Università degli Studi di Padova

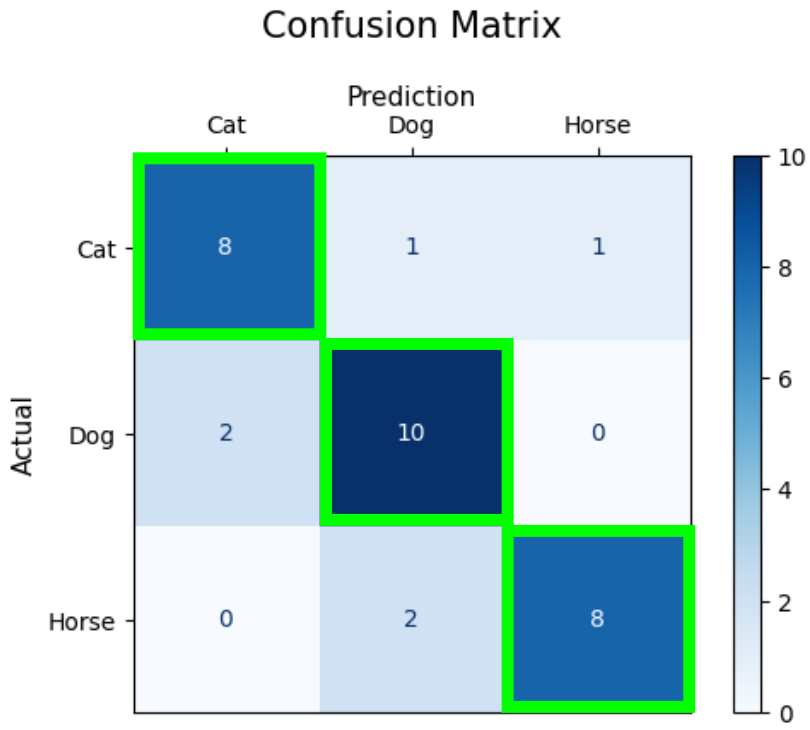
Machine Learning
2024/2025

AMCO
Artificial Intelligence, Machine Learning and Control Research Group

Lecture #13
Logistic Regression

Gian Antonio Susto

Recap: Classification & Classification rate



In classification, the classification rate (or accuracy) is a performance metric used in classification tasks to measure the proportion of correctly classified instances over the total number of instances in a dataset.

$$\text{Classification Rate} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Recap: Binary classification

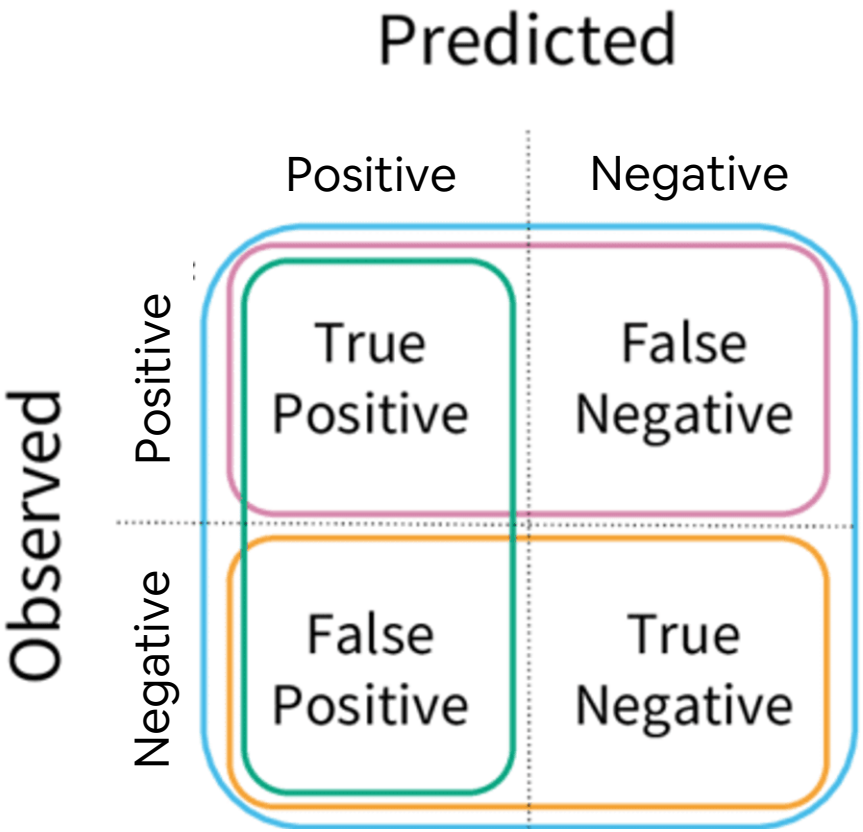
In binary classification, the terms "positive" and "negative" refer to how classes are assigned in the problem. These terms are used to define false positives (FP), false negatives (FN), true positives (TP), and true negatives (TN) when evaluating model performance.

- The positive class is the one that represents the condition or outcome of interest.
- The negative class typically represents the absence of the condition.

Type I error (false positive)

Type II error (false negative)

Recap: Binary classification



$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

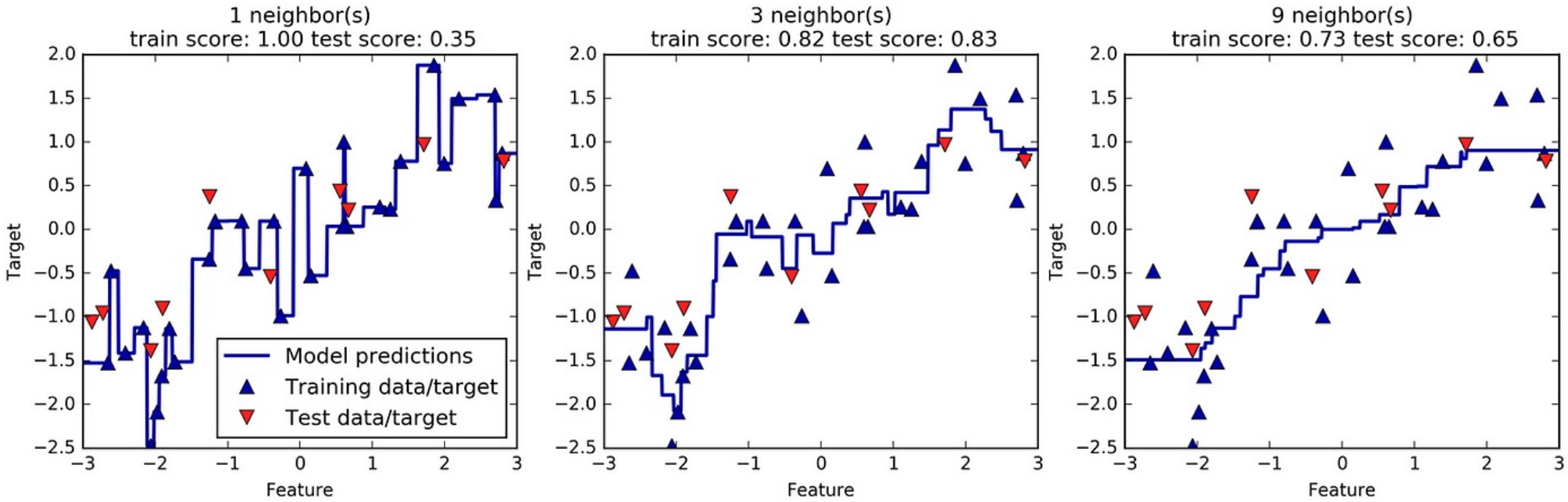
$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

	Spam (Predicted)	Non-Spam (Predicted)	Accuracy
Spam (Actual)	27	6	81.81
Non-Spam (Actual)	10	57	85.07
Overall Accuracy			83.44

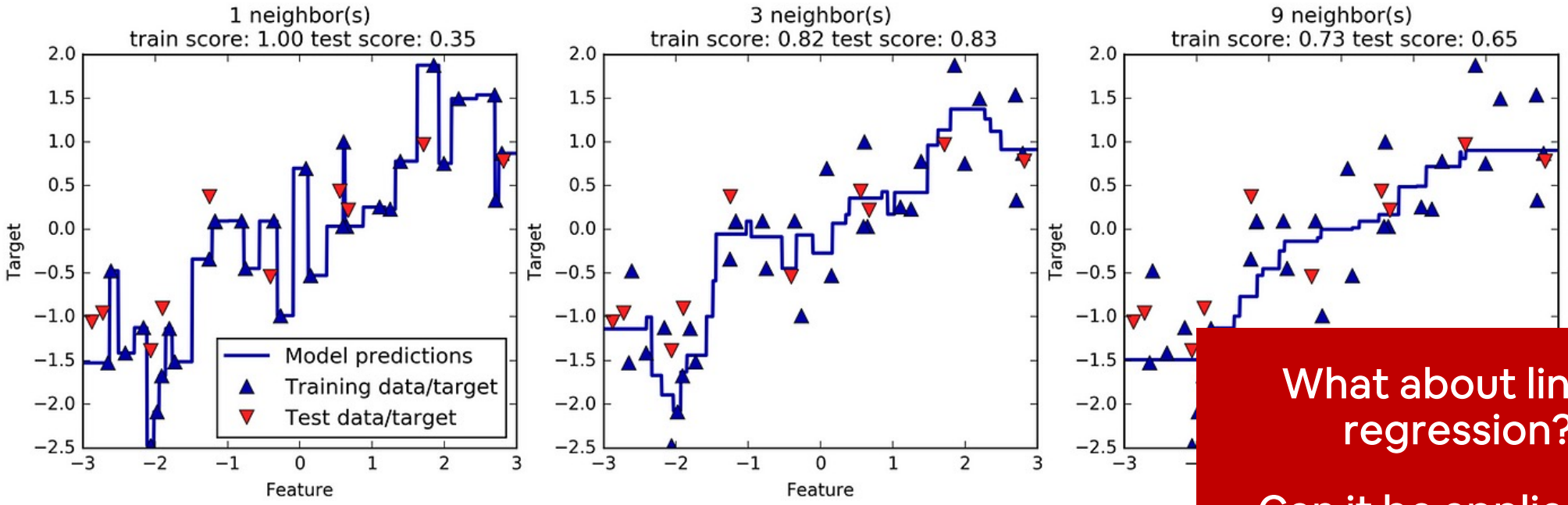
	Spam (Predicted)	Non-Spam (Predicted)	Accuracy
Spam (Actual)	0	10	0.0
Non-Spam (Actual)	0	990	100.0
Overall Accuracy			99

Recap: With small changes algorithms can be adapted from/to classification to/from regression



1. **Compute Choose/Given a value for k** (number of neighbors to consider).
2. **The distance** between the new data point and all training points.
3. **Select the k nearest neighbors** (data points closest to the new point).
4. **Make a prediction**:
    1. **For classification**: Assign the class that appears most frequently among the k neighbors
    2. **For regression**: Take the **average** (or weighted average) of the neighbors' values.

Recap: With small changes algorithms can be adapted from/to classification to/from regression



What about linear regression?

Can it be applied for (binary) classification?

Ideas?

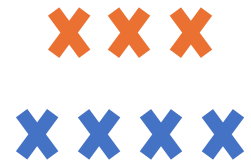
1. Compute Choose/Given a value for k (number of neighbors to co
2. The distance between the new data point and all training points.
3. Select the k nearest neighbors (data points closest to the new poi
4. Make a prediction:
    1. For classification: Assign the class that appears most frequently among the k neighbors
    2. For regression: Take the average (or weighted average) of the neighbors' values.

Let’s try to adapt linear regression to a classification problem

Let’s consider a binary classification problem:

‘Negative Class’ (not spam, absence of a disease…)

‘Positive Class’ (spam, presence of a disease…)



Historical data

Inspired by Andrew Ng and his Stanford course on ‘Machine Learning’

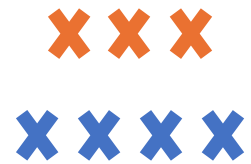
Let’s try to adapt linear regression to a classification problem

$$y \in \{0, 1\}$$

Let’s consider a binary classification problem:

‘Negative Class’ (not spam, absence of a disease…) -> ‘0’

‘Positive Class’ (spam, presence of a disease…) -> ‘1’



Historical data

To work with regression (to fit a line, to compute RMSE, …) we need a quantitative output: we arbitrarily assign values ‘0’ and ‘1’ to the two classes

Inspired by Andrew Ng and his Stanford course on ‘Machine Learning’

Let’s try to adapt linear regression to a classification problem

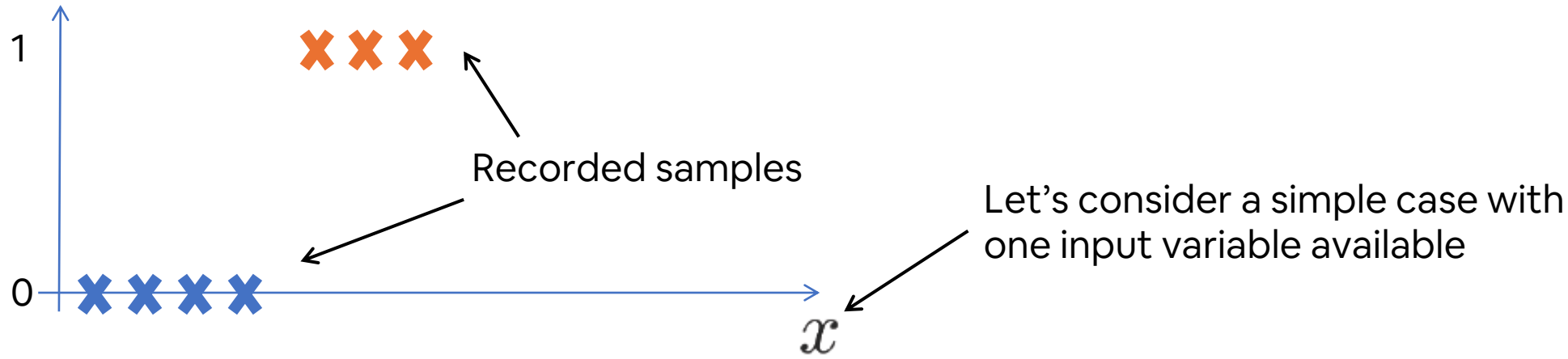
$$y \in \{0, 1\}$$

Let’s consider a binary classification problem:

‘Negative Class’ (not spam, absence of a disease…) -> ‘0’

‘Positive Class’ (spam, presence of a disease…) -> ‘1’

Historical data



Inspired by Andrew Ng and his Stanford course on ‘Machine Learning’

Let’s try to adapt linear regression to a classification problem

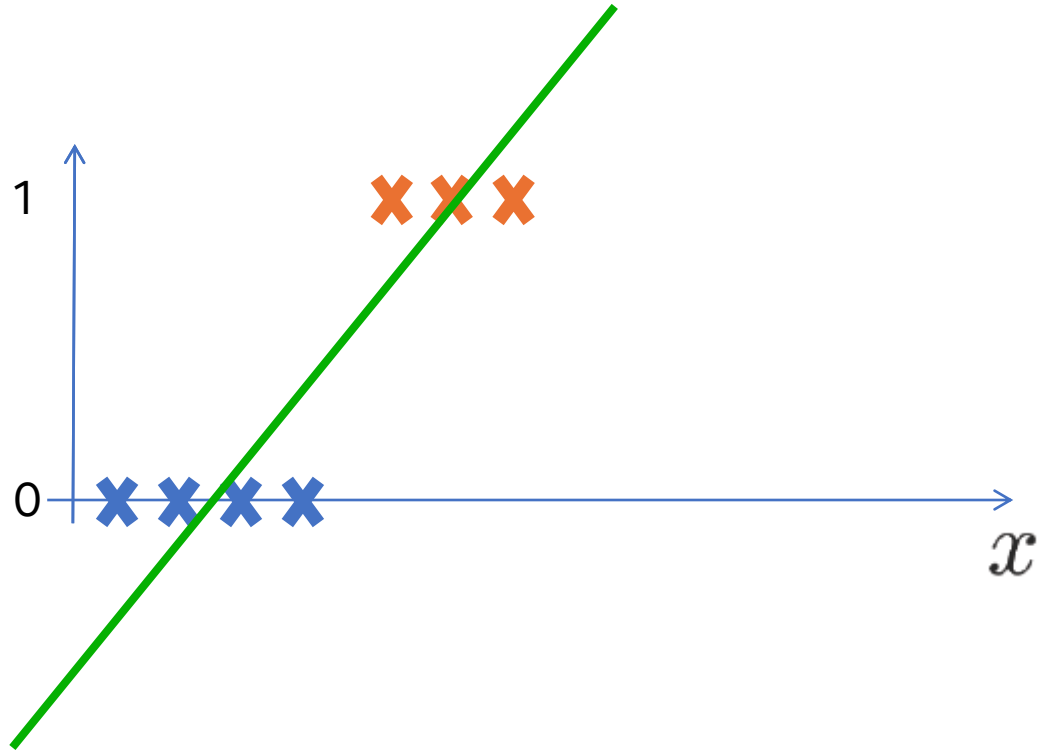
$$y \in \{0, 1\}$$

Let’s consider a binary classification problem:

‘Negative Class’ (not spam, absence of a disease…) ✖✖✖ -> ‘0’

‘Positive Class’ (spam, presence of a disease…) ✖✖✖✖ -> ‘1’

Historical data



We fit a line using OLS: how can we use it for classification? Ideas?

