

Réseaux Neuronaux
Annick Valibouze
2024 - 2025

M2 Mathématiques et Applications
Ingénierie mathématique
Ingénierie Statistique et Data Science

Projet Réseaux Neuronaux

Diana CUERVO-PALOMA

Q 1 - PMC

Le Perceptron Multi-Couche (PMC) est un réseau de neurones utilisé comme approximateur universel de fonctions. Il est très utilisé dans la prédiction et la classification des données. Une seule couche cachée avec une activation non linéaire et une sortie linéaire peut approximer toute fonction continue.

- **Architecture :**
 - Composé d'une couche d'entrée, d'une ou plusieurs couches cachées, et d'une couche de sortie.
 - L'activation se propage successivement de l'entrée vers les couches cachées, puis vers la sortie.
- N neurones avec états ou activations $a = [a_1, \dots, a_N] \in A^N$ et connexions définies par la matrice de poids $W = [W_{i,j}]$.
- Fonction d'entrée affine : $h(i) = W_i a - b_i$.

Q 1 - PMC

- Fonctions d'activation f_i :
 - Couche cachée :
 - Sigmoïde : $f(x) = \frac{1}{1+\exp(-x)}$.
 - ReLU : $f(x) = \max(x, 0)$.
 - Couche de sortie :
 - Classification binaire : Sigmoïde.
 - Multi-classification : Softmax : $f(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$.
 - Régression : Identité : $f(x) = x$.

Q 1 - Apprentissage supervisé

- L'objectif est de trouver les poids W adaptés au corpus d'apprentissage $C = \{(x_i, d_i) \mid i = 1, 2, \dots, n\}$ avec $x \in \mathbb{R}^k$ (entrée) et $d \in \mathbb{R}^m$ (sortie désirée).
- $s = s(x)$: sortie du réseau après propagation de x .
- **Fonctions d'erreur $E(s, d)$:**
 - Régression : Erreur quadratique $E = \frac{1}{2} \sum_i (d_i - s_i)^2$.
 - Classification : Entropie croisée $E = - \sum_i d_i \log(s_i)$.

Q 1 - Apprentissage supervisé

Objectif : Trouver W qui minimise l'erreur via les cycles d'apprentissage.

Initialisation : Définir $W = W(0)$.

Soit $W(t)$ l'état des poids W à l'instant t .

Cycle d'apprentissage : Pour $t \geq 0$, exécuter :

- Sélectionner aléatoirement $(x, d) \in C$.
- Calculer la sortie s du réseau et l'erreur $E(s, d)$.
- **Règle d'apprentissage delta** : Calculer le gradient de l'erreur et mettre à jour les poids selon la rétropropagation :

$$\Delta W_{i,j} = \lambda a_j \text{err}_i$$

Q 1 - Apprentissage supervisé

avec

$$\text{err}_i = \begin{cases} -\frac{\partial E}{\partial s_i} f'(h(i)) & \text{si } i \text{ est un neurone de sortie,} \\ f'(h(i)) \sum_k W_{k,i} \text{err}_k & \text{si } i \text{ est un neurone caché.} \end{cases}$$

où λ est le taux d'apprentissage.

- Incrémenter $t : t = t + 1$.

La séquence $W(t)$ pour $t \geq 0$ converge vers un minimum de E , qui est une fonction de Lyapunov pour le système dynamique défini par la règle delta.

Critères d'arrêt :

- **Seuil de tolérance** $\epsilon \geq 0$:

$$\|W(t+1) - W(t)\| \leq \epsilon$$

- **Durée maximale** $t_{\text{final}} = \text{Epochs} \times n$, où Epochs est le nombre total de passages sur le corpus d'apprentissage, et n la taille de ce corpus.

Plan

Q 2



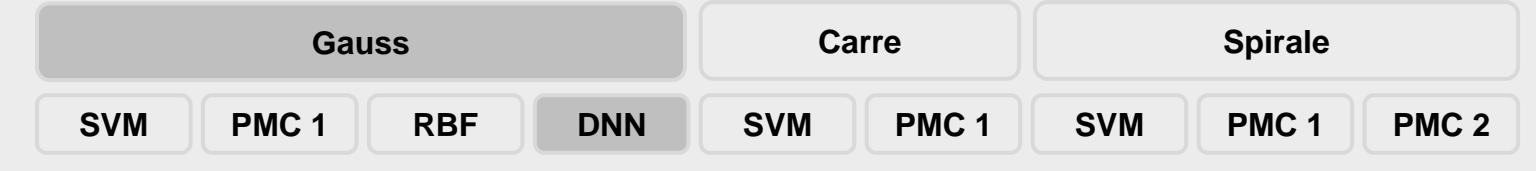
Q 3



Q4

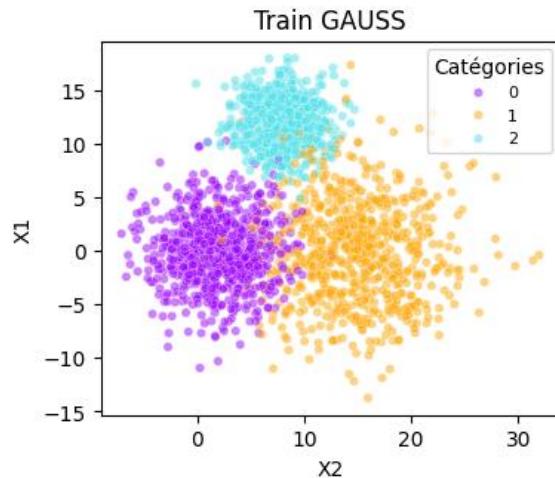


Q 6

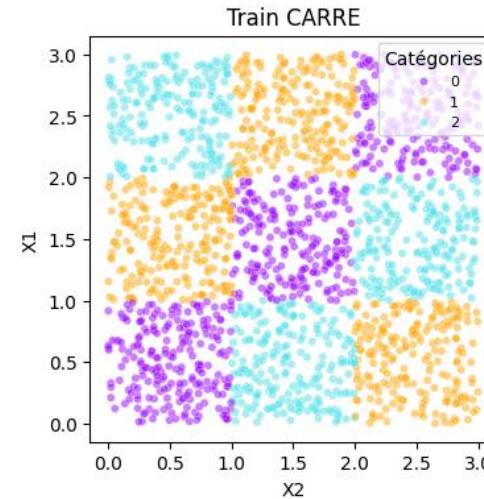


Corpus - Classification

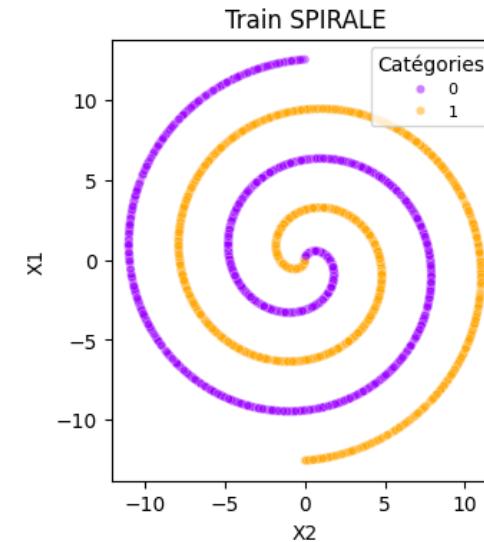
Gauss



Carre



Spirale



- Simulation de 3 distributions normales en R² avec différentes moyennes (μ) et écarts-types (σ)
- 1000 points par classe
- 2100 points pour l'ensemble d'entraînement

- Simulation de distributions uniformes entre 0 et 1 en R², déplacées dans le plan.
- 900 points par classe
- 1890 points pour l'ensemble d'entraînement

- Simulation à partir de coordonnées polaires
- 2000 points par classe
- 2800 points pour l'ensemble d'entraînement

L'ensemble de données est divisé en 70 % pour l'entraînement et 30 % pour les tests.

Q 2 – SVM

- **Objectif** : Assigner un exemple x à l'une des M classes possibles, où $M \geq 2$.
Le corpus d'apprentissage est défini comme :

$$C = \{(x_i, d_i) \mid i = 1, 2, \dots, n\},$$

où $x_i \in \mathbb{R}^k$ représente les caractéristiques de l'exemple, et $d_i \in \{1, 2, \dots, M\}$ indique sa classe.

