Polytech Nice Sophia

Rapport de projet : Machine Learning

BEN KHALIFA Emna, HONAKOKO Giovanni, ZIAD Zineb

10/06/2025

1 Introduction

Le *Machine Learning* (**ML**) ou littéralement l'apprentissage automatique est un domaine scientifique. Plus spécifiquement c'est une de la branche de l'intelligence artificielle (**IA**). Le ML consiste à apprendre à une machine des « *patterns* » c'est-à-dire des motifs récurrents dans un ensemble de données. Dans notre cas nous avions à disposition un dataset `load_digits` provenant de la librairie Python `sci-kit learn` (**sklearn**), cette base de donnée contient des images de chiffres manuscrits. Notre objectif final est donc d'apprendre à la machine via un algorithme optimisé au cours de cette semaine : Comment reconnaître un chiffre manuscrit sur une image ?
Dans le cadre de ce projet on s'est contenté de faire de l'apprentissage supervisé.

2 Découverte et visualisation de la base de données

Notre base de données contient des images de chiffres manuscrits en niveau de gris, ainsi on l'a stock dans une matrice appelée : matrice de donnée X .
Notre base `digits` contient 1797 `data` et 64 `target`, donc X est de dimension $(1797, 64)$
A l'intérieur de cette dernière chaque coefficient représente un pixel pouvant prendre des valeurs dans $[\![0, 16]\!]$. On a :

— □ pour des valeurs proches de 16.

— ■ pour des valeurs proches de 0.

On peut découper X en sous-matrices 8×8, chacune de ces sous-matrices représentant un chiffre manuscrit, une image.

Avant de commencer à entraîner notre machine il est important de faire du data engineering c'est à dire traiter nos données par du :

3 Preprocessing

3.1 Preprocessing local

On commence par normaliser les intensités des pixels contenu dans X en divisant par le $\max_{x \in [\![0,16]\!]} |x|$: $\mathtt{X} = \mathtt{X} \backslash 16$
Ainsi X contient des valeurs dans $[0, 1]$.

Features 1 : ACP/PCA

On va maintenant utiliser un algorithme d'*analyse en composantes principales* (**PCA**). Dans notre cas le PCA trouve des hyperplans dans $\mathbb{R}^{64}$ qui approximent les données au mieux selon la méthode des moindres carrés.
Dans notre cas on vise à diagonaliser la matrice $\Gamma = X^T X \in M_{64,64}(\mathbb{R})$: la matrice de corrélation/covariance. Cette matrice étant symétrique réelle elle est diagonalisable et il existe une relation de passage :

$$\Gamma = PDP^{-1}$$

avec P la matrice de passage et D est la matrice diagonale contenant les valeurs propres de Γ.
Les valeurs propres de Γ sont aussi appelées : **composantes principales**. Ce qui nous intéresse c'est les vecteurs propres associés aux plus grandes valeurs propres en valeur absolue. Puisque l'essentiel de la

dispersion notre jeu de donnée, sa variance sera portée par ces vecteurs. C'est effectivement ce que l'on peut observer sur le graphique suivant :