Inspired by Andrew Ng and his Stanford course on ‘Machine Learning’

Let’s try to adapt linear regression to a classification problem

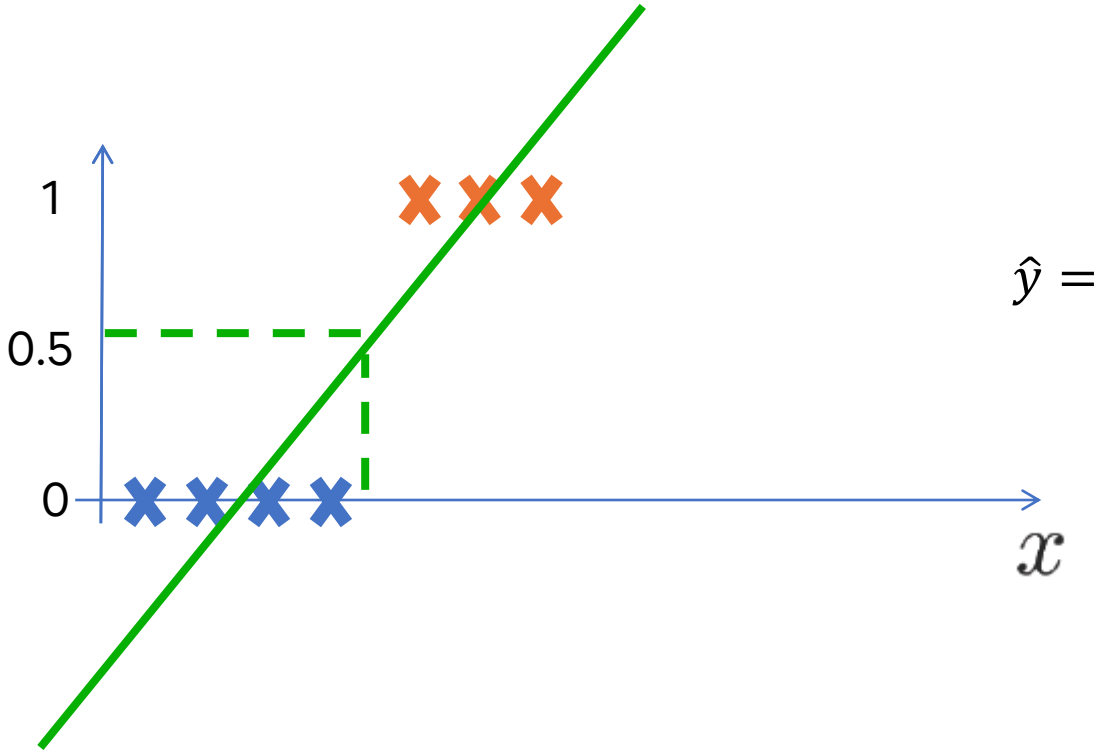
$$y \in \{0, 1\}$$

Let’s consider a binary classification problem:

‘Negative Class’ (not spam, absence of a disease…) ✖✖✖ -> ‘0’

‘Positive Class’ (spam, presence of a disease…) ✖✖✖✖ -> ‘1’

Historical data



$$\hat{y} = \begin{cases} negative & if\ \beta x < 0.5 \\ positive & if\ \beta x \geq 0.5 \end{cases}$$

Inspired by Andrew Ng and his Stanford course on ‘Machine Learning’

Let's try to adapt linear regression to a classification problem

$$y \in \{0, 1\}$$

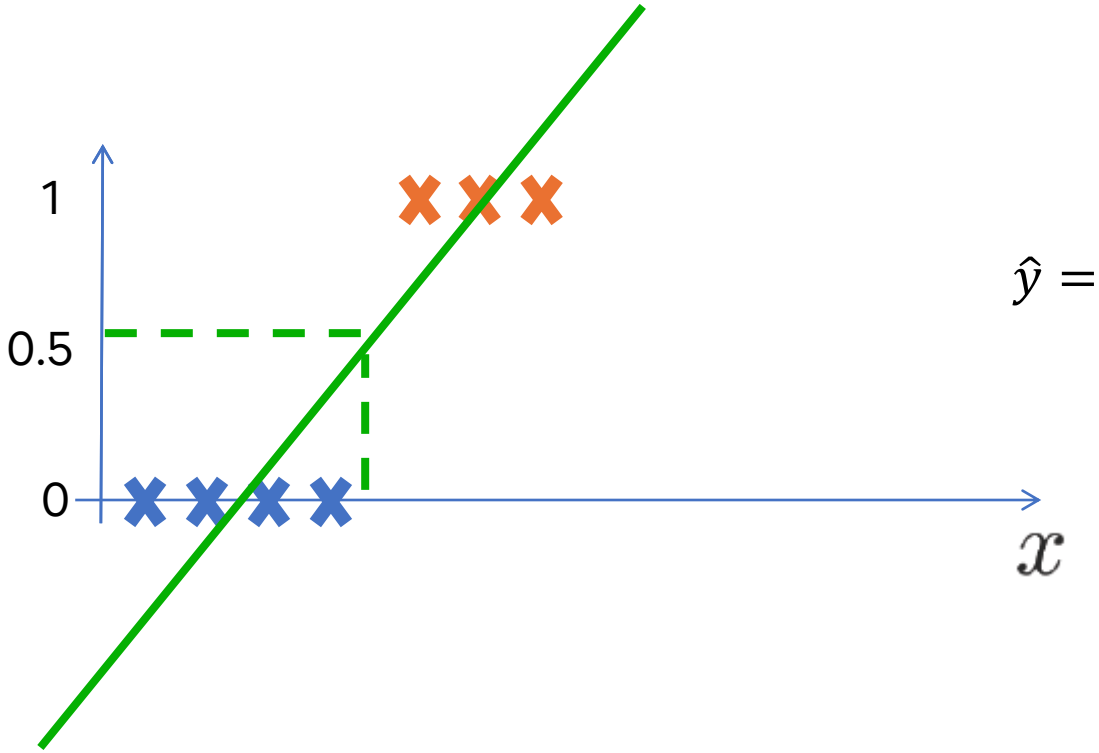
Let's consider a binary classification problem:

'Negative Class' (not spam, absence of a disease...) ✖✖✖ -> '0'

'Positive Class' (spam, presence of a disease...) ✖✖✖✖ -> '1'

Historical data

We call 0.5 the 'decision boundary': the threshold that makes our classification decision change



$$\hat{y} = \begin{cases} negative & if\ \beta x < 0.5 \\ positive & if\ \beta x \geq 0.5 \end{cases}$$

Inspired by Andrew Ng and his Stanford course on 'Machine Learning'

Let's try to adapt linear regression to a classification problem

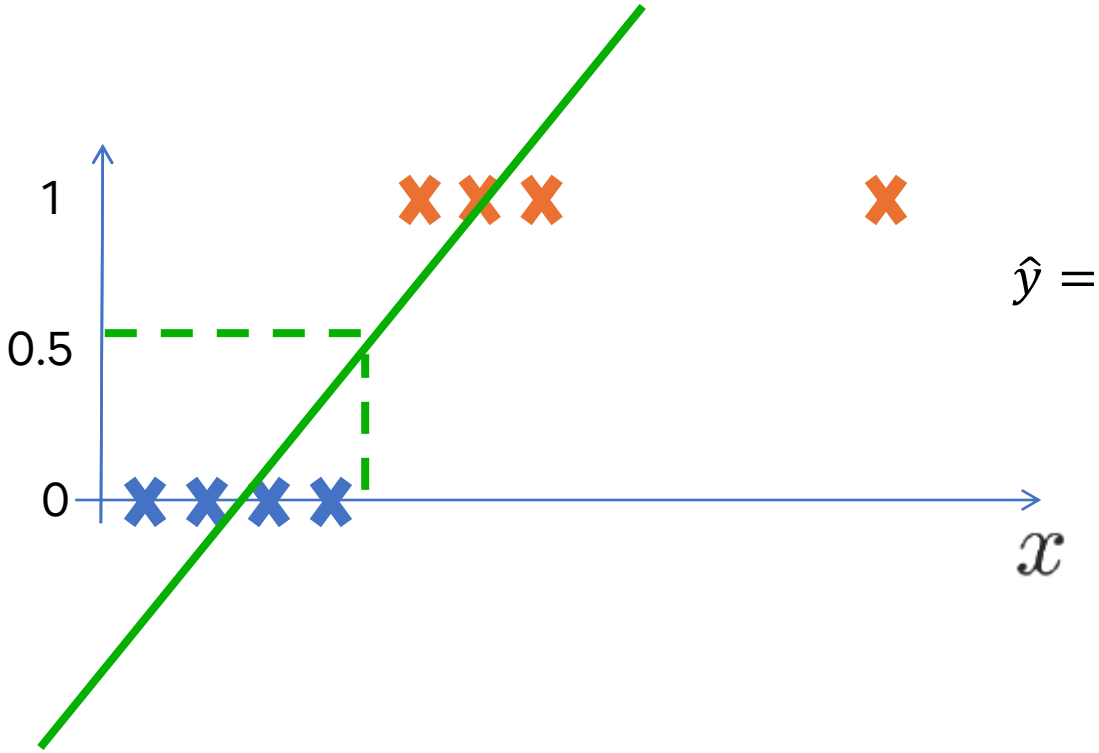
$$y \in \{0, 1\}$$

Let's consider a binary classification problem:

'Negative Class' (not spam, absence of a disease...) ✖✖✖ -> '0'

'Positive Class' (spam, presence of a disease...) ✖✖✖✖ -> '1'

Historical data



$$\hat{y} = \begin{cases} negative & if\ \beta x < 0.5 \\ positive & if\ \beta x \geq 0.5 \end{cases}$$

What if we have data far away from the decision boundary domain? This typically happens in classification

Inspired by Andrew Ng and his Stanford course on 'Machine Learning'

Let's try to adapt linear regression to a classification problem

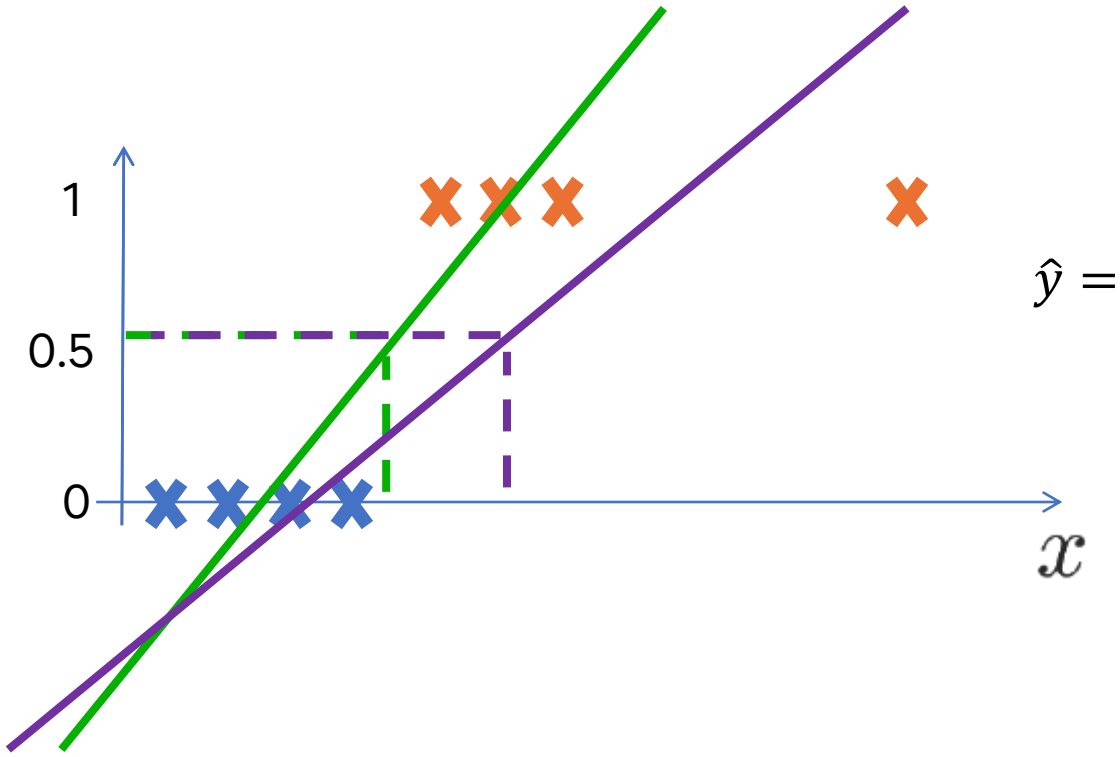
$$y \in \{0, 1\}$$

Let's consider a binary classification problem:

'Negative Class' (not spam, absence of a disease...) ✖✖✖ -> '0'

'Positive Class' (spam, presence of a disease...) ✖✖✖✖ -> '1'

Historical data



$$\hat{y} = \begin{cases} neg \\ pos \end{cases}$$

Fitting the data with Least Square is not an appropriate methodology for 'dividing' two classes.

Moreover, the function can obtain values outside [0,1]

We need for a more appropriate function to describe 2 different levels: ideas?

Let's try to adapt linear regression to a classification problem

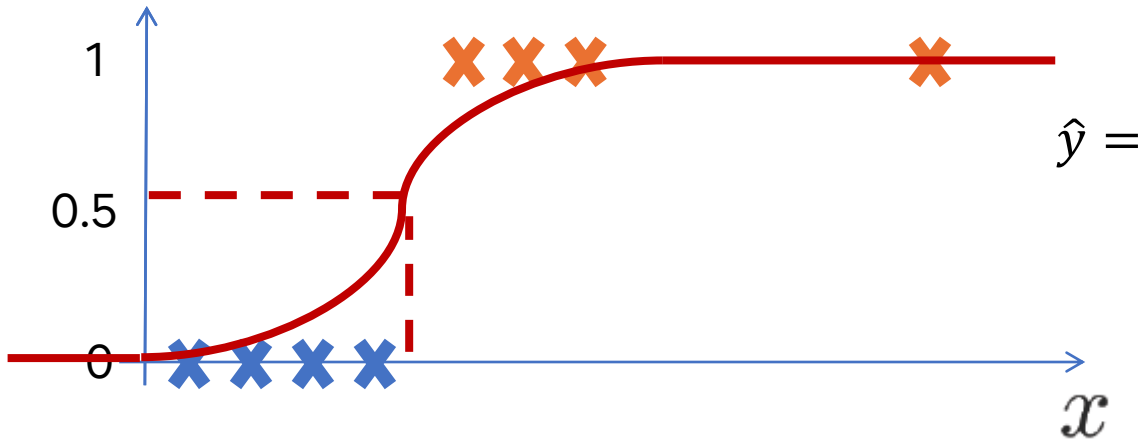
$$y \in \{0, 1\}$$

Let's consider a binary classification problem:

'Negative Class' (not spam, absence of a disease...) ✖✖✖ -> '0'

'Positive Class' (spam, presence of a disease...) ✖✖✖✖ -> '1'