- **Sortie attendue** : Un vecteur $s \in \mathbb{R}^M$, souvent représenté sous forme *one-hot* :

$$s_m = \begin{cases} 1 & \text{si l'exemple appartient à la classe } m, \\ 0 & \text{sinon.} \end{cases}$$

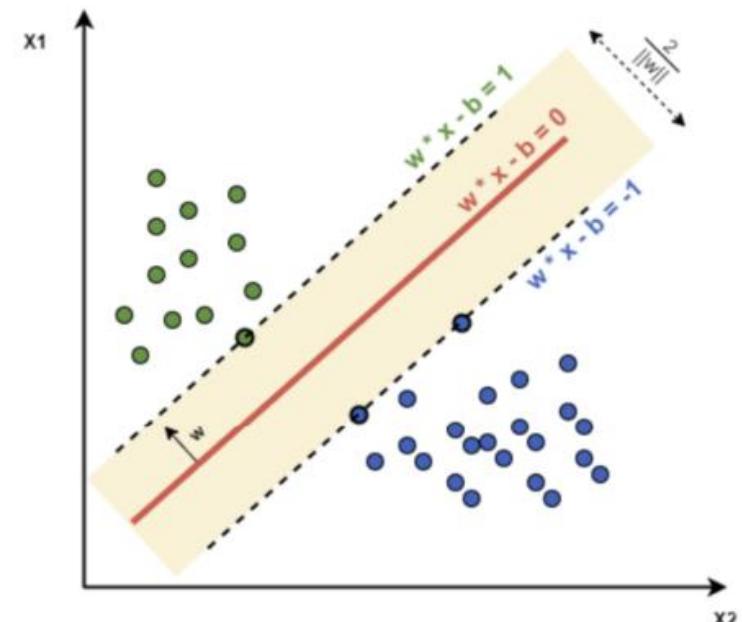
Q 2 – SVM

Le classifieur SVM est une méthode d'apprentissage supervisé utilisée pour les problèmes de classification.

- **Contexte binaire ($M = 2$) :** Trouver un hyperplan de séparation $H : \langle w, x \rangle + b = 0$ avec $w \in \mathbb{R}^k$, $b \in \mathbb{R}$, tel que :

$$\begin{aligned}\langle w, x \rangle + b &> 0 & \text{si } d = 1, \\ \langle w, x \rangle + b &< 0 & \text{si } d = -1,\end{aligned}$$

et maximiser la marge entre les classes. Les points de données les plus proches de l'hyperplan sont appelés *vecteurs de support*.



Q 2 – SVM

- **Hyperplan optimal** : Résolution du problème d'optimisation :

$$\min_{w,b,s} \quad \frac{1}{2} \|w\|^2 + \alpha \sum_{i=1}^n s_i$$

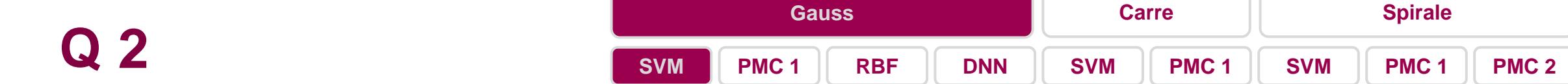
sous les contraintes :

$$d_i(\langle w, x_i \rangle + b) \geq 1 - s_i, \quad s_i \geq 0, \quad \forall i \in \{1, \dots, n\}.$$

où :

- s_i : Variables pour tolérer les erreurs de classification.
- α : Paramètre qui contrôle le compromis entre la maximisation de la marge et la minimisation des erreurs.

Q 2



Call:

```
svm(formula = d ~ X1 + X2, data = sets$train_data, type = "C-classification",
  kernel = "linear")
```

Parameters:

```
  SVM-Type: C-classification
  SVM-Kernel: linear
    cost: 1
```

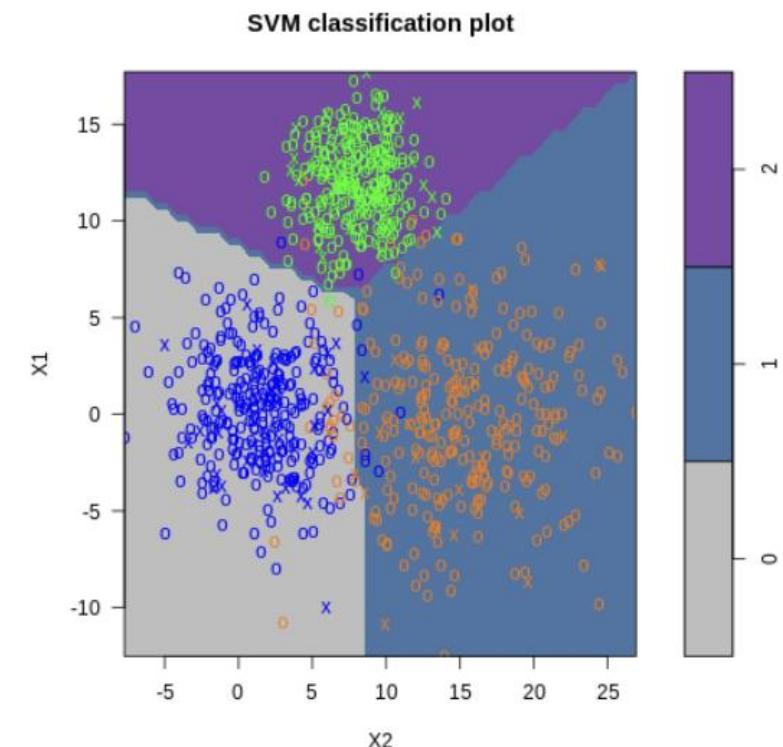
Number of Support Vectors: 287

(101 66 120)

Number of Classes: 3

Levels:

0 1 2



Q 2

Gauss

Carre

Spirale

SVM

PMC 1

RBF

DNN

SVM

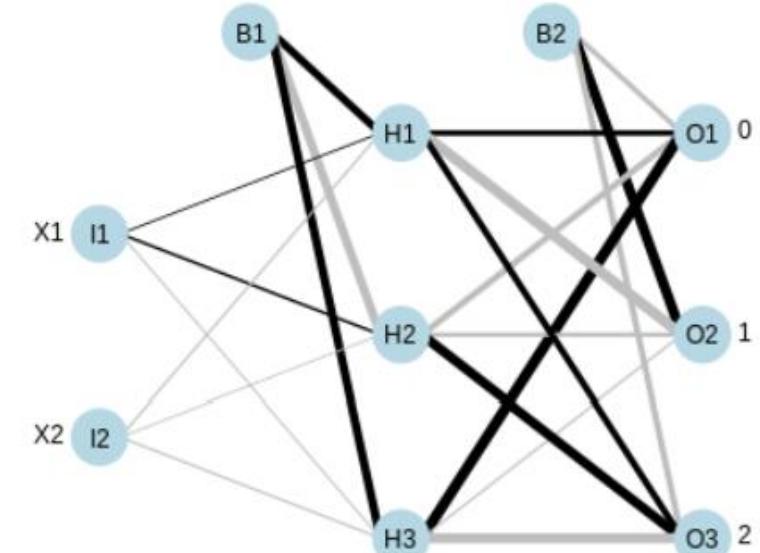
PMC 1

SVM

PMC 1

PMC 2

Caractéristiques	Notation	Valeur
Type de réseau		PMC avec une couche cachée
Architecture		2 – 3 – 3
Fonction d'entrée	$h(i)$	Identité – Affine – Affine
Fonction d'Activation	$e(i)$	Identité - Sigmoïde – Softmax
Fonction d'Erreur	$E(s, d)$	Entropie croisée
Algorithme d'Optimisation		Itération règle du delta
Taux d'Apprentissage	λ	0.2
Époques		maxit = 1000 ou convergence d'Erreur
Ajustement des Hyperparamètres		# de neurones dans la couche cachée et taux d'apprentissage
Logiciel		R - nnet



Q 2

Gauss

SVM

PMC 1

RBF

DNN

Carre

SVM

PMC 1

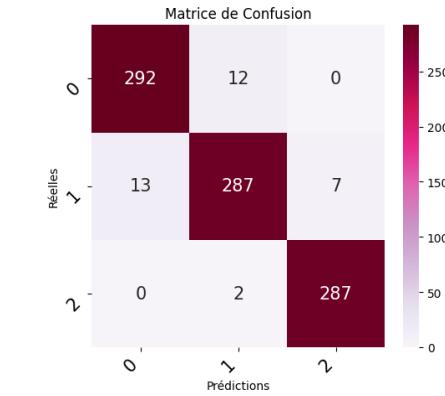
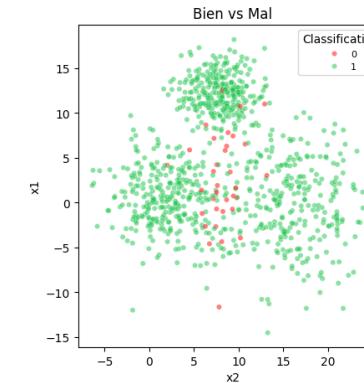
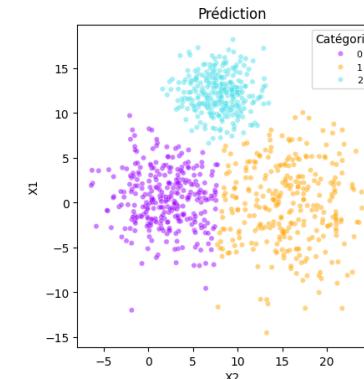
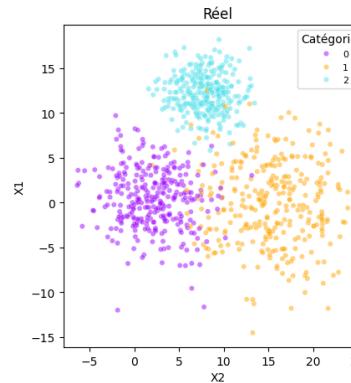
Spirale

