Sorbonne Université

Projet Réseaux Neuronaux 2025-2026

Étudiant :
WANG, Fei

Professeur :
Annick Valibouze

5 janvier 2026

Table des matières

1 L'apprentissage supervisé du PMC

1.1 Introduction

Le PMC est un modèle capable de traiter des variables complexes et adapté aux relations non linéaires, ce qui le rend particulièrement approprié pour le traitement d'ensembles de données étiquetées. Afin d'étudier la manière dont le PMC effectue l'apprentissage supervisé, nous avons sélectionné un ensemble de données étiquetées `Thyroid_Diff.csv`, pour accomplir la tâche consistant à prédire si un cancer de la thyroïde va récidiver (un problème de classification binaire

1.2 Données et pré-traitement

Nous utilisons le fichier `Thyroid_Diff.csv`, contenant $n = 383$ observations sans valeurs manquantes. La variable cible `Recurred` correspond à une classification binaire (recurred / non-recurred). Les variables explicatives sont au nombre de $p = 16$ et comprennent à la fois des variables continues (par exemple l'âge) et des variables catégorielles (stade, anatomopathologie, fonction thyroïdienne, etc.). Afin de fournir une entrée numérique au perceptron multicouche (PMC) ainsi qu'aux méthodes statistiques comparées, nous appliquons un encodage one-hot sur les variables catégorielles. Après encodage, la dimension du vecteur de caractéristiques devient $d = 40$, et chaque individu est représenté par $x \in \mathbb{R}^d$.

Pour évaluer la capacité de généralisation, nous séparons les données en un ensemble d'entraînement et un ensemble de test selon un ratio 7 :3, en privilégiant un échantillonnage stratifié afin de conserver des proportions de classes similaires. Ensuite, nous standardisons les variables (z-score) :

$$x'_j = \frac{x_j - \mu_j}{\sigma_j},$$

où μ_j et σ_j sont estimés uniquement sur l'ensemble d'entraînement puis appliqués à l'ensemble de test. Cette procédure évite la fuite d'information (data leakage) et place les variables sur une échelle comparable, ce qui stabilise l'optimisation par descente de gradient.

1.3 Principe de l'apprentissage supervisé du PMC

1.3.1 Introduction

Le perceptron multicouche (PMC) est constitué d'une couche d'entrée, d'une couche cachée et d'une couche de sortie. La couche d'entrée reçoit un vecteur $x \in \mathbb{R}^d$ (ici $d = 40$ après one-hot). La couche cachée contient $h = 8$ neurones (`size`=8) et introduit une non-linéarité via une fonction d'activation $\phi(\cdot)$, ce qui permet de modéliser des relations non linéaires. La couche de sortie comporte 2 neurones et utilise la fonction softmax afin de produire des probabilités de classe.

1.3.2 Propagation avant

À partir de x, la couche cachée calcule :

$$z = W^{(1)}x + b^{(1)}, \qquad a = \phi(z).$$

La couche de sortie produit ensuite des logits u puis des probabilités via softmax :

$$u = W^{(2)}a + b^{(2)}, \qquad \hat{p}_k = \frac{e^{u_k}}{\sum_{r=1}^{2} e^{u_r}}, \quad k \in \{1, 2\}.$$

Ainsi, $\hat{p} = (\hat{p}_1, \hat{p}_2)$ et $\hat{p}_2$ peut être interprété comme $P(y = \text{Recurred} \mid x)$.

1.3.3 Fonction de perte et optimisation

En apprentissage supervisé, l'objectif est de rapprocher la prédiction des étiquettes réelles. Pour une classification binaire, on utilise classiquement l'entropie croisée :

$$\mathcal{L}_{CE} = -\sum_{k=1}^{2} y_k \log(\hat{p}_k),$$

où y est l'étiquette one-hot et $\hat{p}$ la sortie du réseau. Les paramètres sont mis à jour par rétropropagation du gradient et une méthode de descente de gradient, de façon itérative, afin de minimiser la perte.

Pour réduire le surapprentissage, nous ajoutons une régularisation L_2 contrôlée par le paramètre `decay` :

$$\mathcal{L} = \mathcal{L}_{CE} + \lambda \|\theta\|_2^2,$$

où θ représente l'ensemble des poids du réseau et $\lambda > 0$ règle l'intensité de la pénalisation. Cette pénalisation limite l'amplitude des poids et améliore la robustesse sur les données de test.

1.3.4 Backpropagation

Après la propagation avant, le réseau fournit des probabilités $\hat{p}$. La fonction de perte mesure l'écart entre la prédiction et l'étiquette réelle y. Pour la perte totale, La rétropropagation consiste à calculer, via la règle de dérivation en chaîne, les gradients de la perte par rapport aux paramètres du réseau :

$$\frac{\partial \mathcal{L}}{\partial W^{(2)}}, \quad \frac{\partial \mathcal{L}}{\partial b^{(2)}}, \quad \frac{\partial \mathcal{L}}{\partial W^{(1)}}, \quad \frac{\partial \mathcal{L}}{\partial b^{(1)}}.$$

Intuitivement, l'erreur est d'abord produite à la sortie (désaccord entre $\hat{p}$ et y), puis elle est propagée couche par couche vers l'amont afin d'attribuer à chaque poids sa contribution à l'erreur (le gradient).

Les paramètres sont ensuite mis à jour par descente de gradient, avec un taux d'apprentissage η :

$$W \leftarrow W - \eta \frac{\partial \mathcal{L}}{\partial W}, \qquad b \leftarrow b - \eta \frac{\partial \mathcal{L}}{\partial b}.$$

1.4 Résultats du PMC

Sur l'ensemble de test ($n = 116$), la matrice de confusion du MLP donne : $TN = 83$, $FP = 0$, $FN = 4$, $TP = 29$. Le modèle ne produit aucun faux positif ($FP = 0$), d'où une spécificité de 100% :

$$\text{Specificity} = \frac{TN}{TN + FP} = \frac{83}{83 + 0} = 1.00.$$

Pour la classe *Recurred*, il détecte 29 cas mais en manque 4, soit une sensibilité de 87.9% :

$$\text{Sensibilité} = \frac{TP}{TP + FN} = \frac{29}{29 + 4} = 0.879.$$

L'exactitude globale est de 96.6% :

$$\text{Exactitude} = \frac{TP + TN}{n} = \frac{29 + 83}{116} = 0.966.$$

	Pred	
True	Non.Recurred	Recurred
Non.Recurred	83	0
Recurred	4	29

FIGURE 1 – Matrice de confusion

2 Comparaison avec LDA

2.1 L'introduction de la LDA(méthode statistique)

La LDA (*analyse discriminante linéaire*) est une méthode de classification **statistique** : elle part d'un modèle probabiliste des données, puis choisit la classe la plus probable pour un individu x.

