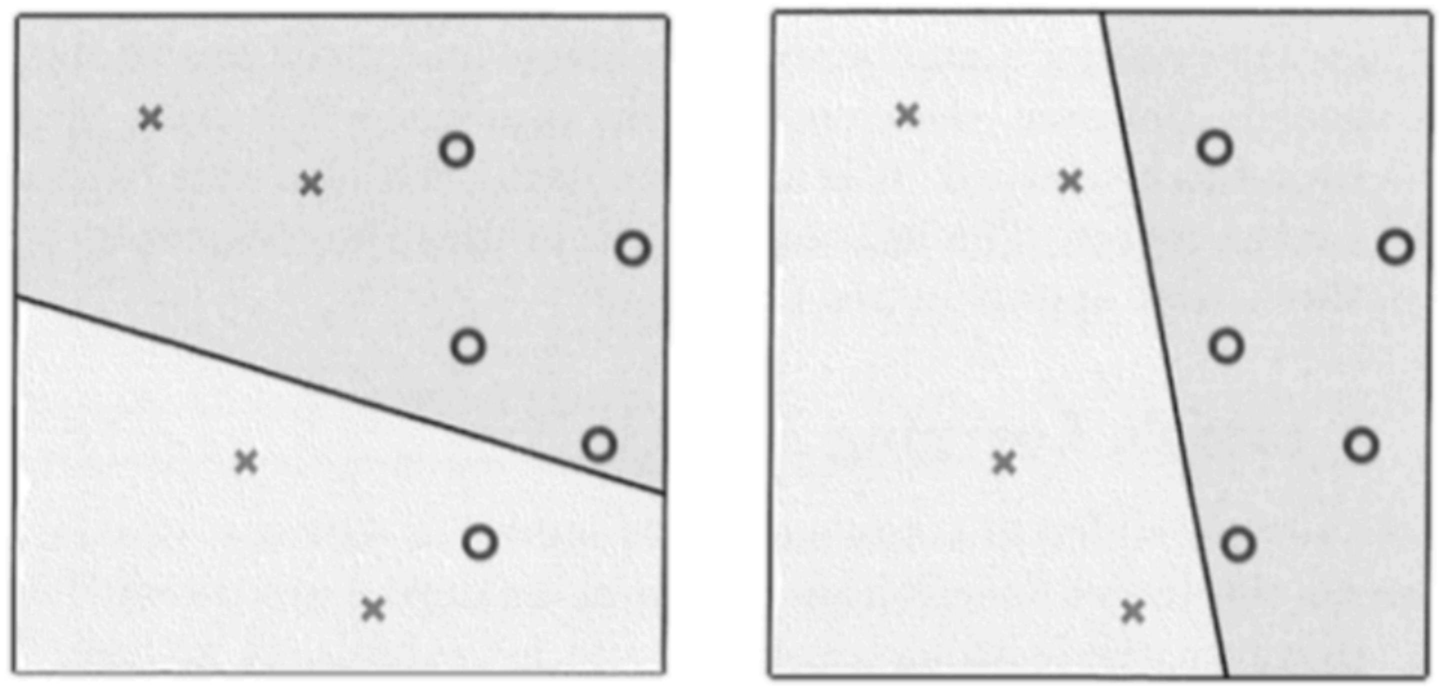
Linear classifiers



(a) Misclassified data (b) Perfectly classified data

Perceptron

- The perceptron is one of the simplest forms of neural networks, introduced by Frank Rosenblatt in 1957.
- It's a type of linear classifier used for binary classification tasks.
- This model of H is called the perceptron.
- The learning algorithm will search H by looking for **weights** and **bias** that perform well on the dataset.



(a) Misclassified data

(b) Perfectly classified data

Perceptron

- Mathematically, the output of the perceptron can be expressed as

$$h(x) = \begin{cases} 1 & \text{if } \sum_i x_i(w_i) + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