SVM

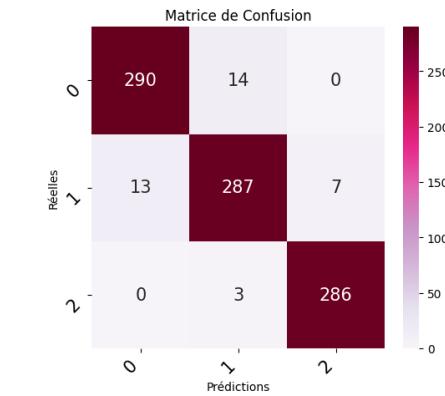
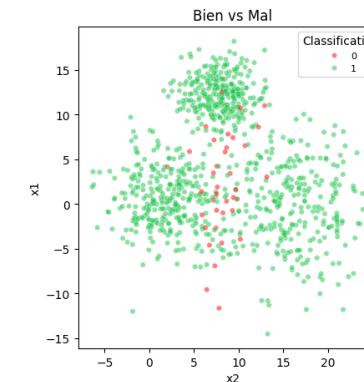
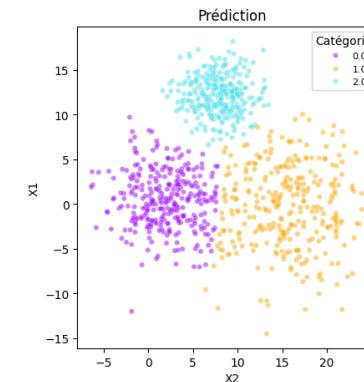
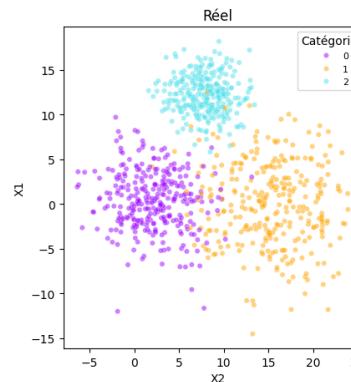
PMC 1

PMC 2

Modèle SVM sur l'ensemble de test avec une précision de : 0.96222



Modèle PMC sur l'ensemble de test avec une précision de : 0.95889



Q 2



Call:

```
svm(formula = d ~ X1 + X2, data = sets$train_data, type = "C-classification",
  kernel = "linear")
```

Parameters:

```
  SVM-Type: C-classification
  SVM-Kernel: linear
    cost: 1
```

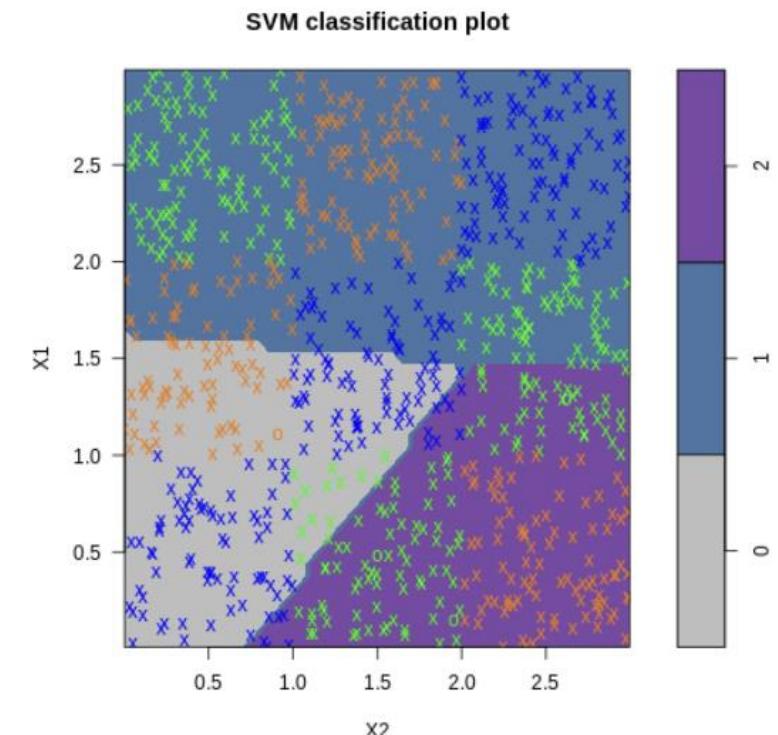
Number of Support Vectors: 1880

```
( 626 634 620 )
```

Number of Classes: 3

Levels:

```
0 1 2
```



Q 2

Gauss

Carre

Spirale

SVM

PMC 1

RBF

DNN

SVM

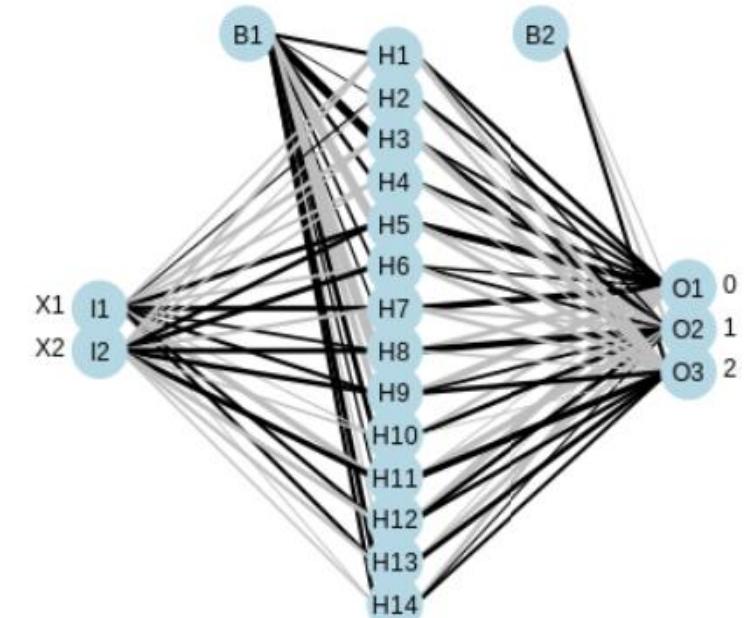
PMC 1

SVM

PMC 1

PMC 2

Caractéristiques	Notation	Valeur
Type de réseau		PMC avec une couche cachée
Architecture		2 – 14 – 3
Fonction d'entrée	$h(i)$	Identité – Affine – Affine
Fonction d'Activation	$e(i)$	Identité - Sigmoïde – Softmax
Fonction d'Erreur	$E(s, d)$	Entropie croisée
Algorithme d'Optimisation		Itération règle du delta
Taux d'Apprentissage	λ	0.001
Époques		maxit = 1000 ou convergence d'Erreur
Ajustement des Hyperparamètres		# de neurones dans la couche cachée et taux d'apprentissage
Logiciel		R - nnet