— On suppose que, **dans chaque classe** $(y = k)$, les observations x sont réparties "autour d'un centre" et suivent une forme en nuage proche d'une loi normale :

$$x \mid (y = k) \sim \mathcal{N}(\mu_k, \Sigma).$$

— Chaque classe a son propre centre μ_k (moyenne), mais on suppose que la **dispersion** des données est comparable entre classes (même matrice Σ).

— On estime alors ces quantités sur l'ensemble d'entraînement (les centres de chaque classe et la dispersion globale), puis pour un nouveau x on calcule une probabilité *a posteriori* :

$$P(y = k \mid x) \;\propto\; P(y = k)\, f_k(x),$$

où $P(y = k)$ est la proportion de la classe k dans les données et $f_k(x)$ la densité normale associée à la classe k.

— La règle de décision est simplement :

$$\hat{y}(x) = \arg\max_k P(y = k \mid x),$$

c'est-à-dire **on prédit la classe la plus probable**.

Pour $k \in \{0, 1\}$, la décision revient à comparer $\delta_1(x)$ et $\delta_0(x)$. En posant

$$w = \Sigma^{-1}(\mu_1 - \mu_0), \qquad b = -\frac{1}{2}\left(\mu_1^\top \Sigma^{-1}\mu_1 - \mu_0^\top \Sigma^{-1}\mu_0\right) + \log\frac{\pi_1}{\pi_0},$$

on obtient une règle très simple :

$$\text{prédire 1 si } w^\top x + b \geq 0, \qquad \text{sinon prédire 0.}$$

La frontière de décision est donc donnée par l'équation

$$w^\top x + b = 0,$$

c'est-à-dire un **hyperplan**. Intuitivement, la LDA projette x sur une direction w et coupe l'espace en deux régions.

2.2 LDA vs PMC

2.2.1 Éléments évalués

Nous comparons le PMC et la LDA sur l'ensemble test ($n = 116$) à l'aide de : (i) la courbe ROC (ii) la matrice de confusion (iii) une visualisation PCA du test avec *vrai* (forme) et *prédit* (couleur) .

2.2.2 Résultats du PMC

Nous avons récemment discuté de la performance de PMC sur le jeu de test et avons obtenu la matrice de confusion. Sur l'ACP du test , les erreurs se concentrent dans la zone de recouvrement entre les deux groupes, ce qui est cohérent avec un problème partiellement non séparé sur les deux premières composantes.

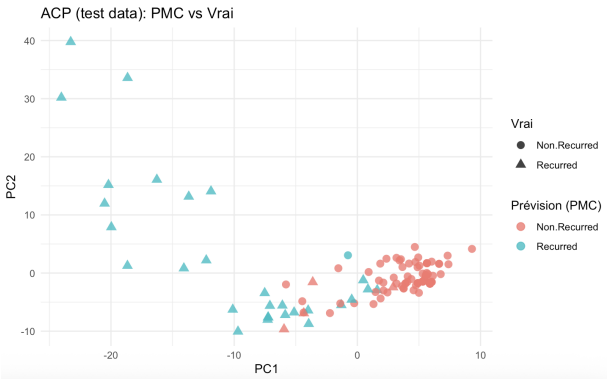

FIGURE 2 – ACP—PMC

2.2.3 Résultats de la LDA

D'après la matrice de confusion , on obtient :

$$TN = 82,\ FP = 1,\ FN = 5,\ TP = 28.$$

La spécificité reste élevée mais n'est plus parfaite :

$$\text{Specificity}_{LDA} = \frac{82}{82+1} = 0.988.$$

La sensibilité est légèrement plus faible :

$$\text{Sensitivity}_{LDA} = \frac{28}{28+5} = 0.848.$$

L'exactitude globale est :

$$\text{Accuracy}_{LDA} = \frac{82+28}{116} = 0.948.$$

Sur l'ACP du test , on observe davantage de points *Recurred* prédits comme *Non.Recurred*, ce qui correspond à l'augmentation de FN.

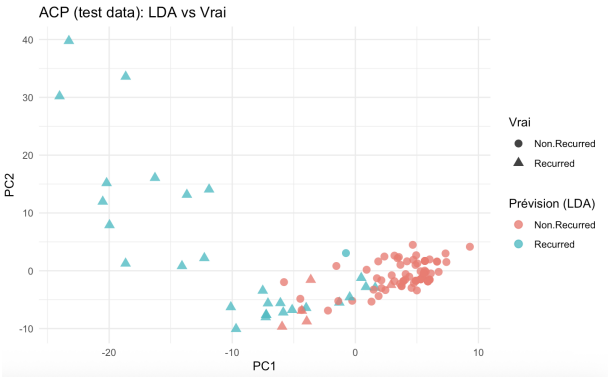

FIGURE 3 – ACP—LDA

2.2.4 Analyse ROC et interprétation

La comparaison ROC suggère des performances globales proches. Cependant, au seuil 0.5 utilisé pour la décision, le PMC commet moins d'erreurs : il ne génère aucun faux positif (FP=0) et il réduit légèrement les faux négatifs par rapport à la LDA. Concrètement, il identifie mieux les cas de récidive tout en évitant les fausses alertes.

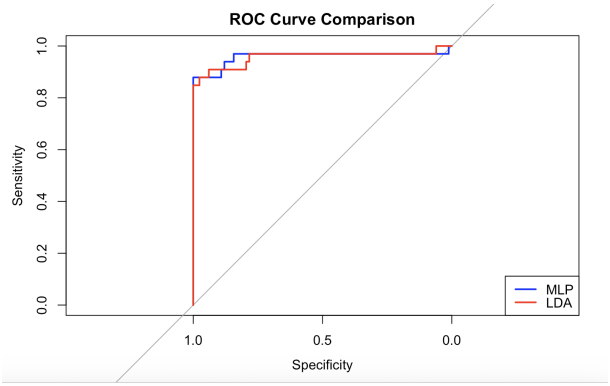

FIGURE 4 – ROC

2.3 Conclusion

Sur ce jeu de données, le PMC surpasse légèrement la LDA sur l'ensemble test (meilleure sensibilité et meilleure exactitude, avec spécificité supérieure). La LDA est une méthode statistique simple et interprétable, mais sa contrainte linéaire peut limiter la performance lorsque la séparation réelle entre classes n'est pas strictement linéaire.