Historical data



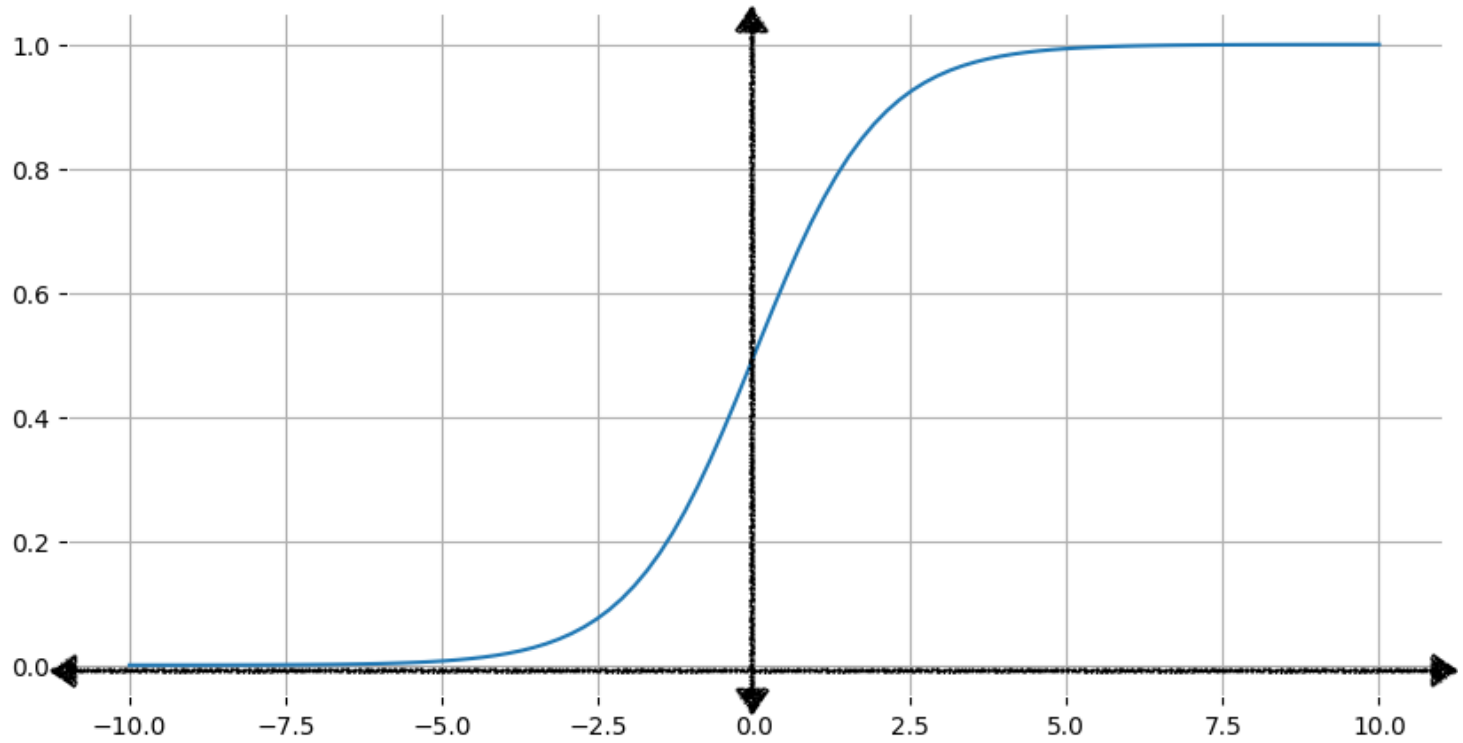
$$\hat{y} = \begin{cases} negative & if\ \sigma(x) < 0.5 \\ positive & if\ \sigma(x) \geq 0.5 \end{cases}$$

A logistic (sigmoid) function will be a better idea! It naturally defines 'two' levels of output

Inspired by Andrew Ng and his Stanford course on 'Machine Learning'

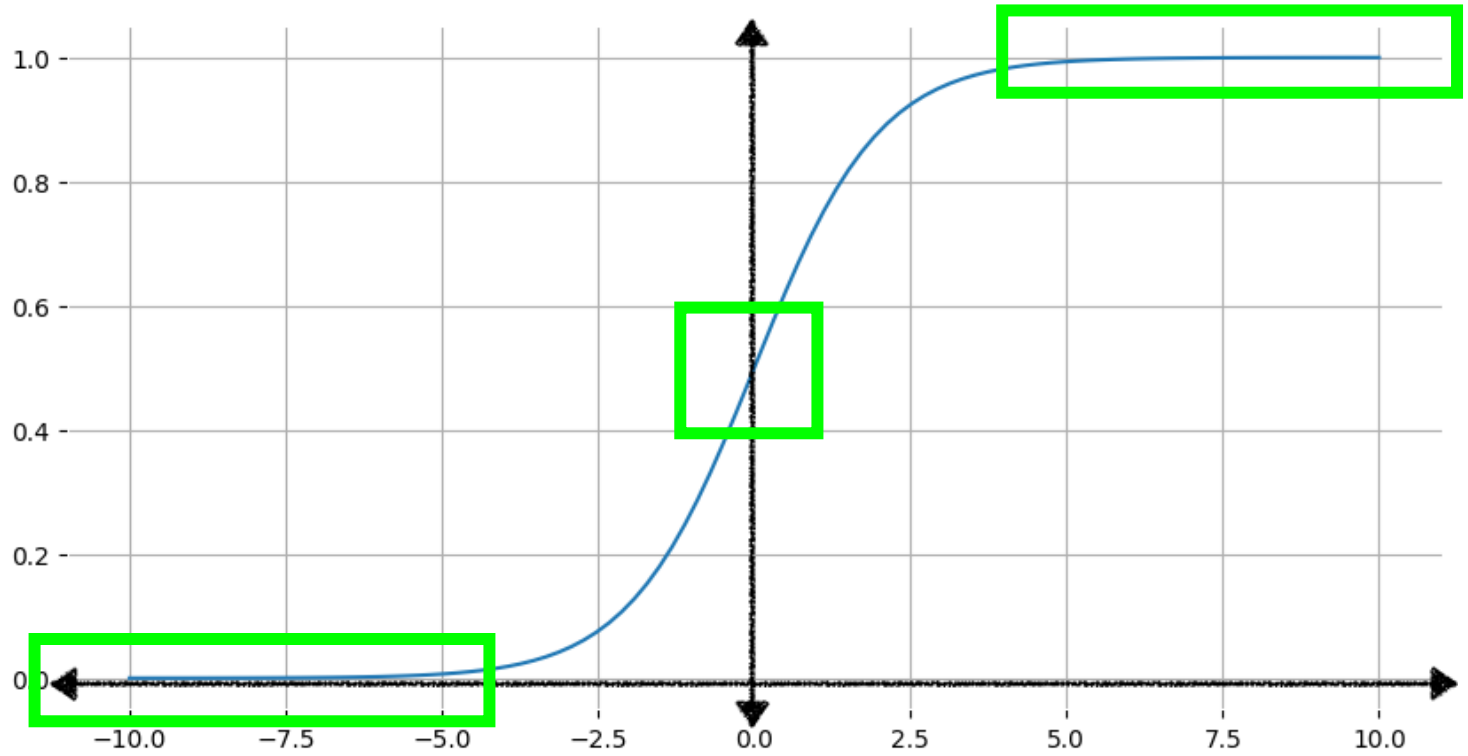
The logistic (sigmoid) function

$$\sigma(x^\top \beta) = \frac{1}{1 + e^{-x^\top \beta}}$$



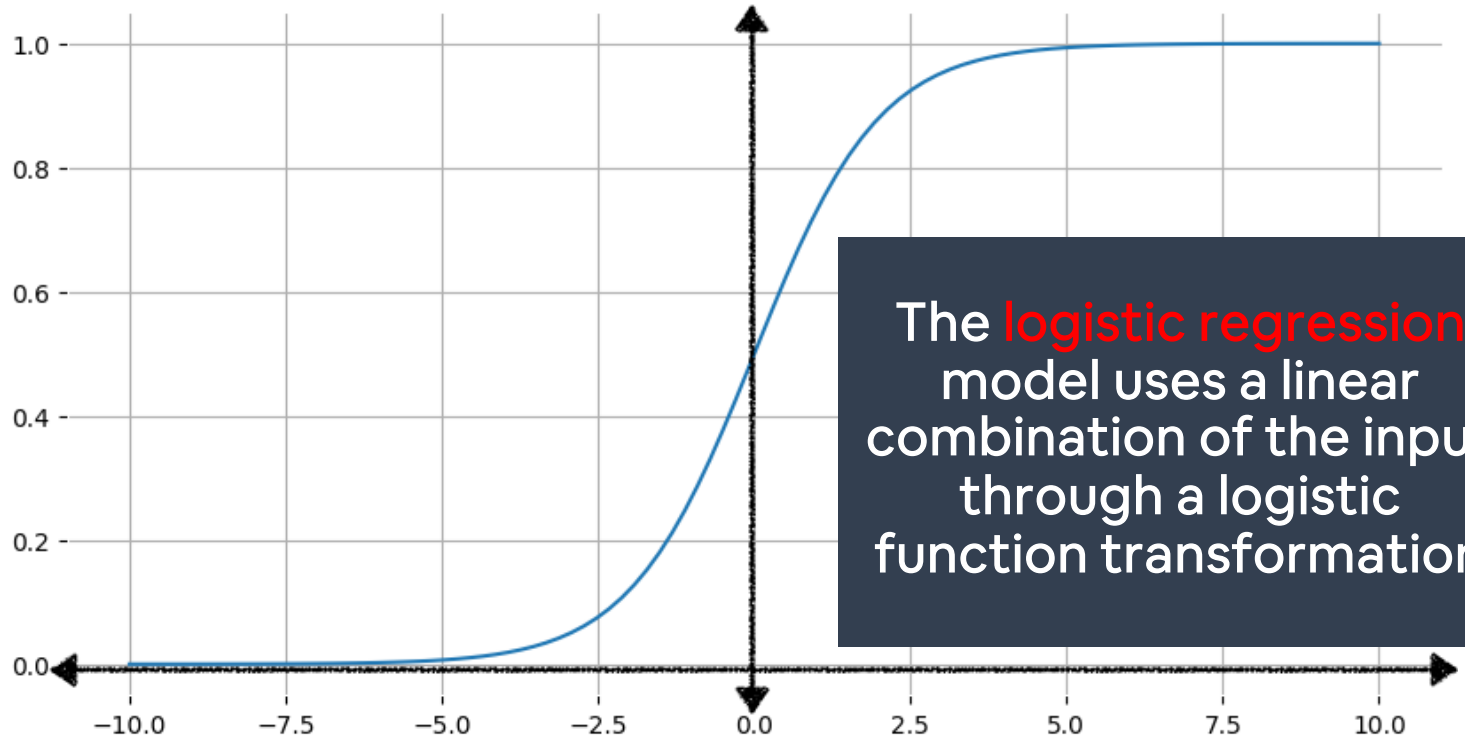
The logistic (sigmoid) function

$$\sigma(x^\top \beta) = \frac{1}{1 + e^{-x^\top \beta}}$$



The logistic (sigmoid) function

$$\sigma(x^\top \beta) = \frac{1}{1 + e^{-x^\top \beta}}$$



The logistic regression model uses a linear combination of the input through a logistic function transformation

The logistic (sigmoid) function

$$P(y = 1 \mid x) = \sigma(x^\top \beta) = \frac{1}{1 + e^{-x^\top \beta}}$$

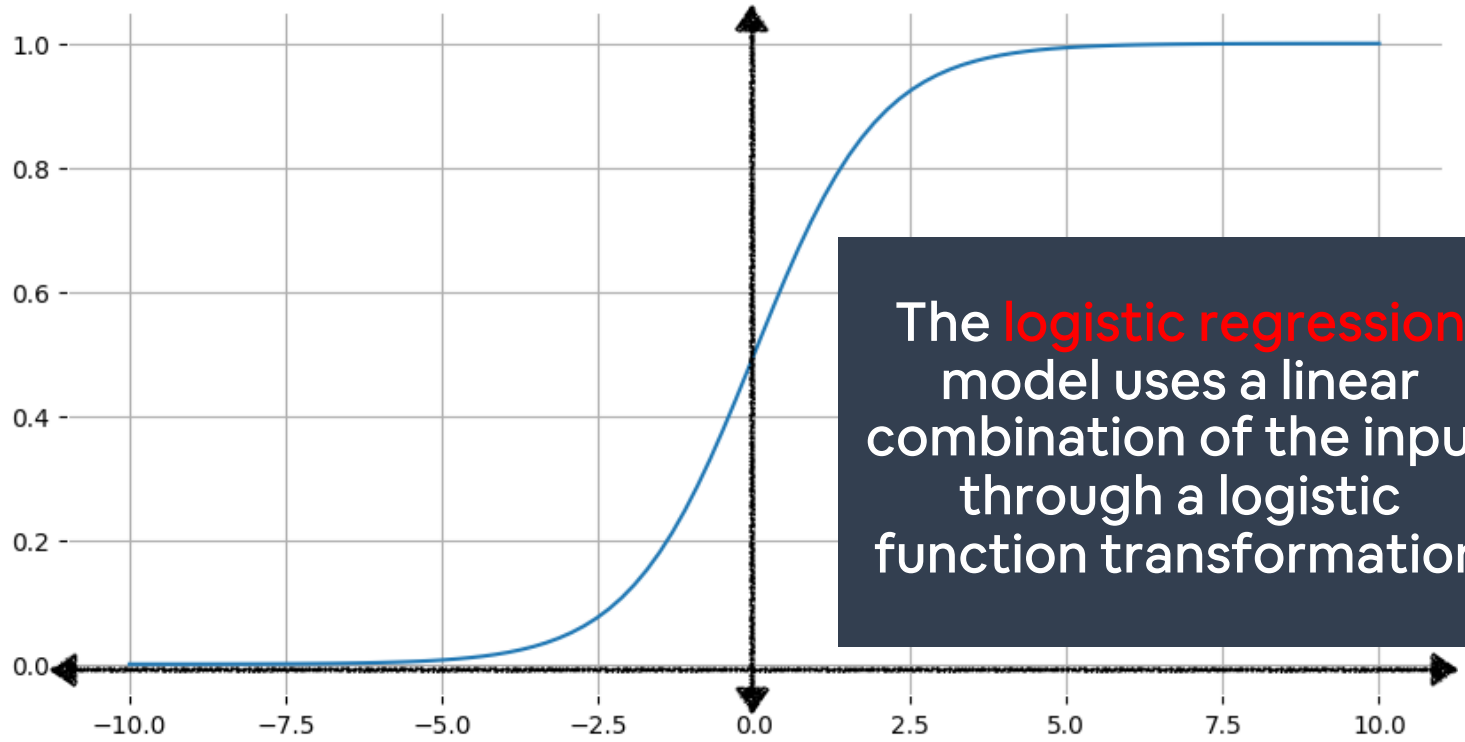
$$P(y = 0 \mid x) = 1 - \sigma(x^\top \beta)$$

The logistic function has a probabilistic interpretation:

$$1 \geq P(y = 1 \mid x) \geq 0$$

$$1 \geq P(y = 0 \mid x) \geq 0$$

$$P(y = 1 \mid x) + P(y = 0 \mid x) = 1$$



The logistic regression model uses a linear combination of the input through a logistic function transformation

The logistic (sigmoid) function

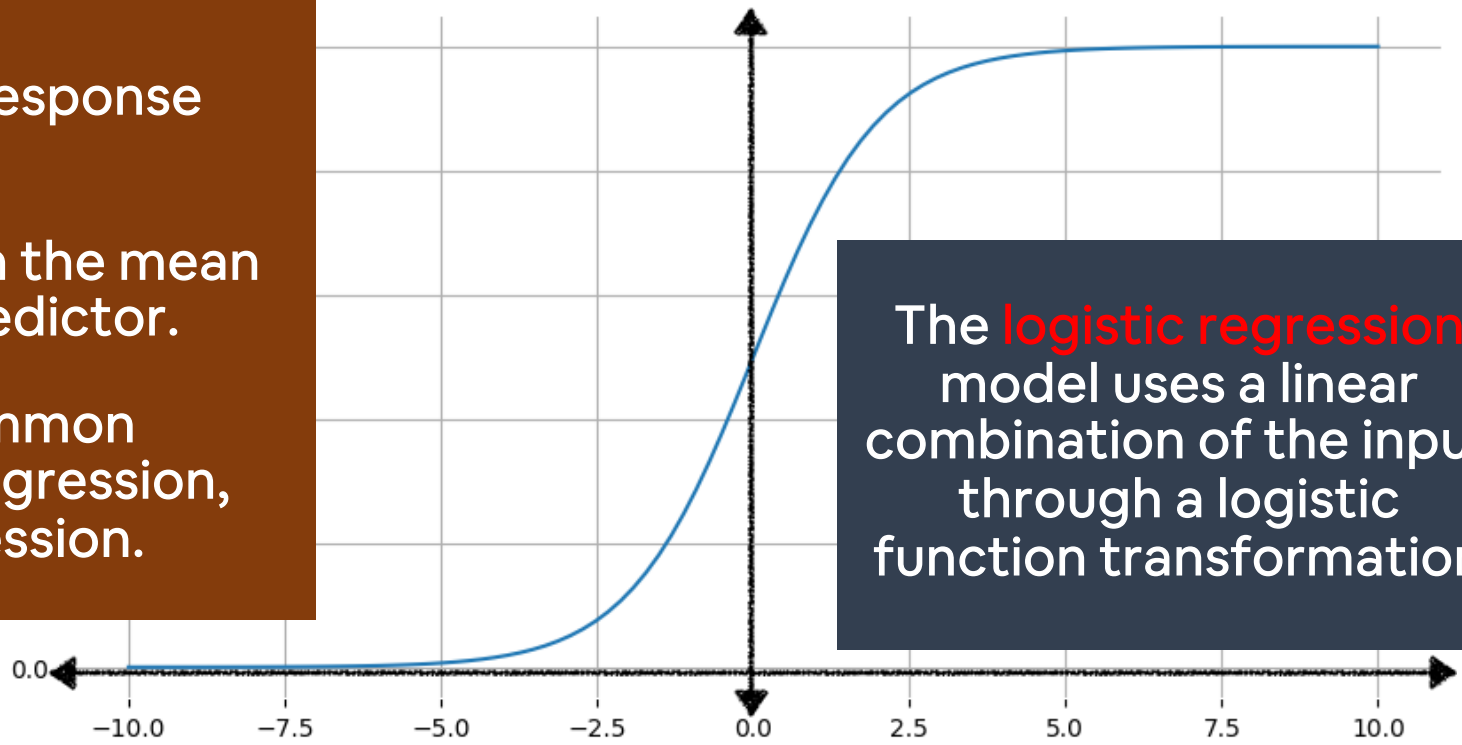
$$P(y = 1 \mid x) = \sigma(x^\top \beta) = \frac{1}{1 + e^{-x^\top \beta}}$$

$$P(y = 0 \mid x) = 1 - \sigma(x^\top \beta)$$

A Generalized Linear Model (GLM) is a flexible generalization of OLS that allows for:

- Non-normal distributions of the response variable (e.g., binomial, Poisson);
- A nonlinear link function between the mean of the response and the linear predictor.

