

IFT 4031/7031,  
Machine Learning for Signal Processing  
**Week6: Machine Learning 3,  
Classification**

Cem Subakan



UNIVERSITÉ  
**Laval**



- How is homework 1 going? The deadline is Oct 28th (as indicated on teams)
  - ▶ Comment va le devoir 1? Le deadline est le Oct 28. (comme indiqué sur teams)
- If you have questions on your project just let me know.
  - ▶ S'il y a des hesitations laissez nous savoir.
- Did you manage to get into VALERIA? There will be a tutorial.
  - ▶ Ca va bien avec VALERIA? Il y aura un tutoriel.
- Aujourd'hui: Classification

# Table of Contents

---

## Classification Intro

## Generative Classification

## Discriminative Classification

Linear Classifiers

The perceptron algorithm

Logistic Regression

## Non-Linear Classification

Kernel Logistic Regression

Neural Network Classification

# Supervised Learning

---

- So far we have mostly done unsupervised learning to discover structures.
  - ▶ Jusqu'à maintenant on a majoritairement fait de l'apprentissage non-supervisé. Le but était de découvrir la structure.
- Now, we will do classification / detection, or supervised learning in other words.
  - ▶ Maintenant on va faire de l'apprentissage supervisé.

# A simple example

---



# A simple example

---



How can we assign the query to a class? / Comment peut-on assigner l'exemple en question à une classe?

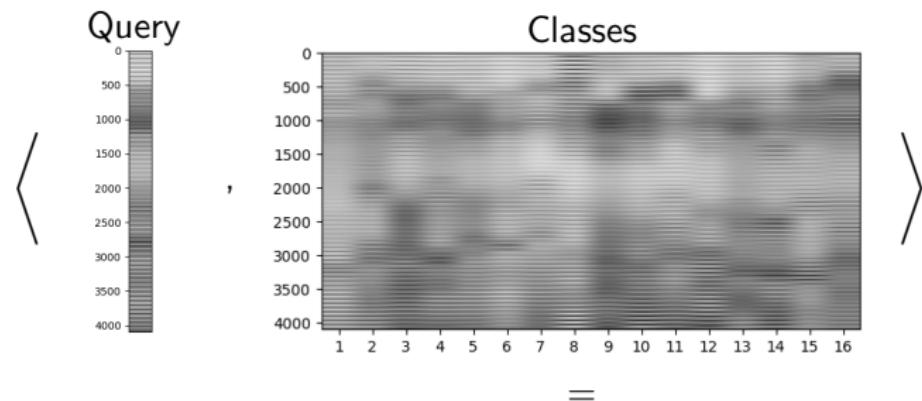
## We can simply calculate inner products!

---

We can calculate inner products! / On peut calculer des produits scalaires!

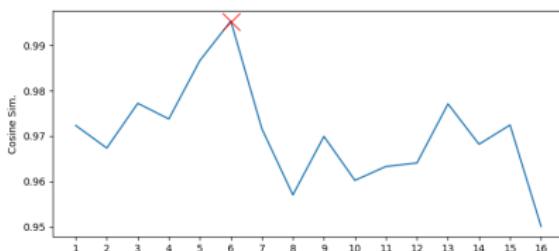
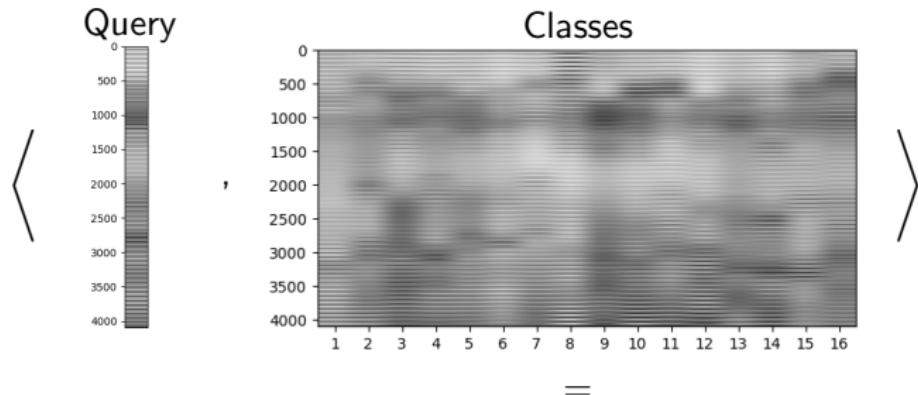
# We can simply calculate inner products!

We can calculate inner products! / On peut calculer des produits scalaires!



# We can simply calculate inner products!

We can calculate inner products! / On peut calculer des produits scalaires!



# It worked but will it this time?

---

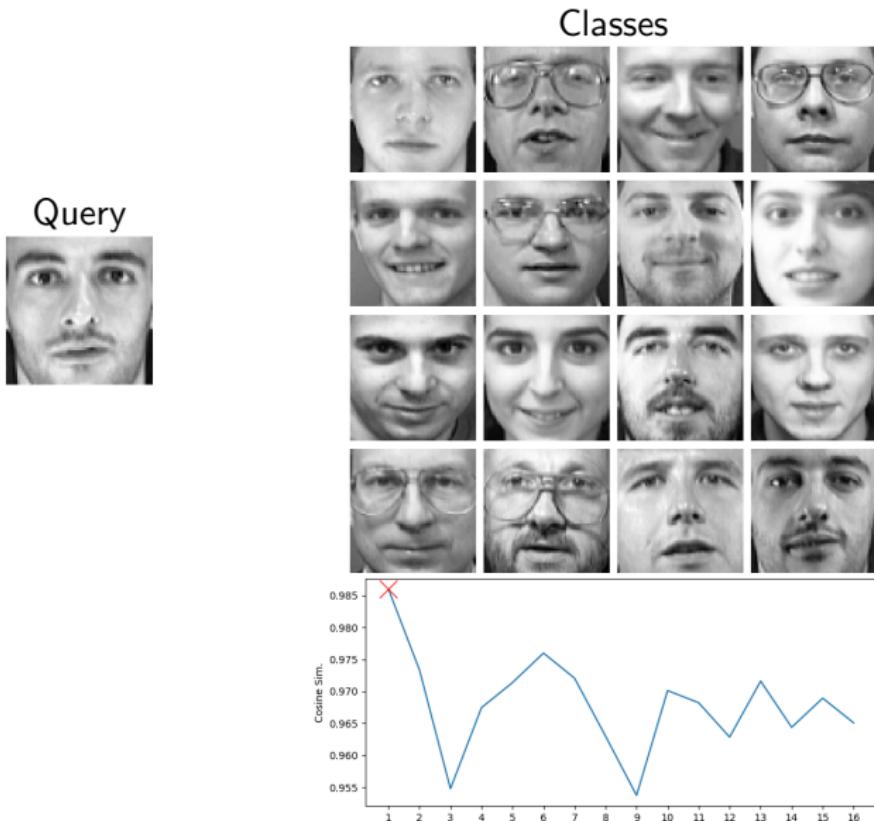


# It worked but will it this time?

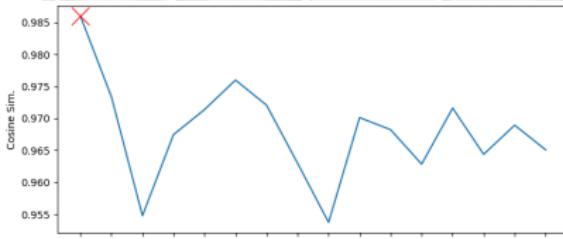
---



# It worked but will it this time?



# It worked but will it this time?



Didn't work this time! / Ça pas fonctionné!

## What can we do to fix this?

---

- We will take a statistical approach (as usual). This will consider the class distributions. / On va prendre une approche de statistique comme d'habitude qui va prendre en compte les distributions de classes.

# What can we do to fix this?

---

- We will take a statistical approach (as usual). This will consider the class distributions. / On va prendre une approche de statistique comme d'habitude qui va prendre en compte les distributions de classes.
- Approach 1: Generative Classification
  - ▶ Approche 1: Classification Générative
    - ▶ We will learn a distribution over samples in the class. / On va fitter une distribution sur les échantillons dans la classe.
- Approach 2: Discriminative Classification
  - ▶ Approche 2: Classification Discriminative
    - ▶ We will learn how the class distribution separate. / On va fitter une fonction pour comprendre comment les classes se séparent.

# What can we do to fix this?

---

- We will take a statistical approach (as usual). This will consider the class distributions. / On va prendre une approche de statistique comme d'habitude qui va prendre en compte les distributions de classes.
- Approach 1: Generative Classification
  - ▶ Approche 1: Classification Générative
    - ▶ We will learn a distribution over samples in the class. / On va fitter une distribution sur les échantillons dans la classe.
- Approach 2: Discriminative Classification
  - ▶ Approche 2: Classification Discriminative
    - ▶ We will learn how the class distribution separate. / On va fitter une fonction pour comprendre comment les classes se séparent.
- We will build step by step towards neural nets, and why we need them in this lecture. / On va builder graduellement à pourquoi on a besoin des réseaux de neurones dans ce cours.
  - ▶ Linear Classifiers (Perceptron Algo. Logistic Regression, Kernel Methods, and then Multilayer perceptron)

# Table of Contents

---

Classification Intro

## Generative Classification

## Discriminative Classification

Linear Classifiers

The perceptron algorithm

Logistic Regression

## Non-Linear Classification

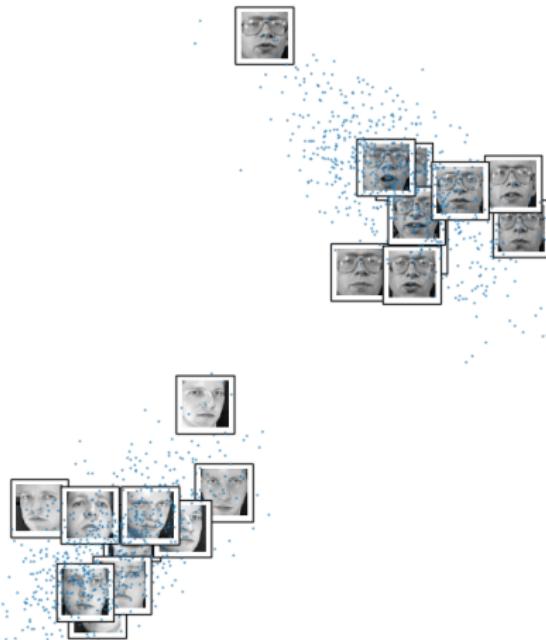
Kernel Logistic Regression

Neural Network Classification

# Generative Classification

---

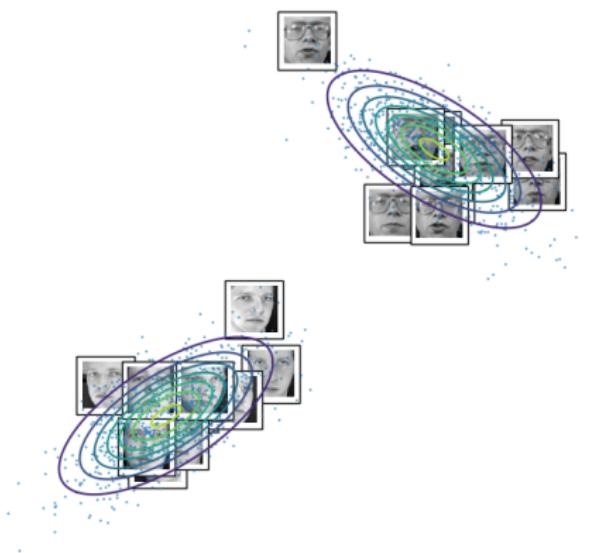
- Generative Classification fits distributions to each class /  
Classification générative fit une distribution à chaque classe.



# Generative Classification

---

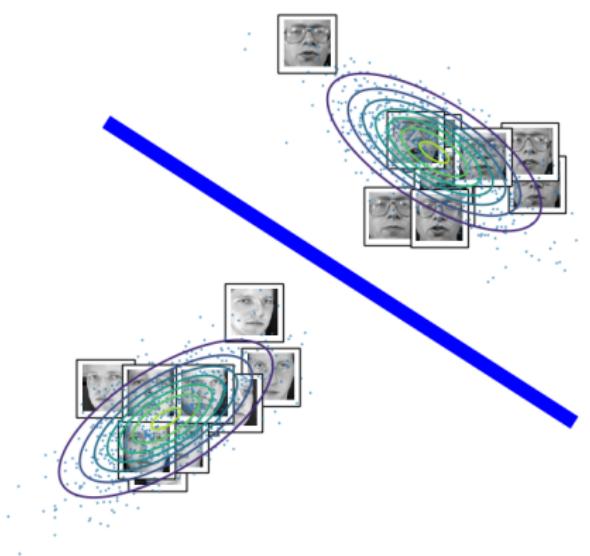
- Generative Classification fits distributions to each class /  
Classification générative fit une distribution à chaque classe.



# Generative Classification

---

- Generative Classification fits distributions to each class /  
Classification générative fit une distribution à chaque classe.



# Generative Classification

---

- Training Time: Fit a distribution  $p(x|\theta_k)$  to each class  $k$  with maximum likelihood.
  - ▶ L'entraînement: On va fitter une distribution  $p(x|\theta_k)$  pour chaque classe  $k$  avec maximum likelihood.

$$\max_{\theta_k} \sum_{n \in \text{class } k} \log p(x_n|\theta_k)$$

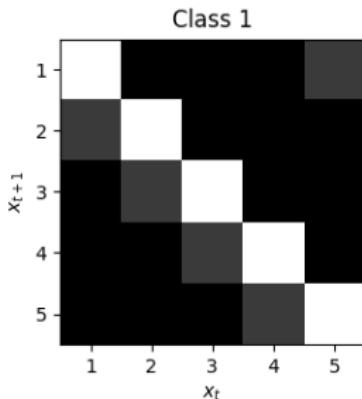
- Test Time: Evaluate the likelihood for each model. Assign to the largest likelihood class!
  - ▶ L'entraînement: Évaluez le likelihood pour chaque modèle. Assignez à la classe avec le likelihood plus grand.

$$\hat{c} = \arg \max_k \log p(x_{\text{test}}|\theta_k)$$

## Example Application: Classifying Sequences

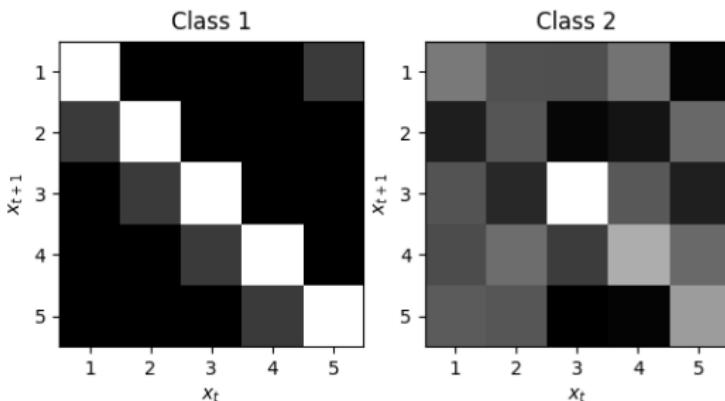
---

- class 1  $\sim \text{Markov}(A_1)$
- class 2  $\sim \text{Markov}(A_2)$



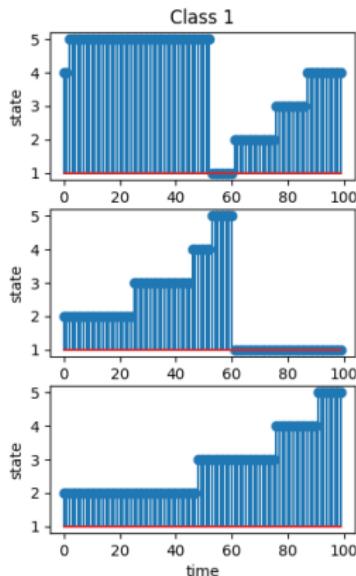
# Example Application: Classifying Sequences

- class 1  $\sim \text{Markov}(A_1)$
- class 2  $\sim \text{Markov}(A_2)$



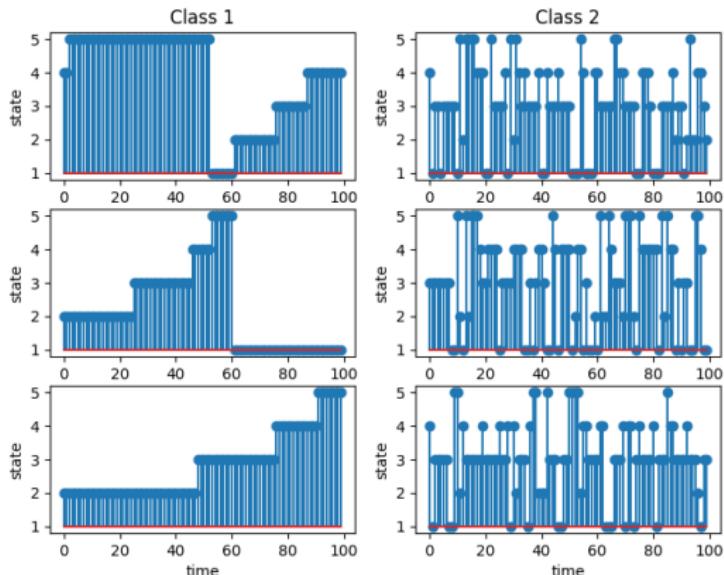
# Example Application: Classifying Sequences

- class 1  $\sim \text{Markov}(A_1)$
- class 2  $\sim \text{Markov}(A_2)$



# Example Application: Classifying Sequences

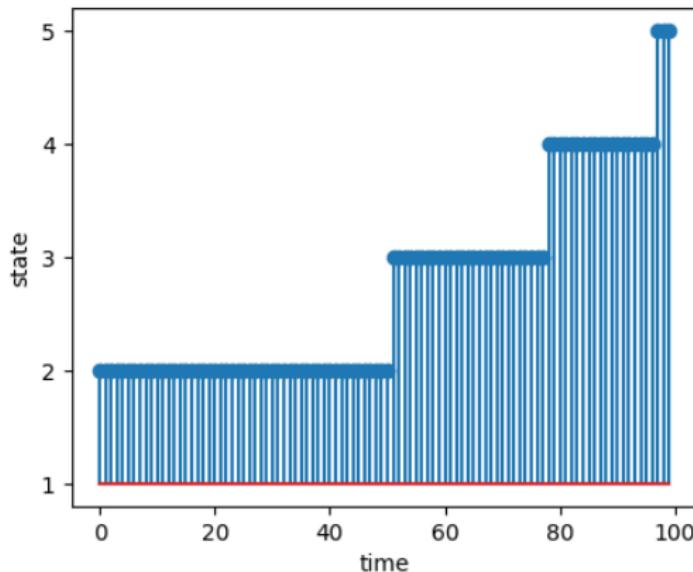
- class 1  $\sim \text{Markov}(A_1)$
- class 2  $\sim \text{Markov}(A_2)$



# Test time

---

- Here's a test sequence / Une séquence de test



- Do you think it belongs to which class? / Vous pensez que cette séquence appartient à quelle classe?