Q 2

Gauss

SVM

PMC 1

RBF

DNN

Carre

SVM

PMC 1

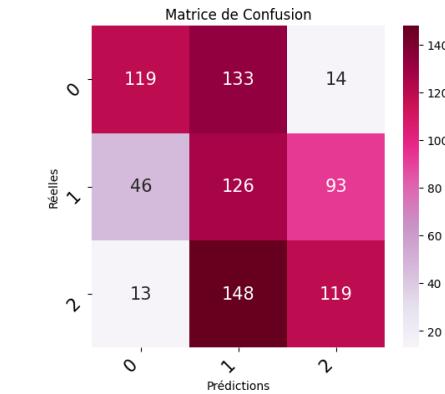
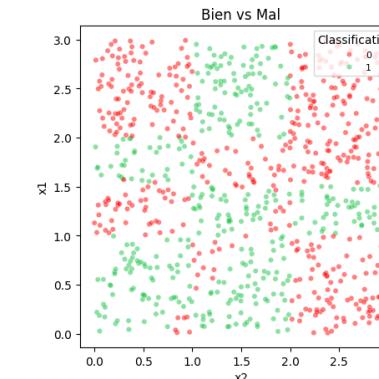
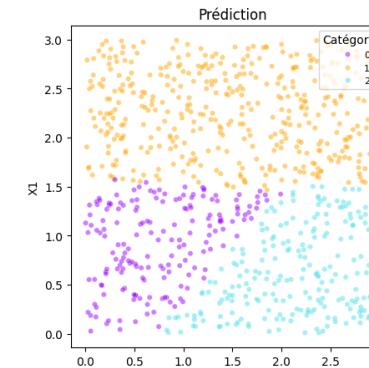
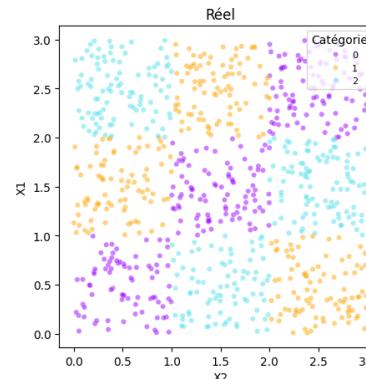
Spirale

SVM

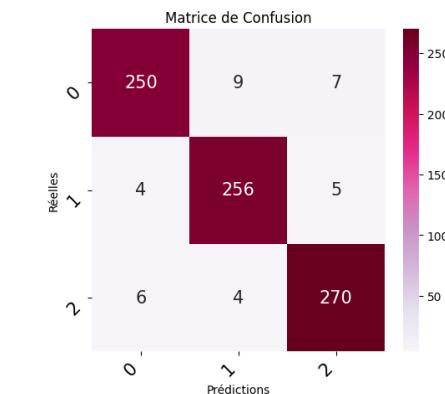
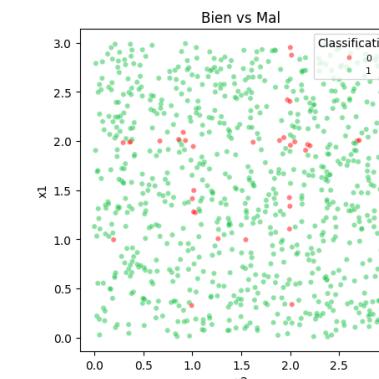
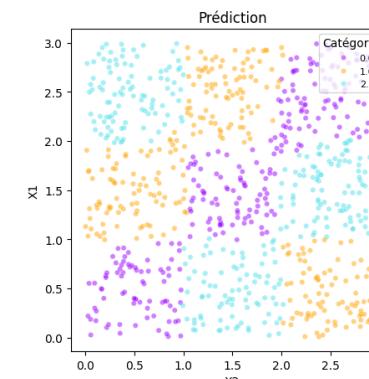
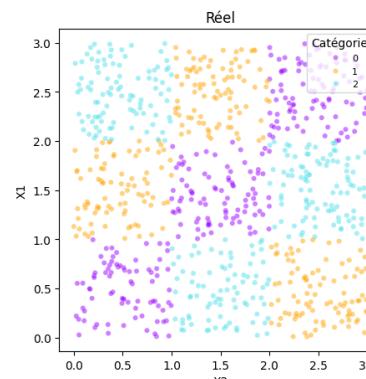
PMC 1

PMC 2

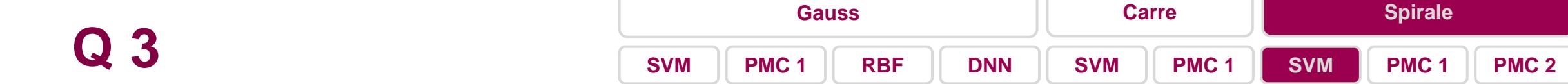
Modèle SVM sur l'ensemble de test avec une précision de : 0.44883



Modèle PMC sur l'ensemble de test avec une précision de : 0.95684



Q 3



Call:

```
svm(formula = d ~ X1 + X2, data = sets$train_data, type = "C-classification",
  kernel = "linear")
```

Parameters:

 SVM-Type: C-classification
 SVM-Kernel: linear
 cost: 1

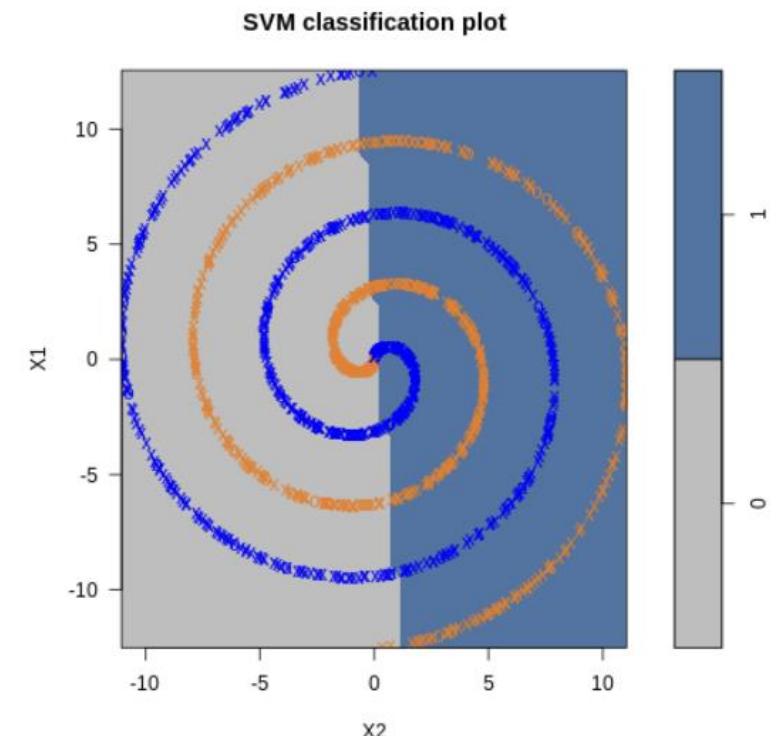
Number of Support Vectors: 2536

(1268 1268)

Number of Classes: 2

Levels:

0 1



Q 3

Gauss

Carre

Spirale

SVM

PMC 1

RBF

DNN

SVM

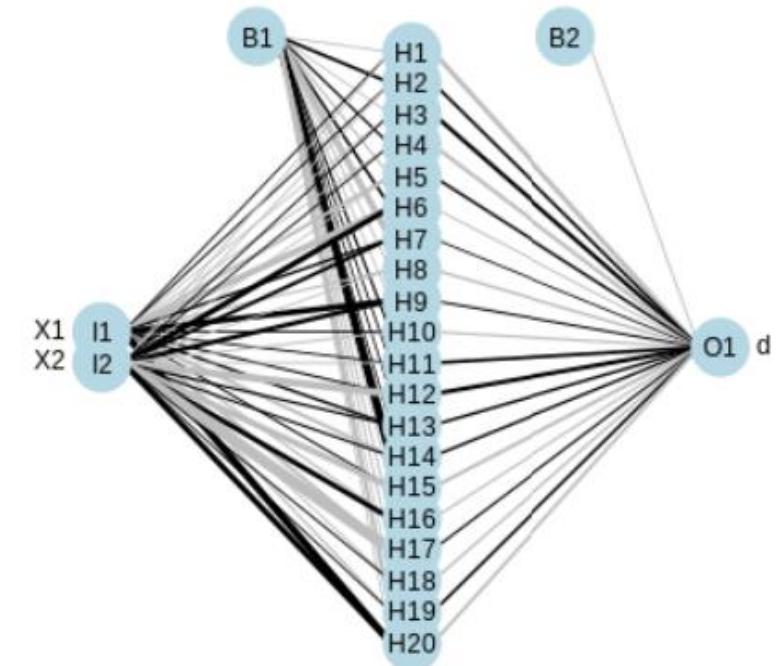
PMC 1

SVM

PMC 1

PMC 2

Caractéristiques	Notation	Valeur
Type de réseau		PMC avec une couche cachée
Architecture		2 – 20 – 3
Fonction d'entrée	$h(i)$	Identité – Affine – Affine
Fonction d'Activation	$e(i)$	Identité - Sigmoïde – Softmax
Fonction d'Erreur	$E(s, d)$	Entropie croisée
Algorithme d'Optimisation		Itération règle du delta
Taux d'Apprentissage	λ	0.001
Époques		maxit = 1000 ou convergence d'Erreur
Ajustement des Hyperparamètres		# de neurones dans la couche cachée et taux d'apprentissage
Logiciel		R - nnet