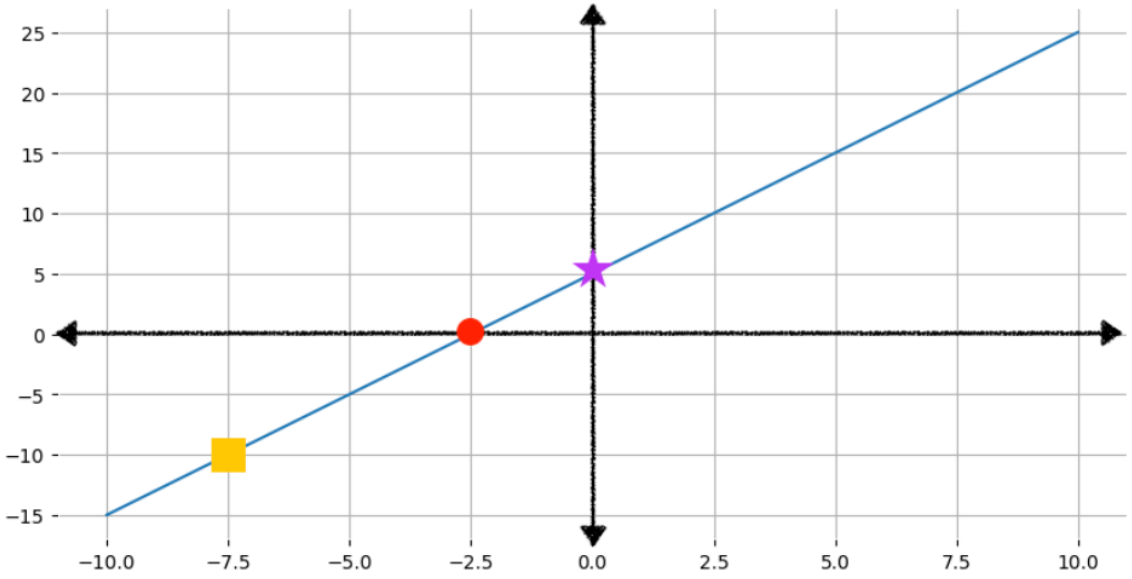
It's a unified framework for many common regression models, including linear regression, logistic regression, and Poisson regression.



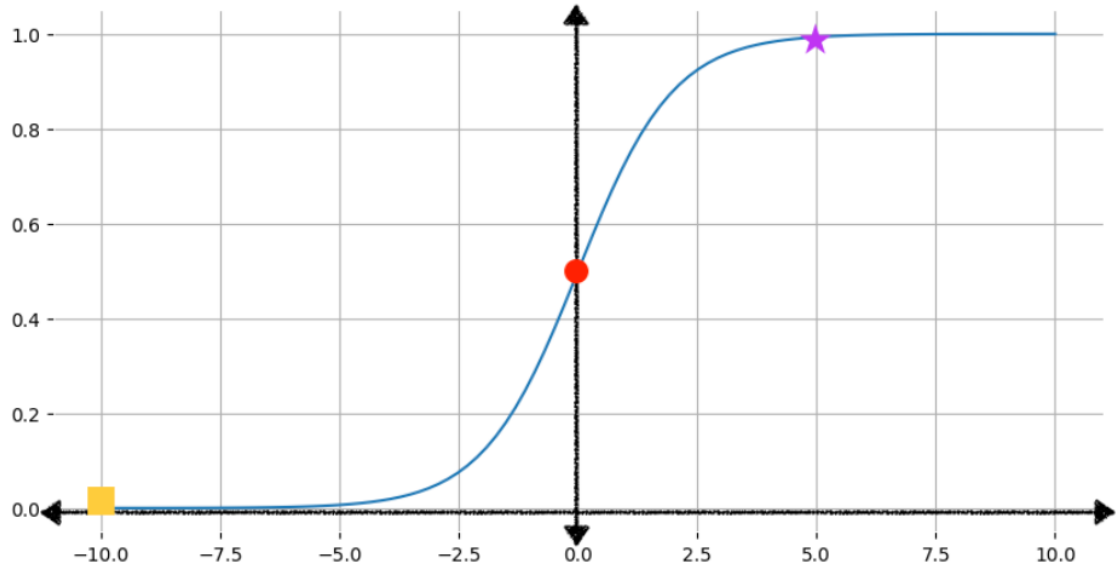
The logistic regression model uses a linear combination of the input through a logistic function transformation

The logistic (sigmoid) function: an example

$z = 2x + 5$



$y' = 1 / (1 + e^{-z})$



z = 5

z = 0

z = -10

y = 0.99

y = 0.5

y = 0

Solving Logistic Regression

$$J(\vec{w}, b) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2$$

No closed-form solution: we need to resort to gradient descent

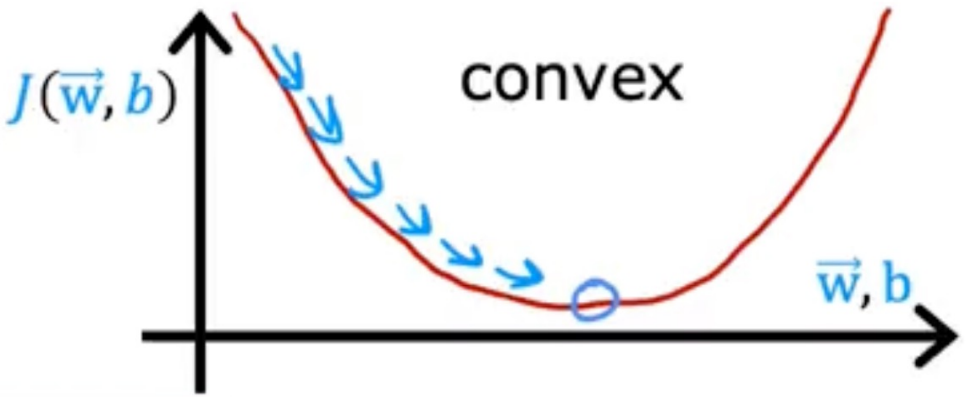
Solving Logistic Regression

$$J(\vec{w}, b) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right) - y^{(i)}\right)^2$$

No closed-form solution: we need to resort to gradient descent

linear regression

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$



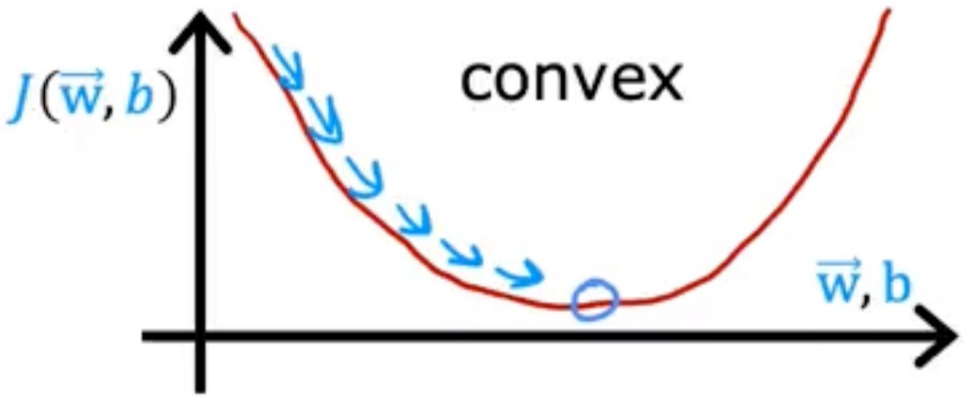
Solving Logistic Regression

$$J(\vec{w}, b) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2$$

No closed-form solution: we need to resort to gradient descent

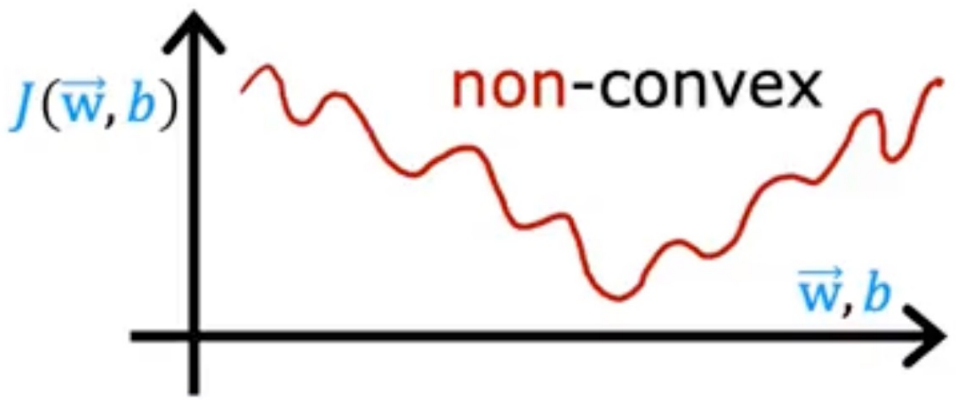
linear regression

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$



logistic regression

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

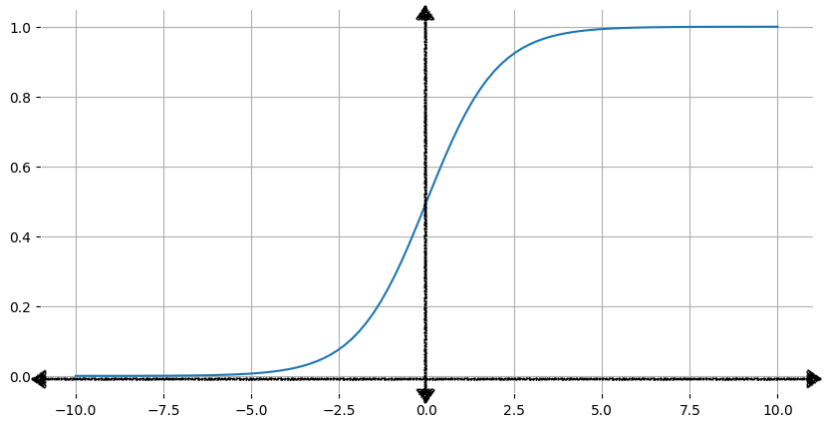


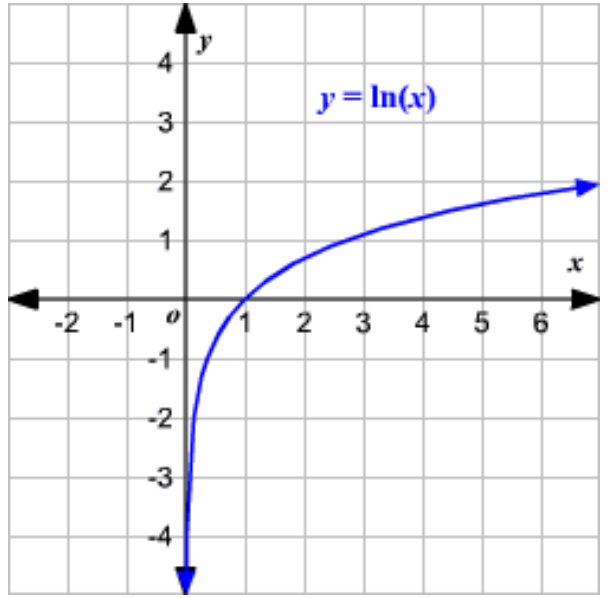
Logistic Regression Loss Function: log-likelihood

$$\mathcal{L}(\beta) = -\sum_{i=1}^{n} \left[y_i \log \sigma(x_i^\top \beta) + (1 - y_i) \log(1 - \sigma(x_i^\top \beta)) \right]$$

$$\sigma(x^\top \beta) = \frac{1}{1 + e^{-x^\top \beta}}$$

	$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$		
$y = 1$		





We would like something that captures a classification metric (not a regression one):

$$\text{Classification Rate} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Logistic Regression Loss Function: log-likelihood

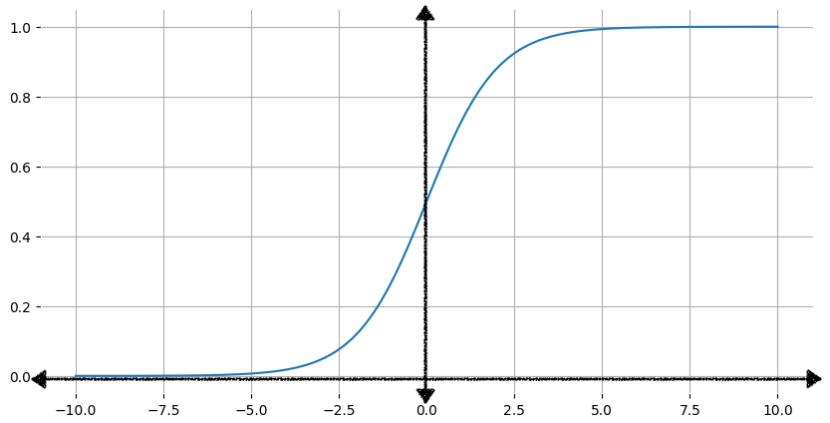
$$\mathcal{L}(\beta) = -\sum_{i=1}^{n} \left[y_i \log \sigma(x_i^\top \beta) + (1 - y_i) \log(1 - \sigma(x_i^\top \beta)) \right]$$

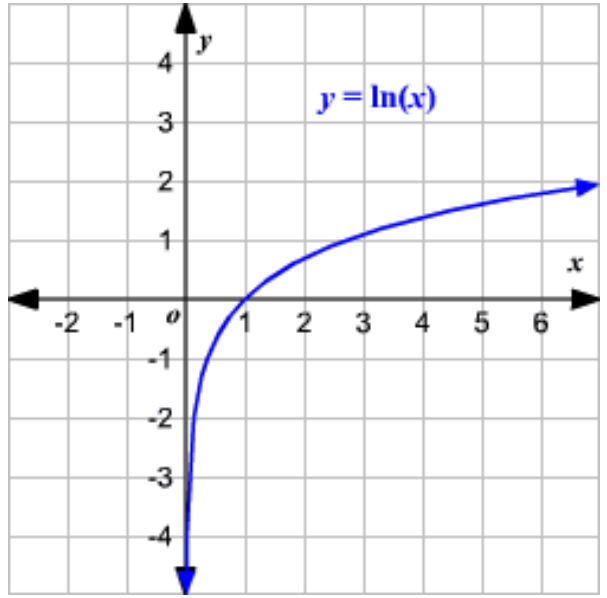
y_i: 0 — $\log \sigma(x_i^\top \beta)$: 1 — $(1 - y_i)$: 0