## Test time

---

$$\log p(x_{1:T} | A_k, \pi_k) = \log \left( p(x_1 | \pi_k) \prod_{t=2}^T p(x_t | x_{t-1}, A_k) \right)$$

## Test time

---

$$\begin{aligned}\log p(x_{1:T} | A_k, \pi_k) &= \log \left( p(x_1 | \pi_k) \prod_{t=2}^T p(x_t | x_{t-1}, A_k) \right) \\ &= \log p(x_1 | \pi_k) + \sum_{t=2}^T \log p(x_t | x_{t-1}, A_k)\end{aligned}$$

## Test time

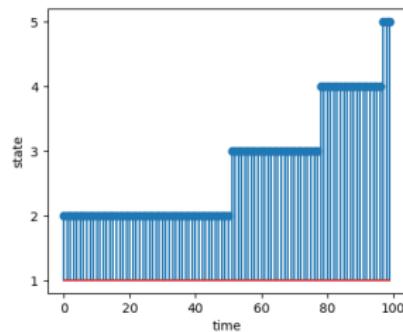
---

$$\begin{aligned}\log p(x_{1:T} | A_k, \pi_k) &= \log \left( p(x_1 | \pi_k) \prod_{t=2}^T p(x_t | x_{t-1}, A_k) \right) \\ &= \log p(x_1 | \pi_k) + \sum_{t=2}^T \log p(x_t | x_{t-1}, A_k) \\ &= \log \pi_k + \sum_{t=2}^T \log A_k(x_t, x_{t-1})\end{aligned}$$

# Now, let's calculate

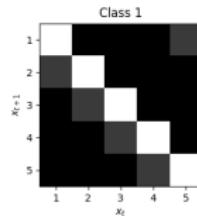
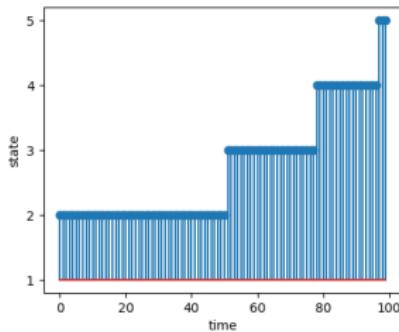
---

## ■ Observation sequence / Séquence observé



# Now, let's calculate

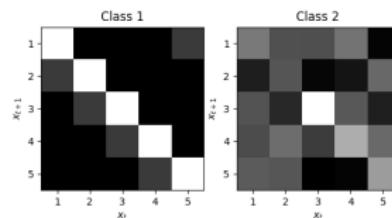
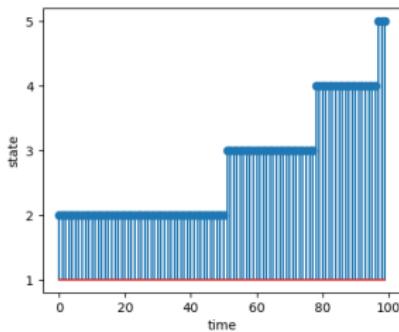
## ■ Observation sequence / Séquence observé



$$\begin{aligned}\log p(x_{1:T} | A_1, \pi_1) &= \log \left( \pi_1 \prod_{t=2}^T A_1(x_t, x_{t-1}) \right) \\ &= \log \left( \frac{1}{5} \cdot 0.95 \cdot 0.95 \cdot 0.95 \dots \right) \\ &= -15.52\end{aligned}$$

# Now, let's calculate

## ■ Observation sequence / Séquence observé



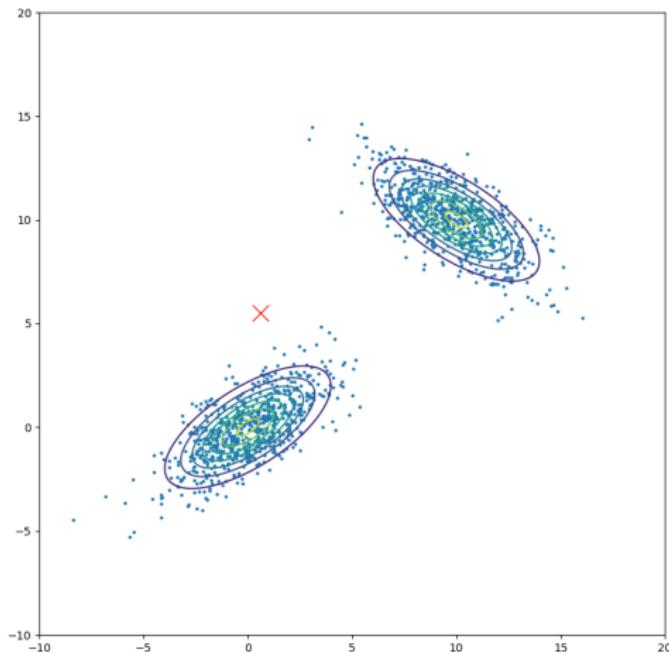
$$\begin{aligned}\log p(x_{1:T} | A_1, \pi_1) &= \log \left( \pi_1 \prod_{t=2}^T A_1(x_t, x_{t-1}) \right) \\ &= \log \left( \frac{1}{5} \cdot 0.95 \cdot 0.95 \cdot 0.95 \dots \right) \\ &= -15.52\end{aligned}$$

$$\begin{aligned}\log p(x_{1:T} | A_2, \pi_2) &= \log \left( \pi_1 \prod_{t=2}^T A_2(x_t, x_{t-1}) \right) = \log \left( \frac{1}{5} \cdot 0.21 \cdot 0.21 \cdot 0.21 \dots \right) \\ &= -117.93\end{aligned}$$

# Let's do something simpler

---

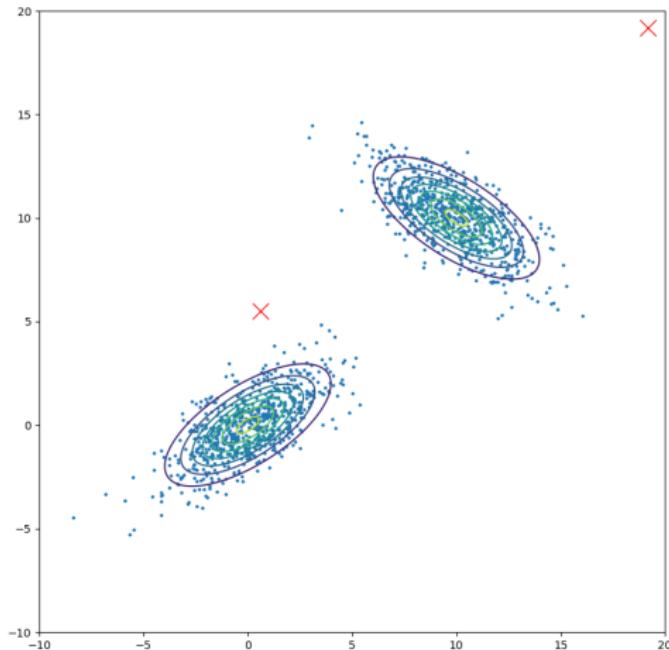
Which class should we assign the red point? / Quel classe doit-on assigner au point rouge?



# How about this?

---

How about this second point? / Et le deuxième point?



# The principled way to derive the decision boundaries

---

- The discriminant function / La fonction de discrimination

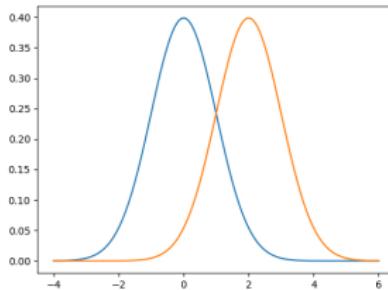
$$d(x) = \log \frac{p(x|c=1)}{p(x|c=2)} = \log \frac{p(x|\theta_1)}{p(x|\theta_2)}$$

- The decision boundary:  $d(x) = 0$ .
- For Gaussians / simple densities this can be derived analytically.

# Deriving the decision boundary

---

- Derive the boundary in 1d case / Dérivons dans le cas 1d.

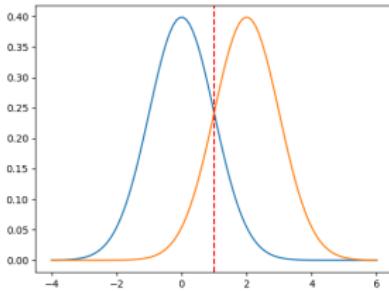


$$d(x) = \log \frac{p(x|\theta_1)}{p(x|\theta_2)} = \log \frac{\mathcal{N}(0; 1)}{\mathcal{N}(2; 1)}$$

# Deriving the decision boundary

---

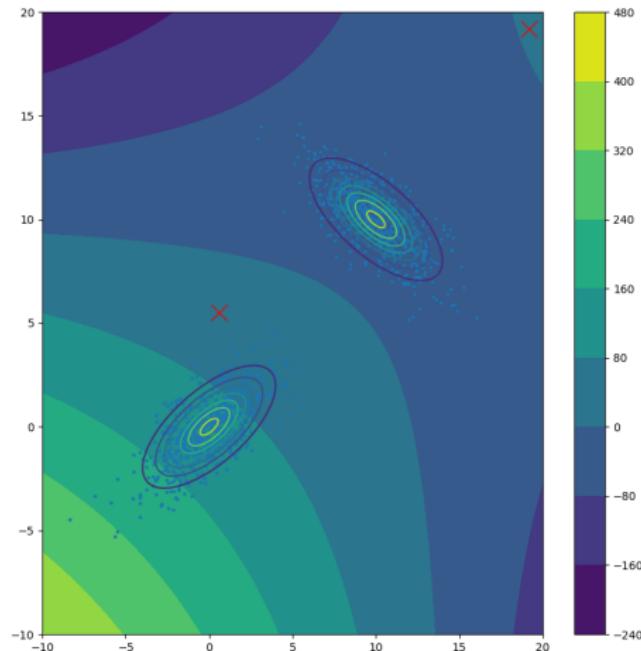
- Derive the boundary in 1d case / Dérivons dans le cas 1d.



$$\begin{aligned} d(x) &= \log \frac{p(x|\theta_1)}{p(x|\theta_2)} = \log \frac{\mathcal{N}(0; 1)}{\mathcal{N}(2; 1)} \\ &= \log \frac{x^2}{(x - 2)^2} = 0 \\ \rightarrow x^2 &= (x - 2)^2 \rightarrow x = 1 \end{aligned}$$

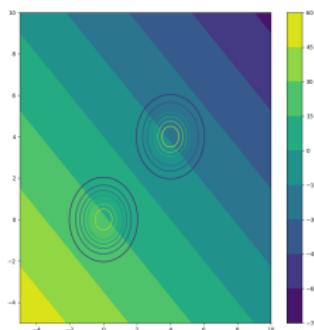
# The 2d decision boundary

$$\text{Plot } d(x) = \log \frac{\mathcal{N}([x,y]^\top; \mu_1, \Sigma_1)}{\mathcal{N}([x,y]^\top; \mu_2, \Sigma_2)}$$



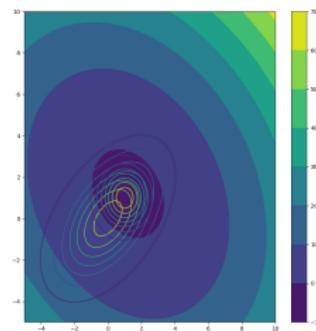
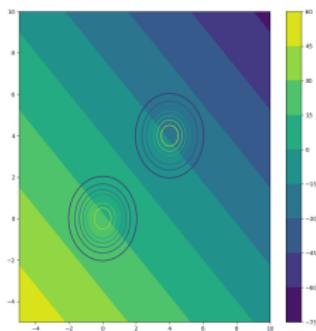
## Few other cases

---



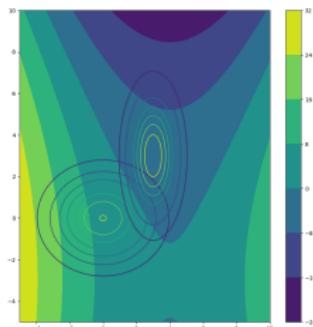
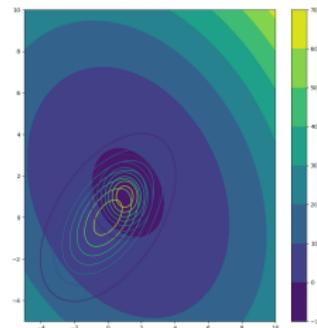
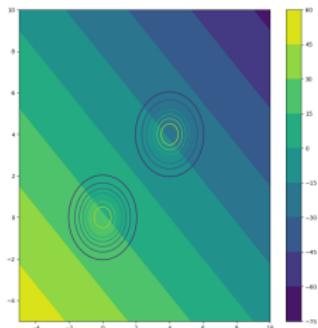
## Few other cases

---



## Few other cases

---



All these cases can be derived analytically! / C'est possible analytiquement  
calculer

# Table of Contents

---

Classification Intro

Generative Classification

## Discriminative Classification

Linear Classifiers

The perceptron algorithm

Logistic Regression

## Non-Linear Classification

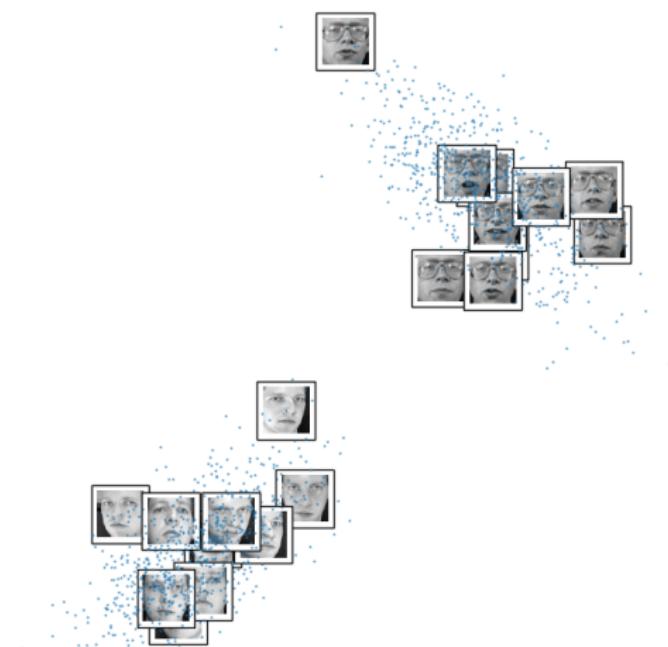
Kernel Logistic Regression

Neural Network Classification

# Discriminative Classification

---

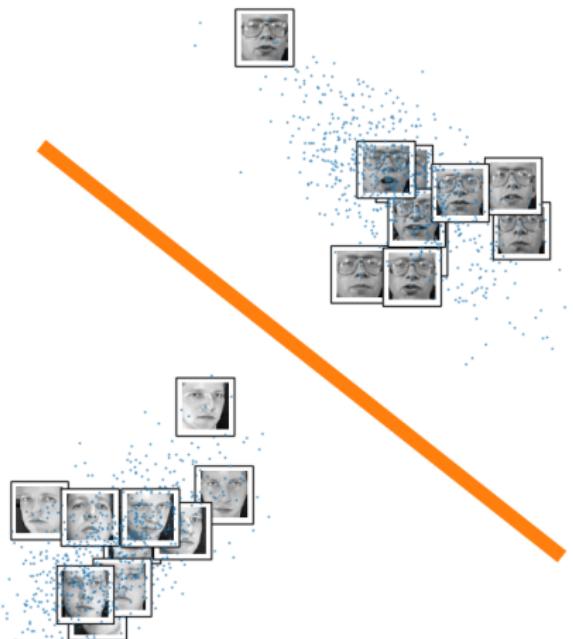
- Discriminative classification directly learns a decision boundary /  
Classification discriminative apprend directement un borne de décision.