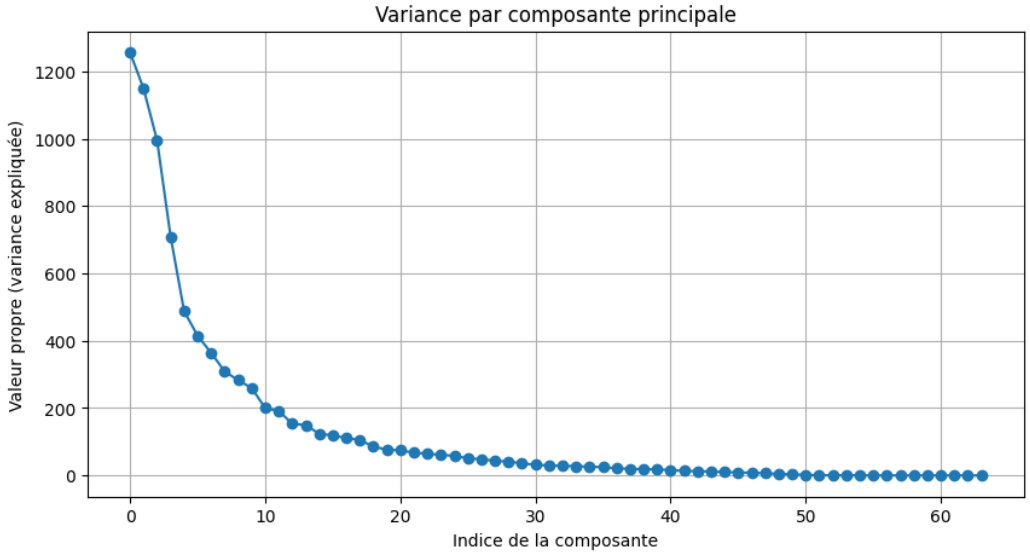

FIGURE 1 – Variance par composante principale

On pourrait se demander combien faut-il de composante principale pour conserver le plus d'information tout en réduisant le flot de donnée que l'on traite. C'est la variance cumulée qui nous donne une telle mesure :

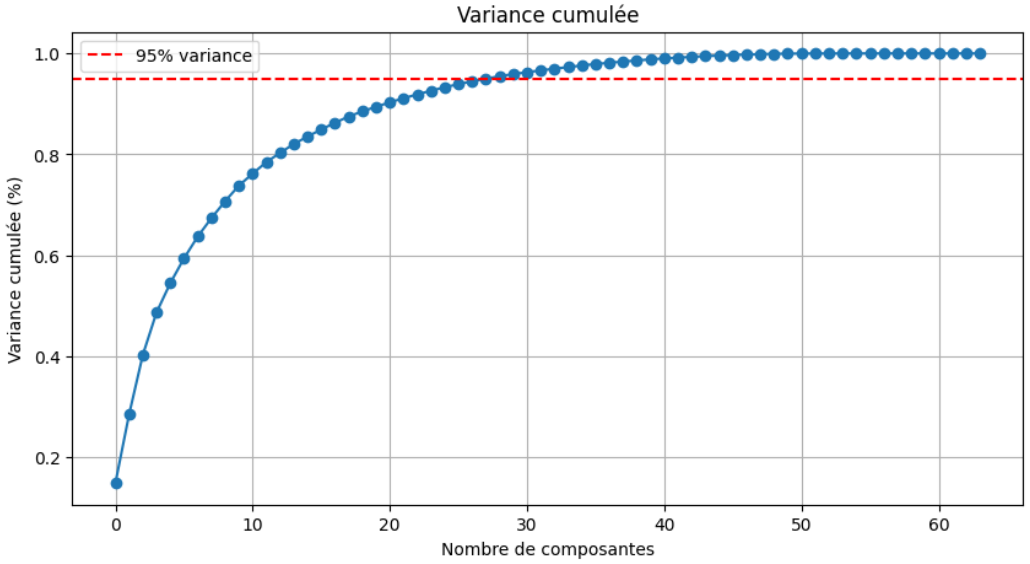

FIGURE 2 – Variance cumulée

Ici prendre 30 composantes principales nous permets de capter 95% de l'information sur une image tout en réduisant le débit de donnée traitée.

Finalement l'intérêt de cette transformation et que l'on puisse projeter X l'espace afin de traiter nos images dans des dimensions réduites ce qui améliore la complexité et l'efficacité.
Par exemple pour 15 composantes principales, on a une base spectrale de 15 vecteurs propres dont voici un aperçu :