Q 3

Gauss

Carre

Spirale

SVM

PMC 1

RBF

DNN

SVM

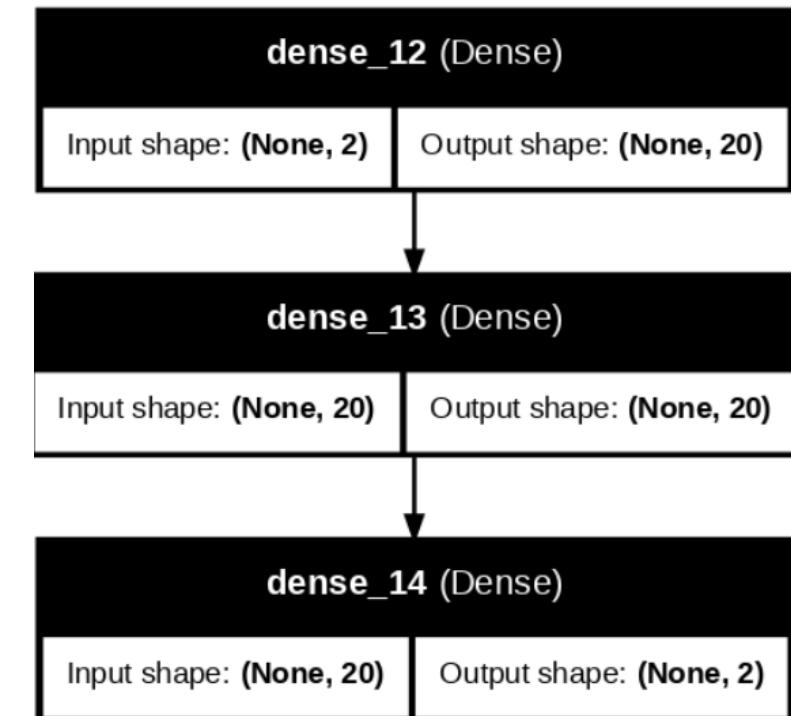
PMC 1

SVM

PMC 1

PMC 2

Caractéristiques	Notation	Valeur
Type de réseau		PMC avec deux couches cachées
Architecture		2 – 20 – 20 – 1
Fonction d'entrée	$h(i)$	Identité – Afine – Affine – Affine
Fonction d'Activation	$e(i)$	Identité – ReLu – ReLu – Softmax
Fonction d'Erreur	$E(s, d)$	Entropie croisée
Algorithme d'Optimisation		Adam
Taux d'Apprentissage	λ	Valeur initiale par défaut (0.001)
Époques		epochs=50, batch_size=10
Ajustement des Hyperparamètres		Non
Logiciel		Python - Keras



Q 3

Gauss

SVM

PMC 1

RBF

Carre

SVM

PMC 1

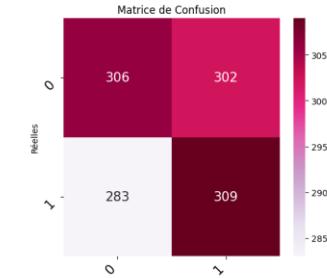
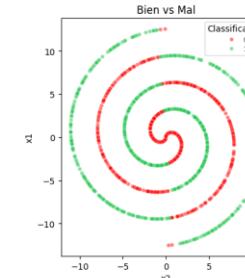
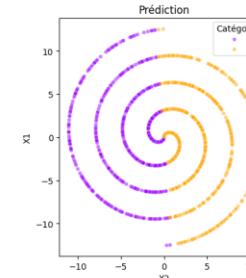
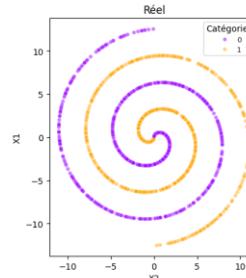
Spirale

SVM

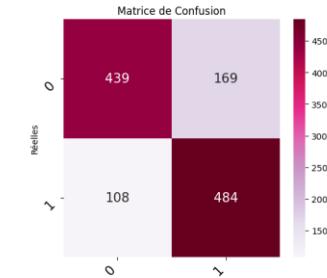
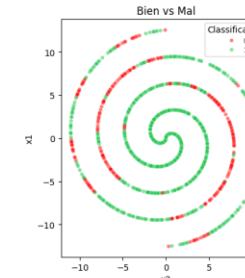
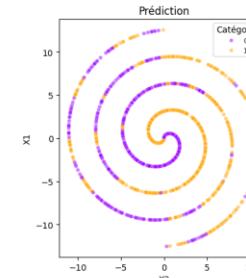
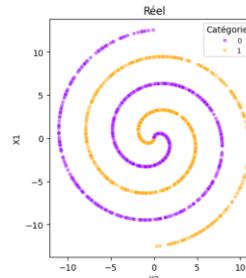
PMC 1

PMC 2

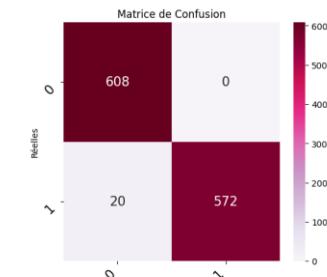
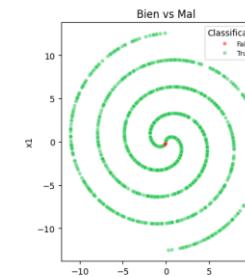
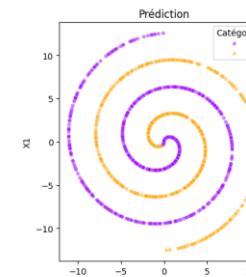
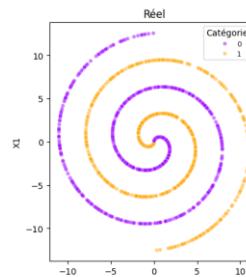
Modèle SVM sur l'ensemble de test avec une précision de : 0.5125



Modèle PMC sur l'ensemble de test avec une précision de : 0.76917



Modèle KERAS sur l'ensemble de test avec une précision de : 0.98333



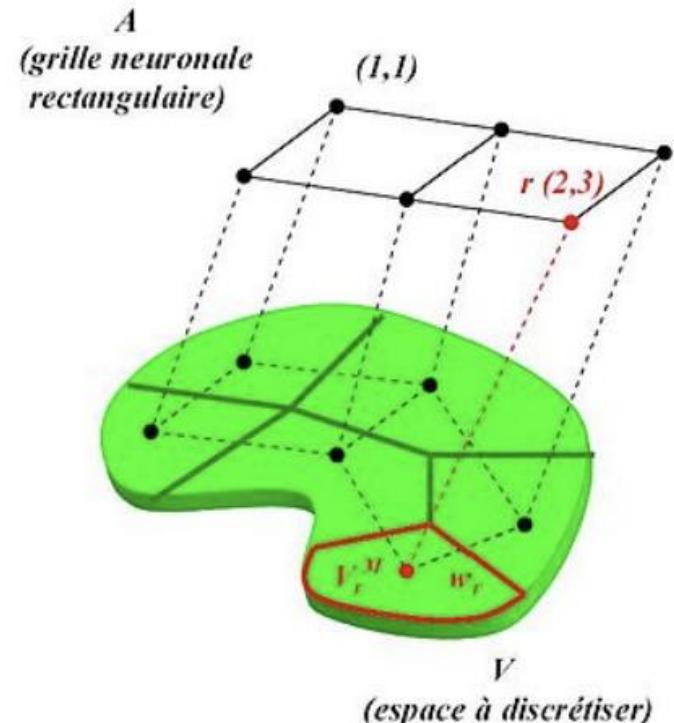
Q 4 Cartes auto-organisées de Kohonen

- Réseau de neurones non supervisé utilisé pour analyser la répartition d'un jeu des données V et simplifier sa représentation.
- Le réseau est constitué d'une grille A de noeuds ou neurones. Chaque neurone $r \in A$ est associé à un vecteur référent $w(r)$, responsable de représenter une zone dans V , définie par :

$$R_r = \{v \in V : \|v - w(r)\| \leq \|v - w(r')\|, \forall r' \in A\}.$$

- La carte A associe à chaque vecteur d'entrée $v \in V$ un neurone $r \in A$, tel que le vecteur référent $w(r)$ soit le plus proche de v :

$$r = \phi_w(v) = \arg \min_{r' \in A} \|v - w(r')\|.$$



Q 4 Cartes auto-organisées de Kohonen

Algorithme d'apprentissage : Adapter les vecteurs référents à l'espace d'entrée.