3 Réseau de neurones avec Keras

3.1 Introduction

Dans cette partie, nous introduisons un modèle de régression basé sur le framework `Keras` (package `keras3`), afin de prédire le prix des logements.Ce choix vise à illustrer une implémentation plus flexible des réseaux neuronaux, et à la comparer à la PMC , dont la structure reste relativement limitée. En tant qu'interface de haut niveau, Keras permet de définir de manière explicite l'architecture du réseau, le processus d'apprentissage ainsi que les stratégies de régularisation, ce qui le rend particulièrement adapté aux tâches de régression plus complexes.

3.2 Données et prétraitement

Le jeu de données utilisé dans cette étude contient 545 observations, chacune correspondant à un logement décrit par plusieurs caractéristiques ainsi que son prix. Les variables explicatives comprennent à la fois des variables continues (telles que la surface ou le nombre de pièces) et des variables catégorielles binaires ou multinomiales (par exemple la présence d'équipements ou l'état d'ameublement).

Étant donné que les réseaux neuronaux ne peuvent traiter que des entrées numériques, toutes les variables catégorielles ont été transformées par encodage one-hot. Après cette transformation, le jeu de données comporte 13 variables explicatives, qui constituent les entrées du réseau neuronal. L'ensemble des variables a ensuite été standardisé afin d'obtenir une moyenne nulle et une variance unitaire, ce qui permet d'améliorer la stabilité du processus d'apprentissage.

Les données ont été divisées aléatoirement en un ensemble d'apprentissage (70%) et un ensemble de test (30%), afin d'évaluer les performances du modèle sur des données non observées lors de l'entraînement.

3.3 Principe du réseau neuronal Keras

Le modèle Keras utilisé dans ce projet repose sur une architecture similaire à celle de la PMC. Soit un vecteur d'entrée

$$\mathbf{x} \in \mathbb{R}^{13},$$

correspondant aux 13 variables explicatives issues de l'encodage one-hot et de la standardisation.

Chaque couche du réseau réalise une transformation affine suivie d'une fonction d'activation non linéaire. Pour la couche l, la sortie s'écrit :

$$\mathbf{h}^{(l)} = f\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\right),$$

où $\mathbf{W}^{(l)}$ et $\mathbf{b}^{(l)}$ désignent respectivement la matrice de poids et le vecteur de biais associés à la couche l.

Les couches cachées utilisent la fonction d'activation ReLU :

$$\mathrm{ReLU}(z) = \max(0, z),$$

qui, comparée aux fonctions sigmoid ou tanh couramment utilisées dans la PMC, est moins sujette au problème de disparition du gradient dans les réseaux profonds, améliorant ainsi l'efficacité et la stabilité de l'apprentissage.

La couche de sortie est linéaire et permet de prédire la variable continue $\hat{y}$, correspondant au prix du logement. L'objectif de l'apprentissage est de minimiser l'erreur quadratique moyenne (MSE) :

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

L'optimisation des paramètres est réalisée à l'aide de l'algorithme Adam, qui combine une estimation adaptative du taux d'apprentissage et un mécanisme de momentum, permettant une convergence plus rapide et plus stable que la descente de gradient classique.

3.4 Dropout et Early Stopping

Afin de réduire le risque de surapprentissage, une régularisation de type *dropout* est introduite dans le réseau Keras. Lors de l'apprentissage, chaque neurone est désactivé aléatoirement avec une probabilité de 20%, ce qui empêche le modèle de dépendre excessivement de certaines connexions locales.

Par ailleurs, une stratégie d'arrêt anticipé (*early stopping*) est mise en place durant l'entraînement. Lorsque la perte sur l'ensemble de validation ne s'améliore plus pendant plusieurs itérations consécutives, l'apprentissage est interrompu automatiquement, évitant ainsi un ajustement excessif aux données d'apprentissage. Cette stratégie contribue de manière significative à la stabilité du modèle.

3.5 Analyse des résultats et performances du modèle

Les résultats obtenus sur l'ensemble de test montrent que le modèle Keras atteint un coefficient de détermination d'environ

$$R^2 = 0.65,$$

ce qui signifie qu'environ 65% de la variance du prix des logements est expliquée par le modèle.

Le graphique comparant les valeurs réelles et prédites indique que la majorité des points se situe à proximité de la diagonale, traduisant une bonne capacité du réseau à capturer la tendance globale des données.

Cependant, une dispersion plus marquée est observée pour les logements à prix élevé, suggérant que le modèle reste limité dans la prédiction des valeurs extrêmes.

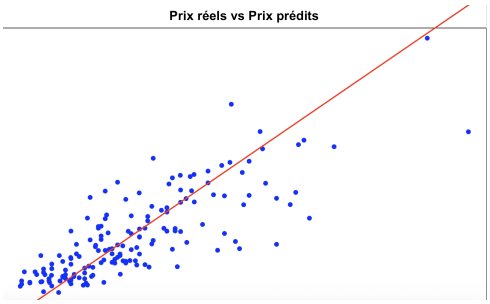

FIGURE 5 – Prix réels vs Prix prédits

4 Carte de Kohonen

4.1 Introduction

La segmentation de la clientèle consiste à diviser un marché en plusieurs groupes de clients présentant des caractéristiques similaires. Elle permet d'identifier plus clairement les différences de besoins entre groupes, afin de soutenir des décisions plus précises en matière de conception de produits, de services et de marketing.

Dans cette étude, nous utilisons des données de clients d'un supermarché (membres) contenant des informations de base (par exemple identifiant client, âge, sexe, revenu annuel, etc.). Nous appliquons une méthode non supervisée, la carte auto-organisatrice de Kohonen (*Self-Organizing Map, SOM*), afin de réaliser une visualisation en deux dimensions et d'analyser la structure de similarité entre clients.

4.2 Données et Prétraitement

Le jeu de données contient environ 2000 observations et provient de Kaggle : https://www.kaggle.com/datasets/dev0914sharma/customer-clustering. Chaque observation correspond à un client et inclut des informations socio-démographiques telles que le sexe (*Sex*), le statut marital (*Marital status*), l'âge (*Age*), le niveau d'éducation (*Education*), le revenu (*Income*), la profession (*Occupation*) et la taille du lieu de résidence (*Settlement size*).

Les données contiennent plusieurs variables catégorielles (par exemple *Sex*, *Marital status*, *Education*, *Occupation*, *Settlement size*). Même si certaines modalités sont codées sous forme numérique (0/1/2), elles représentent des catégories et ne doivent pas être interprétées comme des valeurs continues ni comme un ordre réel. Nous appliquons donc un encodage *one-hot* afin de transformer ces variables en indicatrices 0/1. Après cet encodage, on obtient une matrice d'entrée X composée de 12 variables (caractéristiques).

