

Devoir 1 - Partie pratique

- Ce devoir doit être fait et envoyé sur Gradescope individuellement. Vous pouvez discuter avec d'autres étudiants mais les réponses et le code que vous soumettez doivent être les vôtres. À noter que nous utiliserons l'outil de détection de plagiat de Gradescope. Tous les cas suspectés de plagiat seront enregistrés et transmis à l'Université pour vérification.
- La partie pratique doit être codée en python (avec les librairies numpy et matplotlib), et envoyée sur Gradescope sous fichier python. Pour permettre l'évaluation automatique, vous devez travailler directement sur le modèle donné dans le répertoire de ce devoir. Ne modifiez pas le nom du fichier ou aucune des fonctions signatures, sinon l'évaluation automatique ne fonctionnera pas. Vous pouvez bien sûr ajouter de nouvelles fonctions et importations python
- Les figures, courbes et parties pratiques du rapport doivent être envoyées au format pdf sur Gradescope. Pour le rapport il est recommandé d'utiliser un Jupyter notebook, en écrivant les formules mathématiques avec MathJax et en exportant vers pdf. Vous pouvez aussi écrire votre rapport en \LaTeX ; \LyX ; Word. Dans tout les cas, exportez votre rapport vers un fichier pdf que vous enverrez. Vous êtes bien sûr encouragés à vous inspirer de ce qui a été fait en TP.
- Vous devez soumettre vos solutions sur Gradescope en utilisant le devoir intitulé `Devoir 1 / Homework 1 - Pratique/Practical - 6390A/B` pour le code et `Devoir 1 / Homework 1 - Pratique/Practical - Rapport/Report - 6390A/B` pour le rapport.

Vous devez travailler sur le modèle `solution.py` du répertoire et compléter les fonctions basiques suivantes en utilisant numpy et python

Parzen avec fenêtre continue

Pour ce devoir nous utiliserons le dataset banknote authentication comme exemple. Il contient 1372 points (un par rangée), chacun avec 4 attributs (les 4 premières colonnes) et un label dans $\{0,1\}$ (la 5ème colonne). Il est recommandé de le télécharger [ici](#) puis de tester votre code en l'important comme ceci :

```
import numpy as np
banknote = np.genfromtxt('data_banknote_authentication.txt',
                        delimiter=',')
```

Quand le modèle de réponse dans solution.py contient "banknote" comme argument, vous pouvez considérer que cet argument est le jeu de données au format numpy. Votre fonction doit utiliser cet argument pour faire les calculs, et pas une version du jeu de données que vous auriez chargé vous-même

1. [4 points]

Question. Écrivez des fonctions qui prennent le jeu de données en entrée et retournent les statistiques suivantes:

- (a) `Q1.feature_means` : Un tableau contenant pour chaque attribut la moyenne de sa valeur sur l'ensemble des points du jeu de données. Faites attention à maintenir l'ordre des attributs.
e.g. : `Q1.feature_means(banknote) = [$\mu_1, \mu_2, \mu_3, \mu_4$]`
- (b) `Q1.covariance_matrix` : Une matrice 4×4 donnant la matrice de covariance empirique des attributs sur le jeu de données entier
- (c) `Q1.feature_means_class_1` : Un tableau contenant pour chaque attribut la moyenne de sa valeur sur l'ensemble des points dont le label est 1. Les classes possibles dans banknote sont 0 et 1.
e.g. : `Q1.feature_means_class_1(banknote) = [$\mu_1, \mu_2, \mu_3, \mu_4$]`
- (d) `Q1.covariance_matrix_class_1` : Une matrice 4×4 donnant la matrice de covariance empirique des attributs sur l'ensemble des points dont le label est 1.

2. [1 points]

Question. Implémentez la méthode de Parzen avec une fenêtre discrète de paramètre h . Utilisez la distance euclidienne classique sur les attributs du jeu de données. Votre solution doit avoir le comportement

suivant :

f = HardParzen(h) initialise l'instance de l'algorithme avec le paramètre h

f.train(X, Y) entraîne l'algorithme, où X est une matrice $n \times m$ de n exemples d'entraînement avec m attributs, et Y est un tableau contenant les n labels. Les labels sont représentés par des entiers, mais le nombre de classes dans Y peut varier.

f.compute_predictions(X_test) calcule les labels prédits et les retournent sous forme d'un tableau de même taille que X_test . X_test est une matrice $k \times m$ de k exemples de test avec le même nombre d'attributs que X . Cette fonction n'est appelée qu'après s'être entraîné sur (X, Y) . Si un exemple de test x n'a pas de voisin dans la fenêtre de paramètre h , l'algorithme doit choisir un label au hasard en utilisant **draw_rand_label(x, label_list)**, une fonction qui est fournie dans le fichier **solution.py**, avec **label_list** la liste des différentes classes présentes dans Y , et x le vecteur des attributs du point correspondant.