= 0

$$\sigma(x^\top \beta) = \frac{1}{1 + e^{-x^\top \beta}}$$

	$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$		
$y = 1$		





We would like something that captures a classification metric (not a regression one):

$$\text{Classification Rate} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Logistic Regression Loss Function: log-likelihood

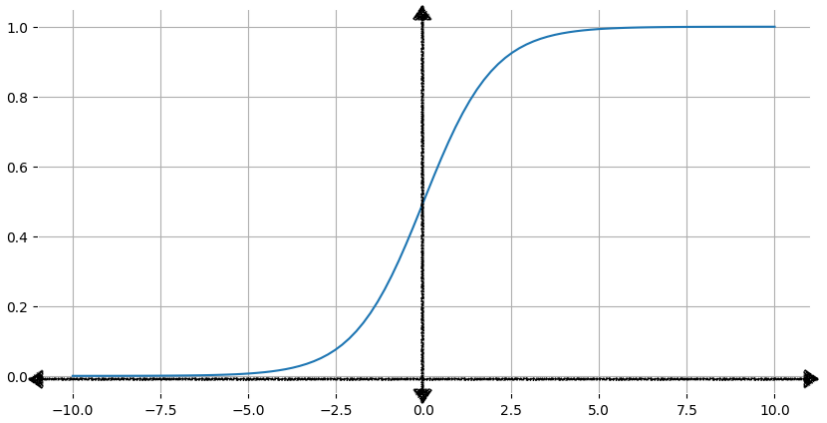
$$\mathcal{L}(\beta) = -\sum_{i=1}^{n} \left[y_i \log \sigma(x_i^\top \beta) + (1 - y_i) \log(1 - \sigma(x_i^\top \beta)) \right]$$

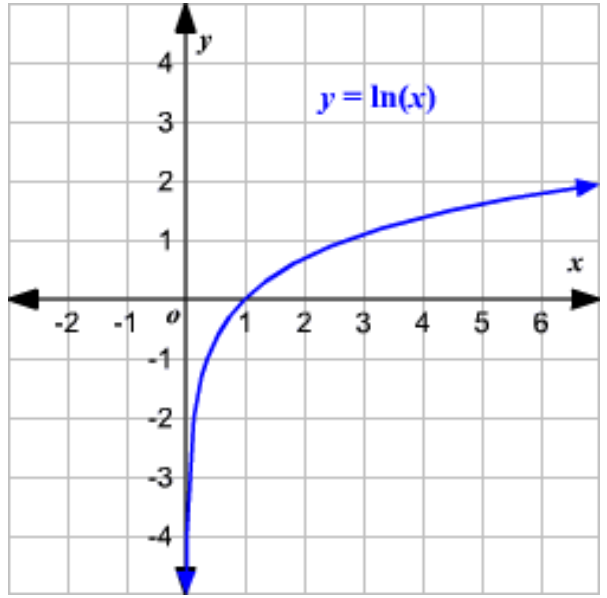
1 0 0

= 0

$$\sigma(x^\top \beta) = \frac{1}{1 + e^{-x^\top \beta}}$$

	$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$		
$y = 1$		





We would like something that captures a classification metric (not a regression one):

$$\text{Classification Rate} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Logistic Regression Loss Function: log-likelihood

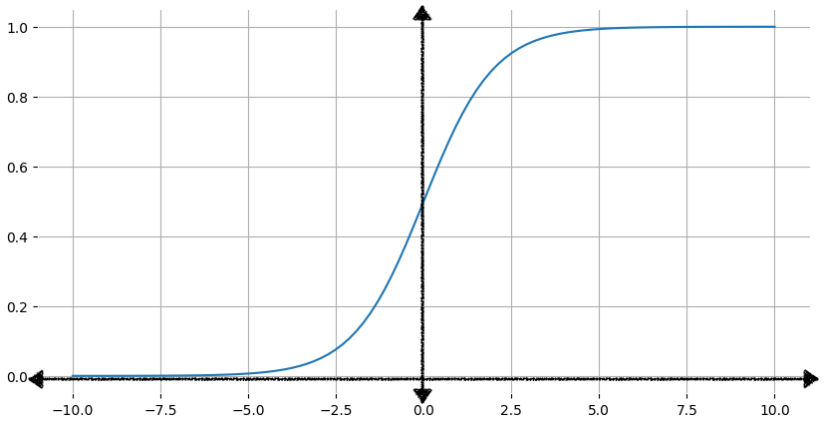
$$\mathcal{L}(\beta) = -\sum_{i=1}^{n} \left[y_i \log \sigma(x_i^\top \beta) + (1 - y_i) \log(1 - \sigma(x_i^\top \beta)) \right]$$

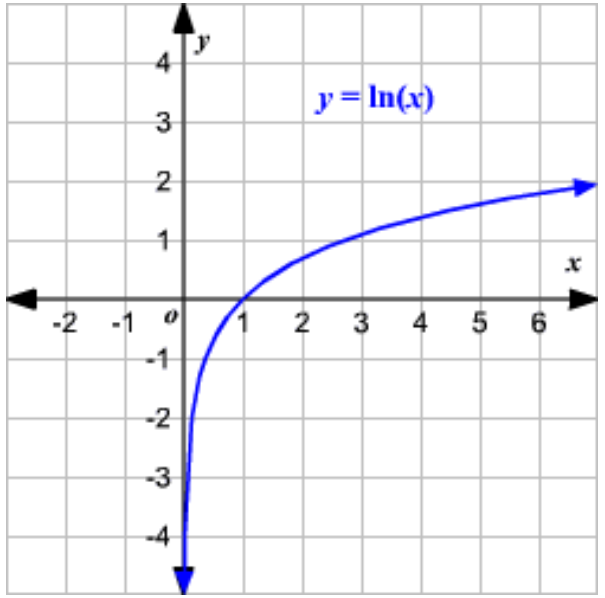
0 — 1 — - ‘high’

= ‘high’

$$\sigma(x^\top \beta) = \frac{1}{1 + e^{-x^\top \beta}}$$

	$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$		
$y = 1$		





We would like something that captures a classification metric (not a regression one):

$$\text{Classification Rate} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Logistic Regression Loss Function: log-likelihood

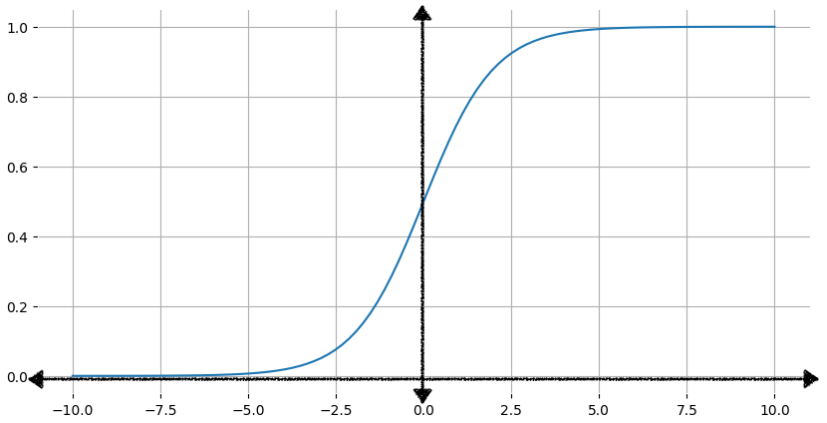
$$\mathcal{L}(\beta) = -\sum_{i=1}^{n} \left[y_i \log \sigma(x_i^\top \beta) + (1 - y_i) \log(1 - \sigma(x_i^\top \beta)) \right]$$

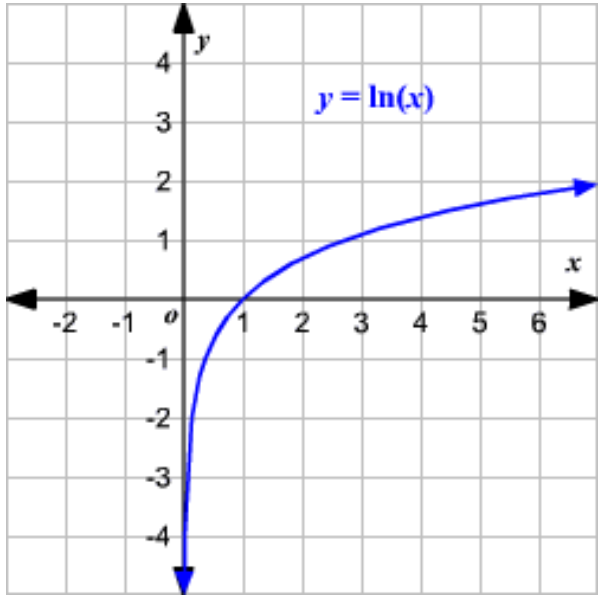
1 -'high' 0

= 'high'

$$\sigma(x^\top \beta) = \frac{1}{1 + e^{-x^\top \beta}}$$

	$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$		
$y = 1$		





We would like something that captures a classification metric (not a regression one):

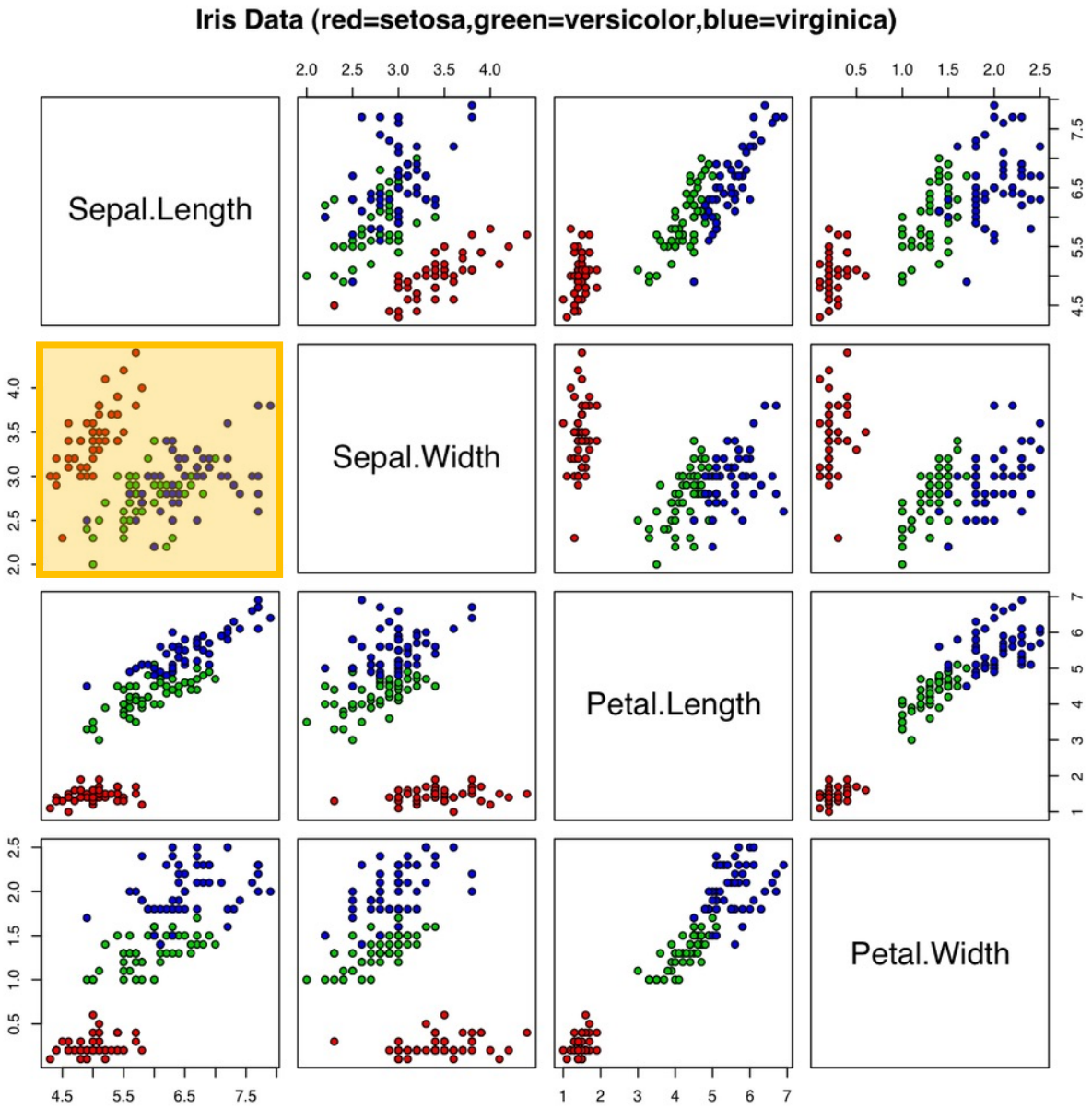
$$\text{Classification Rate} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Logistic Regression Loss Function: log-likelihood

$$\mathcal{L}(\beta) = -\sum_{i=1}^{n} \left[y_i \log \sigma(x_i^\top \beta) + (1 - y_i) \log(1 - \sigma(x_i^\top \beta)) \right]$$

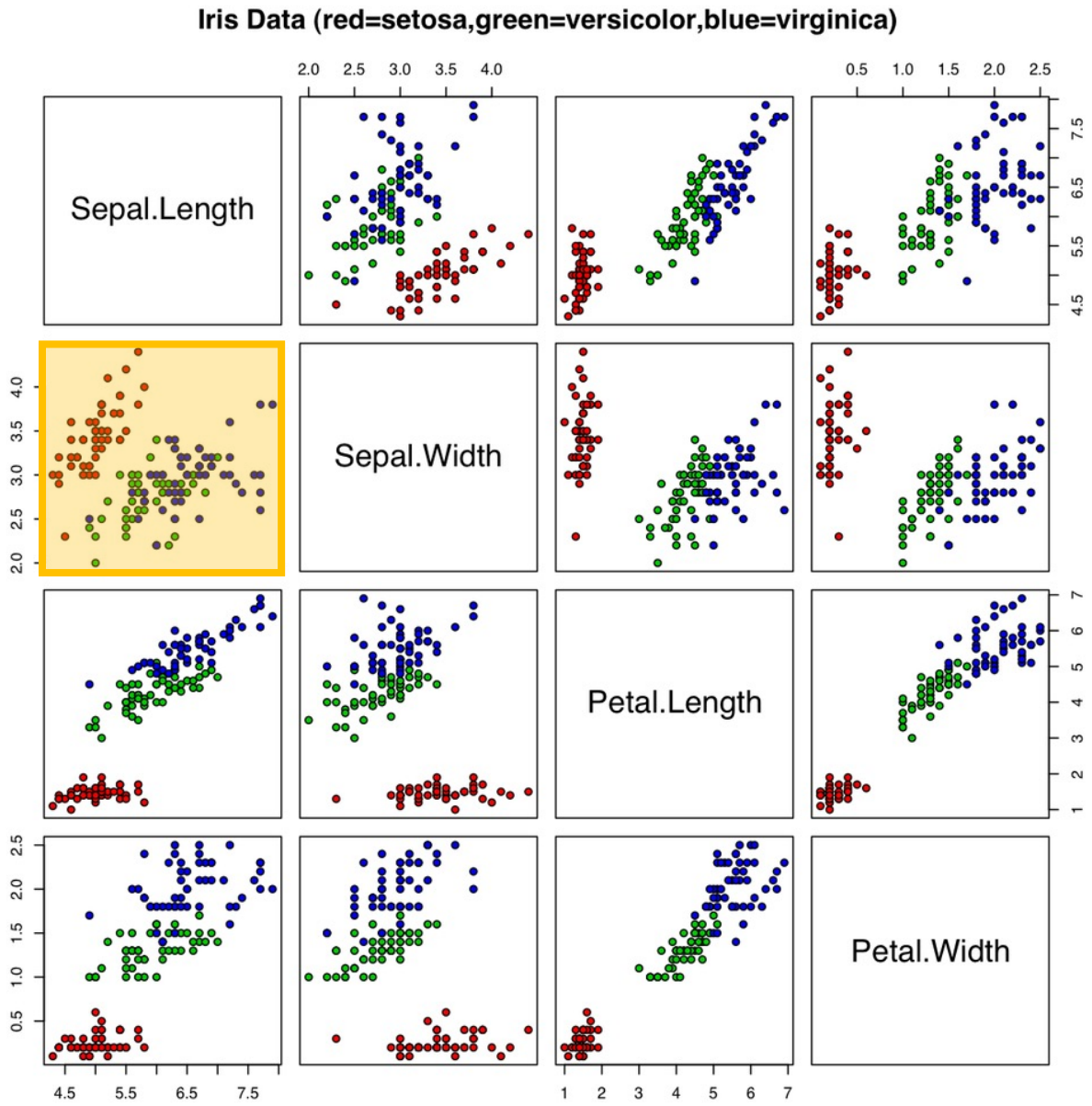
This is convex: we can use gradient descent!