Entrées :

- Données V , taille et forme de la carte A .
- Initialisation aléatoire $\{w_r(0) : r \in A\}$ des vecteurs référents.
- Nombre d'itérations T_{\max} , paramètre d'apprentissage initial ϵ_0 , paramètre de voisinage initial σ_0 .

Soit $\{w_r(t) : r \in A\}$ l'état des vecteurs référents à l'instant t .

Cycle d'adaptation : Pour $t \geq 0$, exécuter :

- Sélectionner un vecteur $v \in V$ aléatoirement.
- Chercher le neurone gagnant $r = \phi_w(v)$.

Q 4 Cartes auto-organisées de Kohonen

- Mise à jour des vecteurs référents :

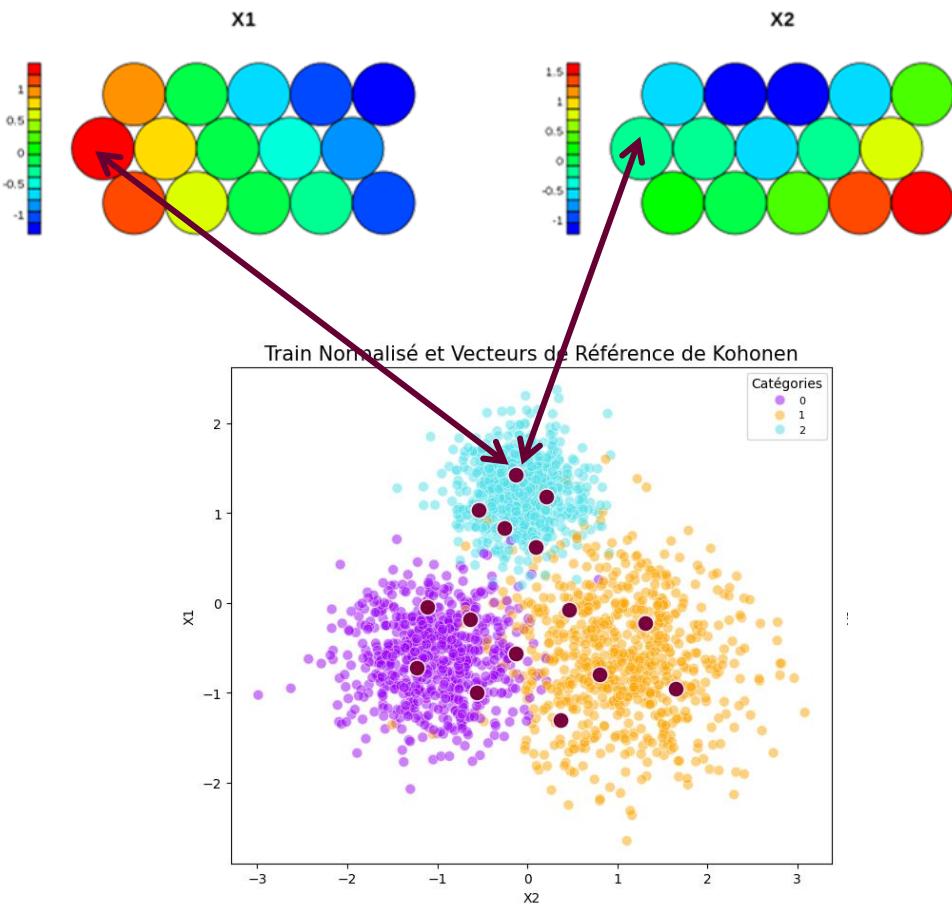
$$w_{t+1}(s) = w_t(s) + \epsilon(t)h_t(s, r)(w_t(s) - v)$$

où :

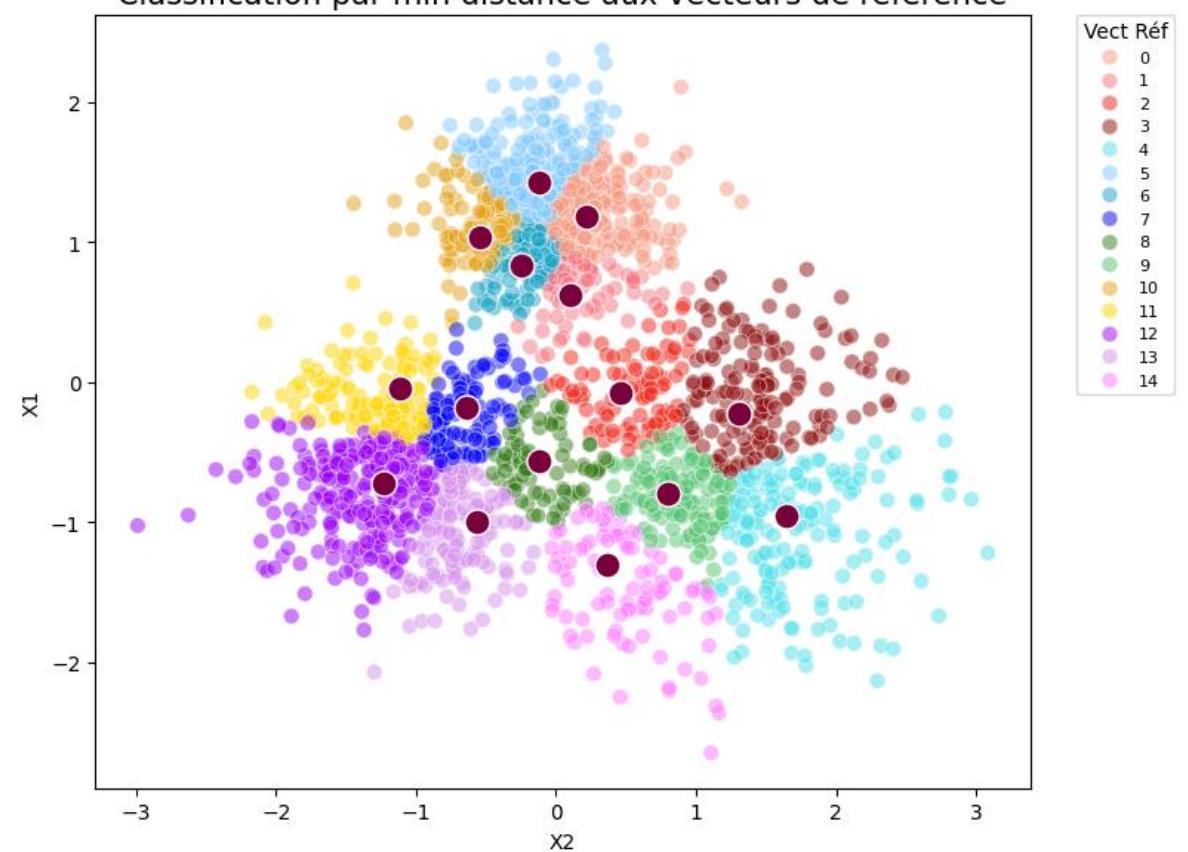
- $\epsilon(t) = \epsilon_0 \exp\left(\frac{-t}{T_{\max}}\right)$ est le pas d'apprentissage. Sa valeur diminue au fil des itérations.
- $h_t(s, r) = \exp\left(\frac{-d^2(s, r)}{2\sigma^2(t)}\right)$ est la fonction de voisinage Gausienne, décrivant comment les neurones proches du vainqueur r sont entraînés.
- $d(s, r)$ est la distance entre les neurones s, r sur la carte A .
- $\sigma(t) = \sigma_0 \exp\left(\frac{-t}{T_{\max}}\right)$ est le coefficient de voisinage.

