Universite de Strasbourg

Intelligence artificielle

Licence 3 - UFR Mathématique - Informatique / Printemps 2023

Projet – Auto-encodeur et classification

Instructions :

- Vous pouvez travailler en binôme (pas de trinôme et plus tolérés)
- Pour l'ensemble des questions, il est permis de reprendre tout ou partie du code écrit en TP par les membres du binôme.
- Vous devrez rendre votre code et un compte rendu de 2 pages maximum au format `pdf` avant le 12 mai 2023 à 20h CET, dans l'espace Moodle prévu à cet effet (1 rendu par groupe). Compte tenu du calendrier resserré entre la date de rendu et la date à laquelle nous devons transmettre les notes à la scolarité, les projets rendus en retard ne seront pas acceptés.
- Vous devez justifier les réponses aux questions qui sont posées dans ce sujet. Toute réponse non justifiée ne sera pas prise en compte lors de la notation.
- Pas de Jupyter Notebook

Barème indicatif

Toutes les questions valent 1 point, à l'exception de l'apprentissage de l'auto-encodeur qui en vaut 2. Le barème se décompose ainsi de la façon suivante :

Données	7
Évaluation	3
Algorithmes (section 4.1)	6
Algorithmes (section 4.2)	4

1 Introduction

Dans ce projet, on se propose d'évaluer l'intérêt d'un certain type de réseaux de neurones pour la classification de données tabulées : les auto-encodeurs. En notant d la dimension des données considérées (*i.e.* le nombre d'attributs caractérisant chaque exemple, à l'exception de la vérité terrain), un auto-encodeur vise à construire une fonction

$$f_{\theta_1} : \mathbb{R}^d \to \mathbb{R}^{d'}$$

(où $d' < d$ et θ_1 sont les paramètres ajustables de f_{θ_1}) afin de pouvoir « compresser » les données. L'ensemble $\mathbb{R}^{d'}$ est alors appelé **espace latent** de l'auto-encodeur.

Comme il est difficile de construire une fonction f_{θ_1} qui compresse intelligemment les données, les auto-encodeurs appliquent la stratégie suivante : plutôt que de chercher explicitement un tel plongement, on introduit une fonction

$$g_{\theta_2} : \mathbb{R}^{d'} \to \mathbb{R}^d$$

et on considère la fonction

$$g_{\theta_2} \circ f_{\theta_1}$$

En ajustant θ_1 et θ_2 de sorte à faire tendre $g_{\theta_2} \circ f_{\theta_1}$ vers la fonction identité, on maximise nos chances d'obtenir un plongement f_{θ_1} pertinent. En effet, pour que la $g_{\theta_2} \circ f_{\theta_1}$ puisse approcher la fonction identité, il est nécessaire que toute l'information contenue dans les données initiales soit encodée dans l'espace latent, puisque g_{θ_2} ne dispose que de cette représentation réduite pour reconstruire la donnée initiale.

Dans le cadre de ce projet, l'auto-encodeur est un réseau de neurones à 3 couches (et donc 1 couche cachée) et dont les fonctions d'activation des deux dernières couches sont des tangentes hyperboliques. Contrairement à ce qui a été vu dans le TP sur les réseaux de neurones, l'entropie croisée ne peut pas être employée comme critère d'optimisation. Puisque nous cherchons à minimiser une distance entre les vecteurs d'entrée et de sortie de l'auto-encodeur, tous deux appartenant à $\mathbb{R}^d$, nous proposons d'employer l'erreur quadratique moyenne comme fonction de coût. La figure 1 résume les principales caractéristiques de cette architecture.

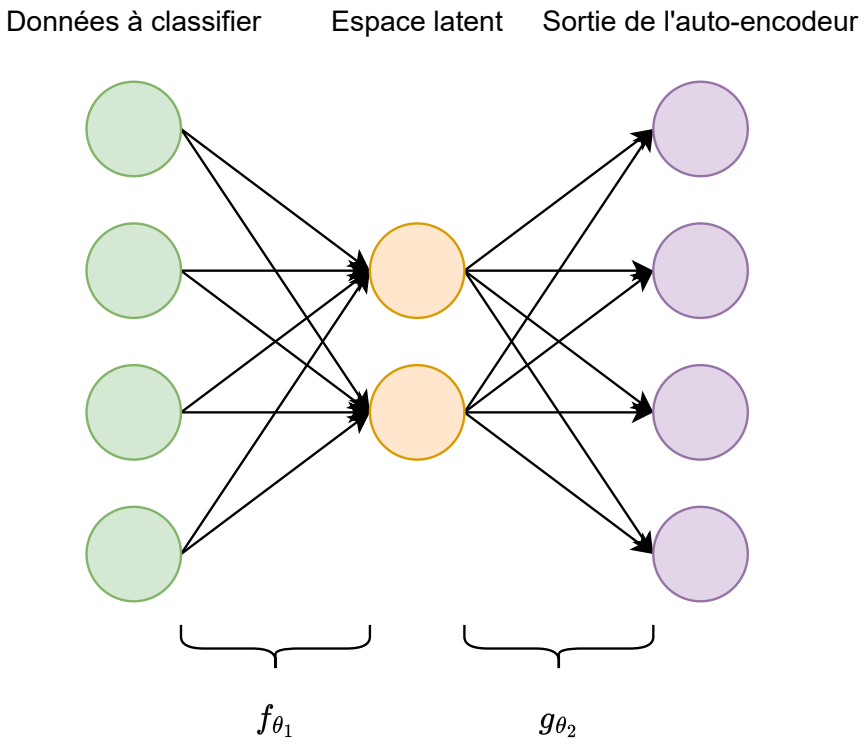

FIGURE 1 – Structure générale d' un auto-encodeur à 3 couches (le nombre de neurones par couche est donné à titre d'illustration et ne correspond pas nécessairement à ce qui est attendu dans ce projet).