Logistic Regression in action: Iris Dataset



Wikipedia

Logistic Regression in action: Iris Dataset



Any problems? Ideas?

Wikipedia

Multiclass classification with binary algorithms

Two strategies:

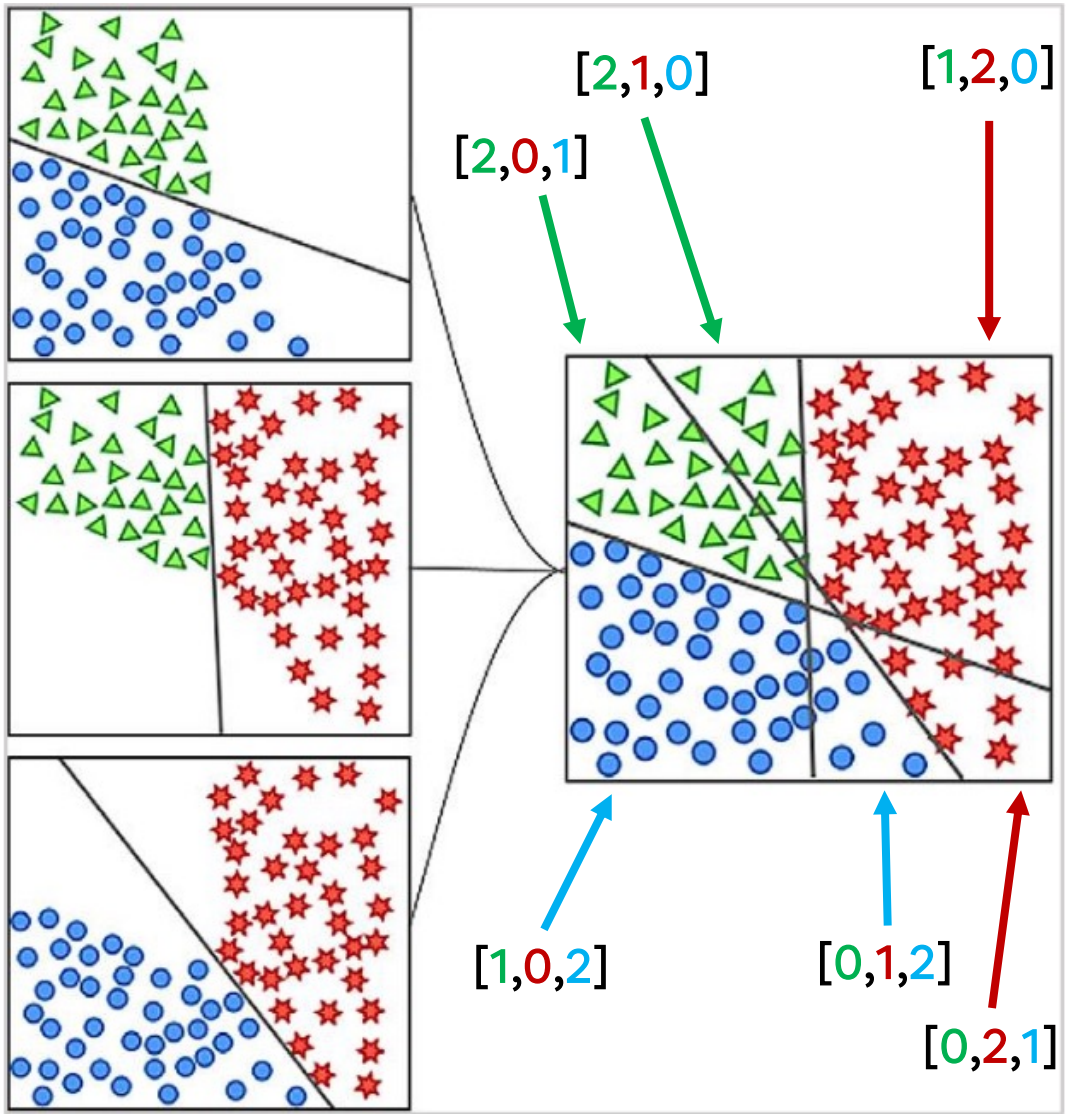
(a) One-vs-one

a. For a *C* classes problem we generate *C(C-1)/2* different classifiers, one for each couple of classes in the dataset

b. Given a new sample to be classified we apply the *C(C-1)/2* classifiers

c. We count how many times each class 'wins' in the related classification against another class

d. The class the has the highest amount of 'victories' is assigned



Multiclass classification with binary algorithms

How to Handle Ties:

- Favor classes based on domain knowledge
- Compute average probabilities for each class
- Random choice

Two strategies:

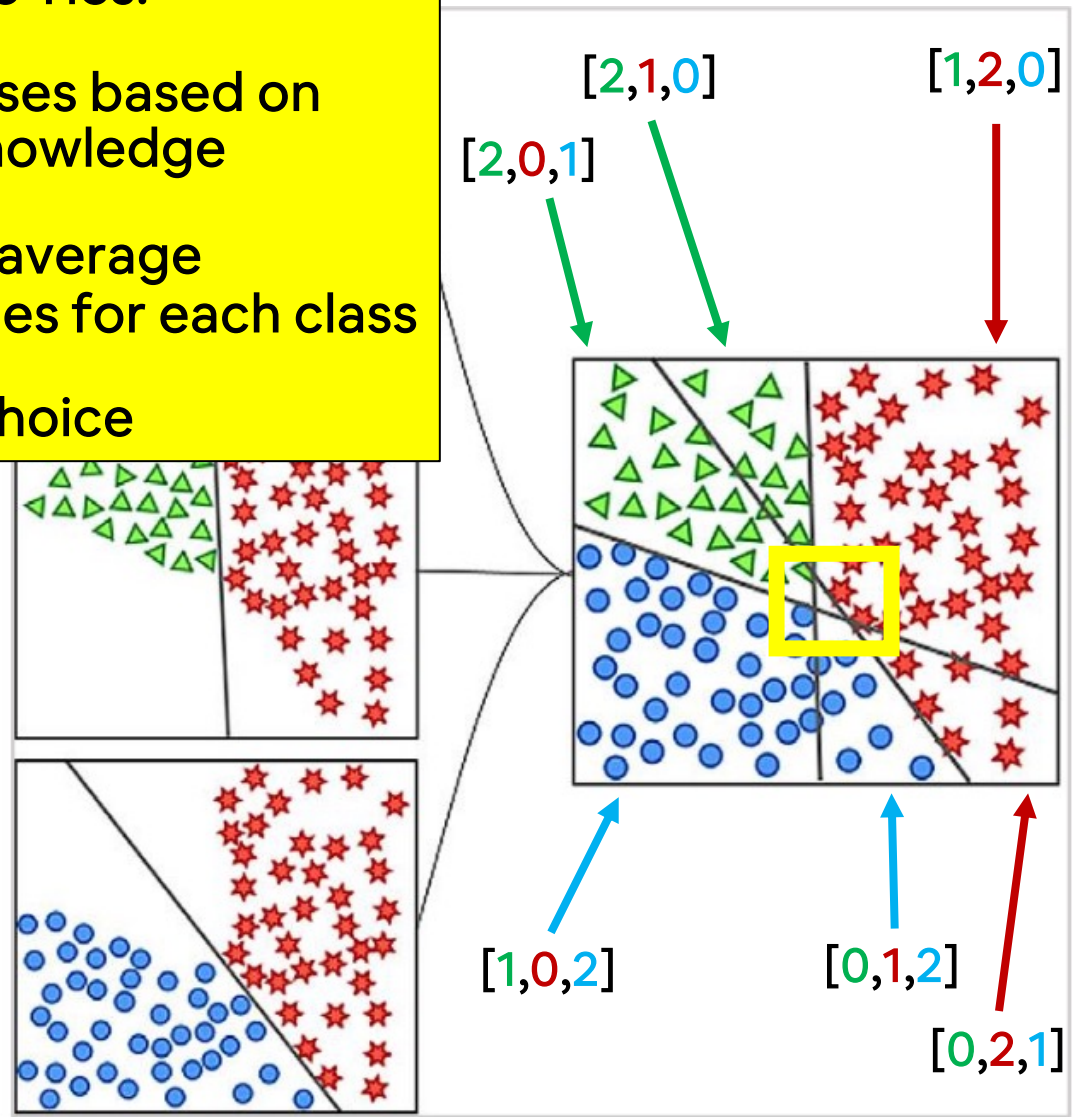
(a) One-vs-one

a. For a *C* classes problem we *C(C-1)/2* different classifie for each couple of classes in the dataset

b. Given a new sample to be classified we apply the *C(C-1)/2* classifiers

c. We count how many times each class 'wins' in the related classification against another class

d. The class the has the highest amount of 'victories' is assigned



Multiclass classification with binary algorithms

Two strategies:

(b) One-vs-all

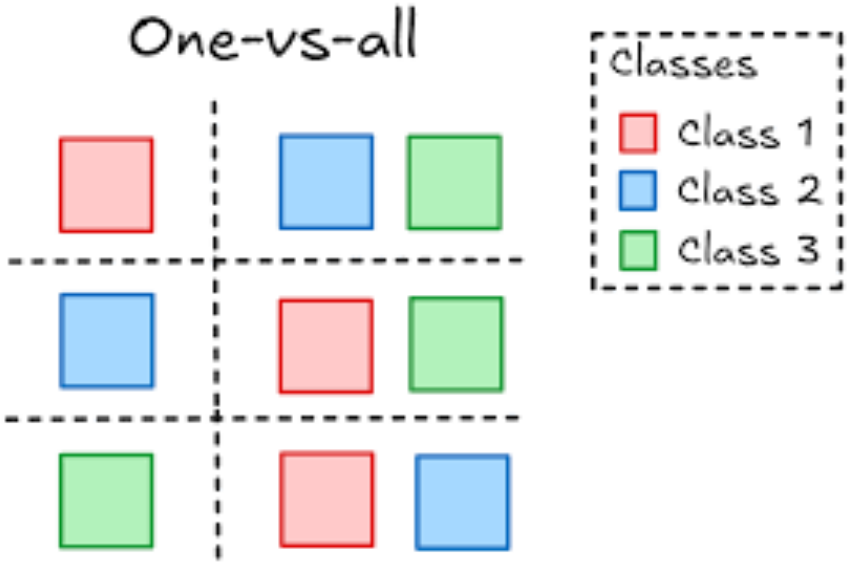
a. For a *C* classes problem, we generate *C* different classifiers

b. For the *c*-th classifier we construct a new label vector, where
   - Yi = 1 if the i-th sample belongs to the c-th class
   - Yi = 0 otherwise

c. The *C* classifiers are computed

d. Given a new sample to be classified we apply the *C* classifiers

e. We assign the class for which the corresponding classifier reports the highest confidence score (in the case of Logistic Regression the associated probability)



Multiclass classification with binary algorithms

Two strategies:

(b) One-vs-all

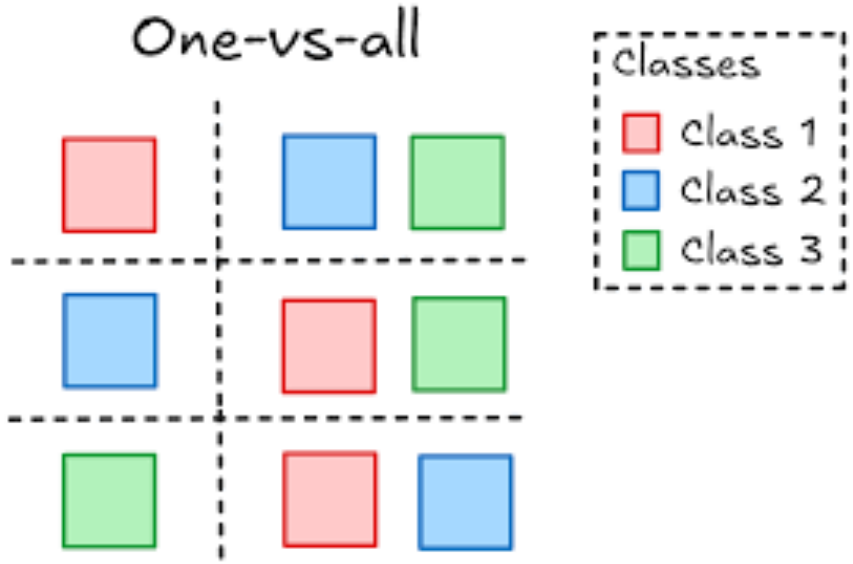
a. For a *C* classes problem, we generate *C* different classifiers

b. For the *c*-th classifier we construct a new label vector, where
   - Yi = 1 if the i-th sample belongs to the c-th class
   - Yi = 0 otherwise

c. The *C* classifiers are computed

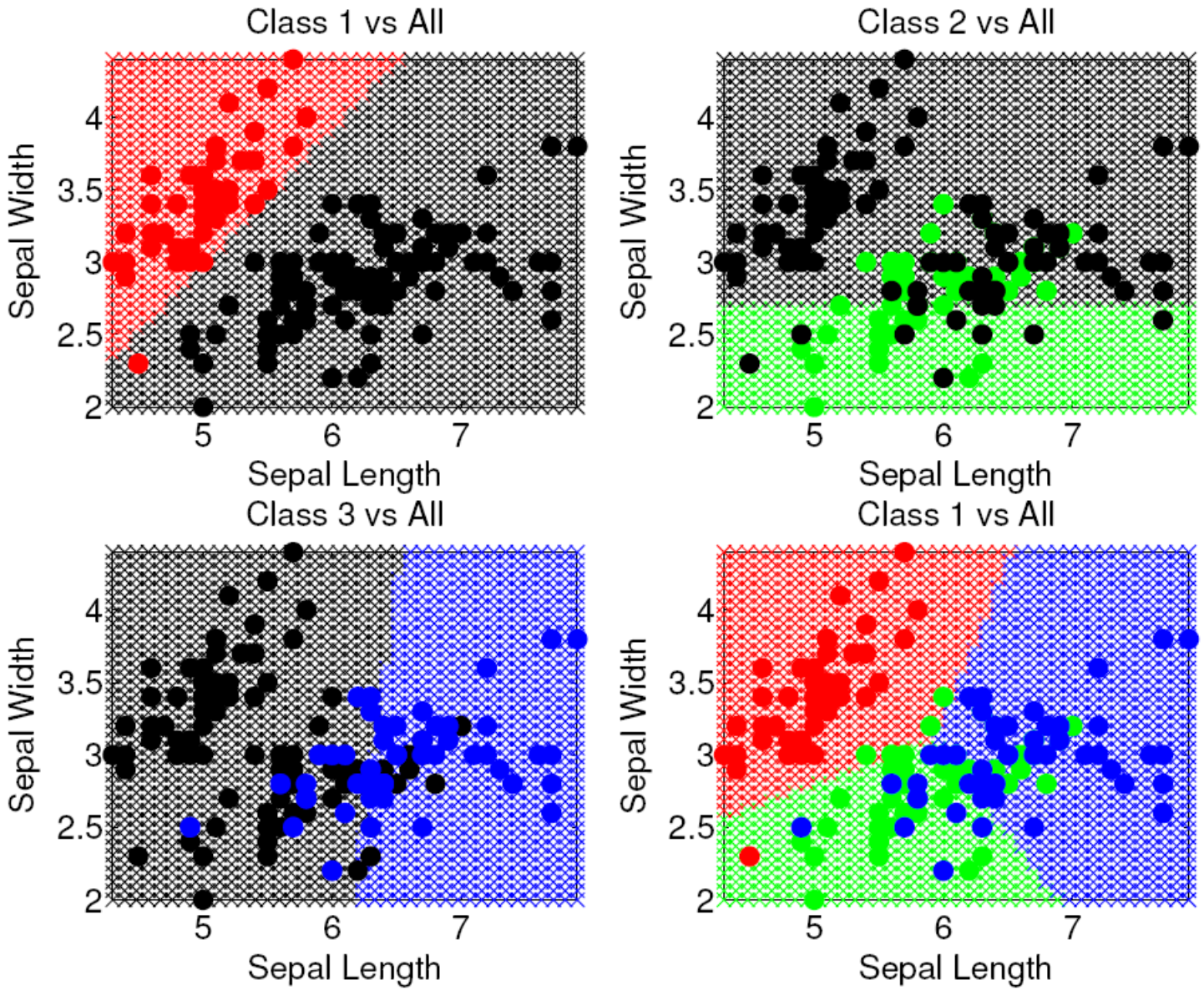
d. Given a new sample to be classified we apply the *C* classifiers

e. We assign the class for which the corresponding classifier reports the highest confidence score (in the case of Logistic Regression the associated probability)