Q 4 Implementation

Position des vecteurs de référence sur la carte de Kohonen



Classification par min distance aux vecteurs de référence



Q 4

Gauss

SVM

PMC 1

RBF

DNN

Carre

SVM

PMC 1

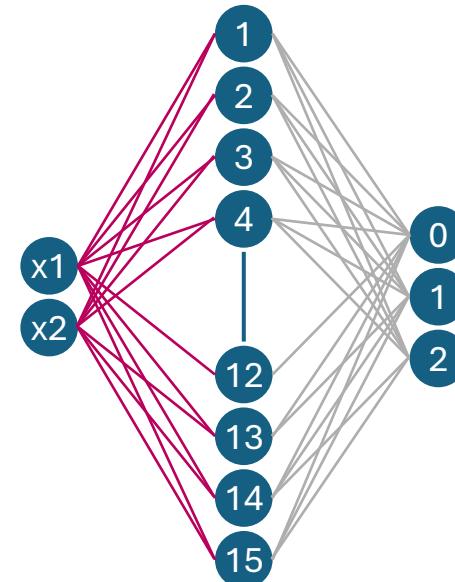
Spirale

SVM

PMC 1

PMC 2

Caractéristiques	Notation	Valeur
Type de réseau		RBF avec une couche cachée
Architecture		2 – 15 – 3
Fonction d'entrée	$h(i)$	Identité – Distance – Affine
Fonction d'Activation	$e(i)$	Standardisation – Gaussienne – Softmax
Fonction d'Erreur	$E(s, d)$	Entropie croisée
Algorithme d'Optimisation		Itération règle du delta
Taux d'Apprentissage	λ	0.001
Époques		2
Ajustement des Hyperparamètres		Non
Logiciel		<ul style="list-style-type: none"> ◦ Couche cachée : R – kohonen ◦ Couche Sortie : Python - NumPy



Cycle d'apprentissage

- Sélectionner un exemple du corpus
- Calcul de l'entrée et activation de la couche cachée

$$h(i) = \|a - W_i^T\| \quad f(x) = \exp\left(-\frac{x^2}{\sigma^2}\right)$$

- Calcul de l'entrée et activation de la couche de sortie

$$h(i) = W_i a - b_i \quad f(x) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

- Calcul l'erre_i couche de sortie et $E = - \sum_i d_i \log(s_i)$

$$\begin{aligned} \text{err}_i &= -\frac{\partial E}{\partial s_i} \cdot f'(h(i)) \\ &= -\frac{d_i}{s_i} \cdot f(h(i)) \cdot (1 - f(h(i))) \\ &= -\frac{d_i}{s_i} \cdot s_i \cdot (1 - s_i) \\ &= -d_i(1 - s_i) \end{aligned}$$

- Calcul la règle d'apprentissage delta

$$\Delta W_{i,j} = \lambda a_j \text{err}_i$$

```
for epoch in range(epoch):
    for i,x in enumerate(train): #x in (#caract.,)
        y = vector_target_train[i] #one hot encoder (#class,)

        # prediction
        a = RBF_couche(x,w_kohonen,sigma) #sorti couche_cachee fonc radial (#kohonen,)
        s = sortie_couche(a,prev_w ) #sorti de couche_sorti / softmax (#class,)

        #apprendisage
        err_i = -y*(1-s) #err_i = -(dE/s_i)*f'(h(i)) i in (1,2,3) /(#class,)
        grad_entropy = (err_i[:, np.newaxis]*a) #(#class, #kohonen)
        result_w = prev_w - lamb*grad_entropy #regle d'apprentissage delta / (#class,#kohonen)

        #loss
        e = entropie_croisee_multiclas(y, s) #loss par chaque class (#class,)
        if prev_w is not None:
            result_e[i] = e
        prev_w = result_w

        # decodage du one-hot
        result_s[i] = arg_max_choice(s) #un reel entre 0 et #class

result_e = np.concatenate((result_e, result_e_))
```

Q 4

Gauss

SVM

PMC 1

RBF

DNN

Carre

SVM

PMC 1

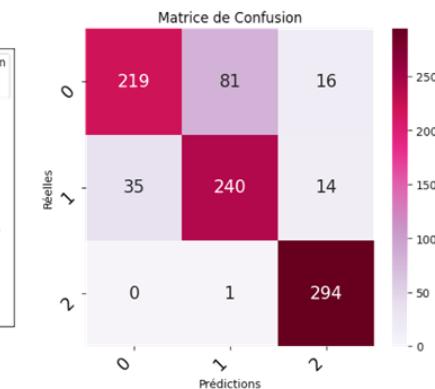
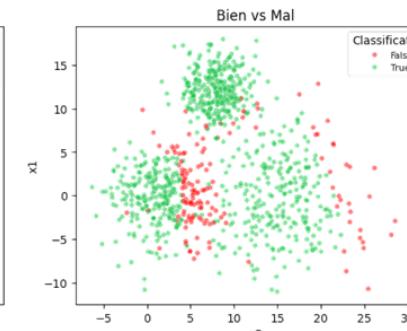
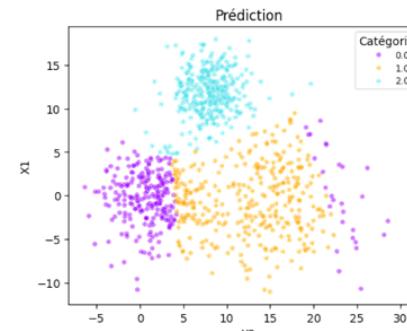
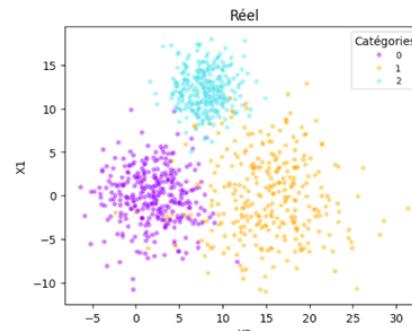
Spirale

SVM

PMC 1

PMC 2

Modèle RBF sur l'ensemble de test avec une précision de : 0.83667



Q 5 Machine de Boltzmann Restreinte (RBM)

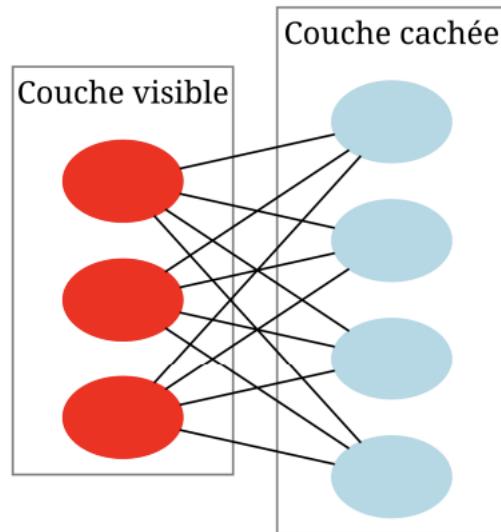
Une Machine de Boltzmann Restreinte (RBM) est un modèle de réseau de neurones pour l'apprentissage non supervisé. Il est composé de deux couches :

- **Couche visible** $\mathbf{x} = (x_1, x_2, \dots, x_N)$: représente les données d'entrée (par exemple, une image ou un vecteur de caractéristiques).
- **Couche cachée** $\mathbf{h} = (h_1, h_2, \dots, h_M)$: apprend des représentations latentes des données visibles.

Les neurones visibles sont connectés aux neurones cachés, sans connexion au sein de chaque couche. L'objectif principal d'une RBM est de modéliser les caractéristiques des données.

Les RBM sont utilisées pour :

- **Réduction de dimension** : extraction de caractéristiques.
- **Initialisation** des poids dans les réseaux profonds.
- **Modélisation générative** : reconstruction et génération de données.



Q 5 Fonction d'énergie et probabilités

L'énergie associée à un état (\mathbf{x}, \mathbf{h}) est définie par :

$$E(\mathbf{x}, \mathbf{h}) = - \sum_{i,j} W_{ij} x_i h_j - \sum_i b_i x_i - \sum_j c_j h_j$$

où W_{ij} est le poids entre x_i et h_j , et b_i , c_j sont les biais des neurones visibles et cachés.

La probabilité d'un état (\mathbf{x}, \mathbf{h}) est donnée par :

$$p(\mathbf{x}, \mathbf{h}) = \frac{\exp(-E(\mathbf{x}, \mathbf{h}))}{Z}, \quad Z = \sum_{\mathbf{x}, \mathbf{h}} \exp(-E(\mathbf{x}, \mathbf{h}))$$

Les probabilités conditionnelles :

$$p(h_j = 1 | \mathbf{x}) = \sigma \left(\sum_i W_{ij} x_i + c_j \right), \quad p(x_i = 1 | \mathbf{h}) = \sigma \left(\sum_j W_{ij} h_j + b_i \right)$$

où $\sigma(x) = \frac{1}{1+\exp(-x)}$ est la fonction sigmoïde.