Nous standardisons les variables continues (par exemple *Age* et *Income*). Sans cette étape, la distance euclidienne peut être dominée par les variables de grande échelle (le revenu a souvent une amplitude bien plus grande que l'âge), ce qui ferait organiser la SOM principalement selon le revenu et réduirait l'influence des autres variables.

Dans les visualisations de type *property map*, les valeurs affichées correspondent donc à des niveaux relatifs (valeurs standardisées), et non aux unités originales des variables.

4.3 Principe de la carte de Kohonen

La carte de Kohonen est une méthode de réseau de neurones non supervisée qui projette des données de grande dimension sur une grille bidimensionnelle. Elle construit une carte visuelle telle que :

— des observations similaires dans l'espace d'origine sont projetées sur des unités voisines sur la grille ;

— des observations différentes apparaissent dans des régions distinctes de la carte ;

— la carte met souvent en évidence des régions homogènes et des zones de transition, ce qui facilite l'interprétation des variations continues et des frontières potentielles entre profils.

4.3.1 Unités de la grille et vecteurs prototypes (codebook vectors)

La grille SOM est composée de plusieurs unités . Chaque unité i est associée à un vecteur prototype (vecteur de poids) de même dimension que l'entrée :

$$\mathbf{w}_i \in \mathbb{R}^p,$$

où p est le nombre de caractéristiques (ici $p = 12$). On peut interpréter $\mathbf{w}_i$ comme le « profil client typique » représenté par l'unité i.

4.3.2 BMU (Best Matching Unit) et projection des observations

Pour chaque observation $\mathbf{x}$, la SOM calcule la distance (souvent euclidienne) entre $\mathbf{x}$ et tous les prototypes :

$$d_i(\mathbf{x}) = \|\mathbf{x} - \mathbf{w}_i\|.$$

L'unité qui minimise cette distance est appelée BMU (*Best Matching Unit*) :

$$b = \arg\min_i \|\mathbf{x} - \mathbf{w}_i\|.$$

L'observation est alors projetée sur la position de son BMU sur la grille. Les visualisations de type *mapping* et *counts* décrivent ainsi la distribution des observations sur les BMU.

4.3.3 Mise à jour avec voisinage

Pendant l'apprentissage, la SOM met à jour non seulement le BMU, mais aussi ses unités voisines. Une forme standard de mise à jour est :

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \alpha(t)\, h_{b,i}(t)\, \big(\mathbf{x} - \mathbf{w}_i(t)\big),$$

où :

— $\alpha(t)$ est le taux d'apprentissage, qui diminue au cours du temps ;

— $h_{b,i}(t)$ est la fonction de voisinage, qui dépend de la distance sur la grille entre l'unité i et le BMU b (plus i est proche de b, plus la mise à jour est importante) ;

— au fil de l'entraînement, le rayon de voisinage diminue, et la carte passe d'une organisation grossière à une organisation plus fine et plus stable.

4.4 Analyse des plots

4.4.1 SOM – nombre d'observations par unité

Cette figure présente le nombre d'observations projetées sur chaque unité (neurone) de la grille SOM. Une couleur plus claire (jaune/blanc) indique qu'un plus grand nombre d'individus est affecté à l'unité, tandis qu'une couleur plus foncée (rouge) indique un effectif plus faible. On observe que les observations se concentrent principalement sur certaines unités situées à gauche et en haut de la grille : la distribution n'est donc pas uniforme et met en évidence des zones « denses » et des zones « rares ». Par ailleurs, quelques unités contiennent très peu d'observations, voire aucune (unités quasi vides). Cela suggère que les profils correspondants sont peu fréquents et peuvent représenter des zones de transition (frontières) ou des types de clients rares.

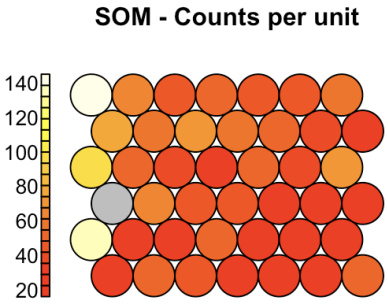

FIGURE 6 – Nombre d'observations par unité

4.4.2 SOM – Mapping (projection des observations)

La carte *Mapping* superpose la position de toutes les observations sur la grille SOM. La forte superposition de points noirs au sein d'une même unité est normale, car environ 2000 observations sont projetées sur un nombre limité d'unités de la grille. Cette visualisation est cohérente avec la carte des *Counts* : les zones où les points sont les plus concentrés correspondent aux unités à effectif élevé. Elle indique que la SOM regroupe des observations similaires dans la même unité ou dans des unités voisines, ce qui produit une structure bidimensionnelle interprétable.

4.4.3 SOM – U-matrix (matrice des distances de voisinage)

La *U-matrix* représente le niveau de dissimilarité entre les vecteurs prototypes (*codebook vectors*) des unités voisines. Une couleur plus claire correspond à une distance de voisinage plus élevée : l'unité diffère davantage de son voisinage et cette zone est souvent associée à une frontière entre groupes. À l'inverse, une couleur plus foncée indique des distances plus faibles, suggérant une région plus homogène où les unités sont similaires. Sur la figure, des zones plus claires apparaissent notamment au centre-bas et vers le bas-droit de la grille, ce qui peut indiquer des bandes de séparation entre différents profils. À l'inverse, les régions plus foncées situées à gauche et en haut semblent former une zone plus continue et relativement homogène.

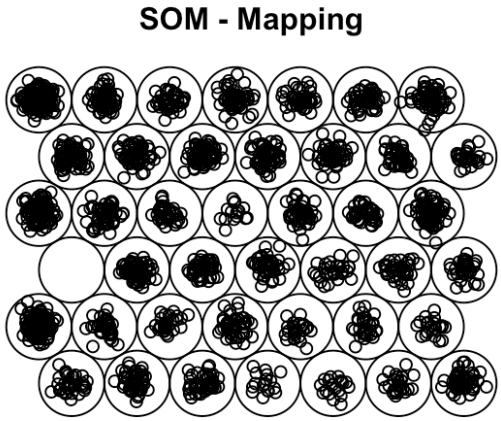

FIGURE 7 – Projection des observations

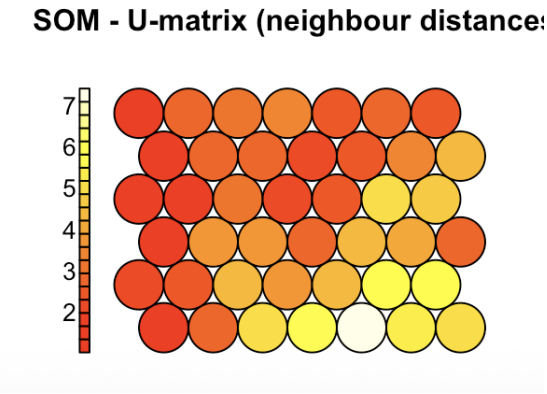

FIGURE 8 – Matrice des distances de voisinage

Les *property maps* représentent, pour chaque unité la grille SOM, la valeur correspondante dans le vecteur prototype.Nous choisissons les deux caractéristiques représentatives que sont le revenu et l'âge pour effectuer une analyse de property map.