where

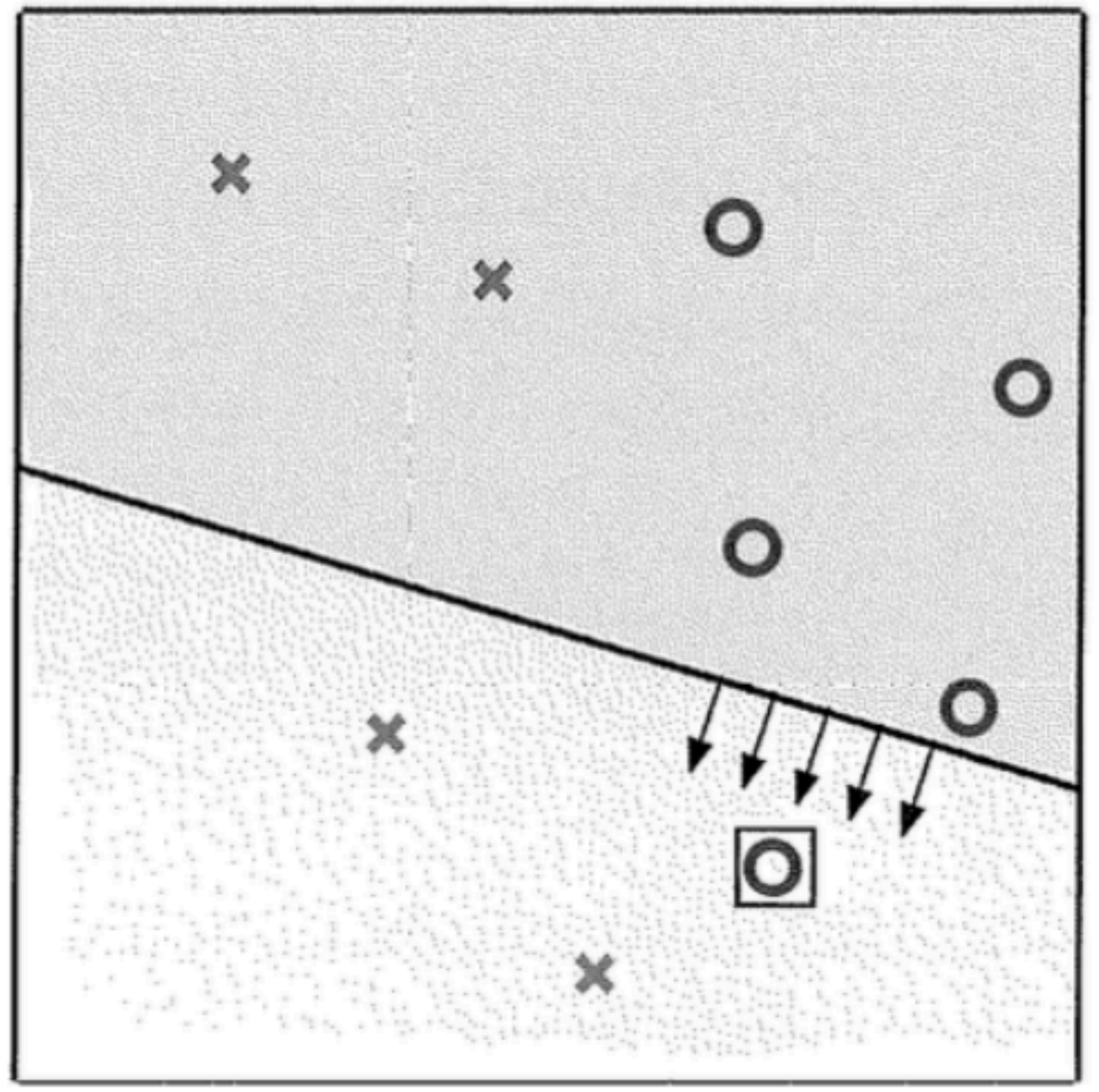
x_i represents the input feature,

w_i represents the weight associated with the feature, and

b is the bias term.

Adjust the perceptron weights automatically

- A perceptron learns by adjusting its weights. It does this automatically based on mistakes it makes during training.
- The perceptron compares its predictions to the correct answers. If it's wrong, it updates its weights to improve its accuracy.
- Start by initializing the weights (w_i) to small random values. The bias term (b) can also be initialized randomly or set to zero.



Adjust the perceptron weights automatically

Iterative Training

- For each training example (x, y), where x is the input feature vector and y is the true class label:
- Compute the predicted output $h(x)$ using the current weights and bias:

$$h(x) = \text{sign}\left(\sum_i (x_i w_i) + b\right)$$

- Compare the predicted output $h(x)$ with the true output y.
- If the predicted output $h(x)$ is incorrect (misclassified), update the weights and bias.

Adjust the perceptron weights automatically

Iterative Training

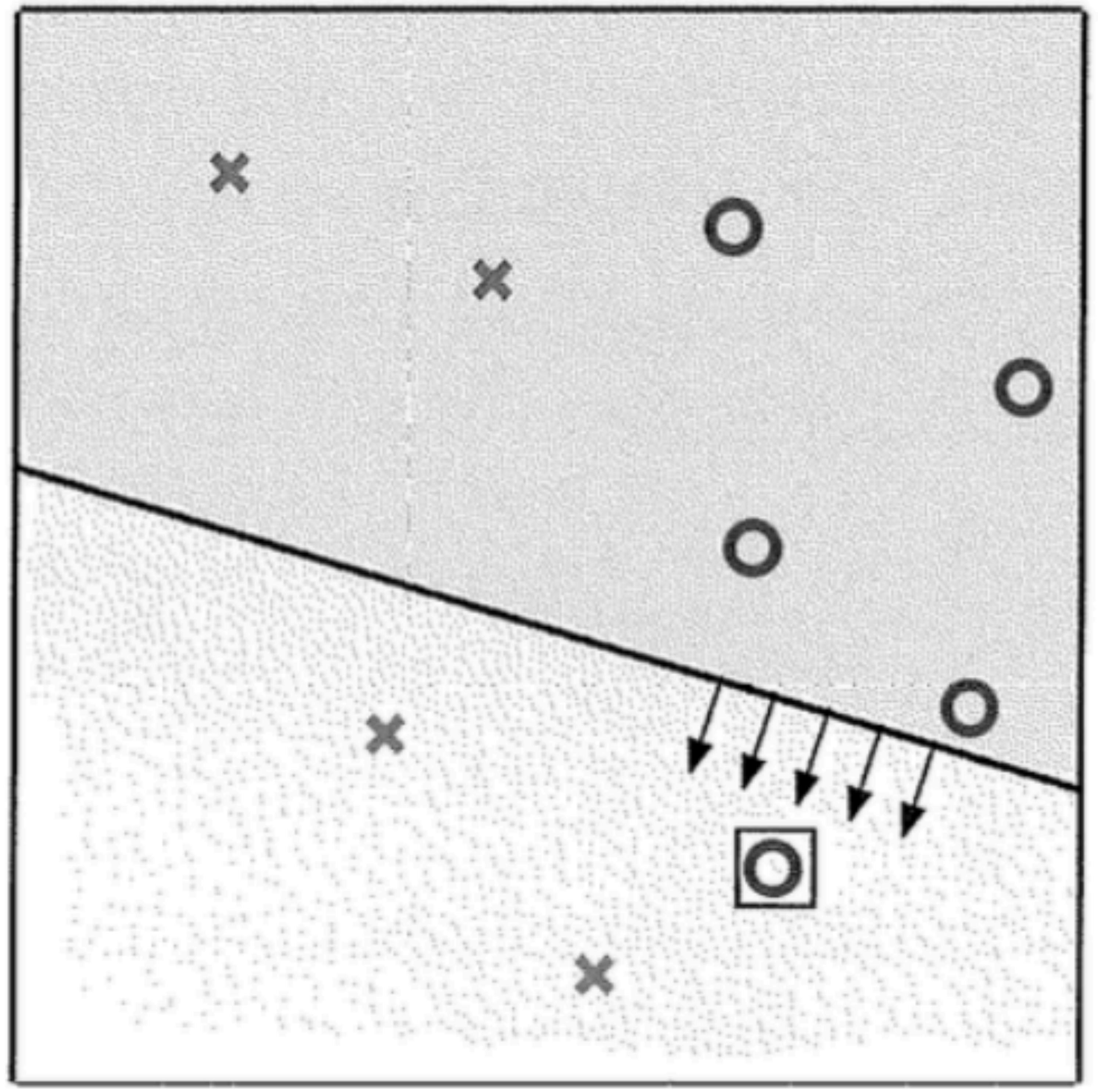
start: The weight vector $\mathbf{w}_0$ is generated randomly,
set $t := 0$

test: A vector $\mathbf{x} \in P \cup N$ is selected randomly,

if $\mathbf{x} \in P$ and $\mathbf{w}_t \cdot \mathbf{x} > 0$ go to *test*,
if $\mathbf{x} \in P$ and $\mathbf{w}_t \cdot \mathbf{x} \leq 0$ go to *add*,
if $\mathbf{x} \in N$ and $\mathbf{w}_t \cdot \mathbf{x} < 0$ go to *test*,
if $\mathbf{x} \in N$ and $\mathbf{w}_t \cdot \mathbf{x} \geq 0$ go to *subtract*.