# Discriminative Classification

---

- Discriminative classification directly learns a decision boundary / Classification discriminative apprend directement un borne de décision.



# Table of Contents

---

Classification Intro

Generative Classification

## Discriminative Classification

### Linear Classifiers

The perceptron algorithm

Logistic Regression

## Non-Linear Classification

Kernel Logistic Regression

Neural Network Classification

# Linear Classifier

---

- Let's learn a vector  $w \in \mathbb{R}^L$ , such that  $c \approx w^\top x$ . / On va apprendre une vecteur  $w$  pour approximer  $c$ .

# Linear Classifier

---

- Let's learn a vector  $w \in \mathbb{R}^L$ , such that  $c \approx w^\top x$ . / On va apprendre un vecteur  $w$  pour approximer  $c$ .
- More specifically, we want to have (on veut avoir)

$$w^\top x \geq 0 \text{ if } c = 1$$

$$w^\top x \leq 0 \text{ if } c = 0$$

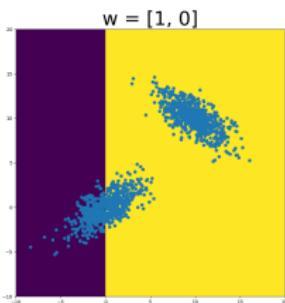
# Linear Classifier

---

- Let's learn a vector  $w \in \mathbb{R}^L$ , such that  $c \approx w^\top x$ . / On va apprendre un vecteur  $w$  pour approximer  $c$ .
- More specifically, we want to have (on veut avoir)

$$w^\top x \geq 0 \text{ if } c = 1$$

$$w^\top x \leq 0 \text{ if } c = 0$$



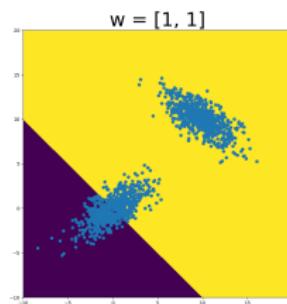
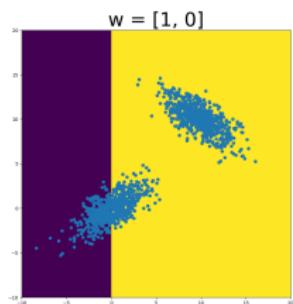
# Linear Classifier

---

- Let's learn a vector  $w \in \mathbb{R}^L$ , such that  $c \approx w^\top x$ . / On va apprendre un vecteur  $w$  pour approximer  $c$ .
- More specifically, we want to have (on veut avoir)

$$w^\top x \geq 0 \text{ if } c = 1$$

$$w^\top x \leq 0 \text{ if } c = 0$$



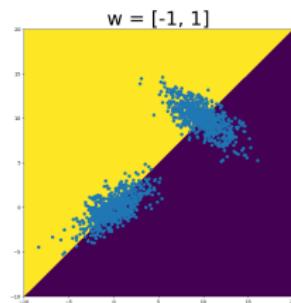
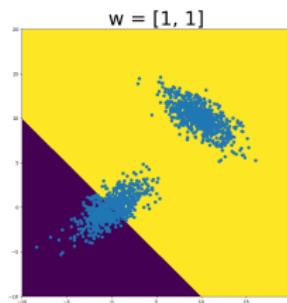
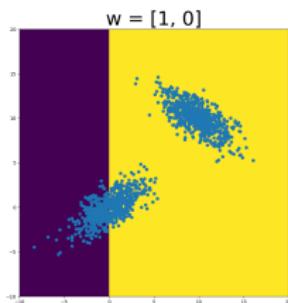
# Linear Classifier

---

- Let's learn a vector  $w \in \mathbb{R}^L$ , such that  $c \approx w^\top x$ . / On va apprendre un vecteur  $w$  pour approximer  $c$ .
- More specifically, we want to have (on veut avoir)

$$w^\top x \geq 0 \text{ if } c = 1$$

$$w^\top x \leq 0 \text{ if } c = 0$$

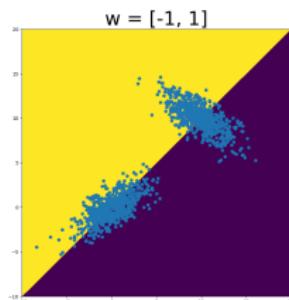
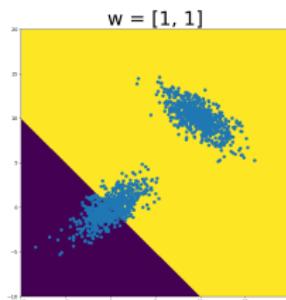
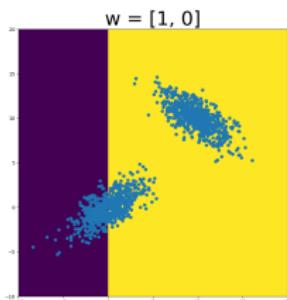


# Linear Classifier

- Let's learn a vector  $w \in \mathbb{R}^L$ , such that  $c \approx w^\top x$ . / On va apprendre une vecteur  $w$  pour approximer  $c$ .
- More specifically, we want to have (on veut avoir)

$$w^\top x \geq 0 \text{ if } c = 1$$

$$w^\top x \leq 0 \text{ if } c = 0$$



- Btw, we will also add a bias term so that  $f(x) = w^\top x + b$  / En passant on va aussi ajouter un biais.

# Table of Contents

---

Classification Intro

Generative Classification

## Discriminative Classification

Linear Classifiers

The perceptron algorithm

Logistic Regression

## Non-Linear Classification

Kernel Logistic Regression

Neural Network Classification

# The perceptron algorithm

---

- But how do we learn  $w$ ? / Comment est-ce qu'on apprend  $w$ ?
- The perceptron algorithm.

# The perceptron algorithm

---

- But how do we learn  $w$ ? / Comment est-ce qu'on apprend  $w$ ?
- The perceptron algorithm.
  - ▶ If  $\text{sgn}(w^\top x_n) = c_n$  do nothing.

# The perceptron algorithm

---

- But how do we learn  $w$ ? / Comment est-ce qu'on apprend  $w$ ?
- The perceptron algorithm.
  - ▶ If  $\text{sgn}(w^\top x_n) = c_n$  do nothing.
  - ▶ If  $\text{sgn}(w^\top x_n) = -c_n$ , then  $w = w + \eta c_n x_n$ . ( $\eta$  is a learning rate)

# The perceptron algorithm

---

- But how do we learn  $w$ ? / Comment est-ce qu'on apprend  $w$ ?
- The perceptron algorithm.
  - ▶ If  $\text{sgn}(w^\top x_n) = c_n$  do nothing.
  - ▶ If  $\text{sgn}(w^\top x_n) = -c_n$ , then  $w = w + \eta c_n x_n$ . ( $\eta$  is a learning rate)
  - ▶ Do these updates until convergence. / On répète jusqu'à ce que converge.

# The perceptron algorithm

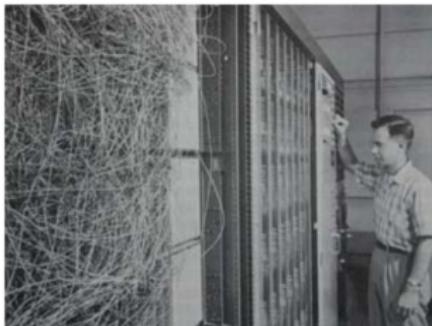
---

- But how do we learn  $w$ ? / Comment est-ce qu'on apprend  $w$ ?
- The perceptron algorithm.
  - ▶ If  $\text{sgn}(w^\top x_n) = c_n$  do nothing.
  - ▶ If  $\text{sgn}(w^\top x_n) = -c_n$ , then  $w = w + \eta c_n x_n$ . ( $\eta$  is a learning rate)
  - ▶ Do these updates until convergence. / On répète jusqu'à ce que converge.
  - ▶ Note that  $c_n \in \{-1, 1\}$ . / Notez que  $c_n \in \{-1, 1\}$ .

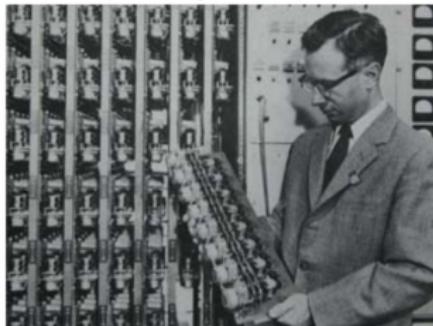
# The perceptron

---

*Feature extraction processor*



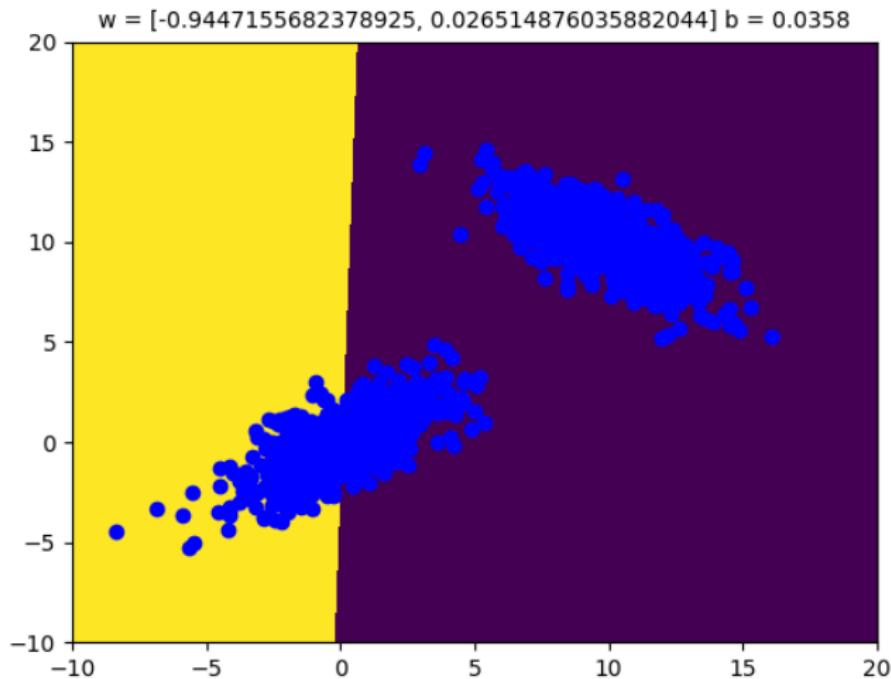
*Perceptron weights  
(motor driven potentiometers)*