Both popular approaches for multiclass classification, they both have flows:
-1vsAll: unbalance distributions (the 'rest' class is generally more represented)
-1vs1: ambiguities if classes get the same number of votes

Logistic Regression in action: Iris Dataset

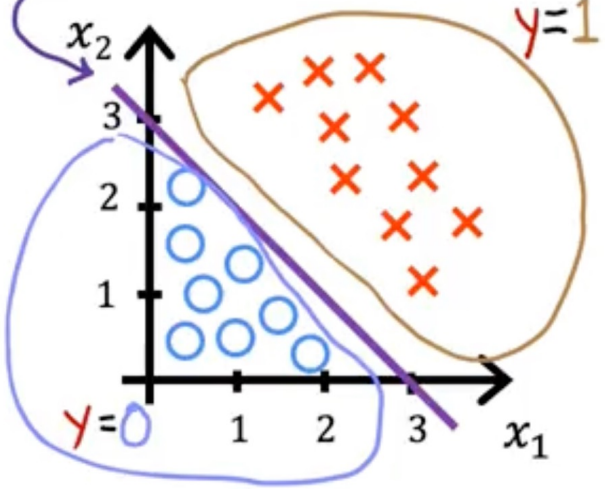


Logistic Regression: Decision Boundaries

$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\underbrace{w_1 x_1 + w_2 x_2 + b}_{z})$$

$w_1 = 1,\ w_2 = 1,\ b = -3$

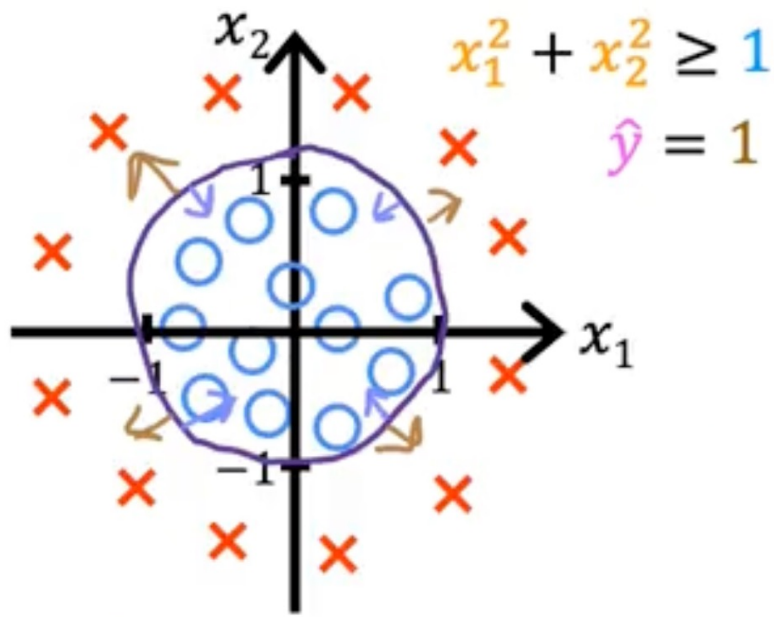
$$z = \vec{w} \cdot \vec{x} + b = 0$$
$$z = x_1 + x_2 - 3 = 0$$
$$x_1 + x_2 = 3$$



$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\underbrace{w_1 x_1^2 + w_2 x_2^2 + b}_{z})$$

$w_1 = 1,\ w_2 = 1,\ b = -1$

$$z = x_1^2 + x_2^2 - 1 = 0$$
$$x_1^2 + x_2^2 = 1$$



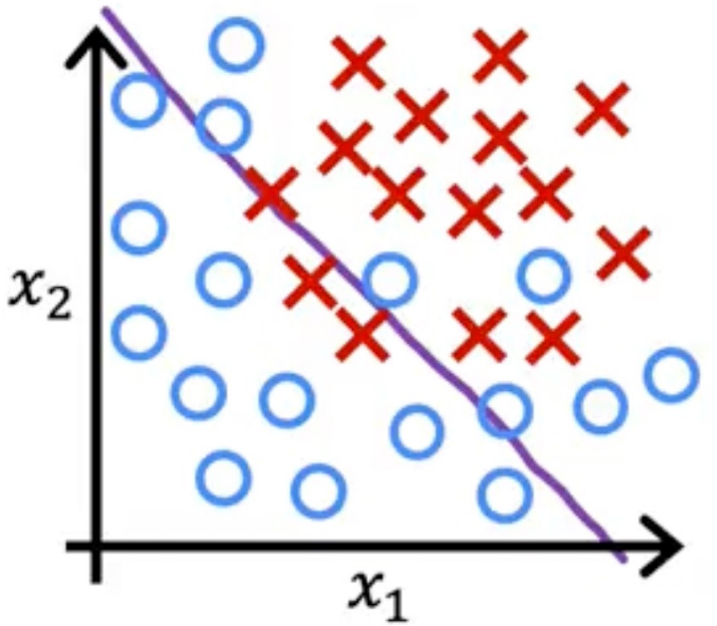
$x_1^2 + x_2^2 \geq 1$

$\hat{y} = 1$

$x_1^2 + x_2^2 < 1$

$\hat{y} = 0$

Logistic Regression (LR): Decision Boundaries

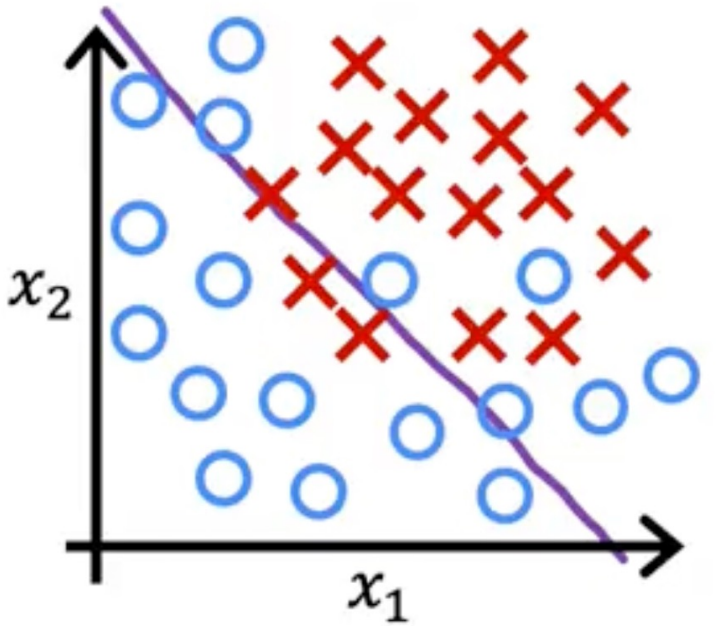


$$z = w_1 x_1 + w_2 x_2 + b$$

$$f_{\vec{w},b}(\vec{x}) = g(z)$$

g is the sigmoid function

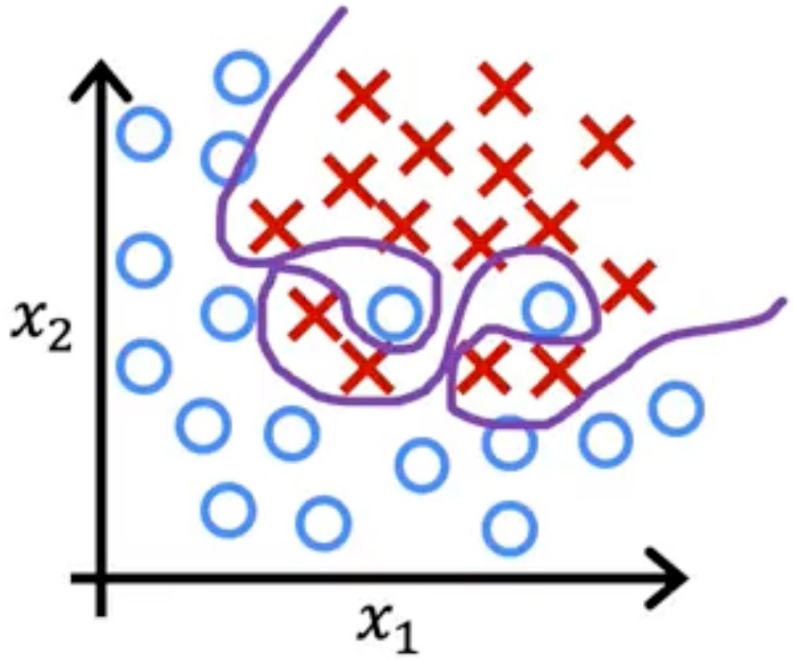
Logistic Regression (LR): Decision Boundaries



$$z = w_1 x_1 + w_2 x_2 + b$$

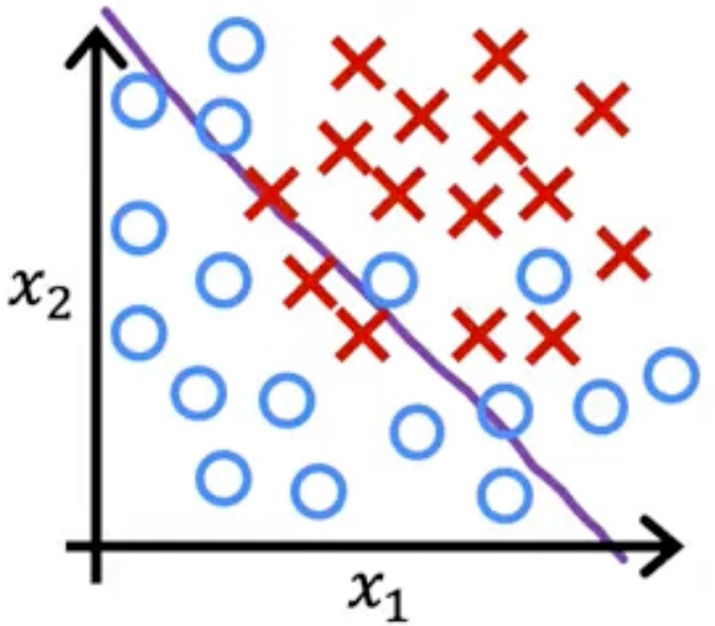
$$f_{\vec{w},b}(\vec{x}) = g(z)$$

g is the sigmoid function



$$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 x_2 + w_4 x_1^2 x_2^2 + w_5 x_1^2 x_2^3 + w_6 x_1^3 x_2 + \cdots + b$$

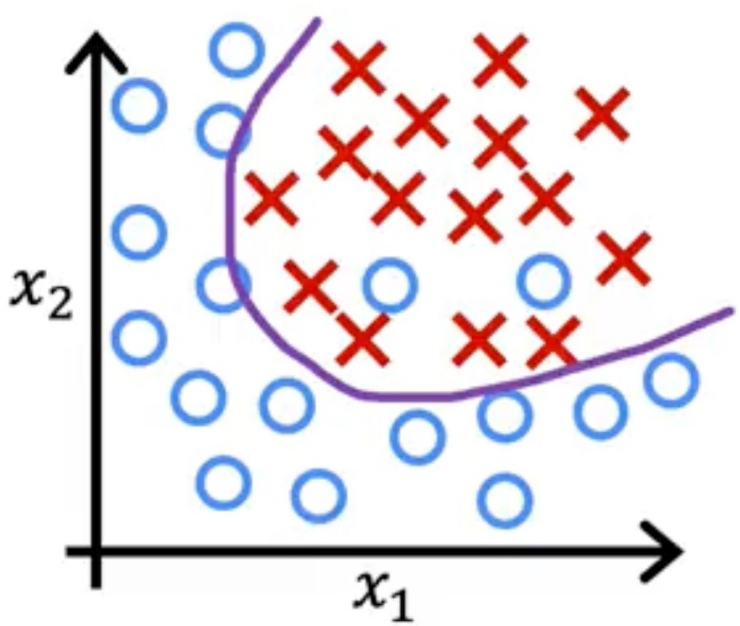
Logistic Regression (LR): Decision Boundaries



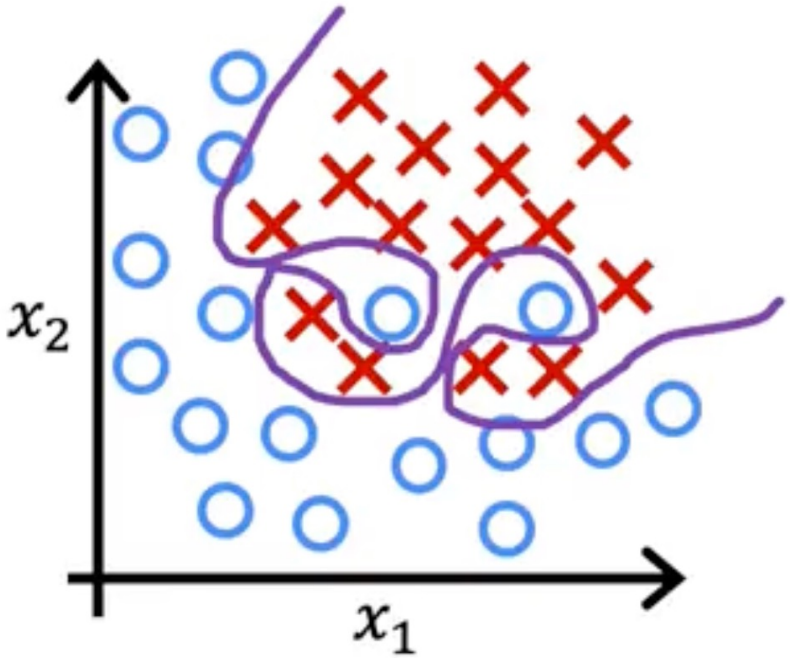
$$z = w_1 x_1 + w_2 x_2 + b$$

$$f_{\vec{w},b}(\vec{x}) = g(z)$$

g is the sigmoid function

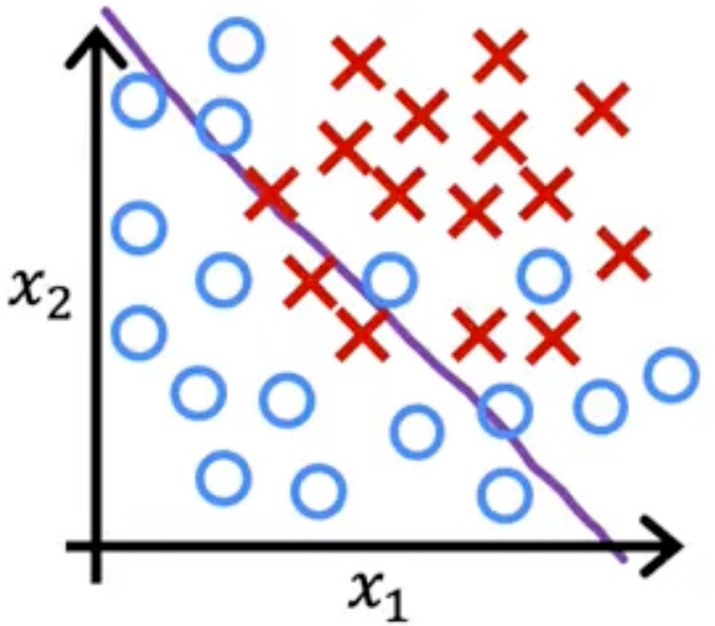


$$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2 + b$$



$$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 x_2 + w_4 x_1^2 x_2^2 + w_5 x_1^2 x_2^3 + w_6 x_1^3 x_2 + \cdots + b$$