add: set $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{x}$ and $t := t + 1$, goto *test*

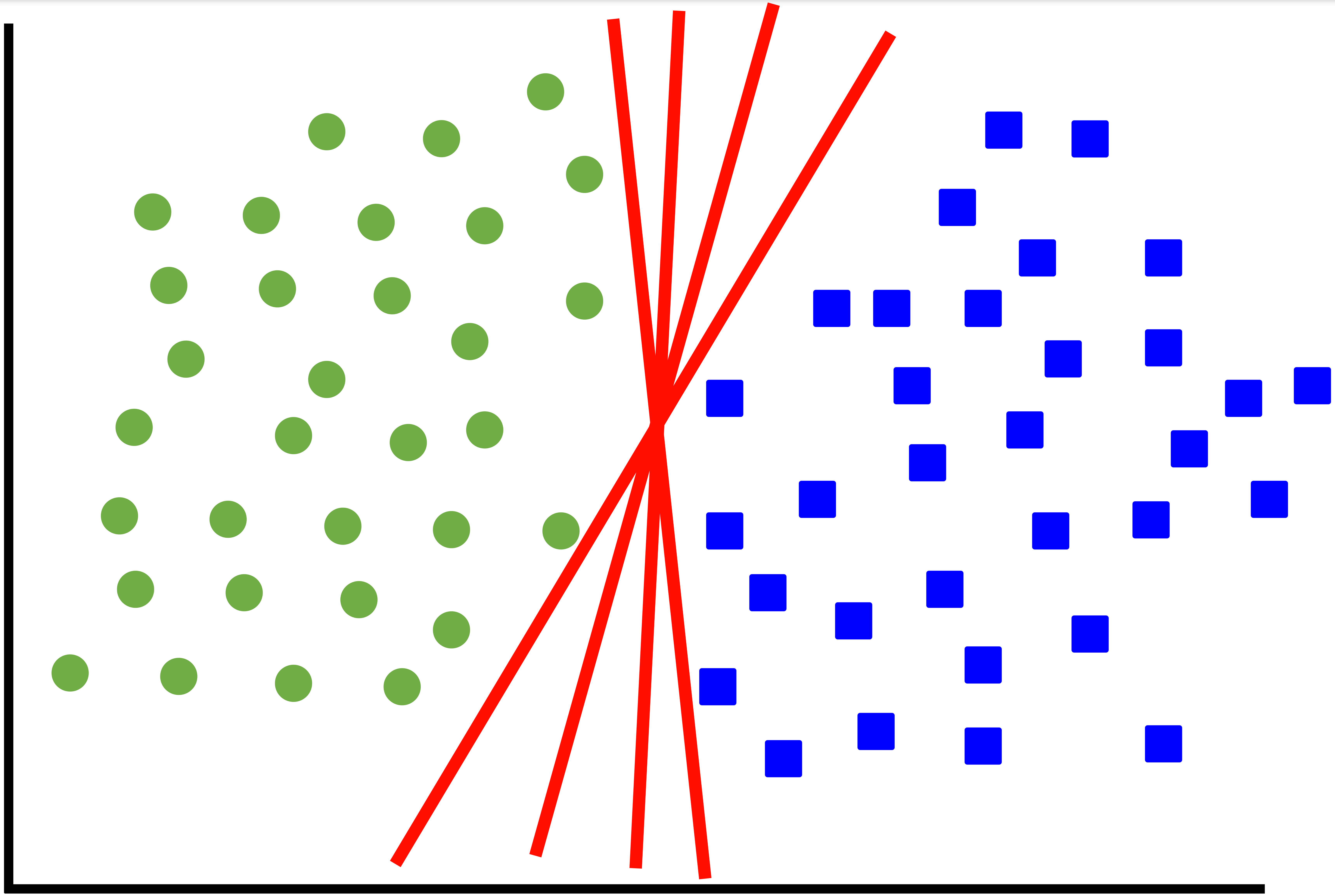
subtract: set $\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{x}$ and $t := t + 1$, goto *test*



Introduction to Support Vector Machines

- The **Support Vector Machine** (SVM) is a supervised machine learning algorithm that can effectively tackle both classification and regression challenges.
- SVM was first introduced in 1992
- SVM is now recognized as an important example of 'kernel methods,' a fundamental area within machine learning.