Interprétation des property maps La *property map* de *Income* montre une zone plus claire au **centre–bas / bas** (revenus **relativement plus élevés**), tandis que le **haut** et la **gauche** sont plus foncés (revenus **plus faibles à intermédiaires**), ce qui suggère un **gradient**. De même, la carte de *Age* indique des âges **relativement plus élevés** vers la **droite**, surtout en **droite–bas**, et des âges **plus faibles** à gauche. Globalement, la SOM regroupe les individus similaires dans des zones proches : les variables continues varient progressivement, et les variables catégorielles révèlent des zones où certaines modalités sont plus fréquentes, avec des régions homogènes et des zones de transition.

5 Des réseaux antagonistes génératifs (GAN)

5.1 Introduction

GAN constitue une famille de modèles génératifs dont le principe est d'approximer la distribution des données réelles par apprentissage antagoniste. Leur objectif n'est pas d'effectuer une tâche de classification ou de régression, mais d'apprendre un mécanisme permettant de générer de nouveaux échantillons à partir d'un bruit aléatoire : étant donné un vecteur aléatoire z, le modèle peut synthétiser une image qui semble provenir des données réelles.

5.2 Principe

Un GAN repose sur deux réseaux de neurones entraînés de manière antagoniste : un générateur G et un discriminateur D. Le générateur transforme un vecteur de bruit $z \sim p_z(z)$ en un échantillon synthétique $G(z)$, tandis que le discriminateur assigne à une entrée x une probabilité

$$D(x) \in (0, 1),$$

interprétée comme la probabilité que x provienne de la distribution réelle $p_{\text{data}}(x)$. L'apprentissage prend la forme d'un jeu à somme nulle : D cherche à distinguer les données réelles des données générées, et G cherche à produire des échantillons susceptibles d'être classés comme réels.

5.2.1 Objectif du discriminateur

Dans une formulation de classification binaire, D observe :

— des échantillons réels $x \sim p_{\text{data}}(x)$ (étiquette 1) ;

— des échantillons générés $\tilde{x} = G(z)$ avec $z \sim p_z(z)$ (étiquette 0).

Le log-vraisemblance (à maximiser) du discriminateur s'écrit alors :

$$\mathcal{L}_D(D;G) = \mathbb{E}_{x\sim p_{\text{data}}}\big[\log D(x)\big] + \mathbb{E}_{z\sim p_z}\big[\log\big(1 - D(G(z))\big)\big].$$

L'entraînement de D consiste à résoudre :

$$\max_D \ \mathcal{L}_D(D;G).$$

5.2.2 Objectif du générateur et formulation min–max

Le générateur ne dispose pas de labels explicites ; il est optimisé uniquement via la rétropropagation du score de D. La formulation canonique de GAN est un problème min–max :

$$\min_G \max_D \ V(D,G),$$

où

$$V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}}\big[\log D(x)\big] + \mathbb{E}_{z\sim p_z}\big[\log\big(1 - D(G(z))\big)\big].$$

Intuitivement, D maximise V en poussant $D(x) \to 1$ pour les données réelles et $D(G(z)) \to 0$ pour les données générées, alors que G cherche à minimiser V en augmentant $D(G(z))$.

En pratique, on emploie fréquemment la perte dite *non saturante*, qui améliore la qualité des gradients en début d'apprentissage :

$$\min_G \ \mathcal{L}_G(G) = -\mathbb{E}_{z\sim p_z}\big[\log D(G(z))\big].$$

Cette variante est équivalente à la formulation min–max au point optimal, tout en étant généralement plus stable numériquement.

5.2.3 Discriminateur optimal

En fixant G, la distribution induite par les échantillons générés est notée $p_g(x)$. L'objectif s'écrit sous forme intégrale :

$$V(D,G) = \int \Big(p_{\text{data}}(x)\log D(x) + p_g(x)\log\big(1 - D(x)\big)\Big)\,dx.$$

Pour un x donné, on maximise la fonction

$$f(D) = p_{\text{data}}(x)\log D + p_g(x)\log(1 - D),$$

en annulant la dérivée :

$$\frac{\partial f}{\partial D} = \frac{p_{\text{data}}(x)}{D} - \frac{p_g(x)}{1-D} = 0 \quad \Longrightarrow \quad D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}.$$

Ainsi, au point optimal, $D^*(x)$ correspond à la proportion de densité réelle dans le mélange $(p_{\text{data}} + p_g)$.

5.2.4 Lien avec la divergence de Jensen–Shannon

En remplaçant D par D^* dans

$$V(D,G) = \int \Big[p_{\text{data}}(x) \log D(x) + p_g(x) \log \big(1 - D(x)\big) \Big] \, dx,$$

avec

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)},$$

on obtient d'abord l'expression développée :

$$V(D^*,G) = \int p_{\text{data}}(x) \, \log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)} \, dx + \int p_g(x) \, \log \frac{p_g(x)}{p_{\text{data}}(x) + p_g(x)} \, dx.$$

En introduisant la densité mélange

$$m(x) = \frac{1}{2}\big(p_{\text{data}}(x) + p_g(x)\big),$$

on réécrit

$$\log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)} = \log \frac{p_{\text{data}}(x)}{2m(x)} = \log \frac{p_{\text{data}}(x)}{m(x)} - \log 2,$$

et de même

$$\log \frac{p_g(x)}{p_{\text{data}}(x) + p_g(x)} = \log \frac{p_g(x)}{2m(x)} = \log \frac{p_g(x)}{m(x)} - \log 2.$$

En substituant et en utilisant $\int p_{\text{data}}(x)\,dx = \int p_g(x)\,dx = 1$, il vient :

$$\begin{aligned} V(D^*,G) &= \int p_{\text{data}}(x) \Big(\log \frac{p_{\text{data}}(x)}{m(x)} - \log 2 \Big) \, dx + \int p_g(x) \Big(\log \frac{p_g(x)}{m(x)} - \log 2 \Big) \, dx \\ &= -2 \log 2 + \int p_{\text{data}}(x) \log \frac{p_{\text{data}}(x)}{m(x)} \, dx + \int p_g(x) \log \frac{p_g(x)}{m(x)} \, dx \\ &= -\log 4 + \int p_{\text{data}}(x) \log \frac{p_{\text{data}}(x)}{m(x)} \, dx + \int p_g(x) \log \frac{p_g(x)}{m(x)} \, dx. \end{aligned}$$