Logistic Regression (LR): Decision Boundaries

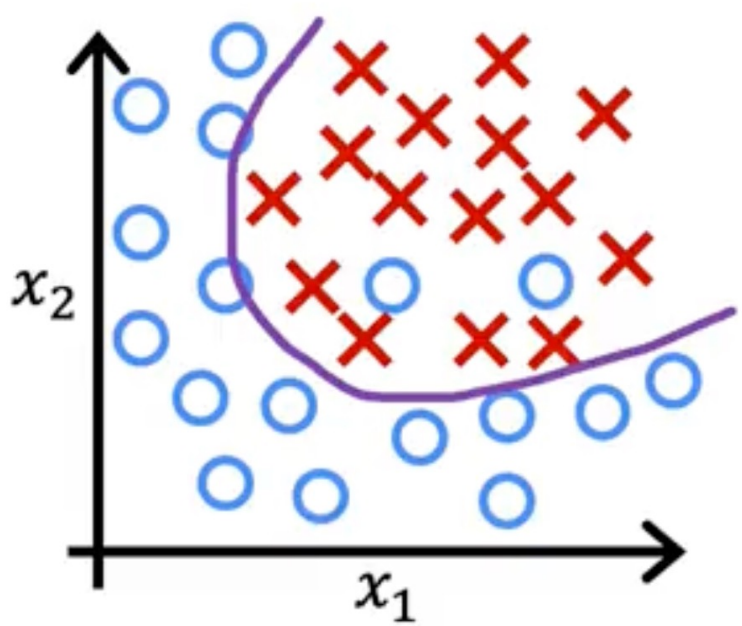


$$z = w_1 x_1 + w_2 x_2 + b$$

$$f_{\vec{w},b}(\vec{x}) = g(z)$$

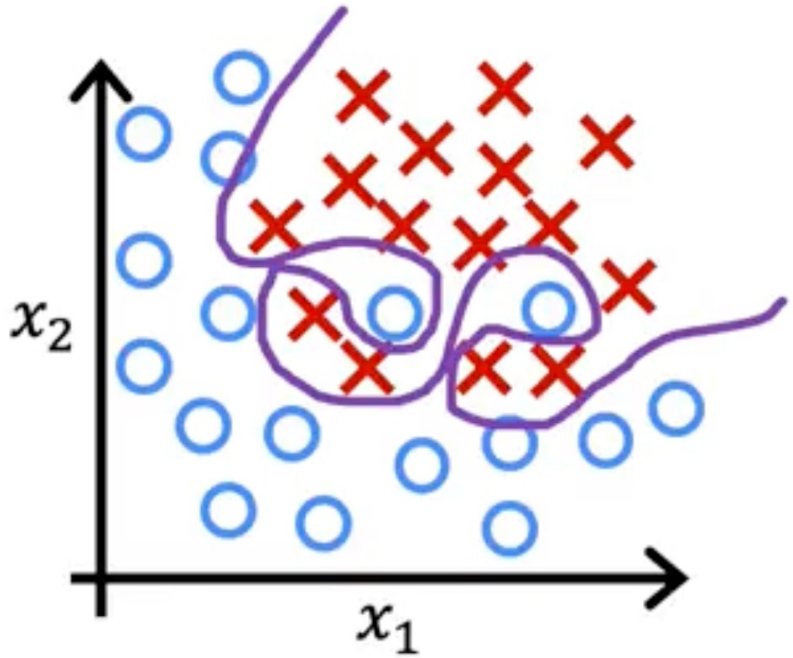
g is the sigmoid function

underfit high bias



$$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2 + b$$

just right



$$z = w_1 x_1 + w_2 x_2 + w_3 x_1^2 x_2 + w_4 x_1^2 x_2^2 + w_5 x_1^2 x_2^3 + w_6 x_1^3 x_2 + \cdots + b$$

We can resort to regularization also with LR!

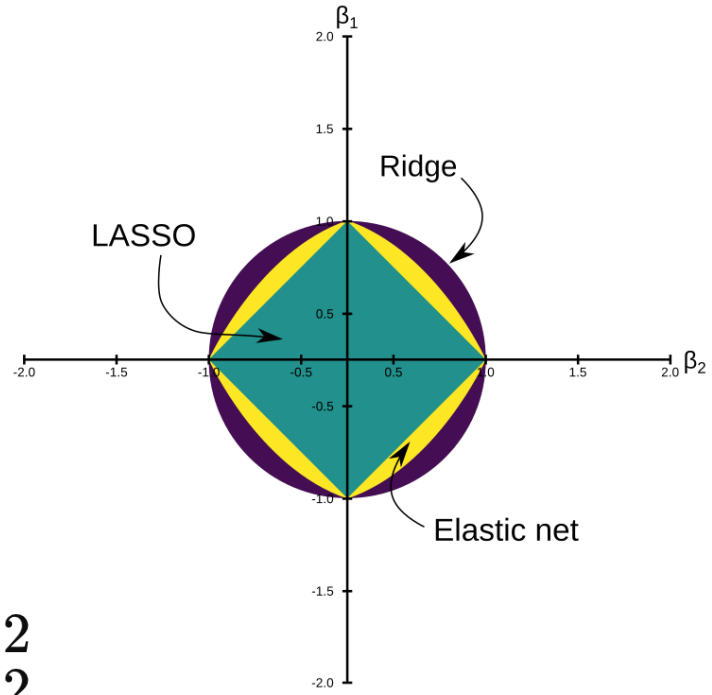
$$\mathcal{L}_{\text{reg}}(\beta) = \mathcal{L}(\beta) + \lambda R(\beta)$$

$$\mathcal{L}(\beta) = -\sum_{i=1}^{n} \left[y_i \log \sigma(x_i^\top \beta) + (1 - y_i) \log(1 - \sigma(x_i^\top \beta)) \right]$$

- Ridge Regression $R(\beta) = \frac{1}{2}\|\beta\|_2^2 = \frac{1}{2}\sum_j \beta_j^2$

- LASSO $R(\beta) = \|\beta\|_1 = \sum_j |\beta_j|$

- Elastic Net $R(\beta) = \alpha\|\beta\|_1 + (1-\alpha)\frac{1}{2}\|\beta\|_2^2$



Linear/Logistic Regression: Kernel Methods (optional)

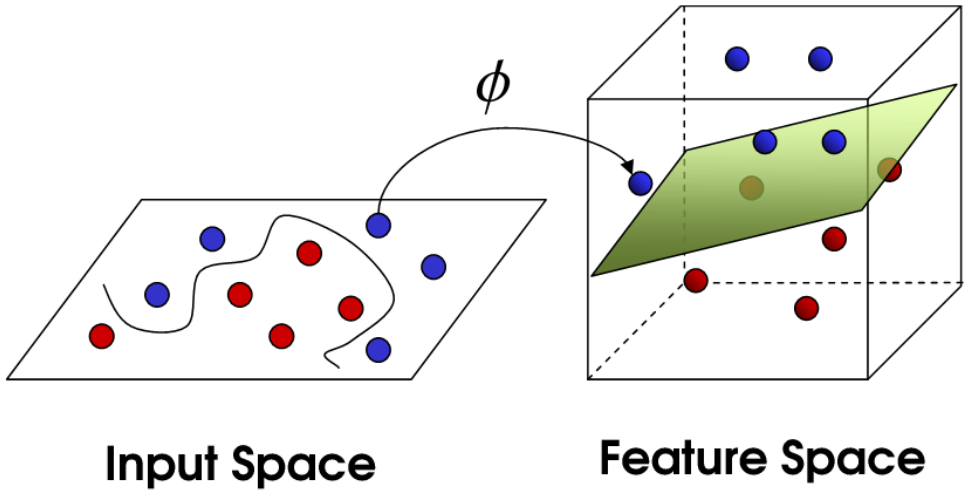
- Kernel methods allow us to apply linear algorithms in nonlinear settings by implicitly mapping data into a higher-dimensional space, using a kernel function (e.g., RBF, polynomial). Instead of computing the mapping explicitly, the kernel trick computes inner products in the new space directly.

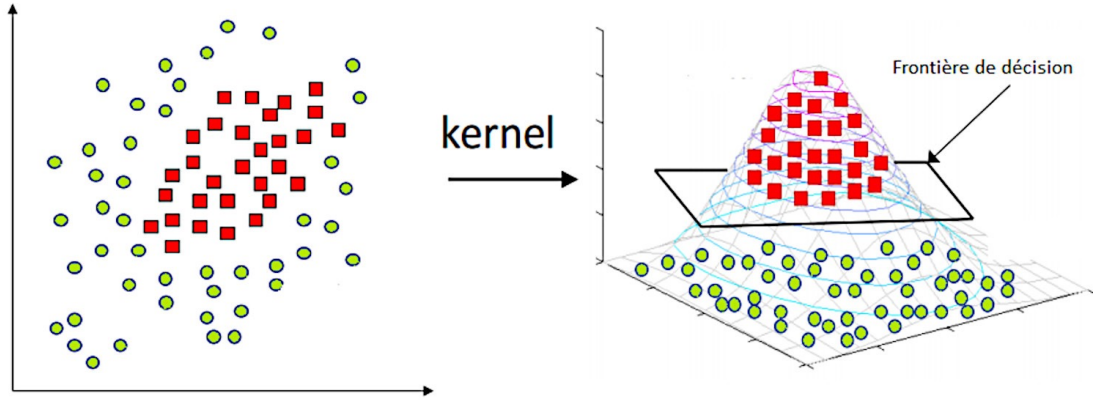
Linear/Logistic Regression: Kernel Methods (optional)

- Kernel methods allow us to apply linear algorithms in nonlinear settings by implicitly mapping data into a higher-dimensional space, using a kernel function (e.g., RBF, polynomial). Instead of computing the mapping explicitly, the kernel trick computes inner products in the new space directly.
- In regularization, kernel methods enable us to control model complexity in this feature space, often leading to better generalization.

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \left(y_i - \mathbf{w}^\top \phi(\mathbf{x}_i)\right)^2 + \lambda \|\mathbf{w}\|^2$$

- More on this when we see Support Vector Machines (SVM)



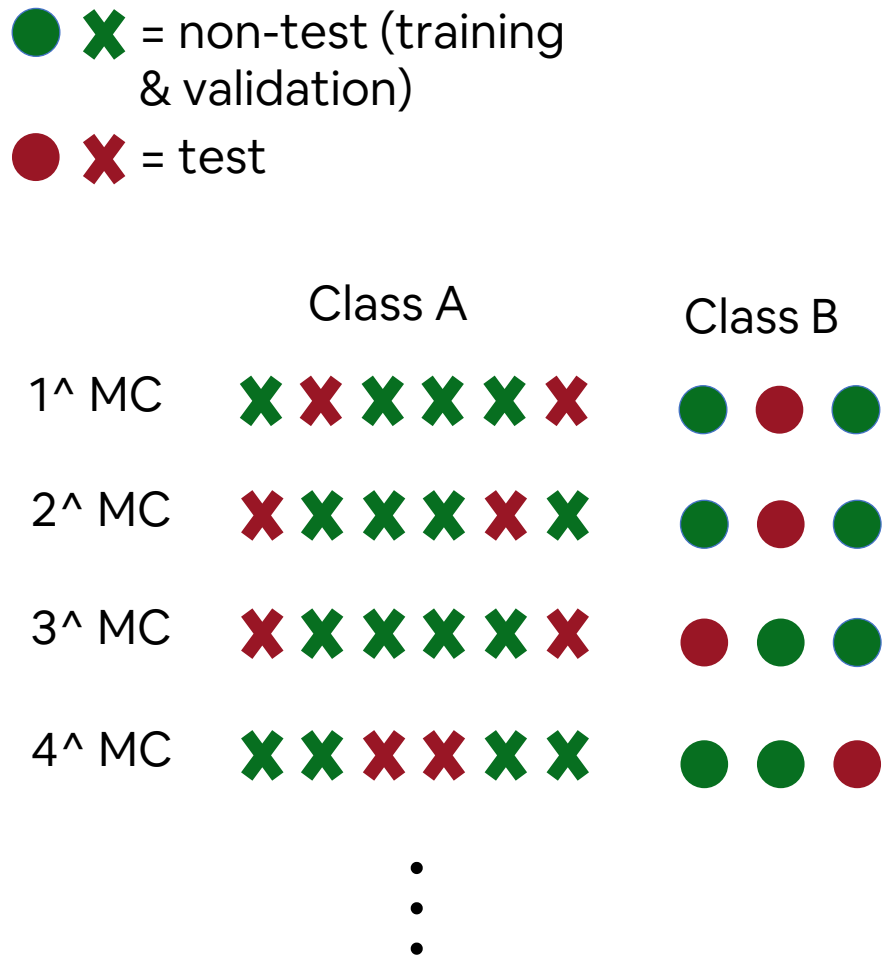


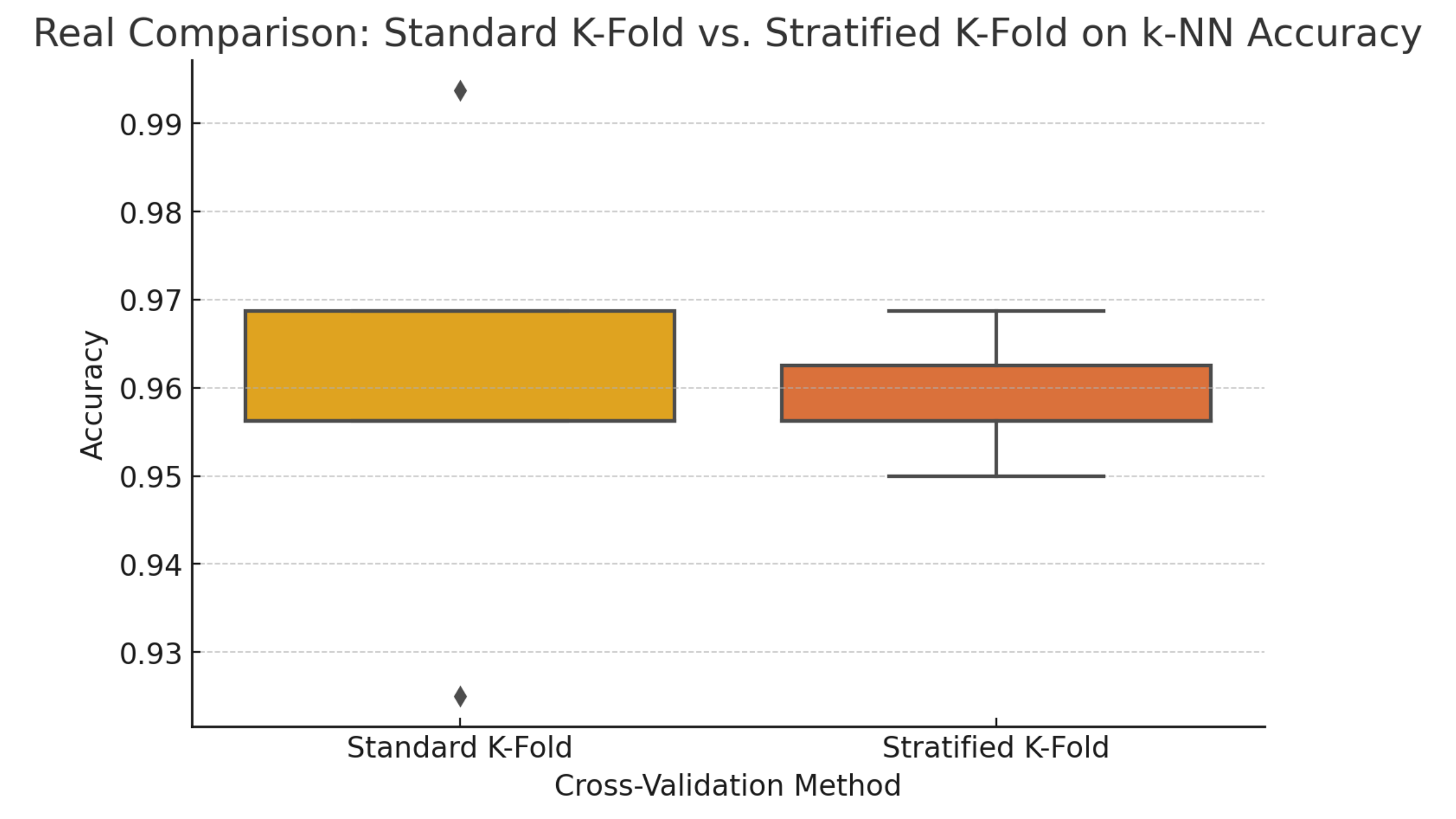
Before leaving: Stratified Cross-Validation

- Stratified Cross-Validation is a variation of cross-validation that preserves the percentage of samples for each class in every fold.
- In other words, each fold has approximately the same class distribution as the entire dataset.

✅ Especially useful when:

- Your classes are imbalanced (e.g., 90% class A, 10% class B)
- You want more reliable model evaluation
- You're working with classification problems




Real Comparison: Standard K-Fold vs. Stratified K-Fold on k-NN Accuracy
Accuracy
0.99
0.98
0.97
0.96
0.95
0.94
0.93
Standard K-Fold
Stratified K-Fold
Cross-Validation Method

Reminder before leaving: Exam – theoretic/numeric exercise part

A 45-60 minutes exam, multiple choices (main reference: slides)

➔ We'll make a 'simulation' during next week lecture (lecture 14 – 27th of March)

➔ The lecture will also be a recap of the first part of the course: if there are topics that you'd like to be discussed, let me know

We are preparing some example exercises that we'll be shared with you.

Thank you!

Gian Antonio Susto