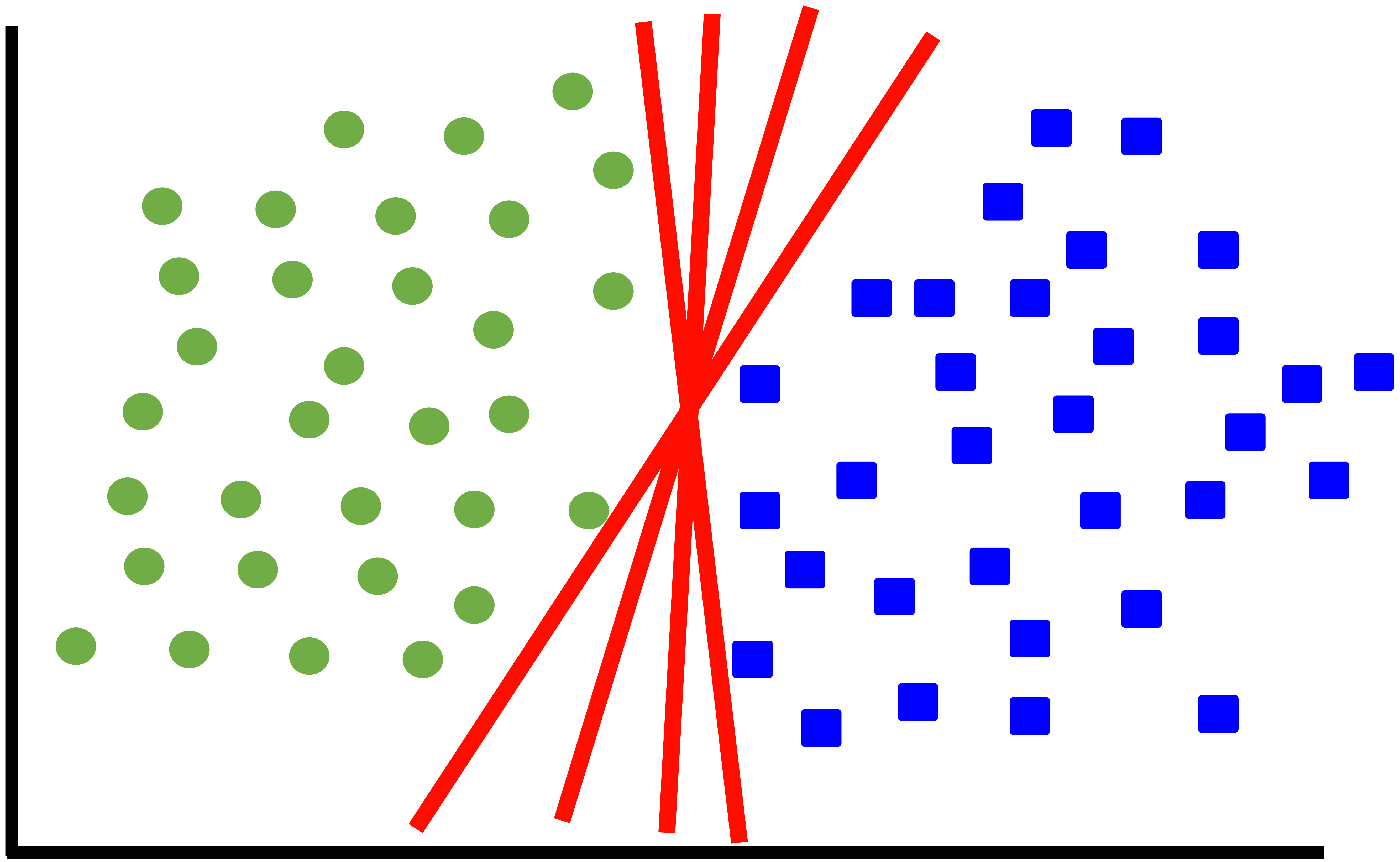
How would you classify this data?



How would you classify this data?

Any of these would be fine..

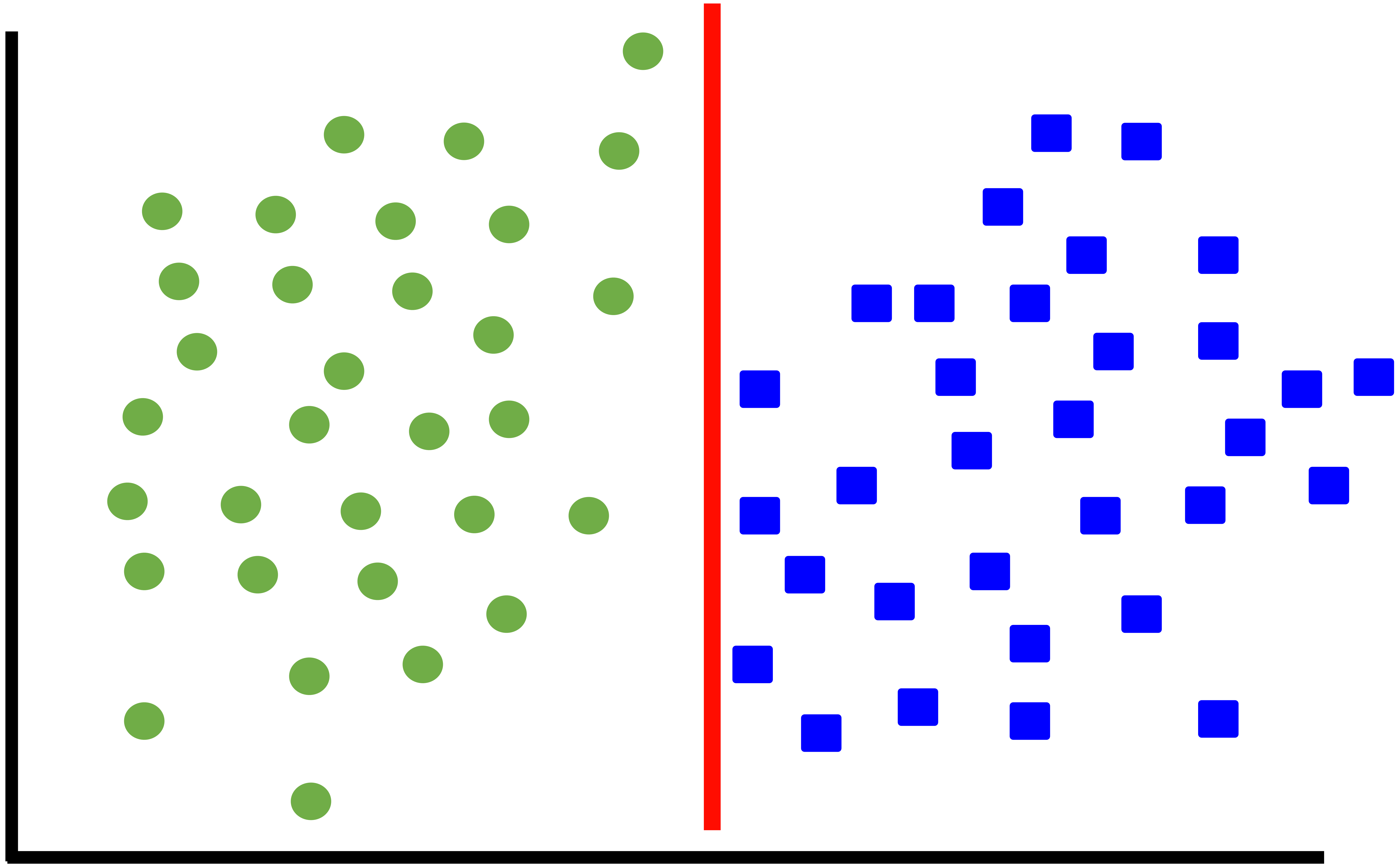
..but which is best?