Or, par définition de la divergence de Jensen–Shannon,

$$\mathrm{JS}(p_{\text{data}} \| p_g) = \frac{1}{2} \int p_{\text{data}}(x) \log \frac{p_{\text{data}}(x)}{m(x)} \, dx + \frac{1}{2} \int p_g(x) \log \frac{p_g(x)}{m(x)} \, dx,$$

d'où

$$\int p_{\text{data}}(x) \log \frac{p_{\text{data}}(x)}{m(x)} \, dx + \int p_g(x) \log \frac{p_g(x)}{m(x)} \, dx = 2 \, \mathrm{JS}(p_{\text{data}} \| p_g) \, .$$

Ainsi, on retrouve la forme classique :

$$V(D^*, G) = -\log 4 + 2\,\mathrm{JS}(p_{\text{data}} \parallel p_g)\,.$$

Comme $\mathrm{JS} \geq 0$ et que $\mathrm{JS} = 0$ si et seulement si $p_g = p_{\text{data}}$, minimiser $V(D^*, G)$ revient à rapprocher la distribution générée de la distribution réelle :

$$\min_G V(D^*, G) \iff \min_G \mathrm{JS}(p_{\text{data}} \parallel p_g)\,, \qquad p_g(x) = p_{\text{data}}(x) \text{ au minimum.}$$

Dans ce cas idéal, le discriminateur ne peut plus distinguer les deux sources et tend vers

$$D(x) = \frac{1}{2}.$$

6 XAI

6.1 Introduction

Cette partie présente un flux de travail logiciel XAI entièrement exécutable : entraînement d'un modèle de régression pour prédire le prix de vente d'une voiture d'occasion, puis application d'outils d'explicabilité afin d'identifier les variables dominantes, d'analyser l'influence locale des variables numériques sur un véhicule donné et de quantifier l'impact du changement de modalités pour les variables catégorielles.

6.2 Jeu de données (Used Car Price Prediction)

Nous utilisons le jeu de données Kaggle `Used_Car_Price_Prediction.csv`. Il contient 3750 observations, 17 variables explicatives et une variable cible `sale_price` (prix de vente/transaction). Lors du prétraitement, nous supprimons les champs manifestement inadaptés au rôle de caractéristiques (par exemple `car_name`, `variant`, `model`, `rto`, `registered_city`, `ad_created_on`, etc.), puis nous éliminons les valeurs manquantes. Après nettoyage, le modèle est entraîné à partir des variables restantes.

6.3 Principe (modèle et logique d'explication XAI)

Nous retenons une *forêt aléatoire* pour la régression du prix, puis nous utilisons des outils de XAI afin d'interpréter ce modèle. L'explication s'organise autour de **deux** niveaux complémentaires.

1.L'importance globale est estimée via l'importance par permutation.En conservant inchangées toutes les autres variables, on *permute aléatoirement* les valeurs d'une variable X_j (ce qui détruit son lien avec la cible), puis on mesure la dégradation de la performance du modèle, ici l'augmentation de la RMSE. Formellement, si RMSE désigne l'erreur sur un jeu d'évaluation et $\mathrm{RMSE}^{\pi(j)}$ l'erreur après permutation de X_j, l'importance de X_j est :

$$I_j \;=\; \mathrm{RMSE}^{\pi(j)} - \mathrm{RMSE}.$$

Une valeur I_j élevée signifie que la permutation de X_j augmente fortement l'erreur, ce qui indique que le modèle dépend fortement de cette variable.

2.La sensibilité locale est analysée au niveau d'un véhicule donné. En fixant un individu x, on fait varier une seule variable à la fois et l'on observe la réponse de la prédiction, les autres caractéristiques restant constantes. Pour une variable numérique x_j, on évalue

$$\hat{y}(t) = f\big(x_{-j}, x_j = t\big),$$

sur une grille de valeurs t. Pour une variable catégorielle x_j (p. ex. `fuel_type`, `transmission`, `body_type`), on remplace la modalité observée par d'autres modalités admissibles et l'on compare les variations de la prédiction à une modalité de référence, ce qui quantifie l'impact local du changement de catégorie sur le prix prédit, toutes choses égales par ailleurs.

6.4 Implémentation

L'analyse de sensibilité des variables numériques montre que le `original_price` est de loin le facteur le plus influent sur le prix prédit, tandis que l'année de fabrication (`yr_mfr`) a un effet modéré et que le kilométrage ainsi que le nombre de consultations ont un impact limité. Pour les variables catégorielles, le `body_type` et la `transmission` entraînent des variations marquées du prix prédit, alors que l'effet du `fuel_type` reste relativement faible. Les résultats d'importance par permutation confirment que le modèle repose principalement sur le prix initial, le type de carrosserie et l'année de fabrication pour estimer la valeur d'un véhicule d'occasion.

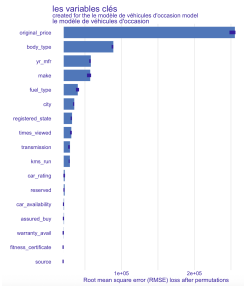

FIGURE 9 – RMSE loss après permutation

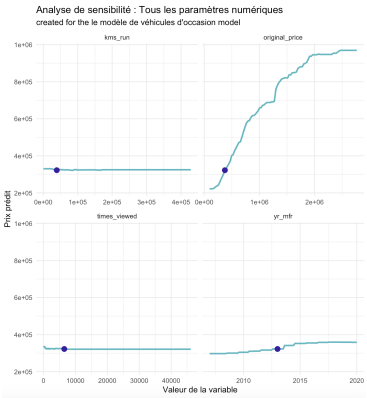

FIGURE 10 – Analyse de sensibilité : paramètres numériques

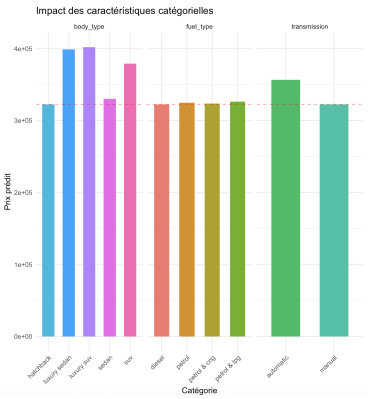

FIGURE 11 – Impact des caractéristiques catégorielles

7 Intégrité

Je déclare que ce travail a été réalisé de manière personnelle et indépendante. Les modèles, les analyses, les interprétations des résultats ainsi que les conclusions présentées dans ce rapport sont le fruit de mon propre travail.