Taken from UIUC MLSP class

# Perceptron Epochs

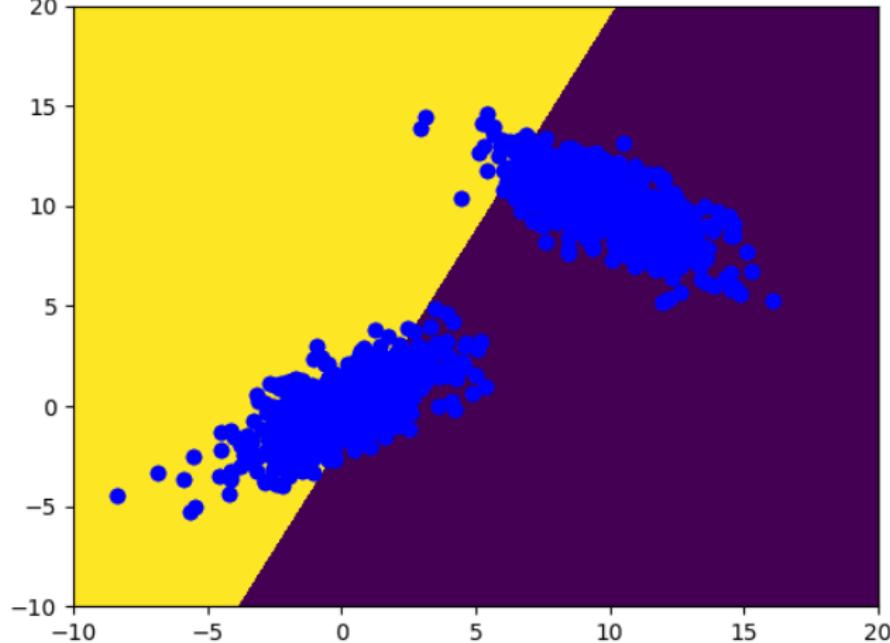
---



# Perceptron Epochs

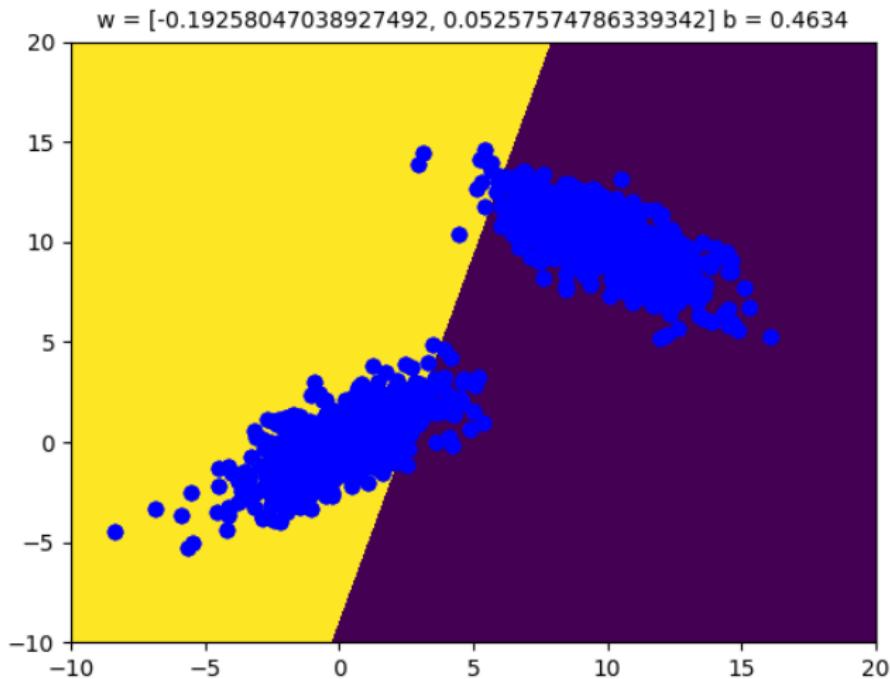
---

$w = [-0.4290201397669718, 0.2016861937502999]$   $b = 0.34500000000000001$



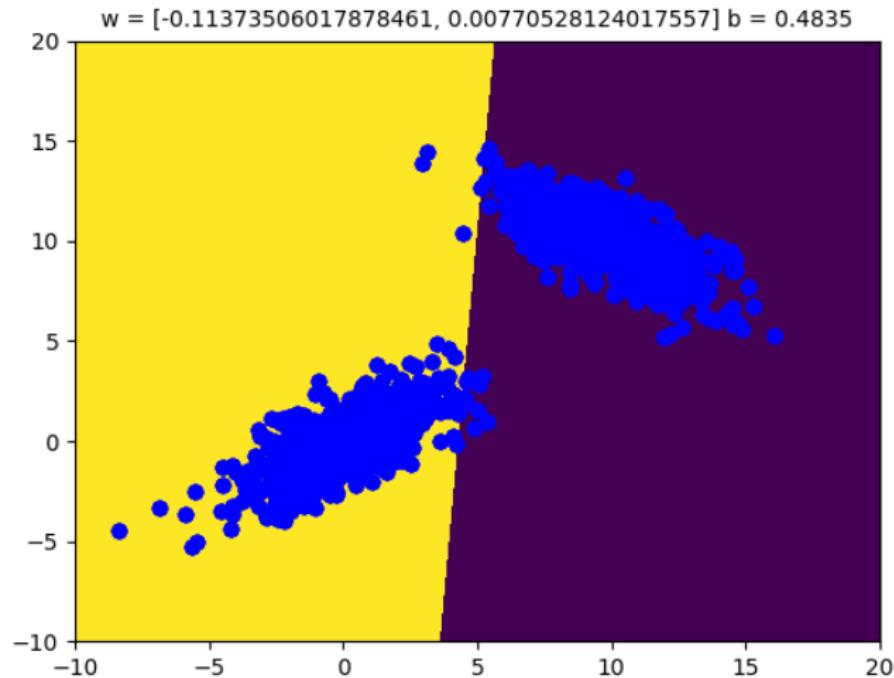
# Perceptron Epochs

---



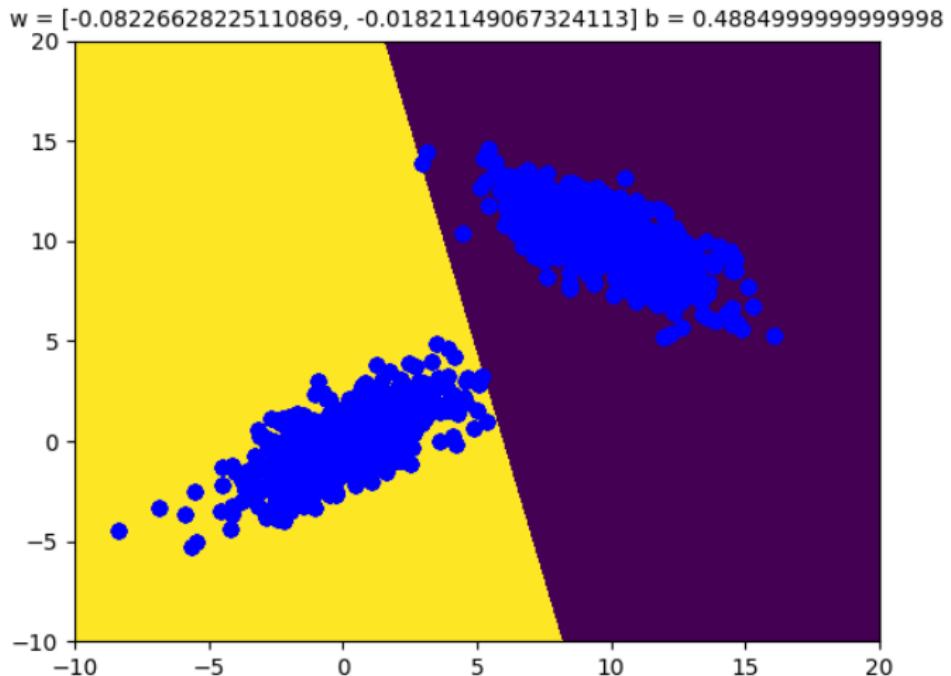
# Perceptron Epochs

---



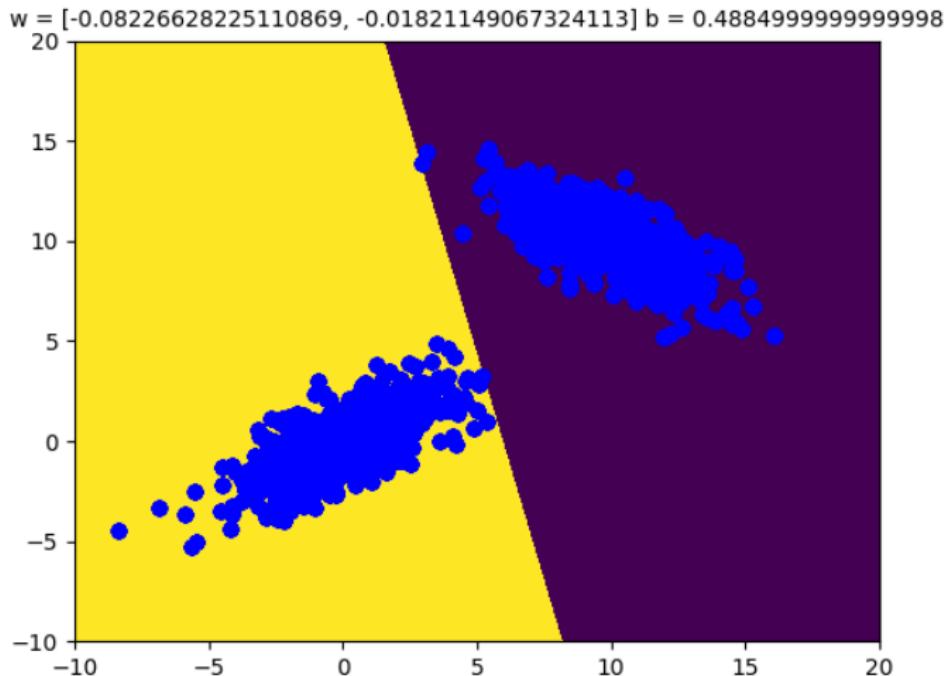
# Perceptron Epochs

---



# Perceptron Epochs

---



# Deriving the perceptron algorithm

---

- Define a cost function / Définissons une fonction de cout:

$$\mathcal{L}(w, b) = - \sum_{\forall c_n \neq \text{sign}(w^\top x_n + b)} c_n (w^\top x_n + b)$$

# Deriving the perceptron algorithm

---

- Define a cost function / Définissons une fonction de cout:

$$\mathcal{L}(w, b) = - \sum_{\forall c_n \neq \text{sign}(w^\top x_n + b)} c_n (w^\top x_n + b)$$

- Do the gradient descent updates: / Faisons les GD updates:

$$w = w - \eta \nabla_w \mathcal{L}(w, b)$$

$$b = b - \eta \nabla_b \mathcal{L}(w, b)$$

# Deriving the perceptron algorithm

---

- Define a cost function / Définissons une fonction de cout:

$$\mathcal{L}(w, b) = - \sum_{\forall c_n \neq \text{sign}(w^\top x_n + b)} c_n (w^\top x_n + b)$$

- Do the gradient descent updates: / Faisons les GD updates:

$$w = w - \eta \nabla_w \mathcal{L}(w, b)$$

$$b = b - \eta \nabla_b \mathcal{L}(w, b)$$

- The gradient: / Le gradient:

$$\nabla_w \mathcal{L}(w) = - \sum_{\forall c_n \neq \text{sign}(w^\top x_n + b)} c_n x_n$$

$$\nabla_b \mathcal{L}(b) = - \sum_{\forall c_n \neq \text{sign}(w^\top x_n + b)} c_n$$

# Deriving the perceptron algorithm

- Define a cost function / Définissons une fonction de cout:

$$\mathcal{L}(w, b) = - \sum_{\forall c_n \neq \text{sign}(w^\top x_n + b)} c_n (w^\top x_n + b)$$

- Do the gradient descent updates: / Faisons les GD updates:

$$w = w - \eta \nabla_w \mathcal{L}(w, b)$$

$$b = b - \eta \nabla_b \mathcal{L}(w, b)$$

- The gradient: / Le gradient:

$$\nabla_w \mathcal{L}(w) = - \sum_{\forall c_n \neq \text{sign}(w^\top x_n + b)} c_n x_n$$

$$\nabla_b \mathcal{L}(b) = - \sum_{\forall c_n \neq \text{sign}(w^\top x_n + b)} c_n$$

- The updates / Les updates: (Same as the perceptron algo! / Les memes updates qu'algo perceptron!)

$$w = w + \eta \sum_{\forall c_n \neq \text{sign}(w^\top x_n + b)} c_n x_n$$

$$b = b + \eta \sum_{\forall c_n \neq \text{sign}(w^\top x_n + b)} c_n$$

## But can we do something better?

---

- Perceptron is basically / Perceptron est simple:

$$f(x) = u(w^\top x + b)$$

where  $u(\cdot)$  is a step function (or  $\text{sign}(\cdot)$  depending on how we define  $c_n$ ) . / où  $u(\cdot)$  est un ‘step function’.

# But can we do something better?

---

- Perceptron is basically / Perceptron est simplement:

$$f(x) = u(w^\top x + b)$$

where  $u(\cdot)$  is a step function (or  $\text{sign}(\cdot)$  depending on how we define  $c_n$ ) . / où  $u(\cdot)$  est un ‘step function’.

- Because of that we can not directly use above within a loss function, and instead we use a modified objective. (Not all items get updated.)
  - ▶ À cause de la step function on ne peut pas utiliser le modèle directement dans le fonctionnement de cout.

# But can we do something better?

---

- Perceptron is basically / Perceptron est simplement:

$$f(x) = u(w^\top x + b)$$

where  $u(\cdot)$  is a step function (or  $\text{sign}(\cdot)$  depending on how we define  $c_n$ ) . / où  $u(\cdot)$  est un ‘step function’.

- Because of that we can not directly use above within a loss function, and instead we use a modified objective. (Not all items get updated.)
  - ▶ À cause de la step function on ne peut pas utiliser le modèle directement dans le fonction de cout.
- What if we want to have smooth, differentiable transitions? / Et si on veut avoir des transitions plus smooth?

# Table of Contents

---

Classification Intro

Generative Classification

## Discriminative Classification

Linear Classifiers

The perceptron algorithm

Logistic Regression

Non-Linear Classification

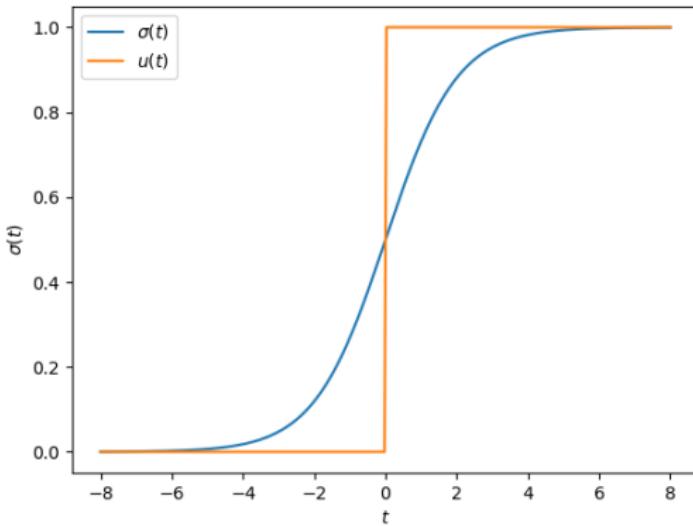
Kernel Logistic Regression

Neural Network Classification

# The logistic function

---

- We can use the logistic function  $\sigma(t) = \frac{1}{1+e^{-t}}$ . / On peut utiliser la fonction logistique.



# Logistic regression

---

- Set the estimator as  $f(w, b) = \sigma(w^\top x + b)$ .

# Logistic regression

---

- Set the estimator as  $f(w, b) = \sigma(w^\top x + b)$ .
- Same neural network, different activation function. / La même réseau de neurones, différente fonction d'activation.

# Logistic regression

---

- Set the estimator as  $f(w, b) = \sigma(w^\top x + b)$ .
- Same neural network, different activation function. / La même réseau de neurones, différente fonction d'activation.
- What will be the loss function? / Quelle sera la fonction de coût?

# Logistic regression

---

- Set the estimator as  $f(w, b) = \sigma(w^\top x + b)$ .
- Same neural network, different activation function. / La même réseau de neurones, différent fonctionne d'activation.
- What will be the loss function? / Quelle sera la fonctionne de coût?
- How about the negative Bernoulli log-likelihood?

# Bernoulli Distribution

---

- $\mathcal{BE}(y; \pi) = \pi^y(1 - \pi)^{1-y}, y \in \{0, 1\}.$
- Let's take the log / Prenons le logarithme

$$\log \mathcal{BE}(y; \pi) = y \log \pi + (1 - y) \log(1 - \pi)$$

- We will parametrize the Bernoulli distribution with  $\pi = w^\top x + b/$   
On est en train de paramétriser la distribution Bernoulli.

# Logistic regression loss function

---

- Training loss / La fonction de cout pour l'entraînement:

$$\begin{aligned}\mathcal{L}(w, b) &= \sum_n (-y_n \log \pi_w(x_n) + (1 - y_n) \log(1 - \pi_w(x_n))) \\ &= \sum_n \left( -y_n \log (\sigma(w^\top x_n + b)) - (1 - y_n) \log (1 - \sigma(w^\top x_n + b)) \right)\end{aligned}$$

- Let's calculate the gradient with respect to  $w$ . / Calculons le gradient par rapport à  $w$ .

# Logistic regression loss function

---

- Training loss / La fonction de cout pour l'entraînement:

$$\begin{aligned}\mathcal{L}(w, b) &= \sum_n (-y_n \log \pi_w(x_n) + (1 - y_n) \log(1 - \pi_w(x_n))) \\ &= \sum_n \left( -y_n \log (\sigma(w^\top x_n + b)) - (1 - y_n) \log (1 - \sigma(w^\top x_n + b)) \right)\end{aligned}$$

- Let's calculate the gradient with respect to  $w$ . / Calculons le gradient par rapport à  $w$ .
- The chain rule:  $\frac{df(g(x))}{dx} = \frac{df(g(x))}{dg(x)} \cdot \frac{dg(x)}{dx} = f'(g(x)) \cdot g'(x)$

# Logistic regression loss function

---

- Training loss / La fonction de cout pour l'entraînement:

$$\begin{aligned}\mathcal{L}(w, b) &= \sum_n (-y_n \log \pi_w(x_n) + (1 - y_n) \log(1 - \pi_w(x_n))) \\ &= \sum_n \left( -y_n \log (\sigma(w^\top x_n + b)) - (1 - y_n) \log (1 - \sigma(w^\top x_n + b)) \right)\end{aligned}$$

- Let's calculate the gradient with respect to  $w$ . / Calculons le gradient par rapport à  $w$ .
- The chain rule:  $\frac{df(g(x))}{dx} = \frac{df(g(x))}{dg(x)} \cdot \frac{dg(x)}{dx} = f'(g(x)) \cdot g'(x)$
- Few more things:  $\frac{\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$ ,  $\frac{\log x}{dx} = 1/x$ .

# The gradient wrt $w$

---

- Just do it! / Faisons-le!

$$\begin{aligned}\frac{d\mathcal{L}(w, b)}{dw} &= - \sum_n y_n \frac{\sigma(w^\top x_n + b)}{\sigma(w^\top x_n + b)} (1 - \sigma(w^\top x_n + b)) x_n \\ &\quad + \sum_n (1 - y_n) \frac{1 - \sigma(w^\top x_n + b)}{1 - \sigma(w^\top x_n + b)} \sigma(w^\top x_n + b) x_n\end{aligned}$$

# The gradient wrt $w$

---

- Just do it! / Faisons-le!

$$\begin{aligned}\frac{d\mathcal{L}(w, b)}{dw} &= - \sum_n y_n \frac{\sigma(w^\top x_n + b)}{\sigma(w^\top x_n + b)} (1 - \sigma(w^\top x_n + b)) x_n \\ &\quad + \sum_n (1 - y_n) \frac{1 - \sigma(w^\top x_n + b)}{1 - \sigma(w^\top x_n + b)} \sigma(w^\top x_n + b) x_n \\ &= - \sum_n y_n \underbrace{\frac{\sigma(w^\top x_n + b)}{\sigma(w^\top x_n + b)}}_{\cancel{\sigma(w^\top x_n + b)}} (1 - \sigma(w^\top x_n + b)) x_n \\ &\quad + \sum_n (1 - y_n) \underbrace{\frac{1 - \sigma(w^\top x_n + b)}{1 - \sigma(w^\top x_n + b)}}_{\cancel{1 - \sigma(w^\top x_n + b)}} \sigma(w^\top x_n + b) x_n\end{aligned}$$

# The gradient wrt $w$

---

- Just do it! / Faisons-le!

$$\begin{aligned}\frac{d\mathcal{L}(w, b)}{dw} &= - \sum_n y_n \frac{\sigma(w^\top x_n + b)}{\sigma(w^\top x_n + b)} (1 - \sigma(w^\top x_n + b)) x_n \\ &\quad + \sum_n (1 - y_n) \frac{1 - \sigma(w^\top x_n + b)}{1 - \sigma(w^\top x_n + b)} \sigma(w^\top x_n + b) x_n \\ &= - \sum_n y_n \cancel{\frac{\sigma(w^\top x_n + b)}{\sigma(w^\top x_n + b)}} (1 - \sigma(w^\top x_n + b)) x_n \\ &\quad + \sum_n (1 - y_n) \cancel{\frac{1 - \sigma(w^\top x_n + b)}{1 - \sigma(w^\top x_n + b)}} \sigma(w^\top x_n + b) x_n \\ &= \sum_n (\sigma(w^\top x_n + b) - y_n) x_n\end{aligned}$$

- Makes a lot of sense! / Ça fait du sens!