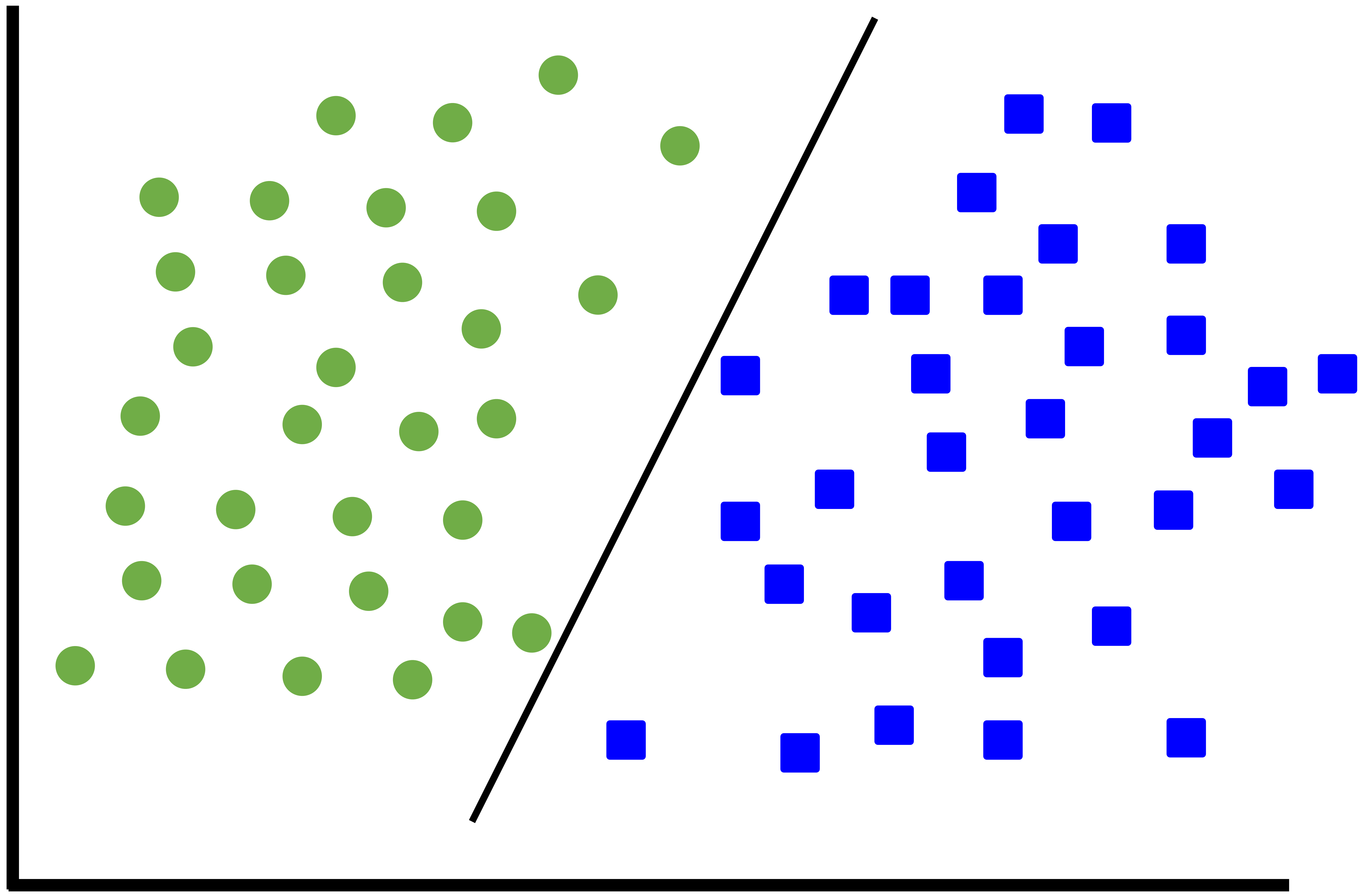
How would you classify this data?

Any of these would be fine..

..but which is best?



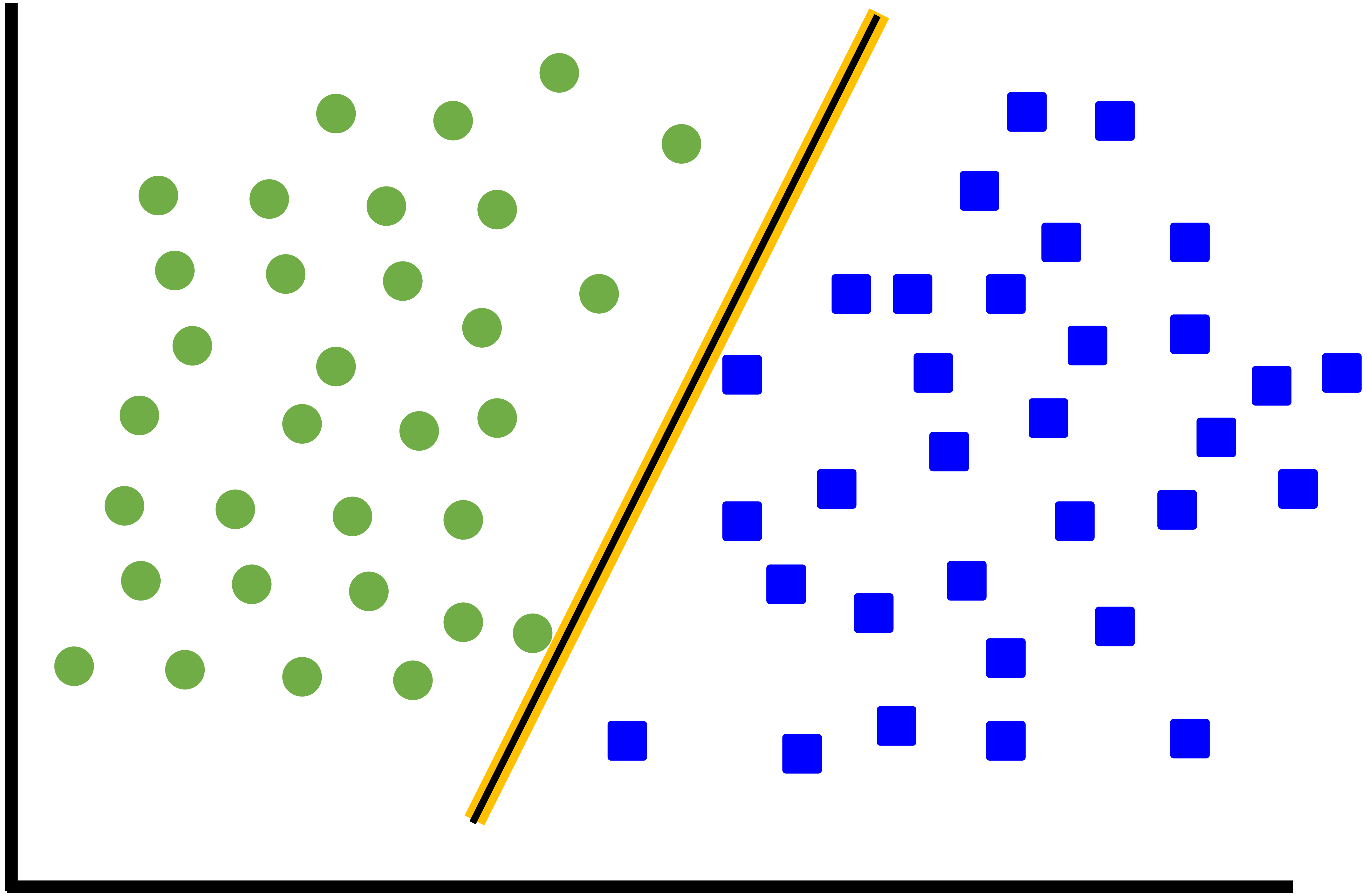
How would you classify this data?