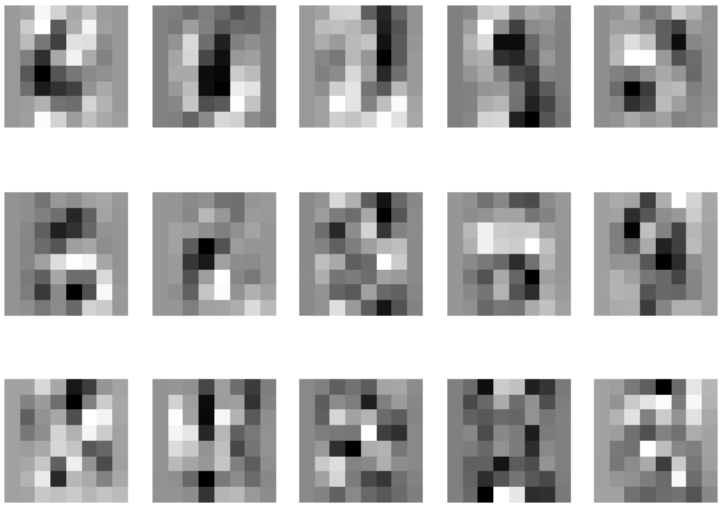

FIGURE 3 – Base spectrale pour 15 composantes principales

On comprends un peu plus visuellement que des combinaisons linéaires de ces vecteurs puissent générer nos chiffres manuscrit.
Évidemment lors d'une projection on a une perte d'information. Donc lorsque l'on reconstruit notre image à partir de l'espace restreint de l'ACP, le résultat est moins nette mais reste satisfaisant :

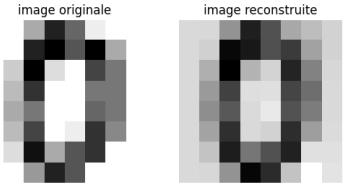

FIGURE 4 – Comparaison image originale vs image reconstruite avec PCA global

Ici nous utilisons le terme *global* pour signifier le fait que nous avons effectué l'ACP sur toute la base de donnée `digits` via X.
Durant ce projet on s'est également amusé à recoder l'ACP à partir des concepts algébristes évoqués plus haut et nous avons retrouvé les mêmes résultats qu'avec les fonctionnalités de `sklearn`.
Par la suite nous avons également scindé notre database initiale en sous-data base contenant uniquement les occurrences de chaque classe. C'est-à-dire que nous avons fabriqué une matrice de donnée X_1 stockant toute les images de « 1 »et de même pour chaque classe de 0 à 9. En appliquant l'ACP on obtient un résultat plus nette :

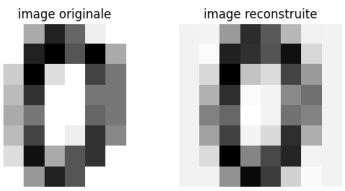

FIGURE 5 – Comparaison image originale vs image reconstruite avec PCA local

Avec 0 les résultats sont assez concluants, néanmoins ce n'est pas forcément le cas pour les autres classes et

cela est dû au clusters des classes. Ces dernières n'ont pas toute de séparation linéaire claire :

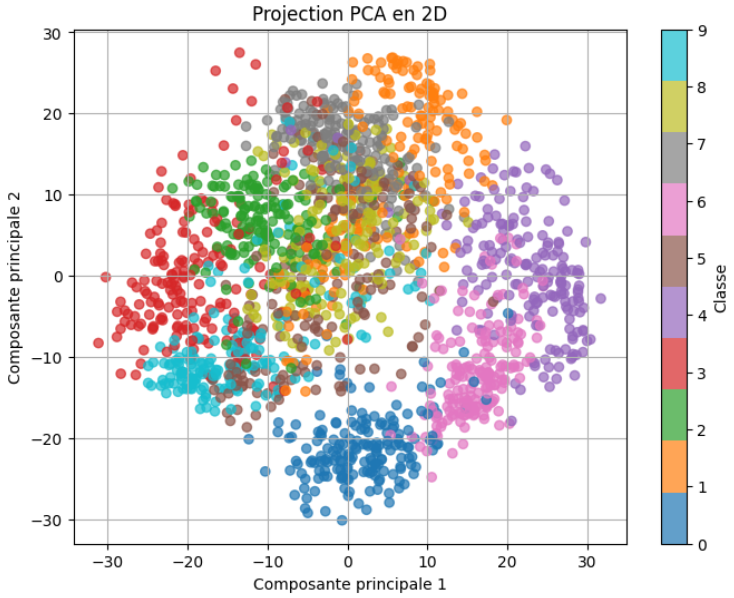

FIGURE 6 – Clusters des classes avec ACP

Features 2 : Partitionnement d'une image

Afin d'être plus effectif dans la détection des variations de gradient d'intensité pour une image 8×8, nous l'avons scindé en 3 zones distinctes.