# The gradient wrt $w$

---

- Just do it! / Faisons-le!

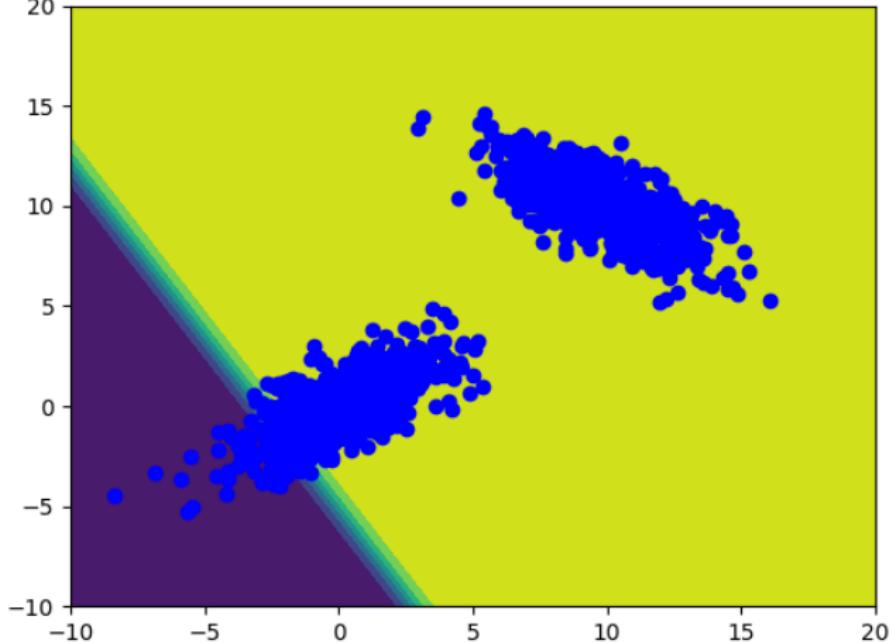
$$\begin{aligned}\frac{d\mathcal{L}(w, b)}{dw} &= - \sum_n y_n \frac{\sigma(w^\top x_n + b)}{\sigma(w^\top x_n + b)} (1 - \sigma(w^\top x_n + b)) x_n \\ &\quad + \sum_n (1 - y_n) \frac{1 - \sigma(w^\top x_n + b)}{1 - \sigma(w^\top x_n + b)} \sigma(w^\top x_n + b) x_n \\ &= - \sum_n y_n \cancel{\frac{\sigma(w^\top x_n + b)}{\sigma(w^\top x_n + b)}} (1 - \sigma(w^\top x_n + b)) x_n \\ &\quad + \sum_n (1 - y_n) \cancel{\frac{1 - \sigma(w^\top x_n + b)}{1 - \sigma(w^\top x_n + b)}} \sigma(w^\top x_n + b) x_n \\ &= \sum_n (\sigma(w^\top x_n + b) - y_n) x_n\end{aligned}$$

- Makes a lot of sense! / Ça fait du sens!
- torch will automatically do this for us. However, one needs to do this at least once in their life!
  - ▶ torch fait ça automatiquement mais on doit faire cette exercice au moins une fois dans notre vie!

# Logistic Regression Epochs

---

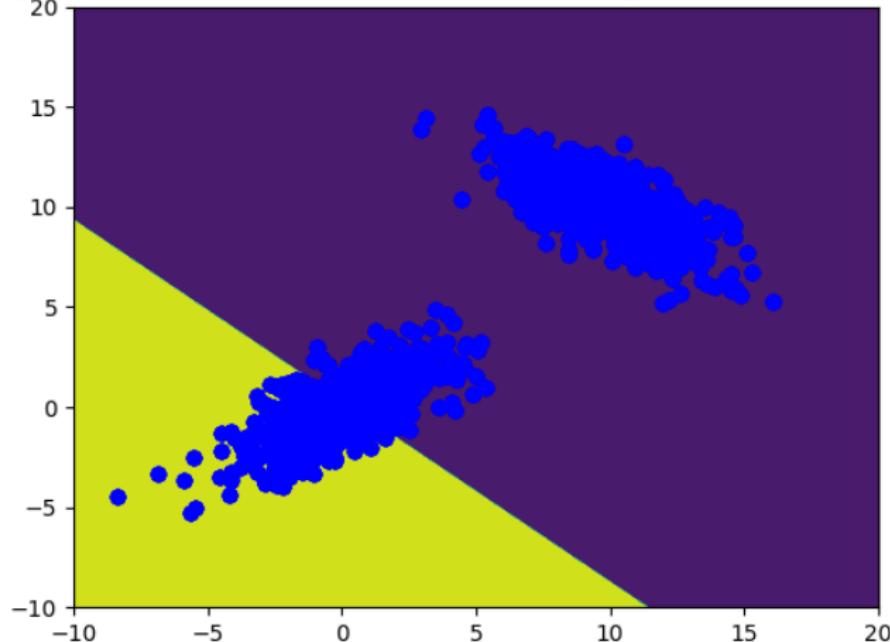
$w = [2.774843454360962, 1.6030927896499634]$   $b = 4.142136573791504$



# Logistic Regression Epochs

---

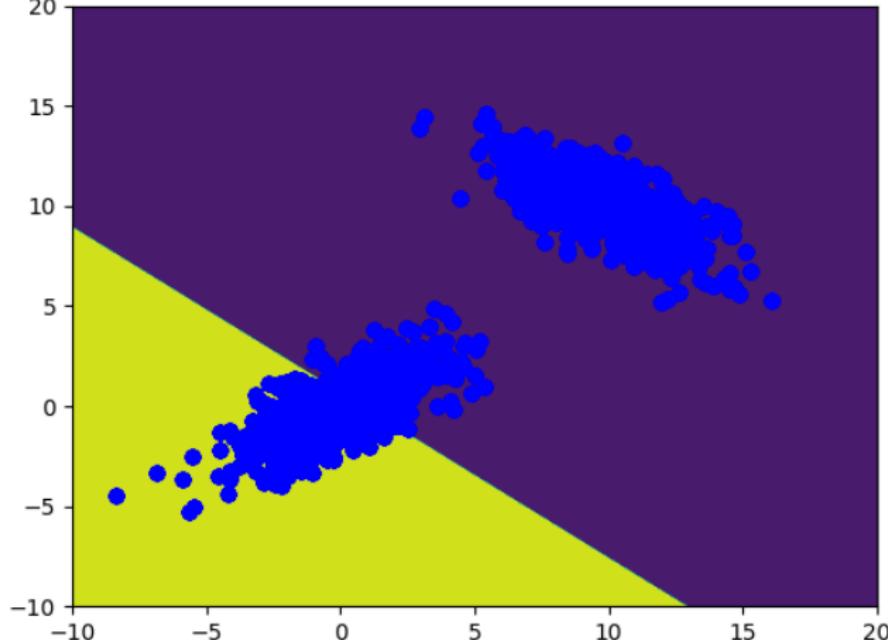
$w = [-64.31231689453125, -71.11492156982422]$   $b = 10.29466533660888$



# Logistic Regression Epochs

---

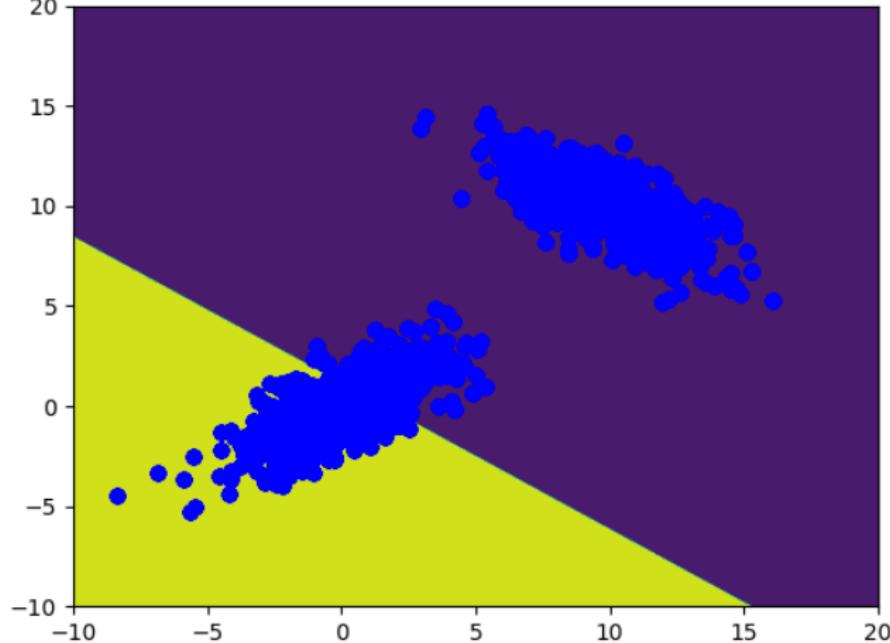
$w = [-57.4975700378418, -69.5396728515625] b = 22.519577026367188$



# Logistic Regression Epochs

---

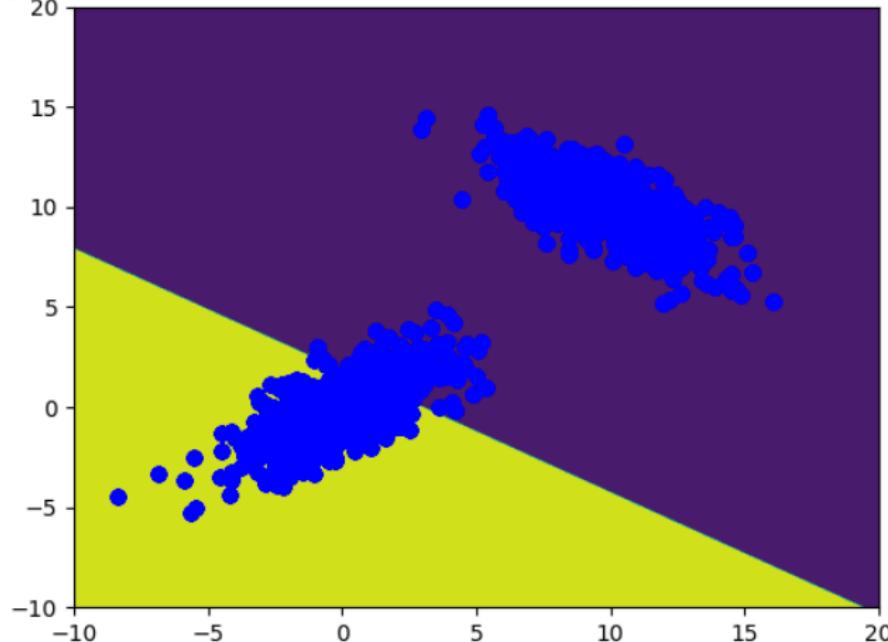
$w = [-46.69367980957031, -63.91728591918945]$   $b = 35.9787139892578$



# Logistic Regression Epochs

---

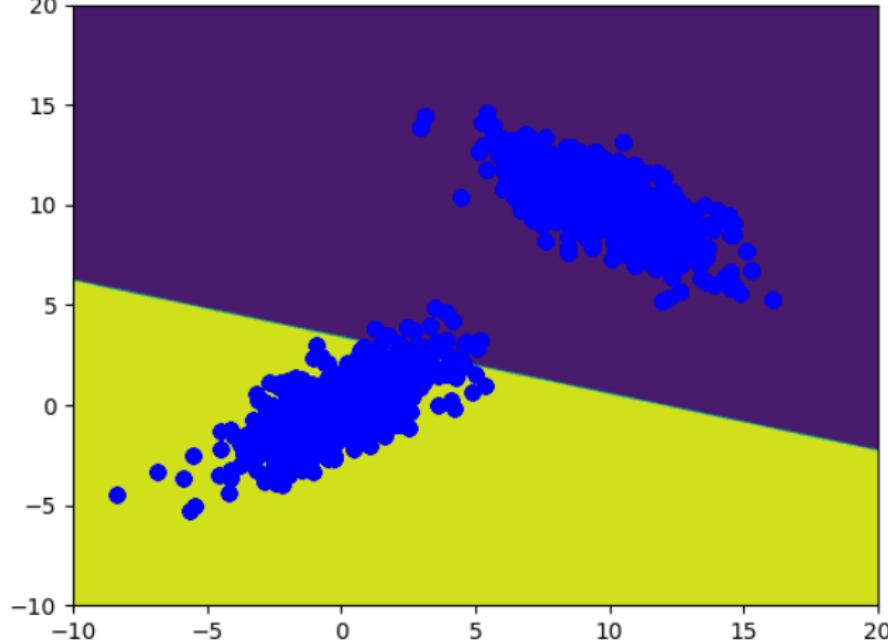
$w = [-32.589866638183594, -53.49893569946289]$   $b = 48.29494476318359$



# Logistic Regression Epochs

---

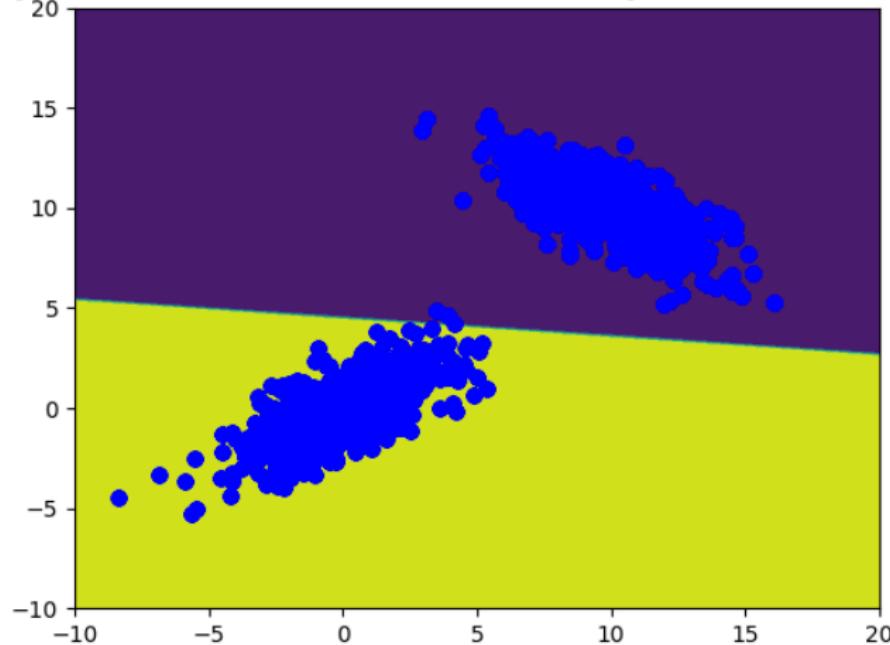
$w = [-9.948322296142578, -35.09688186645508]$   $b = 59.3541374206543$



# Logistic Regression Epochs

---

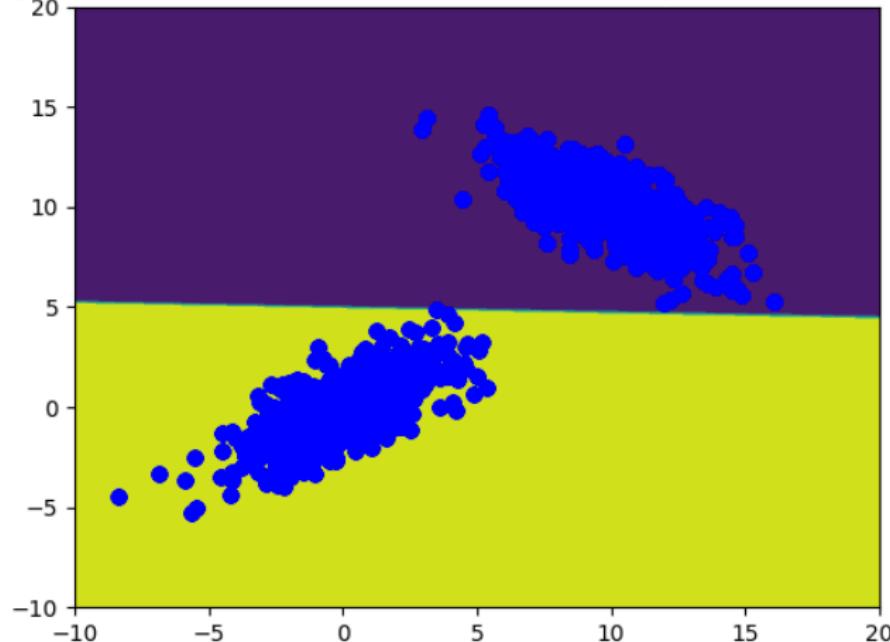
$w = [-2.5054750442504883, -27.3735294342041]$   $b = 61.56295776367187$