Des outils d'intelligence artificielle ont été utilisés de manière ponctuelle et encadrée, notamment pour l'amélioration de la qualité rédactionnelle en langue française, la clarification de certaines formulations, ainsi que pour l'assistance à la mise en forme du code et à l'esthétique des visualisations. Ces outils n'ont en aucun cas été utilisés pour générer automatiquement les analyses, les résultats expérimentaux ou les conclusions scientifiques.

Toutes les sources de données et les méthodes employées sont explicitement mentionnées dans le rapport, et aucune forme de plagiat ou de reproduction non autorisée n'a été réalisée.

8 Conclusion

Dans ce projet, nous avons exploré plusieurs modèles issus de l'apprentissage supervisé, non supervisé et génératif afin d'analyser des données complexes sous différents angles. Le perceptron multicouche (PMC), appliqué à la prédiction de la récidive du cancer de la thyroïde, montre de bonnes performances et surpasse légèrement la LDA, notamment grâce à sa capacité à modéliser des relations non linéaires. La comparaison avec la LDA met en évidence le compromis classique entre interprétabilité statistique et flexibilité des modèles neuronaux.

L'implémentation d'un réseau neuronal avec Keras pour une tâche de régression illustre l'intérêt d'architectures plus flexibles, tout en soulignant les limites du modèle pour la prédiction des valeurs extrêmes. La carte de Kohonen permet, quant à elle, une visualisation structurée des données et révèle des profils latents cohérents, mettant en évidence des zones homogènes et des transitions progressives entre groupes.

Les réseaux antagonistes génératifs (GAN) offrent une perspective complémentaire, centrée sur l'apprentissage de la distribution des données, et illustrent comment un cadre théorique rigoureux peut conduire à une génération réaliste d'échantillons. Enfin, l'étude XAI appliquée à la prédiction du prix des véhicules d'occasion met en évidence l'importance de l'interprétabilité des modèles complexes. Les analyses globales et locales confirment que le prix initial, le type de carrosserie et l'année de fabrication constituent les principaux déterminants du prix prédit.

Dans l'ensemble, ce projet souligne l'intérêt de combiner performance prédictive, interprétabilité et visualisation afin d'obtenir une compréhension plus complète des données et des modèles.

ANNEXE

1.Le code de PMC et LDA

```
bc <- read.csv("~/Downloads/神经网络 Projet/Thyroid_Diff.csv",
                      stringsAsFactors = FALSE, check.names = TRUE)
bc$Recurred <- factor(bc$Recurred,
                                levels = c("No", "Yes"),
                                labels = c("Non.Recurred", "Recurred"))
pred_cols <- setdiff(names(bc), "Recurred")
char_cols <- pred_cols[sapply(bc[, pred_cols], is.character)]
bc[char_cols] <- lapply(bc[char_cols], factor)

X0 <- model.matrix(Recurred ~ . , data = bc)[, -1]
y   <- bc$Recurred
dim(X0)
colnames(X0)
ncol(bc) - 1
ncol(X0)

set.seed(123)
idx_1 <- which(y == "Recurred")
idx_0 <- which(y == "Non.Recurred")

train_idx <- c(
   sample(idx_1, floor(0.7 * length(idx_1))),
   sample(idx_0, floor(0.7 * length(idx_0)))
)

y_train <- y[train_idx]
y_test   <- y[-train_idx]
X_train0 <- X0[train_idx, , drop = FALSE]
X_test0   <- X0[-train_idx, , drop = FALSE]
X_train <- scale(X_train0)
X_test   <- scale(X_test0,
                        center = attr(X_train, "scaled:center"),
                        scale   = attr(X_train, "scaled:scale"))

pca <- prcomp(X_train, center = FALSE, scale. = FALSE)
pca_all <- rbind(
   data.frame(PC1 = pca$x[, 1], PC2 = pca$x[, 2],
                   Set = "Train", true = y_train),
   data.frame(PC1 = predict(pca, X_test)[, 1],
                   PC2 = predict(pca, X_test)[, 2],
                   Set = "Test", true = y_test))

#PMC
```

```
library(nnet)
y_train_ind <- class.ind(y_train)
colnames(y_train_ind) <- levels(y_train)

pmc <- nnet(x = X_train, y = y_train_ind,
                  size = 8, decay = 0.01, maxit = 1000,
                  softmax = TRUE, trace = TRUE)
p_mlp <- predict(pmc, X_test, type = "raw")[, "Recurred"]
pred_mlp <- factor(ifelse(p_mlp >= 0.5, "Recurred", "Non.Recurred"),
                           levels = levels(y_test))

#LDA
library(MASS)
lda_fit <- lda(x = X_train, grouping = y_train)
lda_out <- predict(lda_fit, X_test)

p_lda <- lda_out$posterior[, "Recurred"]
pred_lda <- factor(lda_out$class, levels = levels(y_test))

# ROC
library(pROC)
roc_mlp <- roc(y_test, p_mlp, levels = c("Non.Recurred","Recurred"), direction = "<")
roc_lda <- roc(y_test, p_lda, levels = c("Non.Recurred","Recurred"), direction = "<")
plot(roc_mlp, col = "blue", main = "ROC Curve Comparison")
plot(roc_lda, col = "red", add = TRUE)
legend("bottomright", legend = c("MLP", "LDA"), col = c("blue", "red"), lwd = 2)

test_pca <- subset(pca_all, Set == "Test")
test_pca$pred_mlp <- pred_mlp
test_pca$pred_lda <- pred_lda
ggplot(test_pca, aes(PC1, PC2, color = pred_mlp, shape = true)) +
  geom_point(size = 3, alpha = 0.8) +
  labs(title = "ACP (test data): PMC vs Vrai ",
       x = "PC1", y = "PC2",
       color = "Prévision (PMC)", shape = "Vrai") +
  theme_minimal()

ggplot(test_pca, aes(PC1, PC2, color = pred_lda, shape = true)) +
  geom_point(size = 3, alpha = 0.8) +
  labs(title = "ACP (test data): LDA vs Vrai ",
       x = "PC1", y = "PC2",
       color = "Prévision (LDA)", shape = "Vrai") +
  theme_minimal()
```

```
confusion_mlp <- table(True = y_test, Pred = pred_mlp)
print(confusion_mlp)
confusion_lda <- table(True = y_test, Pred = pred_lda)
print(confusion_lda)
```