2 Données

On se propose de travailler sur un *dataset* décrivant les caractéristiques et le prix de *smartphones*. Les données sont disponibles sur Moodle (fichiers `raw_train.csv` et `raw_test.csv`) et proviennent d'un dépôt Kaggle [1].

Notre objectif, avec cette banque de données, est alors de classifier le prix d'un téléphone en fonction de ses caractéristiques.

Travail à réaliser

1. En se rendant sur le site dont est issue la banque de données, expliquer comment est encodé le prix dans les fichiers CSV. Plus généralement, prendre le temps de comprendre la signification des différentes colonnes.
2. Indiquer la valeur de d pour les données utilisées dans ce projet.
3. Indiquer le nombre de classes représentées dans le *dataset*. On note K cette quantité.
4. Indiquer le nombre d'exemples dans chacune des classes et ce, pour chacun des jeux de données. Commenter
5. Concernant la visualisation des données, comme on dénombre une quantité relativement élevée d'attributs, il est difficilement envisageable d'utiliser la fonction `pairplot`. À la place, on se propose d'employer la fonction `kdeplot`, qui génère, pour un attribut donné, une estimation de la densité de probabilité que suivent les exemples selon cet attribut. Dans notre contexte, trois variables sont à renseigner : le *dataframe*, le nom de l'attribut étudié ainsi que celui donnant la classe de l'exemple (afin d'avoir une distribution pour chaque classe). Consulter la documentation pour déterminer le nom exact de ces paramètres, puis écrire une fonction qui produit ce type de visualisation pour chaque attribut.
6. En exploitant les différents résultats de `kdeplot`, déterminer celui qui représente le facteur le plus discriminant quant au prix du téléphone. Joindre une capture d'écran de cette visualisation dans le rapport.
7. Écrire une fonction permettant de normaliser les attributs des exemples entre -1 et +1. Le minimum du *dataset* (*i.e.* la réunion des jeux d'apprentissage de test) est envoyé sur -1 et le maximum sur +1, les autres valeurs étant déterminées par interpolation linéaire entre ces deux extrema.

3 Évaluation

À chaque fois qu'il sera question d'évaluer un modèle, ou encore de comparer plusieurs modèles, on s'appuiera sur sur les indicateurs suivants :

— L'exactitude pondérée EP (*weighted accuracy*)

— Le taux de bonne classification pour chaque classe i, noté T_i

1. `https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification`

Concrètement, en définissant la matrice de confusion $(C_{ij})_{i,j\in[\![1;K]\!]}$ (où K est le nombre de classes considérées) par

$$C_{ij} = \#\{\text{exemples de vérité terrain } i \text{ et de classe prédite } j\}$$

on a

$$EP = \frac{\sum_{i=1}^{K} C_{ii}}{\sum_{i=1}^{K}\sum_{j=1}^{K} C_{ij}}$$
$$T_i = \frac{C_{ii}}{\sum_{j=1}^{K} C_{ij}}$$

Travail à réaliser

1. Rappeler l'intérêt d'évaluer un modèle sur un jeu différent de celui ayant servi à l'apprentissage de ce même modèle.
2. Indiquer la valeur de chacun des T_i lorsque $EP = 1$.
3. Même question lorsque $EP = 0$

4 Algorithmes

Le travail demandé s'articule en deux temps :
- Apprentissage d'un auto-encodeur, pour obtenir une réalisation de f_{θ_1} ;
- Entraînement d'un arbre de décision binaire sur les données projetées dans l'espace latent.

4.1 Apprentissage de l'auto-encodeur

Travail à réaliser

1. Expliquer pourquoi l'entropie croisée n'est pas adaptée comme critère d'optimisation.
2. Compte tenu des valeurs prises par la fonction d'activation de la dernière couche, expliquer pourquoi il faut normaliser les données entre -1 et +1 pour que la sortie de l'auto-encodeur puisse être identique à son entrée.
3. Modifier la formule de rétro-propagation du gradient au sein de la dernière couche. En effet, l'expression

   $$A[L] - y$$

   proposée pour le calcul de l'erreur de la dernière couche n'est valable que pour la fonction d'activation *softmax* et le critère d'entropie croisée. Avec la fonction tanh et le critère MSE, cette expression devient

   $$-\frac{2}{d}(A[L] - y) \times (1 - A[L]^2)$$

4. Indiquer si l'apprentissage de l'auto-encodeur est supervisé ou non.
5. Réaliser l'entraînement de l'auto-encodeur dont l'espace latent est de dimension $d' = 2$ sur l'ensemble des données d'apprentissage et durant 100 époques et un pas d'apprentissage de 10^{-2}. Penser à employer l'erreur quadratique moyenne comme fonction de coût (fonction `MSE_cost` du fichier `utility.py`).

4.2 Apprentissage de l'arbre de décision dans l'espace latent

À présent, on souhaite construire un arbre de décision mais, contrairement à la section précédente, où l'on considérait directement les exemples par leurs attributs, on les traite désormais par le biais de leur représentation dans l'espace latent.

Travail à réaliser

1. Écrire une fonction qui calcule $f_{\theta_1}(x_i)$ pour tout élément x_i des deux jeux de données.
2. En reprenant l'algorithme d'apprentissage des arbres de décision binaires vu en TP, construire un arbre de décision à partir des représentations des données dans l'espace latent.
3. Indiquer pourquoi il est attendu que cet arbre soit moins profond qu'un arbre qui serait appris sur les données « brutes ».
4. Évaluer les performances de ce modèle sur le jeu de test.