# Logistic Regression Epochs

---

$w = [-0.6339982151985168, -24.971982955932617]$   $b = 62.072887420654$



# Generalizing to multiway classification

---

- Define (a new network!) / On définit une nouvelle réseau

$$\pi_k := \exp(w_k^\top x + b_k) / \sum_l \exp(w_l^\top x + b_l)$$

# Generalizing to multiway classification

---

- Define (a new network!) / On définit une nouvelle réseau

$$\pi_k := \exp(w_k^\top x + b_k) / \sum_l \exp(w_l^\top x + b_l)$$

- The loss function is now going to be the log-likelihood of discrete distribution / La fonction de cout sera maintenant le log-likelihood de la distribution discrète.

$$\mathcal{L}(W, b) = -\log \left( \prod_n \text{Discrete}(y_n; \pi_{1:K}) \right)$$

# Generalizing to multiway classification

---

- Define (a new network!) / On définit une nouvelle réseau

$$\pi_k := \exp(w_k^\top x + b_k) / \sum_l \exp(w_l^\top x + b_l)$$

- The loss function is now going to be the log-likelihood of discrete distribution / La fonctionne de cout sera maintenant le log-likelihood de la distribution discrète.

$$\begin{aligned}\mathcal{L}(W, b) &= -\log \left( \prod_n \text{Discrete}(y_n; \pi_{1:K}) \right) \\ &= -\log \left( \prod_n \prod_k \pi_k^{[k=y_n]} \right)\end{aligned}$$

# Generalizing to multiway classification

---

- Define (a new network!) / On définit une nouvelle réseau

$$\pi_k := \exp(w_k^\top x + b_k) / \sum_l \exp(w_l^\top x + b_l)$$

- The loss function is now going to be the log-likelihood of discrete distribution / La fonction de cout sera maintenant le log-likelihood de la distribution discrète.

$$\begin{aligned}\mathcal{L}(W, b) &= -\log \left( \prod_n \text{Discrete}(y_n; \pi_{1:K}) \right) \\ &= -\log \left( \prod_n \prod_k \pi_k^{[k=y_n]} \right) \\ &= \sum_n \sum_k [y_n = k] \log \pi_k\end{aligned}$$

# Generalizing to multiway classification

---

- Define (a new network!) / On définit une nouvelle réseau

$$\pi_k := \exp(w_k^\top x + b_k) / \sum_l \exp(w_l^\top x + b_l)$$

- The loss function is now going to be the log-likelihood of discrete distribution / La fonction de cout sera maintenant le log-likelihood de la distribution discrète.

$$\begin{aligned}\mathcal{L}(W, b) &= -\log \left( \prod_n \text{Discrete}(y_n; \pi_{1:K}) \right) \\ &= -\log \left( \prod_n \prod_k \pi_k^{[k=y_n]} \right) \\ &= \sum_n \sum_k [y_n = k] \log \pi_k \\ &= \sum_n \sum_k [y_n = k] \left( w_k^\top x + b_k - \log \sum_l \exp(w_l^\top x + b_l) \right)\end{aligned}$$

# Table of Contents

---

Classification Intro

Generative Classification

Discriminative Classification

Linear Classifiers

The perceptron algorithm

Logistic Regression

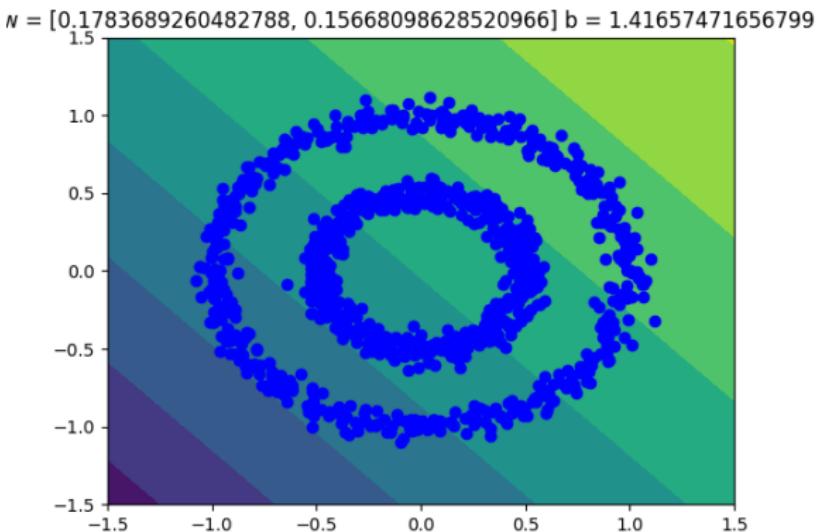
Non-Linear Classification

Kernel Logistic Regression

Neural Network Classification

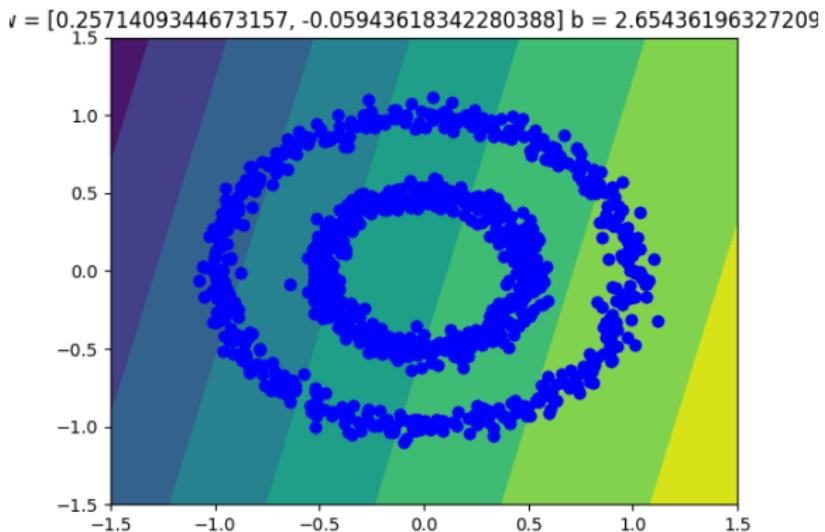
# Non-Linear Classification

- What if we have something like this? / Qu'est-ce qu'on fait si on a ça?



# Non-Linear Classification

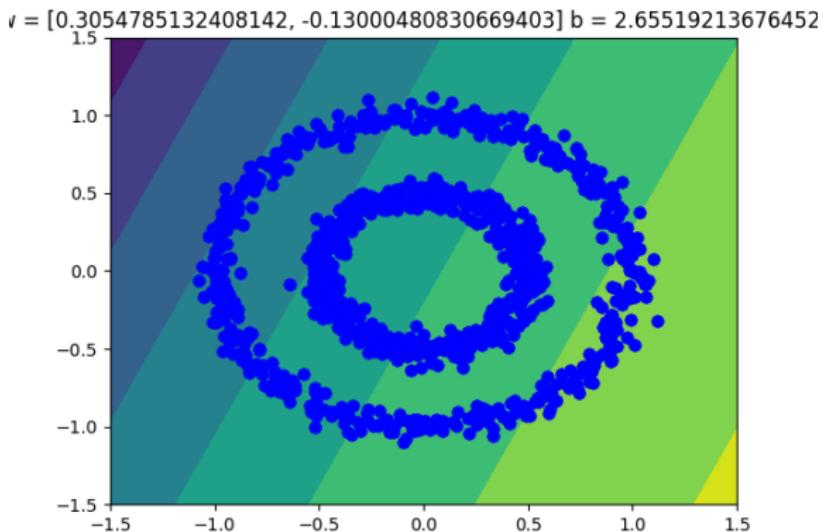
- What if we have something like this? / Qu'est-ce qu'on fait si on a ça?



# Non-Linear Classification

---

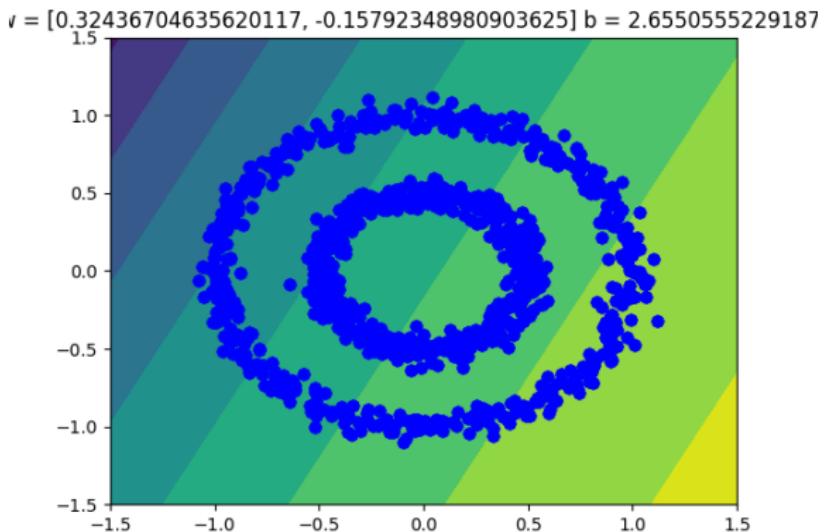
- What if we have something like this? / Qu'est-ce qu'on fait si on a ça?



# Non-Linear Classification

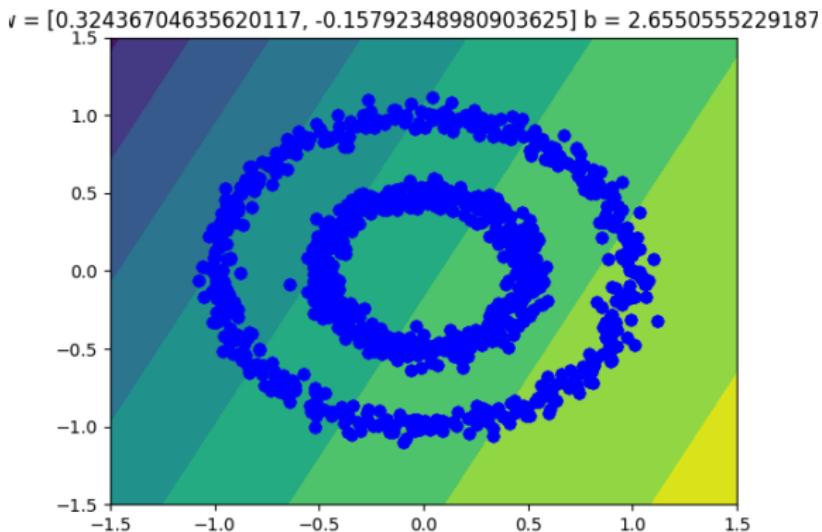
---

- What if we have something like this? / Qu'est-ce qu'on fait si on a ça?



# Non-Linear Classification

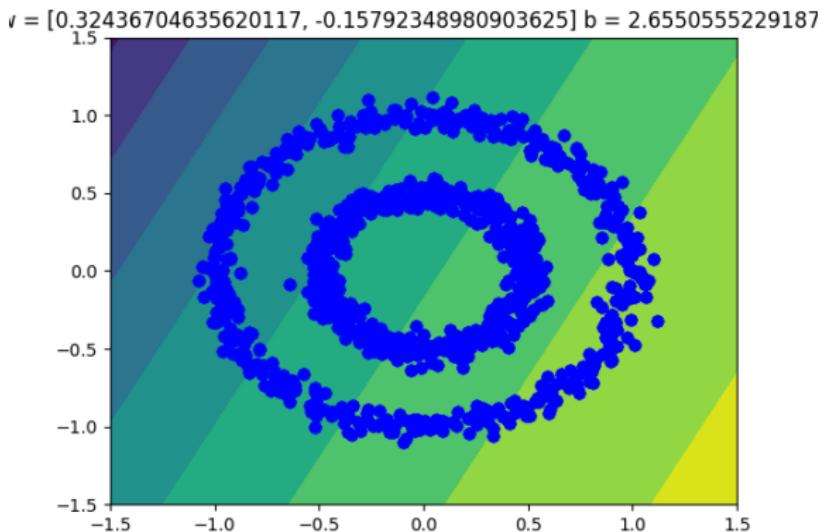
- What if we have something like this? / Qu'est-ce qu'on fait si on a ça?



- Linear classifier is not enough! / Classificateur linéaire n'est pas suffisante!

# Non-Linear Classification

- What if we have something like this? / Qu'est-ce qu'on fait si on a ça?



- Linear classifier is not enough! / Classificateur linéaire n'est pas suffisante!
- What can we do? / Qu'est-ce qu'on peut faire?

# The Kernel Trick

---

- We can extend our linear classifier  $y = w^\top x \rightarrow y = w^\top \phi(x)$ . We introduce a feature transformation  $\phi(\cdot)$ .
  - ▶ On améliore notre classificateur linéaire en introduisant une transformation de feature  $\phi(\cdot)$ .

# The Kernel Trick

---

- We can extend our linear classifier  $y = w^\top x \rightarrow y = w^\top \phi(x)$ . We introduce a feature transformation  $\phi(\cdot)$ .
  - ▶ On améliore notre classificateur linéaire en introduisant une transformation de feature  $\phi(\cdot)$ .
- As we talked about it last week for Kernel-PCA, introducing  $\phi(\cdot)$  is not possible if  $\phi$  is large dimensional. / Comme on en a parlé  $\phi(\cdot)$  n'est pas réalisable si on map a une large nombre de dimensions.

# The Kernel Trick

---

- We can extend our linear classifier  $y = w^\top x \rightarrow y = w^\top \phi(x)$ . We introduce a feature transformation  $\phi(\cdot)$ .
  - ▶ On améliore notre classificateur linéaire en introduisant une transformation de feature  $\phi(\cdot)$ .
- As we talked about it last week for Kernel-PCA, introducing  $\phi(\cdot)$  is not possible if  $\phi$  is large dimensional. / Comme on en a parlé  $\phi(\cdot)$  n'est pas réalisable si on map a une grande nombre de dimensions.
- **Kernel Trick:** We can substitute/ On substitue  $w = \sum_n a_n \phi(x_n)$ , so

$$\begin{aligned}f(x) &= \sigma \left( \sum_n a_n \phi(x)^\top \phi(x_n) \right) \\&= \sigma \left( \sum_n a_n k(x, x_n) \right)\end{aligned}$$

# The Kernel Trick

---

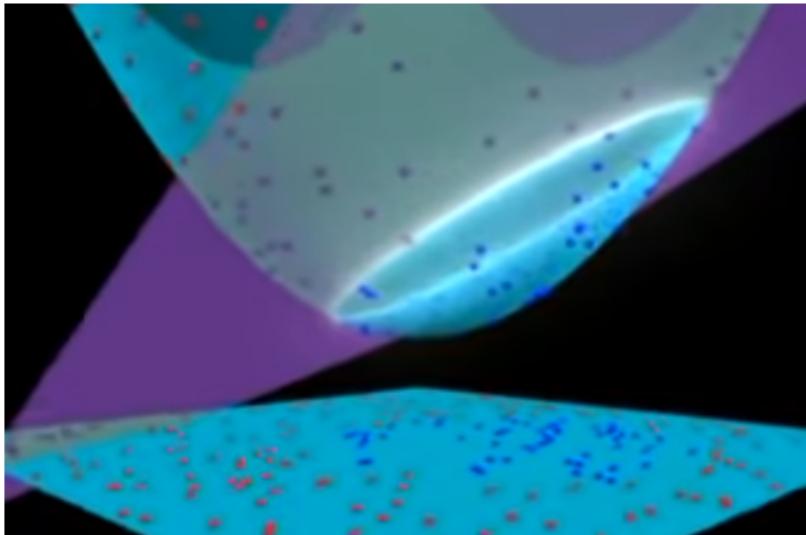
- We can extend our linear classifier  $y = w^\top x \rightarrow y = w^\top \phi(x)$ . We introduce a feature transformation  $\phi(\cdot)$ .
  - ▶ On améliore notre classificateur linéaire en introduisant une transformation de feature  $\phi(\cdot)$ .
- As we talked about it last week for Kernel-PCA, introducing  $\phi(\cdot)$  is not possible if  $\phi$  is large dimensional. / Comme on en a parlé  $\phi(\cdot)$  n'est pas réalisable si on map a une grande nombre de dimensions.
- **Kernel Trick:** We can substitute/ On substitue  $w = \sum_n a_n \phi(x_n)$ , so

$$\begin{aligned}f(x) &= \sigma \left( \sum_n a_n \phi(x)^\top \phi(x_n) \right) \\&= \sigma \left( \sum_n a_n k(x, x_n) \right)\end{aligned}$$

- We can incorporate this in many places. But let's look at injecting this idea in logistic regression.
  - ▶ On peut incorporer cette idée dans différents endroits. Mais d'abord essayons d'injecter cette idée dans la régression logistique.

# Motivating the Kernel Trick

---



Let's watch!