- ❑ Zone 1 : ligne 1-3
- ❑ Zone 2 : ligne 4-5
- ❑ Zone 3 : ligne 6-8

En observant les variations moyennes de ces zones on a une idée globale du potentiel chiffre mystère.

Features 3 : Filtres de Sobel

Pour affiner notre détection du chiffre mystère, on peut appliquer des filtres de convolution à notre image. C'est là qu'intervient la notion de filtre de Sobel permettant de capter les variations de gradient de façon plus localisée. Nous indiquant où se situe des contours droits verticaux ou horizontaux.

Enfin on peux concaténer nos features pour la finalisation.

Maintenant que nous avons un moyen assez fin pour que notre machine devine le chiffre mystère sur l'image et la répertorie dans la bonne classe. On veut pouvoir le faire pour toute les images de notre dataset. D'où la section suivante :

3.2 Preprocessing global

Avec Python et plus particulièrement la librairie `sci-kit` on peut automatiser notre processus de pré-traitement local sur toute notre database. Pour cela il suffit d'utiliser une *pipeline*, c'est un objet qui chaîne une suite ordonnée d'étape.

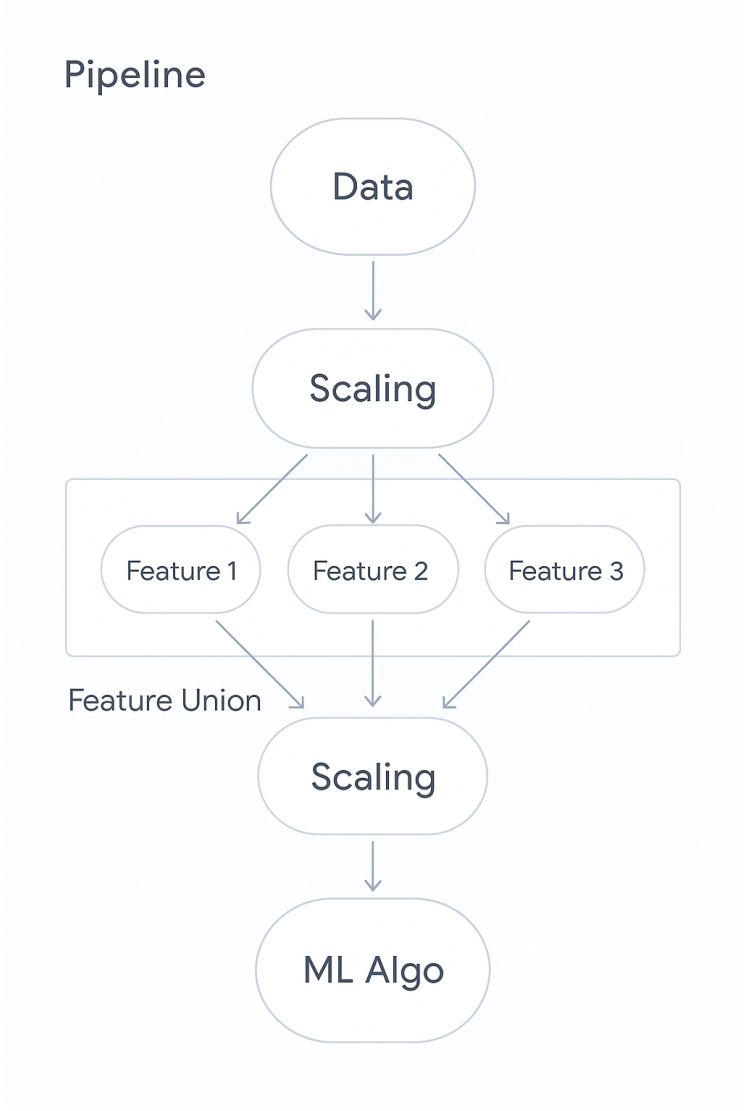

FIGURE 7 – Schéma de la pipeline

4 Apprentissage machine

Tout d'abord on décompose notre database en 2 sous-ensembles.