Classifier Margin

The margin of a linear classifier is the distance from the decision boundary to the closest data point from either class.

It represents the "safety buffer" or "margin of error" of the classifier.

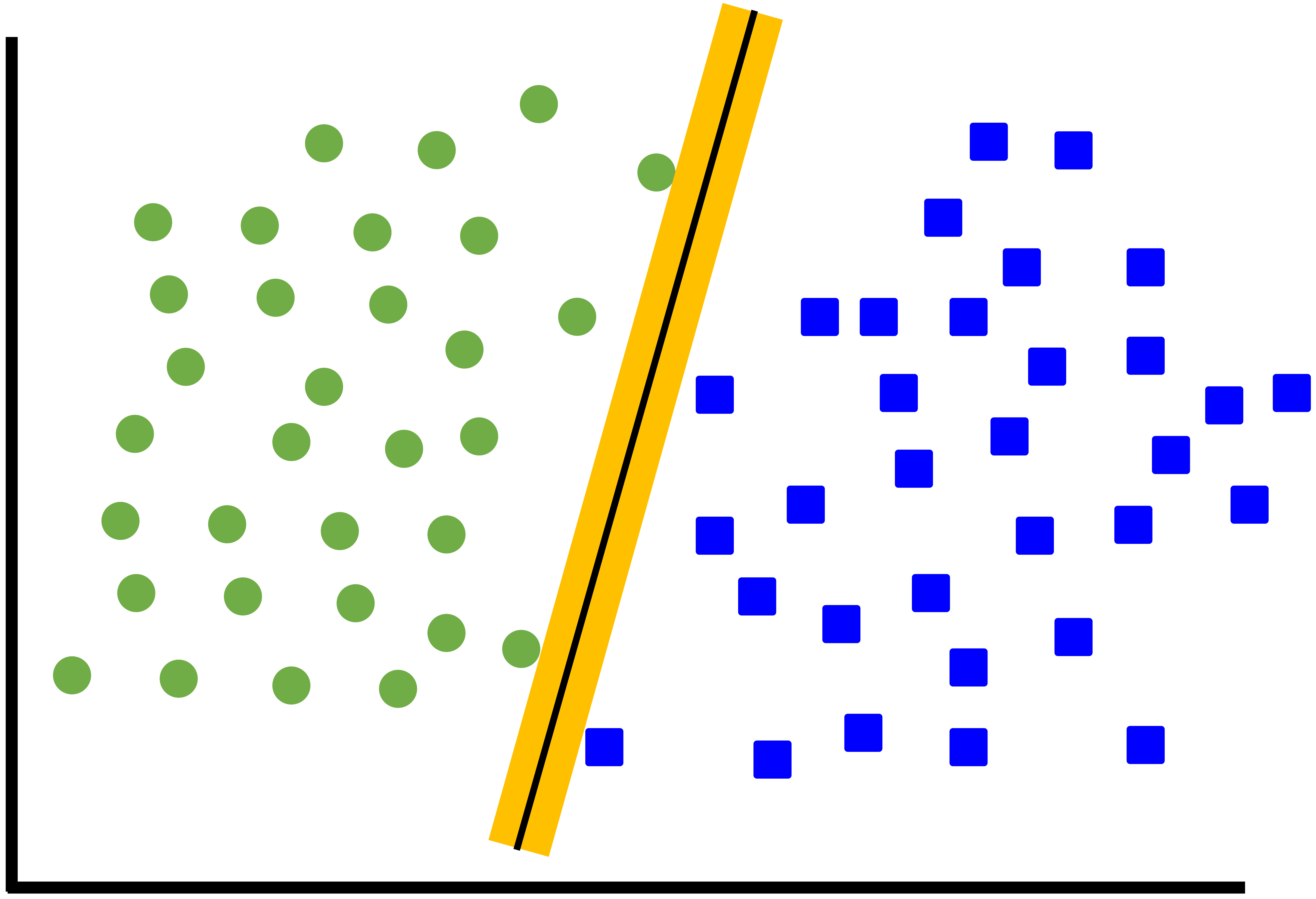


Maximum Margin Separator

A larger margin indicates greater confidence in the classification. Why?!!

.. because there's more separation between classes.