# Table of Contents

---

Classification Intro

Generative Classification

Discriminative Classification

Linear Classifiers

The perceptron algorithm

Logistic Regression

Non-Linear Classification

Kernel Logistic Regression

Neural Network Classification

# Kernel Logistic Regression

---

- The coin bias is now estimated with the kernel-trick / Le biais est maintenant estimé avec le kernel trick.

$$\pi = \sigma \left( \sum_n a_n k(x, x_n) \right)$$

- Then the loss becomes, / le fonctionne de cout devient,

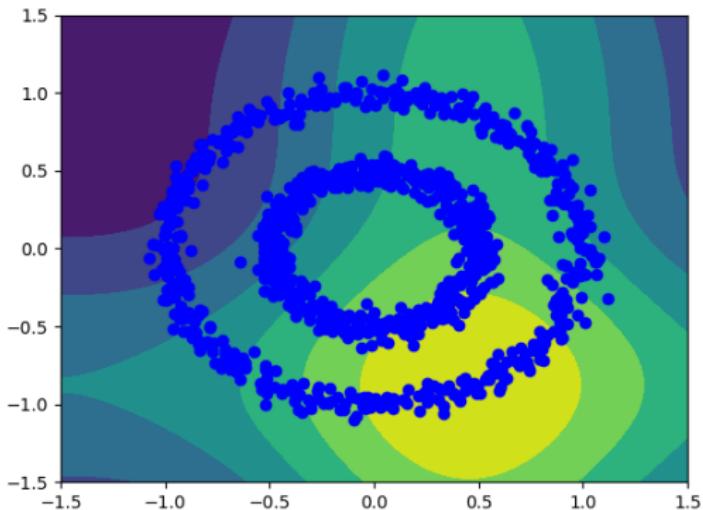
$$\begin{aligned} \mathcal{L}(a) = & \sum_n \left( -y_n \log \left( \sigma \left( \sum_j a_j k(x_n, x_j) \right) \right) \right. \\ & \left. - (1 - y_n) \log \left( 1 - \sigma \left( \sum_j a_j k(x_n, x_j) \right) \right) \right) \end{aligned}$$

- Kernel functions are chosen from options such as RBF Kernel, Polynomial Kernel,.. / On choisit le kernel entre les options qu'on est habitué.

$$k_{\text{rbf}}(x, x_n) = \exp(-\gamma \|x - x_n\|_2)$$

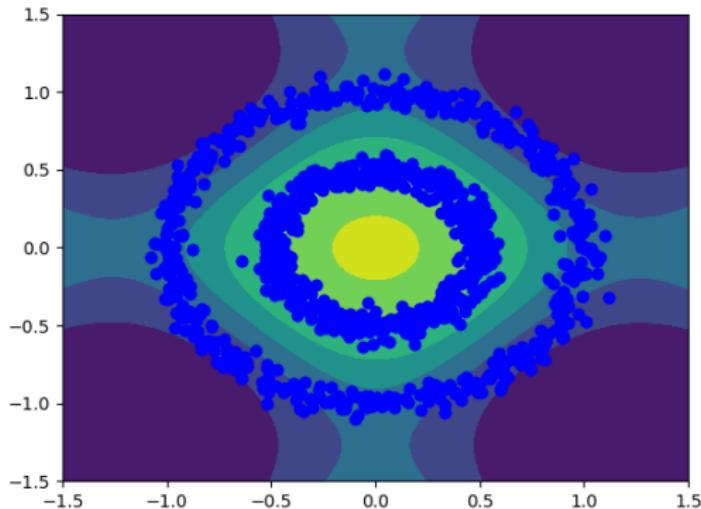
# Kernel Logistic Regression Learning Steps

---



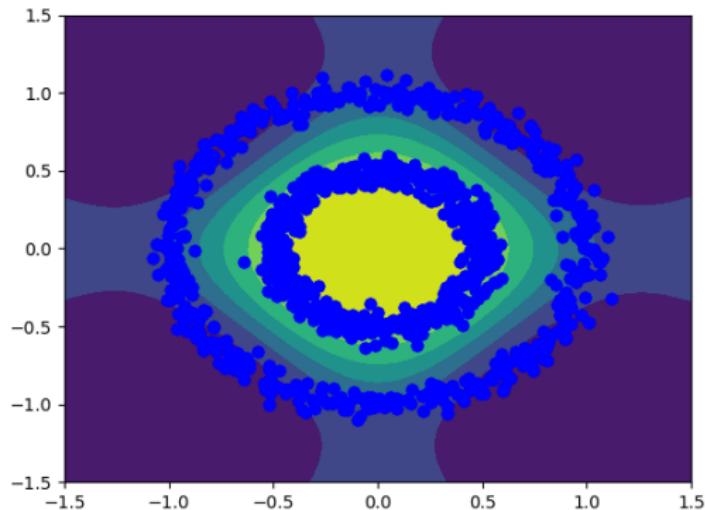
# Kernel Logistic Regression Learning Steps

---



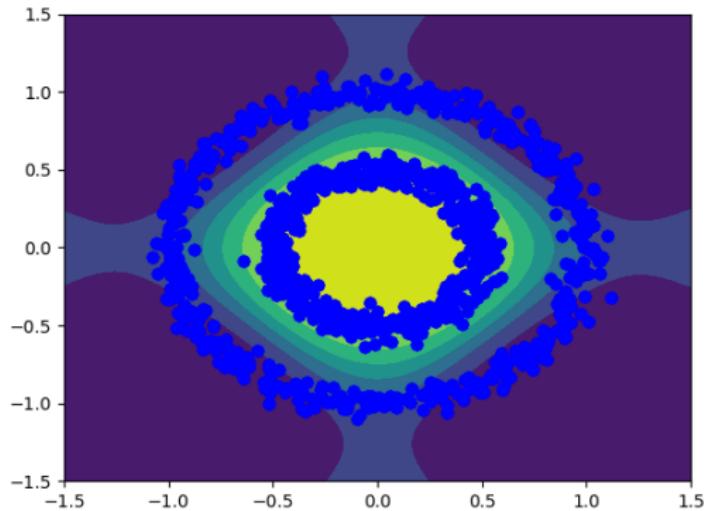
# Kernel Logistic Regression Learning Steps

---



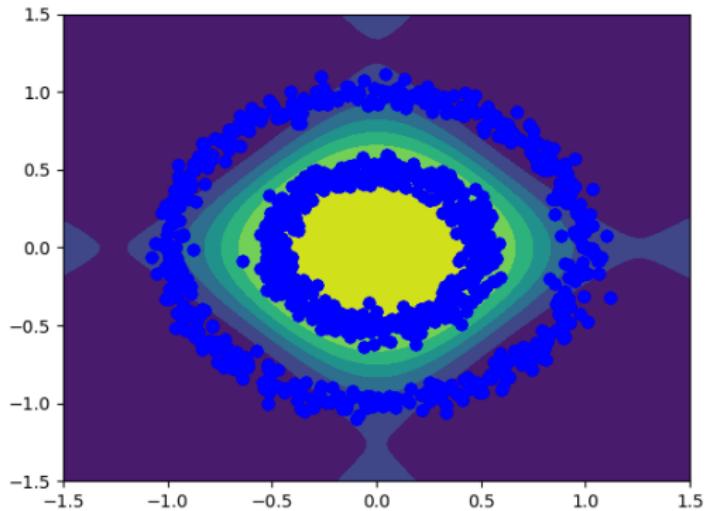
# Kernel Logistic Regression Learning Steps

---



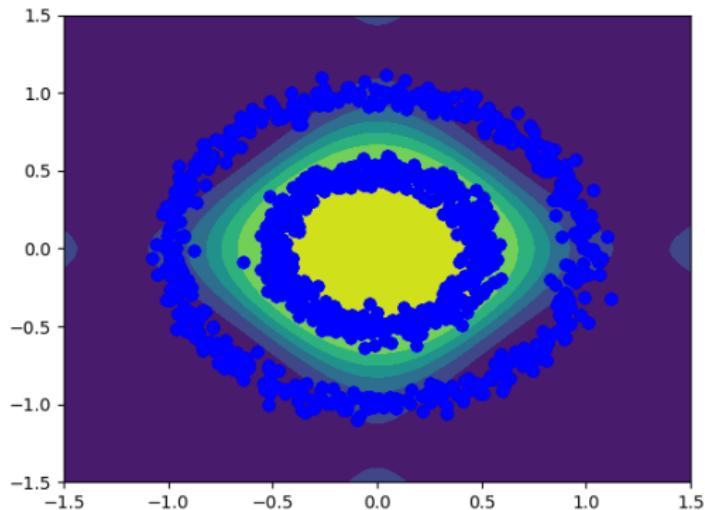
# Kernel Logistic Regression Learning Steps

---



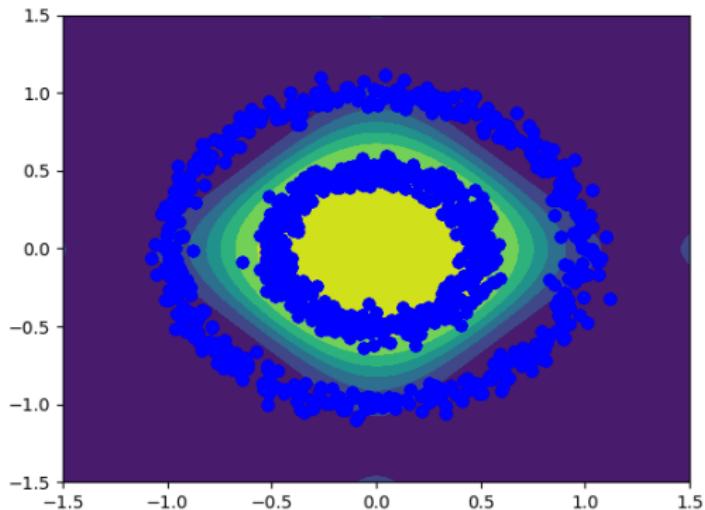
# Kernel Logistic Regression Learning Steps

---



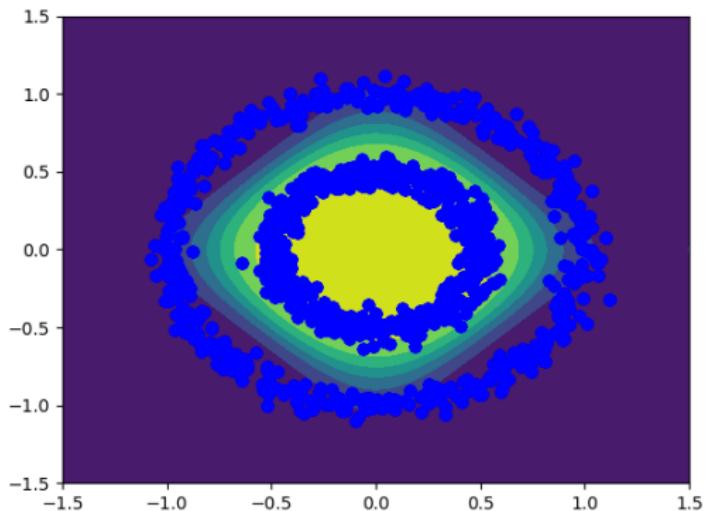
# Kernel Logistic Regression Learning Steps

---



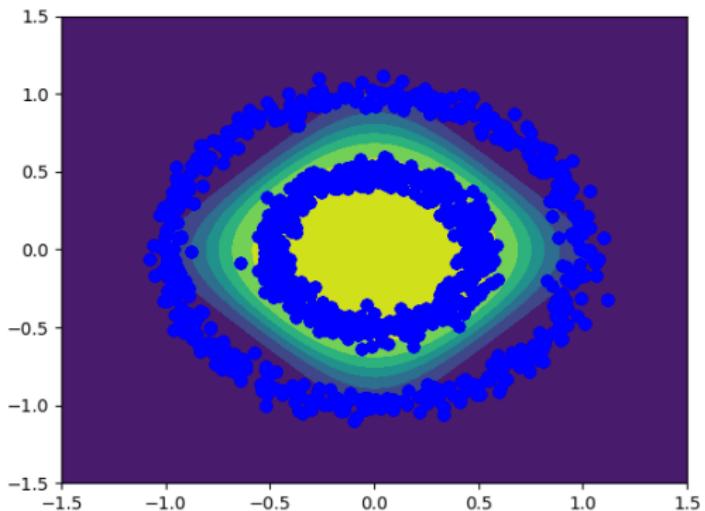
# Kernel Logistic Regression Learning Steps

---



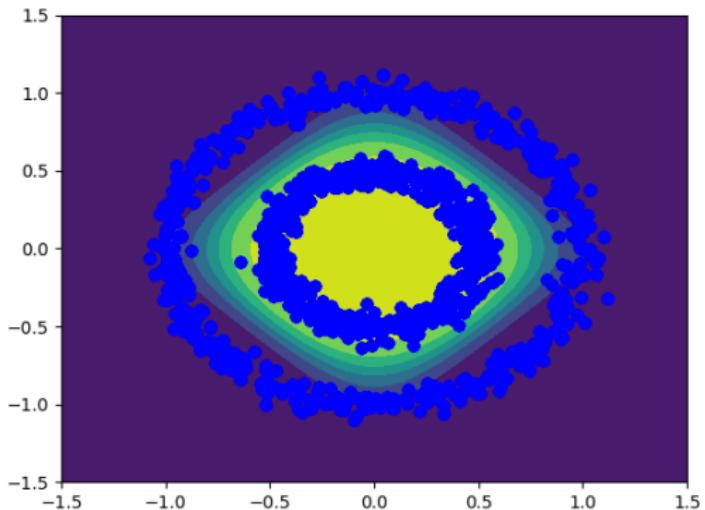
# Kernel Logistic Regression Learning Steps

---



# Kernel Logistic Regression Learning Steps

---



# Table of Contents

---

Classification Intro

Generative Classification

Discriminative Classification

Linear Classifiers

The perceptron algorithm

Logistic Regression

Non-Linear Classification

Kernel Logistic Regression

Neural Network Classification

# Neural Network Classification

---

- The bias parameter can be estimated with a general function  $f(x)$ .

$$\pi = f(x)$$

$$= \sigma(w_M^\top h(W_{M-1}^\top h(W_{M-2}^\top (\dots h(W_1^\top x))))))$$

- The general function maps  $x$  from  $L$  dimensions to  $K_1, K_2, \dots, K_{M-1}$  dimensions. The last layer  $w_M$  is a vector and maps to 1 output dimension. / Chaque couche map à une dimensionnalité différent  $K_l$ .

# Neural Network Classification

---

- The bias parameter can be estimated with a general function  $f(x)$ .

$$\pi = f(x)$$

$$= \sigma(w_M^\top h(W_{M-1}^\top h(W_{M-2}^\top (\dots h(W_1^\top x))))))$$

- The general function maps  $x$  from  $L$  dimensions to  $K_1, K_2, \dots, K_{M-1}$  dimensions. The last layer  $w_M$  is a vector and maps to 1 output dimension. / Chaque couche map à une dimensionnalité différent  $K_l$ .
- The activation functions  $h(\cdot)$  are element-wise functions. Typical examples are ReLU, tanh, softplus, leaky ReLU, ...

# Neural Network Classification

---

- The bias parameter can be estimated with a general function  $f(x)$ .

$$\begin{aligned}\pi &= f(x) \\ &= \sigma(w_M^\top h(W_{M-1}^\top h(W_{M-2}^\top (\dots h(W_1^\top x))))))\end{aligned}$$

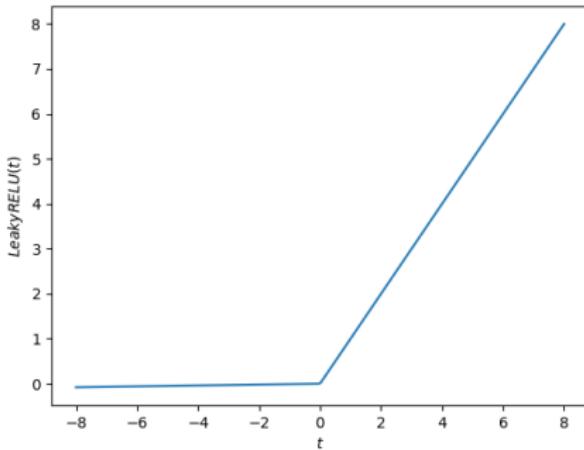
- The general function maps  $x$  from  $L$  dimensions to  $K_1, K_2, \dots, K_{M-1}$  dimensions. The last layer  $w_M$  is a vector and maps to 1 output dimension. / Chaque couche map à une dimensionnalité différent  $K_l$ .
- The activation functions  $h(\cdot)$  are element-wise functions. Typical examples are ReLU, tanh, softplus, leaky ReLU, ...
- The loss function becomes / La fonction de cout devient:

$$\mathcal{L}(\theta) = \sum_n \left( -y_n \log \left( \sigma(f_\theta(x_n)) \right) - (1 - y_n) \log \left( 1 - \sigma(f_\theta(x_n)) \right) \right)$$

# An Example Neural Network

---

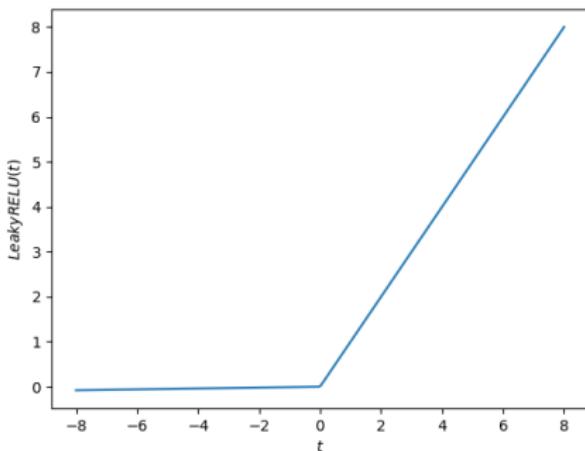
- $f_\theta(x) = w_2 h(W_1^\top x)$ ,  $h(x) = \text{LeakyRELU}(t)$ ,  $W_1 \in \mathbb{R}^{2 \times 100}$ ,  $w_1 \in \mathbb{R}^{100}$ .