2. Le code de Keras

```
bc <- read.csv("~/Downloads/神经网络 Projet/Housing.csv", stringsAsFactors = FALSE,
check.names = TRUE)
pred_cols <- setdiff(names(bc), "price")
char_cols <- pred_cols[sapply(bc[, pred_cols], is.character)]
bc[char_cols] <- lapply(bc[char_cols], factor)

X0 <- model.matrix(price ~ ., data = bc)[, -1]
y  <- bc$price
dim(X0)
colnames(X0)
head(X0)

set.seed(123)
train_idx <- sample(1:nrow(bc), floor(0.7 * nrow(bc)))
y_train <- y[train_idx]
y_test  <- y[-train_idx]
X_train0 <- X0[train_idx, , drop = FALSE]
X_test0  <- X0[-train_idx, , drop = FALSE]
X_train <- scale(X_train0)
X_test  <- scale(X_test0,
                 center = attr(X_train, "scaled:center"),
                 scale  = attr(X_train, "scaled:scale"))


library(keras3)
model <- keras_model_sequential() %>%
  layer_dense(units = 128, activation = 'relu', input_shape = ncol(X_train)) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dense(units = 1)

model %>% compile(
  loss = 'mse',
  optimizer = optimizer_adam(learning_rate = 0.1),
  metrics = c('mae')
)
```

```
history <- model %>% fit(
  X_train, y_train,
  epochs = 100,
  batch_size = 32,
  validation_data = list(X_test, y_test),
  verbose = 1,
  callbacks = list(callback_early_stopping(monitor = "val_loss", patience = 10))
)

predictions <- model %>% predict(X_test)
predictions_df <- data.frame(
  Actual_Price = y_test,
  Predicted_Price = predictions
)
print(predictions_df)



rss <- sum((predictions_df$Actual_Price - predictions_df$Predicted_Price)^2)
tss <- sum((predictions_df$Actual_Price - mean(predictions_df$Actual_Price))^2)
r_squared <- 1 - (rss / tss)
print(paste("R-squared: ", r_squared))


plot(predictions_df$Actual_Price,
       predictions_df$Predicted_Price,
       xlab = "Prix réels ",
       ylab = "Prix prédits",
       main = "Prix réels vs Prix prédits",
       pch = 16,
       col = "blue")

abline(0, 1, col = "red", lwd = 2)
```

3. Le code de la carte de Kohonen

```
library(kohonen)
library(cluster)
library(ggplot2)
library(readxl)
set.seed(123)
bc <- read.csv("~/Downloads/ 神 经 网 络 Projet/segmentation data.csv.xls",
stringsAsFactors = FALSE)
df <- bc[, c("Sex", "Marital.status", "Age", "Education",
             "Income", "Occupation", "Settlement.size")]
```

```
df <- na.omit(df)
cat_cols <- c("Sex", "Marital.status", "Education", "Occupation", "Settlement.size")
df[cat_cols] <- lapply(df[cat_cols], factor)
X <- model.matrix(~ . - 1, data = df)
X[, c("Age", "Income")] <- scale(X[, c("Age", "Income")])
dim(X)
head(X)

xdim <- 7
ydim <- 6
som_grid <- somgrid(xdim = xdim, ydim = ydim, topo = "hexagonal")
som_model <- som(
  X,
  grid = som_grid,
  alpha = c(0.05, 0.01),
  radius = 1.5,
  rlen = 1000,
  keep.data = TRUE
)

par(mfrow = c(2,2))
plot(som_model, type = "counts", main = "SOM - Counts per unit")
plot(som_model, type = "mapping", main = "SOM - Mapping")
plot(som_model, type = "dist.neighbours", main = "SOM - U-matrix (neighbour distances)")

par(mfrow = c(2,2))
plot(som_model, type = "property", property = som_model$codes[[1]][, "Income"],
     main = "Property map: Income")
plot(som_model, type = "property", property = som_model$codes[[1]][, "Age"],
     main = "Property map: Age")
plot(som_model, type = "property", property = som_model$codes[[1]][, "Education1"],
     main = "Property map: Education1")
```

4. Le code de XAI

```
library(DALEX)
library(randomForest)
library(tidyverse)
set.seed(123)
df <- read.csv("~/Downloads/神经网络Projet/Used_Car_Price_Prediction.csv", stringsAsFactors = TRUE)
data_clean <- df %>%
  select(-car_name, -variant, -model, -rto, -registered_city, -ad_created_on,
         -is_hot, -broker_quote, -emi_starts_from, -booking_down_pymnt,
```

```
-total_owners) %>%
  mutate(
    sale_price = as.numeric(sale_price),
    yr_mfr = as.numeric(yr_mfr),
    kms_run = as.numeric(kms_run),
    original_price = as.numeric(original_price),
    sale_price = as.numeric(sale_price)
  ) %>%

  filter(body_type != "") %>%
  filter(fuel_type != "") %>%
  filter(transmission != "") %>%
  na.omit()
cat("le nombre de caractéristiques :", ncol(data_clean) - 1)
cat("la liste des caractéristiques :", paste(colnames(data_clean)))

model <- randomForest(sale_price ~ ., data = data_clean, ntree = 100)
nrow(data_clean)
ncol(data_clean)
#XAI
explainer <- DALEX::explain(
  model = model,
  data = data_clean %>% select(-sale_price),
  y = data_clean$sale_price,
  label = "le modèle de véhicules d'occasion"
)

importance <- model_parts(explainer, B = 5)
plot(importance) +
  ggtitle("les variables clés")


car1 <- data_clean[3, ]
print(car1)

num_vars <- names(data_clean %>% select(where(is.numeric), -sale_price))
cp <- predict_profile(explainer,
                      new_observation = car1,
                      variables = num_vars)
plot(cp) +
  ggtitle("Analyse de sensibilité : Tous les paramètres numériques") +
  labs(y = "Prix prédit", x = "Valeur de la variable") +
  theme_minimal()
```

```
cat_vars <- c("fuel_type", "transmission", "body_type")
valid_vars <- intersect(cat_vars, colnames(data_clean))
cp_cat <- predict_profile(explainer,
                                    new_observation = car1,
                                    variables = valid_vars)
library(ggplot2)
plot_data <- as.data.frame(cp_cat) %>%
  pivot_longer(cols = all_of(valid_vars), values_to = "Category_Value") %>%
  filter(name == `_vname_`)
ggplot(plot_data, aes(x = Category_Value, y = `_yhat_`, fill = Category_Value)) +
  geom_col(width = 0.6) +
  facet_wrap(~ `_vname_`, scales = "free_x") +
  geom_hline(yintercept = predict(explainer, car1),
             linetype = "dashed", color = "red", alpha = 0.5) +
  ggtitle("Impact des caractéristiques catégorielles", ) +
  labs(y = "Prix prédit", x = "Catégorie") +
  theme_minimal() +
  theme(legend.position = "none",
        axis.text.x = element_text(angle = 45, hjust = 1))
```