— l'ensemble `training`, là où la machine apprendra à deviner un chiffre.

— l'ensemble `test`, là où la machine essaiera de prédire, donc de classifier un individus.

Remarquons tout d'abord que même en partitionnant notre database, on a une répartition des classes qui est quasi-uniforme sur chaque ensemble :

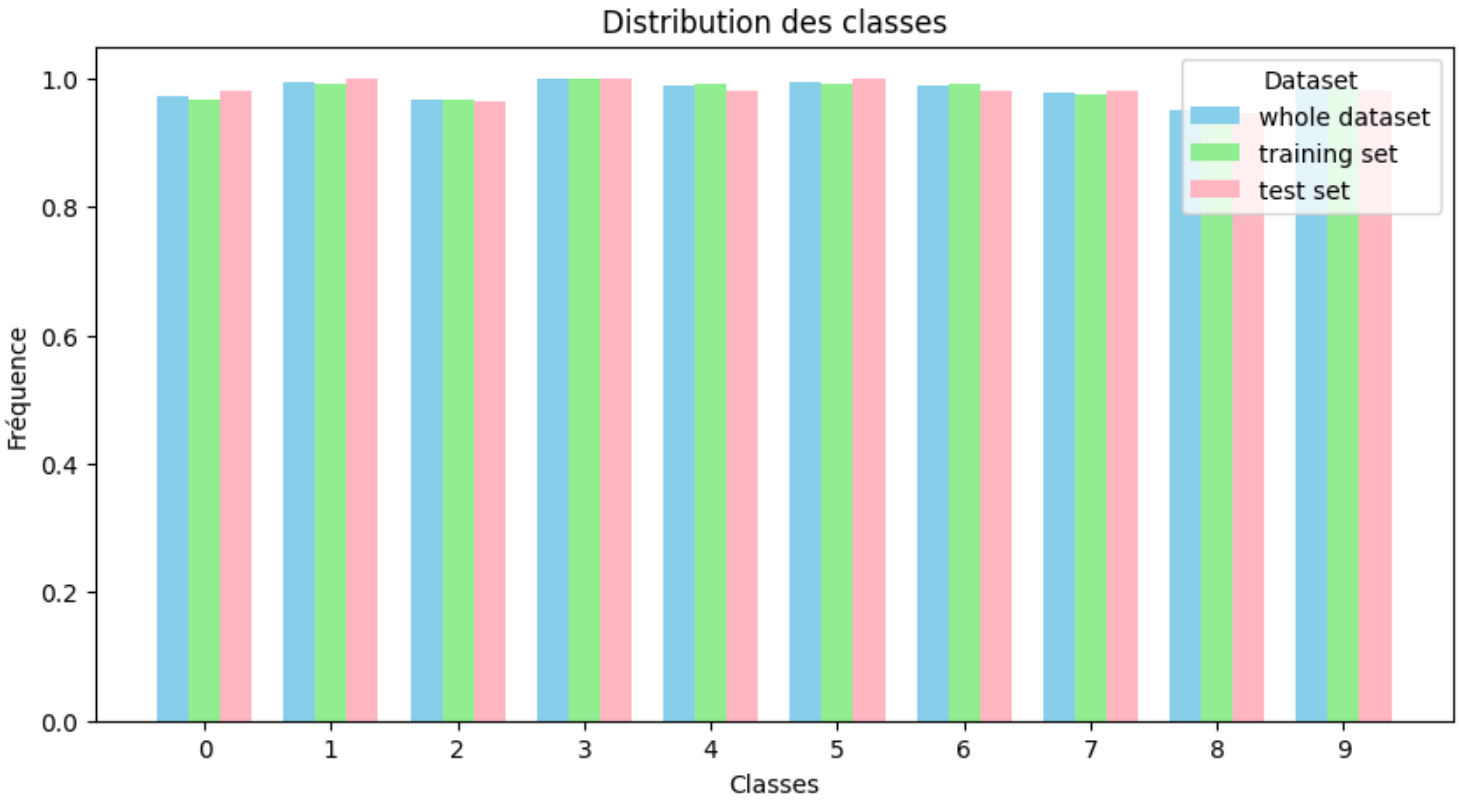

FIGURE 8 – Distribution des classes pour la database scindée

On a une preuve visuelle que nos distributions pour le `training` et `test` sont assez proche. Il existe cependant une métrique de probabilité pouvant appuyer ce propos : la distance de Bhattacharyya. Nous l'avons implémenté et l'écart entre nos deux distributions est de l'ordre de 10^{-4}. Donc on peut considérer qu'elles sont proches voir quasi égale.

Nous avons réalisé 2 modèles :

- Modèle 1 : voir la pipeline avec le preprocessing local.
- Modèle 2 : semblable au modèle 1, mais en ajoutant un feature d'ACP local.

Pour le deuxième modèle nous avons créer des sous-databases contenant uniquement les membres d'une même classe afin de réduire le bruit et les erreurs dans la reconnaissance d'image.
Après entraînement on compare les résultats obtenu avec les matrices de confusion démontrant les performances de classifications des 2 modèles. On a gardé le meilleur modèle qui est le deuxième :
Une fois les meilleurs paramètres calculés avec les meilleurs estimateurs on obtient :