- For the hawk eyed people: We are effectively learning the famous  $\phi(\cdot)$  here. Why? / On est en train d'apprendre le fameux  $\phi(\cdot)$ . Vous voyez pourquoi?

# An Example Neural Network

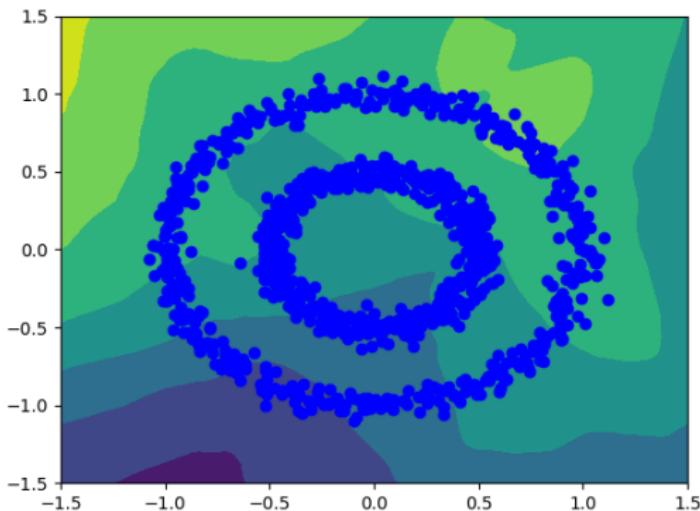
- $f_\theta(x) = w_2 h(W_1^\top x)$ ,  $h(x) = \text{LeakyRELU}(t)$ ,  $W_1 \in \mathbb{R}^{2 \times 100}$ ,  $w_1 \in \mathbb{R}^{100}$ .



- For the hawk eyed people: We are effectively learning the famous  $\phi(\cdot)$  here. Why? / On est en train d'apprendre le fameux  $\phi(\cdot)$ . Vous voyez pourquoi?
- Let's say we define  $\phi_{W_1}(x) := h(W_1^\top x)$ .

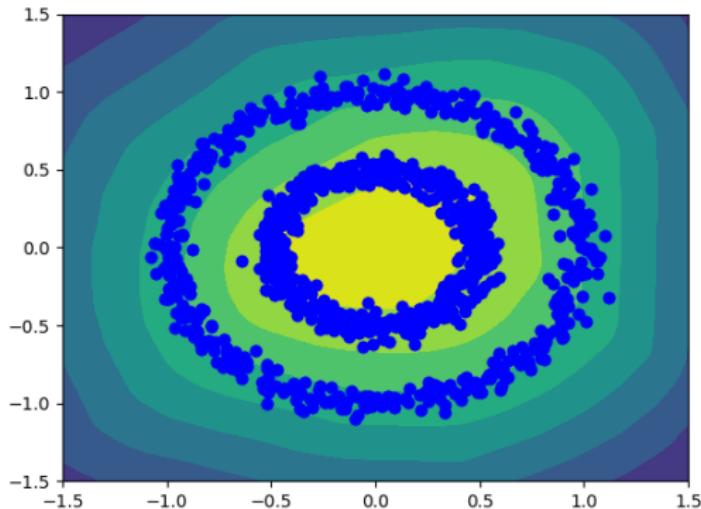
# Neural Network Classifier Learning Steps

---



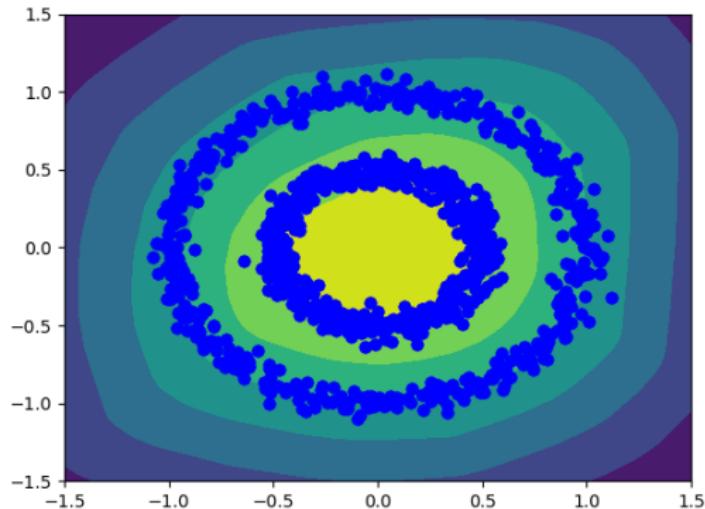
# Neural Network Classifier Learning Steps

---



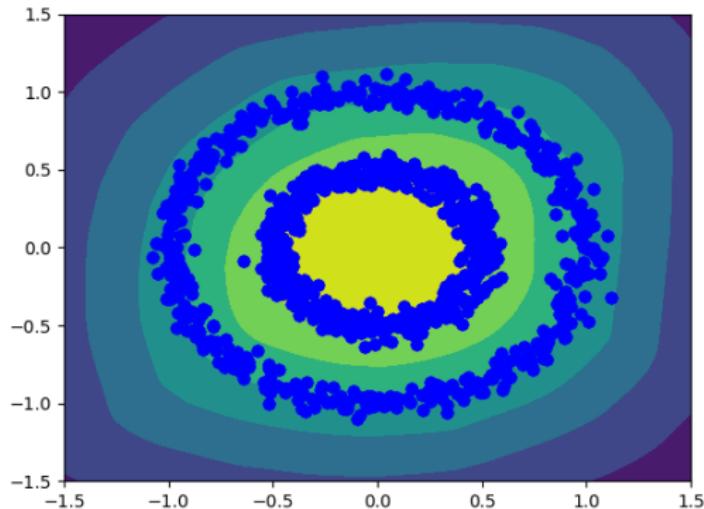
# Neural Network Classifier Learning Steps

---



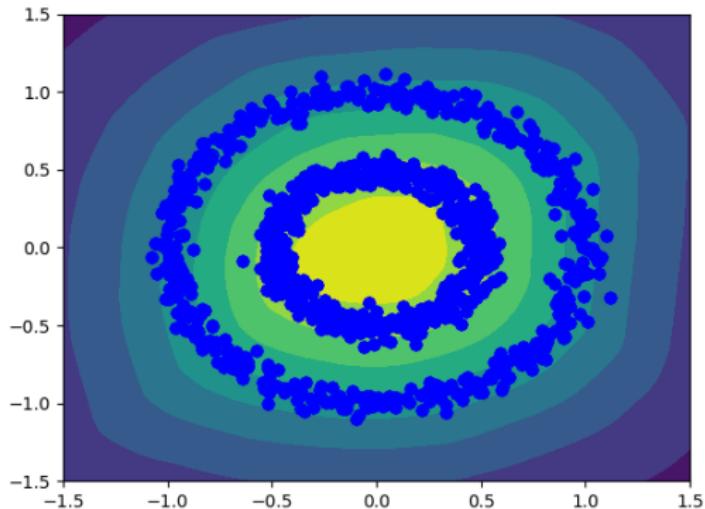
# Neural Network Classifier Learning Steps

---



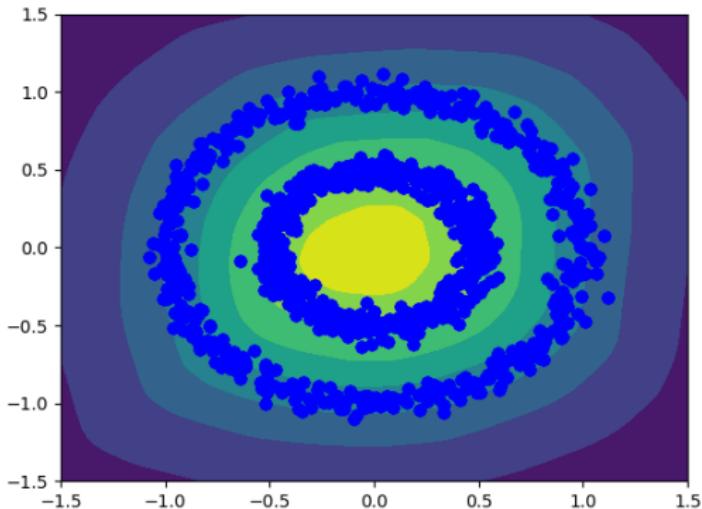
# Neural Network Classifier Learning Steps

---



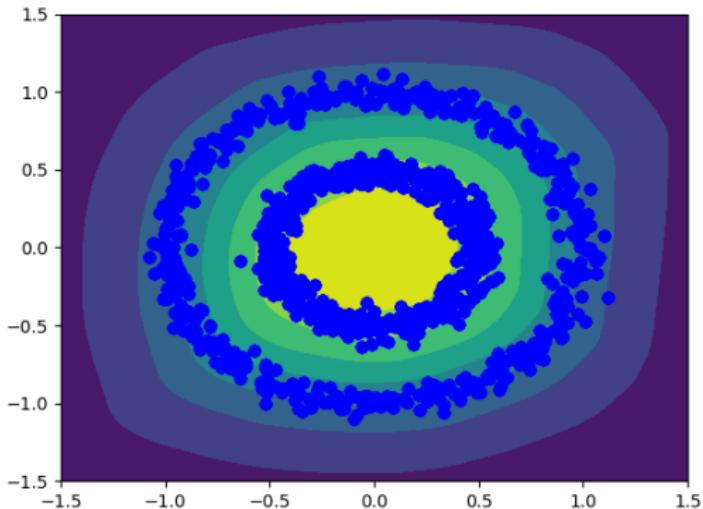
# Neural Network Classifier Learning Steps

---



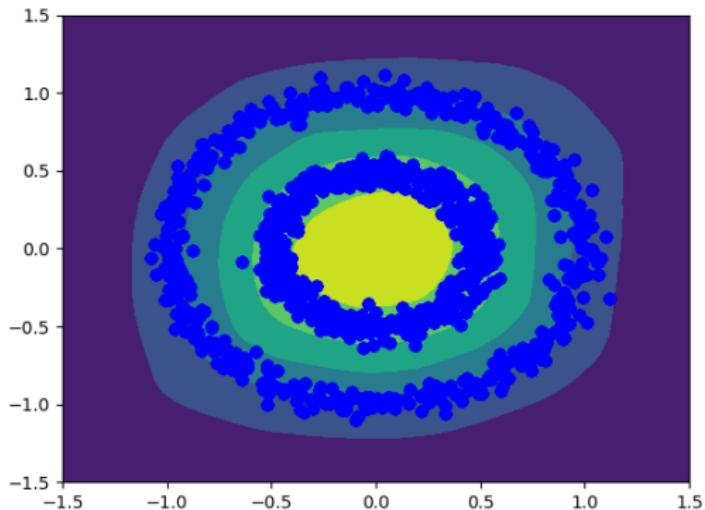
# Neural Network Classifier Learning Steps

---



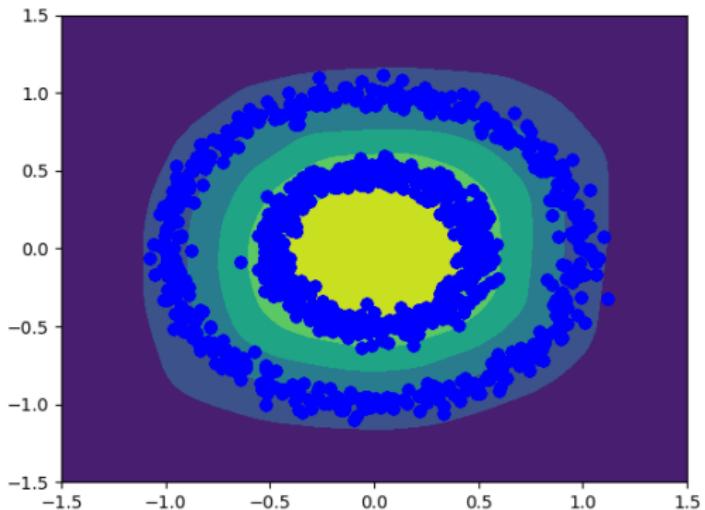
# Neural Network Classifier Learning Steps

---



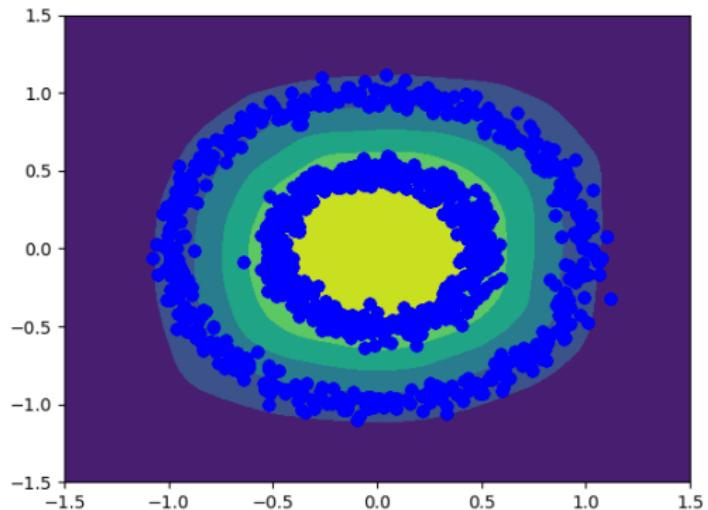
# Neural Network Classifier Learning Steps

---



# Neural Network Classifier Learning Steps

---



# Kernel Methods vs Neural Networks

---

- Kernel Methods are not very suitable for large datasets because of the hard dependency on the training set  $x_1, \dots, x_N$ . / Les méthodes de noyau ne sont pas très adaptées pour des datasets larges parce qu'ils dépendent sur le training set.
- Kernel Methods come with an interpretability advantage. At the end of the day it's a linear model, and we have the importance weights. / Les méthodes de noyaux sont plus interprétables car ils sont linéaires.
- Neural Nets are more modern because of the way we train them. They are more amenable to train with SGD. / Les réseaux de neurones sont plus adaptés à entraîner avec SGD sur des grands jeux de données.

## Recap

---

- We have covered how we approach statistical way of doing classification. / On a parlé sur comment faire classification de manière statistique.
  - ▶ Generative Classification
  - ▶ Discriminative Classification

## Recap

---

- We have covered how we approach statistical way of doing classification. / On a parlé sur comment faire classification de manière statistique.
  - ▶ Generative Classification
  - ▶ Discriminative Classification
- We have talked about how to do linear classification with perceptron algorithm and logistic regression. / On parlé de comment faire régression linéaire avec l'algo de perceptron et avec la regression logistique.

# Recap

---

- We have covered how we approach statistical way of doing classification. / On a parlé sur comment faire classification de manière statistique.
  - ▶ Generative Classification
  - ▶ Discriminative Classification
- We have talked about how to do linear classification with perceptron algorithm and logistic regression. / On parlé de comment faire régression linéaire avec l'algo de perceptron et avec la regression logistique.
- We have talked about Kernel methods, and neural networks for non-linear classification. / On a parlé de méthodes de noyau et les réseaux de neurones pour classification non-linéaire.

# Recap

---

- We have covered how we approach statistical way of doing classification. / On a parlé sur comment faire classification de manière statistique.
  - ▶ Generative Classification
  - ▶ Discriminative Classification
- We have talked about how to do linear classification with perceptron algorithm and logistic regression. / On parlé de comment faire régression linéaire avec l'algo de perceptron et avec la regression logistique.
- We have talked about Kernel methods, and neural networks for non-linear classification. / On a parlé de méthodes de noyau et les réseaux de neurones pour classification non-linéaire.
- We have not talked about SVMs. It's an important idea to know, but you can get that from many other classes. On n'a pas parlé de SVMs.

## Suggested reading

---

- Bishop, chapters 4, 5, 6

## Next week

---

- More deep learning!