Q 5 Apprentissage avec la divergence contrastive

L'apprentissage dans une RBM vise à maximiser la vraisemblance des données d'entrée \mathbf{x} , définie comme :

$$P(\mathbf{x}) = \sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{h}),$$

La maximisation de cette vraisemblance, équivalente à minimiser l'énergie du système, est approximée par la méthode de *Contrastive Divergence (CD)* avec les étapes suivantes :

1. **Initialisation** : Les paramètres W , b_i , et c_j sont initialisés aléatoirement ou selon une distribution gaussienne.
2. **Étape positive** : Pour chaque exemple \mathbf{x} du corpus :
 - Calculer $p(h_j = 1|\mathbf{x})$, les probabilités conditionnelles.
 - Échantillonner $\mathbf{h} \sim p(\mathbf{h}|\mathbf{x})$.
 - Obtenir $\langle x_i h_j \rangle_{\text{data}}$, l'attente des produits croisés sous les données.
3. **Étape négative** :
 - Générer $\mathbf{x}' \sim p(\mathbf{x}|\mathbf{h})$.
 - Calculer $\mathbf{h}' \sim p(\mathbf{h}|\mathbf{x}')$.
 - Obtenir $\langle x_i h_j \rangle_{\text{model}}$, l'attente des produits croisés sous le modèle.
4. **Mise à jour** :
$$\Delta W_{ij} = \epsilon (\langle x_i h_j \rangle_{\text{data}} - \langle x_i h_j \rangle_{\text{model}}),$$
$$\Delta b_i = \epsilon (\langle x_i \rangle_{\text{data}} - \langle x_i \rangle_{\text{model}}), \quad \Delta c_j = \epsilon (\langle h_j \rangle_{\text{data}} - \langle h_j \rangle_{\text{model}}),$$
où ϵ est le taux d'apprentissage.

Q 6

Gauss

Carre

Spirale

SVM

PMC 1

RBF

DNN

SVM

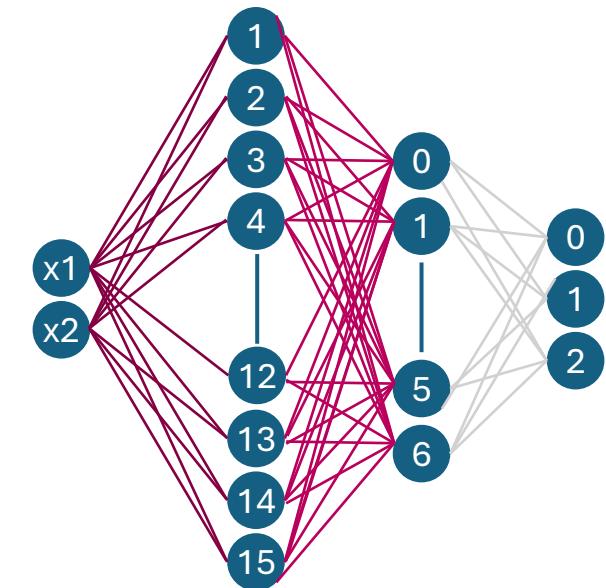
PMC 1

SVM

PMC 1

PMC 2

Caractéristiques	Notation	Valeur
Type de réseau		DNN avec deux couches cachées
Architecture		2 – 15 – 7 – 3
Fonction d'entrée	$h(i)$	Identité – Distance – Affine – Affine
Fonction d'Activation	$e(i)$	Standardisation – Gaussienne – ReLu – Softmax
Fonction d'Erreur	$E(s, d)$	Entropie croisée
Algorithme d'Optimisation		Adam
Taux d'Apprentissage	λ	Valeur initiale par défaut (0.001)
Époques		epoch=50, batch_size=10
Ajustement des Hyperparamètres		Non
Logiciel		<ul style="list-style-type: none"> ◦ Première couche cachée : R – kohonen ◦ Deuxième couche cachée et couche sortie: Python - Keras



Q 6

Gauss

SVM

PMC 1

RBF

DNN

Carre

SVM

PMC 1

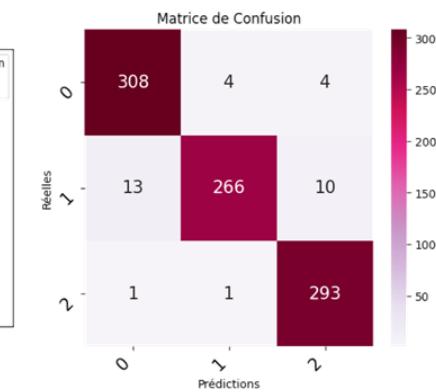
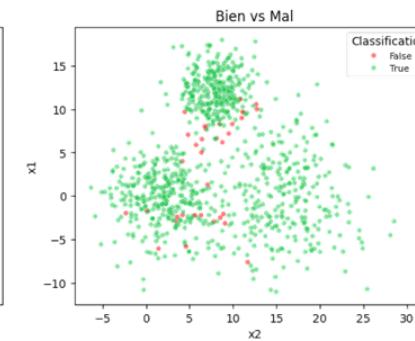
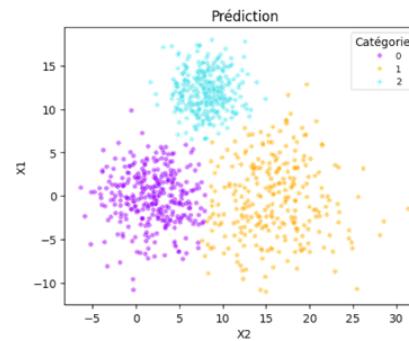
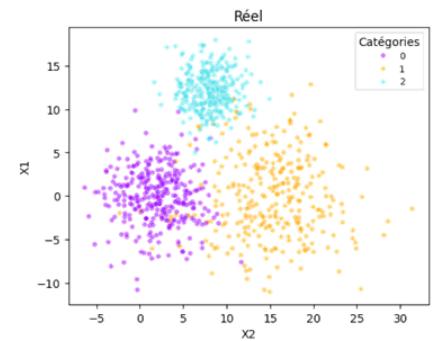
Spirale

SVM

PMC 1

PMC 2

Modèle DNN sur l'ensemble de test avec une précision de : 0.96333



Q7 Conclusions

- **SVM et PMC sur la mixture gaussienne** : Les deux modèles ont obtenu des précisions élevées (0,95 pour le SVM et 0,96 pour le PMC) sur cet ensemble linéairement séparable, illustrant leur efficacité pour résoudre des problèmes bien structurés.
- **SVM et PMC sur l'ensemble Carré** : Le SVM a montré une limite importante (précision de 0,45) sur cet ensemble non linéairement séparable. En revanche, le PMC a démontré sa capacité d'adaptation en atteignant une précision de 0,95 après l'augmentation du nombre de neurones cachés de 3 à 14. Ce résultat met en évidence le potentiel des réseaux neuronaux pour généraliser et s'adapter à des corpus plus complexes.
- **PMC sur la spirale** : L'ensemble en spirale a représenté un défi notable. Avec une seule couche cachée, le PMC a atteint une précision modérée de 0,77. Cependant, l'ajout d'une seconde couche cachée et l'utilisation de la fonction d'activation ReLU ont permis d'atteindre une précision élevée de 0,96. Ce cas démontre la capacité des PMC à séparer efficacement des ensembles très complexes lorsque leur architecture est ajustée.

Q7 Conclusions

- **PMC à une seule couche** : Ce modèle s'est bien comporté sur les trois ensembles de données, soulignant la flexibilité des réseaux neuronaux à s'adapter à divers types de données avec des contraintes moindres par rapport à d'autres modèles comme le SVM.
- **Méthode de Kohonen et réseau RBF** : Bien que la méthode de Kohonen soit puissante pour organiser et réduire la dimensionnalité des données, son implémentation dans un réseau RBF n'a pas donné de résultats significatifs sur l'ensemble le plus simple (classification de 3 gaussiennes). Cela souligne la nécessité d'explorer des configurations supplémentaires pour exploiter pleinement cette méthode.
- **Keras comme outil DNN** : Keras s'est révélé être une bibliothèque versatile et intuitive pour concevoir, tester, et ajuster différentes architectures de réseaux neuronaux. Cette flexibilité en fait un outil de choix pour les expérimentations en apprentissage profond.

Logiciels

○ **svm** :

- Pour la classification : `type = 'C-classification'`
- Type de séparation : `kernel = "linear"`

○ **nnet** :

- Pour la classification : `linout = FALSE`
- Nombre de neurones dans la couche cachée : `size`
- Pas d'apprentissage : `decay`
- Critère d'arrêt : `Maxit`

○ **kohonen** :

- Somgrid:
 - Topo : hexagonal ou rectangulaire
 - neighbourhood.fct : gaussienne
- Som :
 - rlen : nombre d'itérations
 - alpha : taux d'apprentissage



○ **Keras**

```
# Construire le modèle de réseau neuronal
model = Sequential()

# Couche d'entrée (2 caractéristiques)
model.add(Input(shape=(2,)))

# Couches cachées
model.add(Dense(neurons_layer_1, activation='relu'))
model.add(Dense(neurons_layer_2, activation='relu'))

# Couche de sortie (2 classes)
model.add(Dense(2, activation='sigmoid'))

# Construire le modèle
model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

# Entrainer le modèle
model.fit(X_train, y_train, epochs=50, batch_size=10,
validation_data=(X_val, y_val), verbose=0)
```



Merci