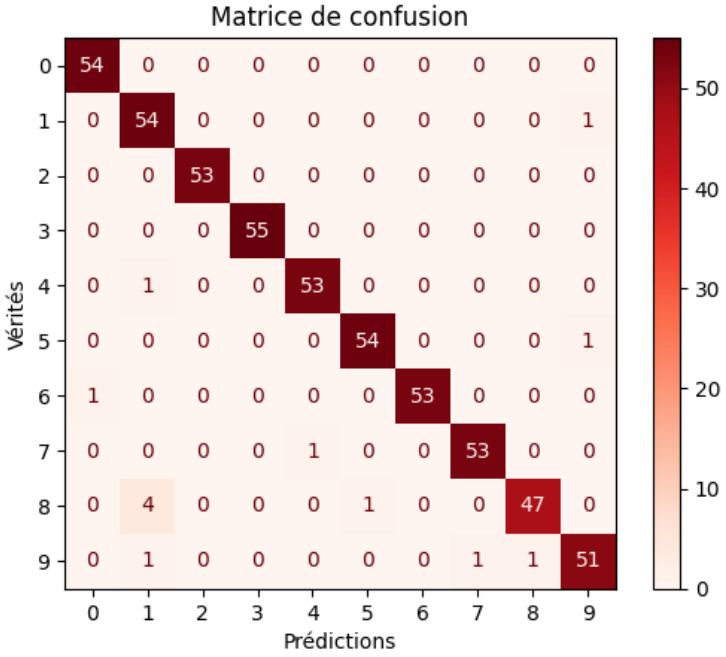

FIGURE 9 – Matrice de confusion du modèle 2

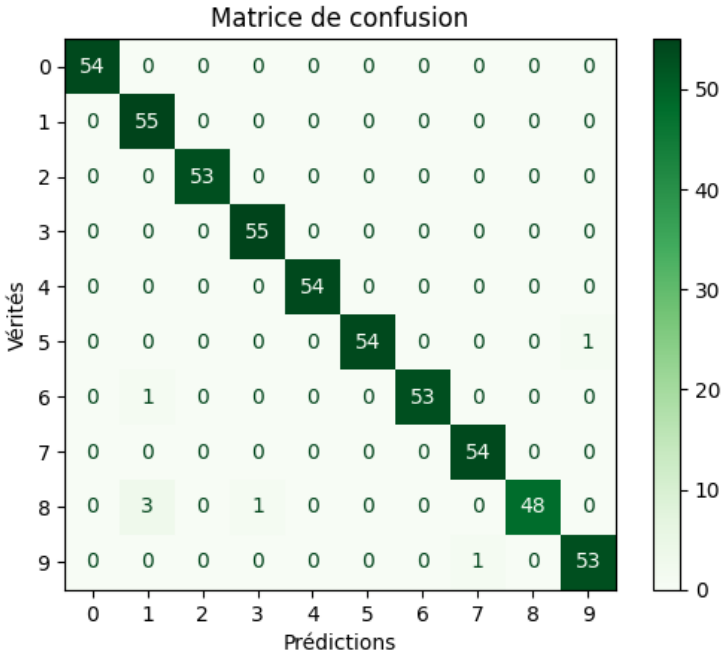

FIGURE 10 – Matrice de confusion du modèle 2 optimisé

Pour le modèle 2 non optimisé on a :

- Précision sur l'ensemble `test` : 0.9759
- Précision sur l'ensemble `training` : 1
- 527 correctement prédit sur 540

Tandis que pour le modèle 2 optimisé on a :

- Précision sur l'ensemble `test` : 0.9889
- Précision sur l'ensemble `training` : 1
- 534 correctement prédit sur 540

La séparation linéaire entre les classe se base sur le principe de :

5 OvO vs OvR