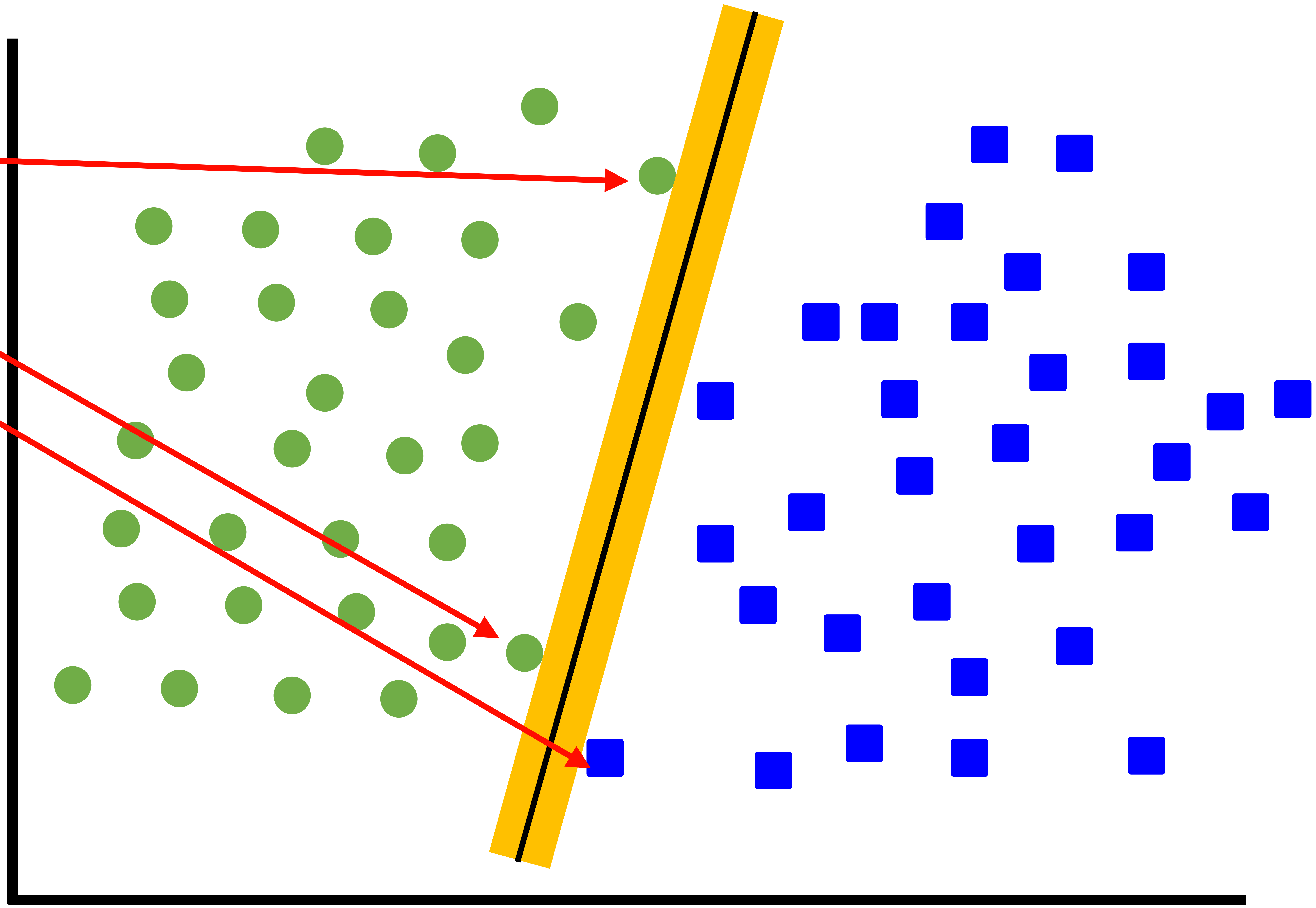
This is the simplest kind of SVM (Called an LSVM)



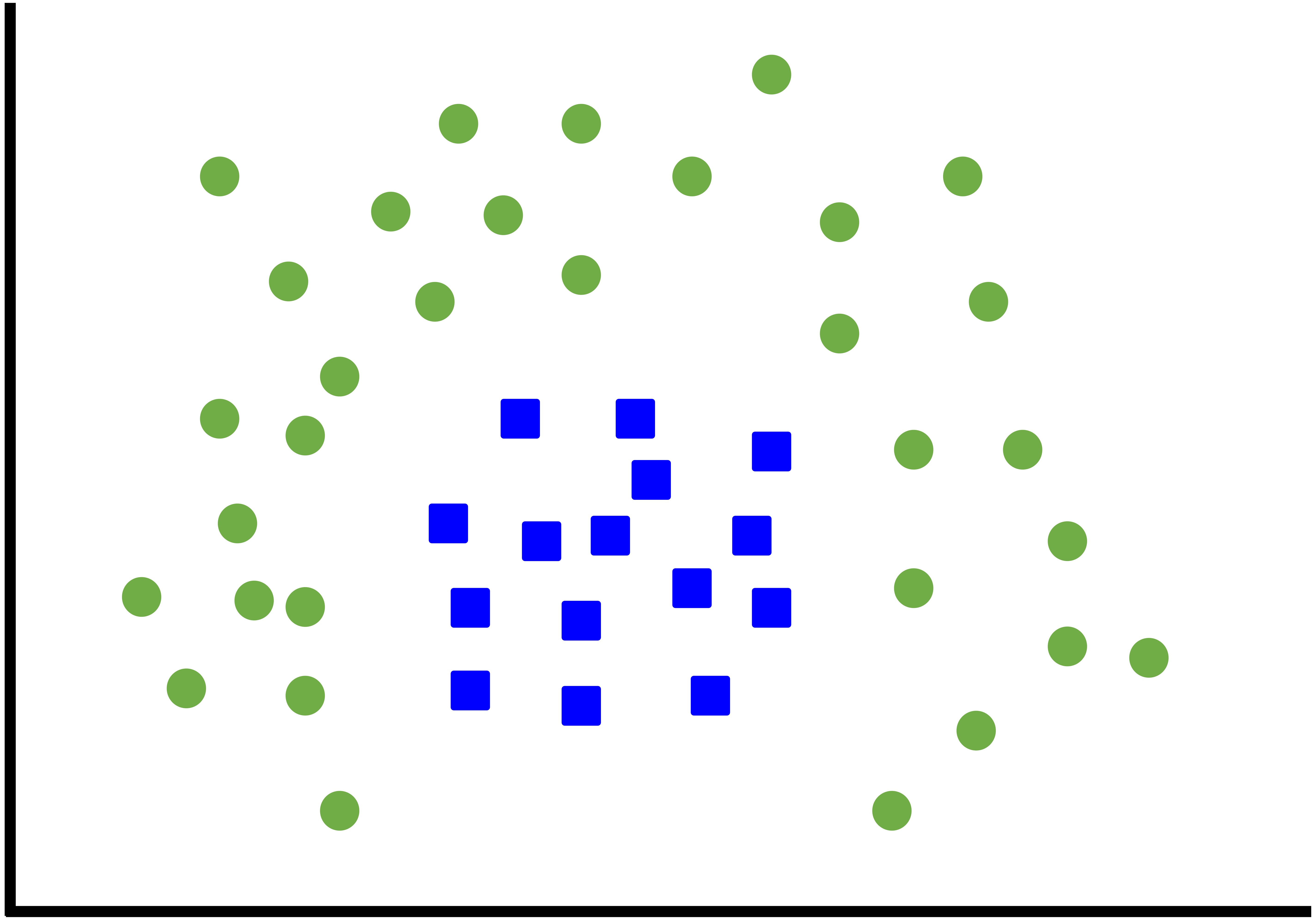
Support Vectors

Support Vectors

are those datapoints that the margin pushes up against

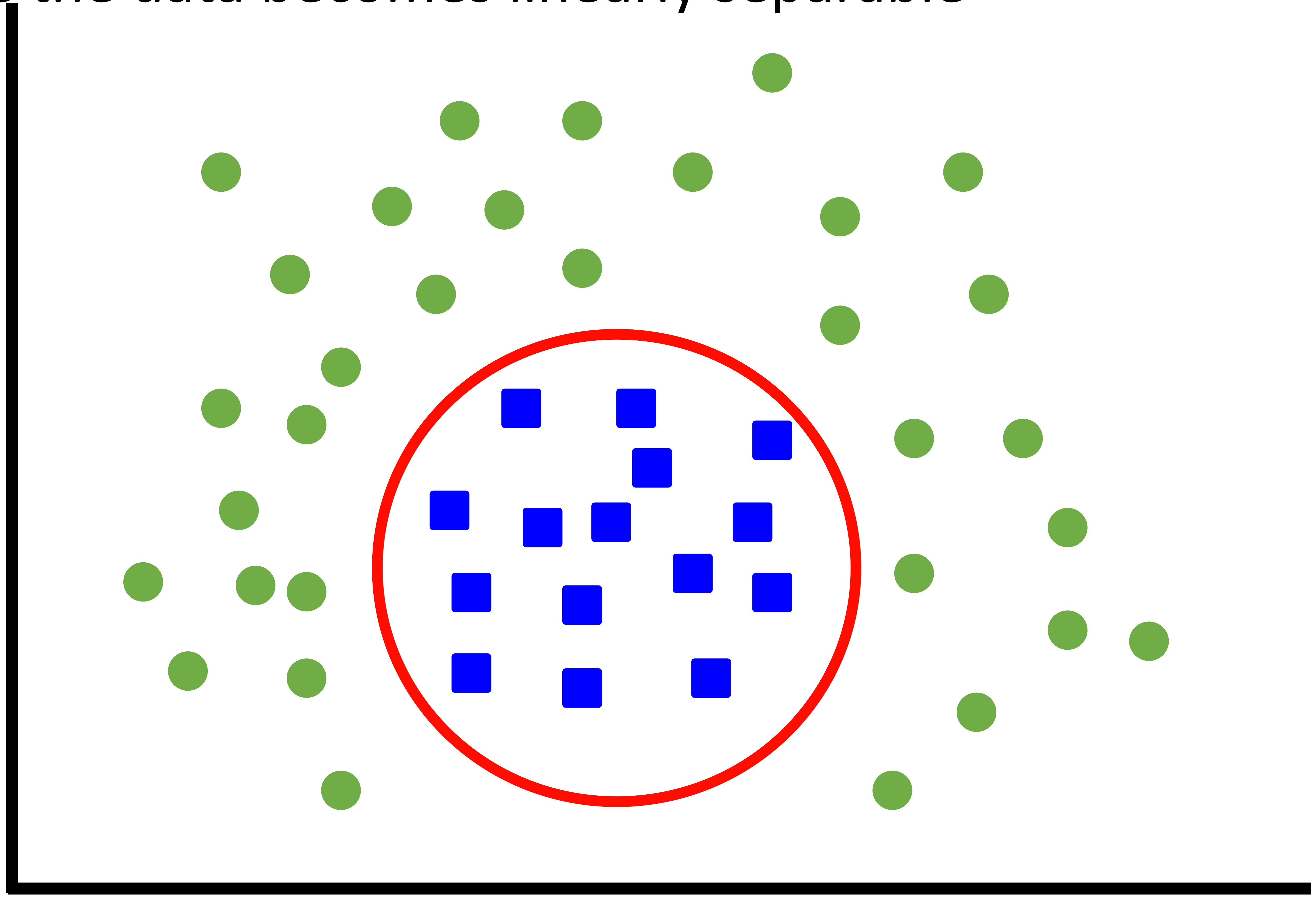


Non-Linear



Non-Linear SVM

- Support Vector Machines (SVMs) can be extended to handle non-linearly separable data by using a technique called the kernel trick.
- The kernel trick allows SVMs to implicitly map the input data into a higher-dimensional feature space where the data becomes linearly separable



Types of Kernels

- SVMs use kernel functions to project data into a higher dimension, improving the ability to find separating boundaries.

- Common kernel functions include:
  - Linear Kernel: $K(x_i, x_j) = x_i^T x_j$
  - Polynomial Kernel: $K(x_i, x_j) = (x_i^T x_j + c)^d$
  - Gaussian RBF (Radial Basis Function) Kernel: $K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}$
  - Sigmoid Kernel: $K(x_i, x_j) = \tanh(\alpha x_i^T x_j + c)$

x_i and x_j are input feature vectors. c, d, γ, α are hyperparameters that can be adjusted to control the behavior of the kernel function.

Thank You