On appelle *One-versus-One* que l'on note OvO la façon d'entraîner un classifieur à différencier les individus par paire de classe.
Là où un *One-versus-Rest* noté OvR entraîne un classifieur à distinguer des individus en considérant toute les classes.
Voici une façon simple de le visualiser :

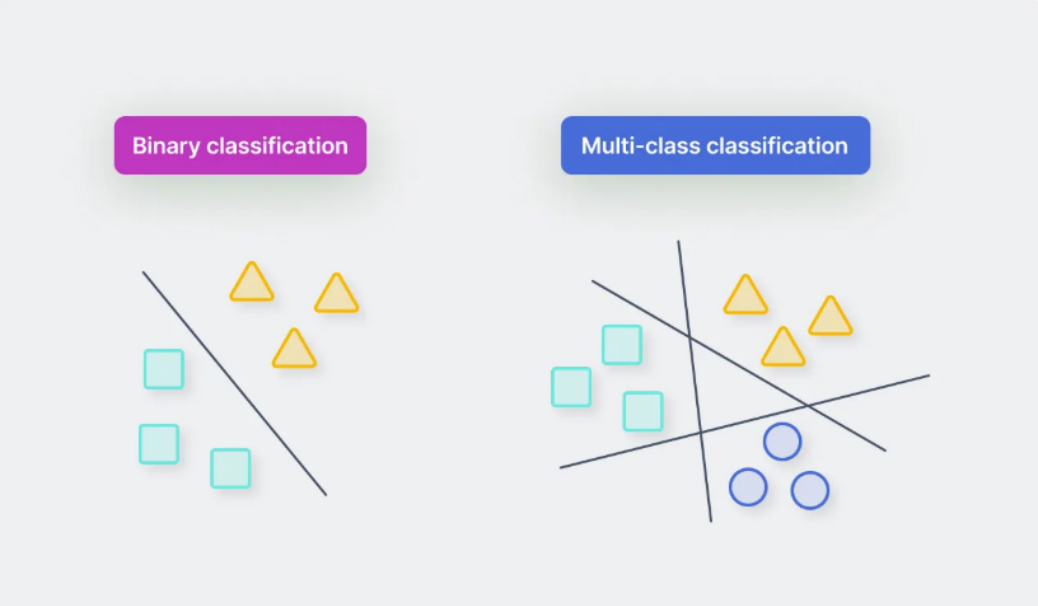

FIGURE 11 – OvO vs OvR

6 Réseau de neurones

7 Conclusion

Nous avons vu qu'il était bien plus simple d'implémenter un réseau de neurones, donc si on arrive à l'affiner assez cette méthode est à prioriser. Dans notre cas nous n'avons pas eu le temps de réaliser une telle optimisation et notre deuxième modèle de ML est plus précis que notre réseau de neurone.