3. [5 points]

Question. Implémentez la méthode de Parzen avec une fenêtre continue. Nous utiliserons comme kernel une fonction à base radiale (RBF) (aussi connue comme *Gaussian* kernel) avec paramètre σ . Utilisez la distance Euclidienne sur les attributs du jeu de données. Veuillez consulter les slides de la deuxième semaine pour la définition. La structure de votre solution devrait être la même que pour la question précédente, mais vous n'aurez jamais besoin de choisir un label au hasard en utilisant **draw_rand_label(x, label_list)**. Le nom de la classe est **SoftRBFParzen**

4. [5 points]

Question. Implémentez une fonction **split_dataset** qui sépare le jeu de données banknote de la façon suivante:

- Un ensemble d'entraînement composé des points du jeu de données dont l'indice a un reste de 0, 1 ou 2 quand divisé par 5
- un ensemble de validation composé des points du jeu de données dont l'indice a un reste de 3 quand divisé par 5
- un ensemble de test composé des points du jeu de données dont l'indice a un reste de 4 quand divisé par 5

Par exemple l'élément dont l'indice est 14 (dans le jeu de données original) doit faire partie de l'ensemble de test, car le reste de 14 divisé par 5 est 4. N'utilisez pas de séparation aléatoire pour cette exercice (bien que ce soit habituellement une très bonne idée). La fonction doit prendre en entrée le jeu de données, et retourner les trois sous-ensembles sous forme d'un tuple (train, validation, test), où chaque élément du tuple est une matrice à 5 colonnes (les 4 attributs et les labels, gardés dans le même ordre).

5. [10 points]

Question. Implémentez deux fonctions **ErrorRate.hard_parzen** et **ErrorRate.soft_parzen** qui calculent le taux d'erreur (i.e. la proportion de mauvaises classifications) des algorithmes HardParzen et SoftRBGParzen. Le comportement attendu est le suivant :

test_error = ErrorRate(x_train, y_train, x_val, y_val) initialise la classe et sauvegarde les ensembles d'entraînement et de validation, où **x_train** et **x_val** sont des matrices d'attributs à 4 colonnes, et **y_train** et **y_val** sont des tableaux contenant les labels.

test_error.hard_parzen(h) prends en entrée le paramètre **h** et retourne sous forme de nombre réel le taux d'erreur sur **x_val** et **y_val** de l'algorithme HardParzen qui a été entraîné sur **x_train** et **y_train**.

test_error.soft_parzen(σ) fait la même chose mais pour SoftRBF-Parzen.

Ensuite, incluez dans votre rapport un graphe unique avec deux lignes :

- (a) l'erreur de classification de Hard Parzen sur l'ensemble de validation de banknote, après avoir été entraîné sur l'ensemble d'entraînement (voir question 4) pour les valeurs suivantes de **h**:

$$h \in \{0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0\}$$

- (b) l'erreur de classification de Soft Parzen sur l'ensemble de validation de banknote, après avoir été entraîné sur l'ensemble d'entraînement (voir question 4) pour les valeurs suivantes de σ :

$$\sigma \in \{0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0\}$$

L'axe des abscisses représentera à la fois **h** et σ . Attention à annoter les axes et lignes du graphe!

Donnez une discussion détaillée de vos observations.

6. [5 points]

Question. Implémentez une fonction `get_test_errors` qui utilise les erreurs de classification sur l'ensemble de validation calculées à la question précédente pour sélectionner h^* et σ^* , puis qui calcule le taux d'erreur sur l'ensemble de test. La valeur h^* est celle (parmi celles proposées à la question 5) qui minimise l'erreur de Hard Parzen sur l'ensemble de validation, et σ^* est le paramètre (parmi ceux proposés à la question 5) qui minimise l'erreur de Soft RBF Parzen sur l'ensemble de validation.

La fonction doit prendre en argument le dataset et le séparer en utilisant la question 4. Le résultat attendu est un tableau de taille 2, dont la première valeur est le taux d'erreur sur l'ensemble de test de Hard Parzen avec paramètre h^* , et la deuxième est le taux d'erreur sur l'ensemble de test de Soft RBF Parzen avec paramètre σ^* .

7. [5 points]

Question. Ajoutez à votre rapport une discussion sur la complexité temporelle (temps de calcul) de ces deux méthodes. Comment cela évolue-t-il pour chaque méthode quand les paramètres h ou σ changent ? Pourquoi ?

8. [5 points]

Question. Implémentez une projection aléatoire (Gaussian sketch) pour l'utiliser sur les données d'entrée :

Votre fonction `project_data` doit accepter en entrée une matrice d'attributs X de dimension $n \times 4$, et une matrice A de dimension 4×2 encodant la projection.

En définissant $p : x \mapsto \frac{1}{\sqrt{2}}x^T A$, utiliser cette projection aléatoire pour réduire la dimension des entrées (attributs des points du jeu de données) de 4 à 2.

Votre fonction doit retourner le résultat de p appliqué à X , sous forme d'une matrice $n \times 2$.

e.g. $project_data(X_{n,4}, A_{4,2}) = X_{n,2}^{proj}$

9. [10 points]

Question. De même que dans la question 5, calculez les erreurs de validation du Hard Parzen entraîné sur 500 projections aléatoires de l'ensemble d'entraînement, pour

$$h \in \{0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0\}$$

Les erreurs de validation devraient être calculées sur la projection de l'ensemble de validation, en utilisant la même matrice A . Pour obtenir des projections aléatoires, vous pouvez tirer A comme 8 variables indépendantes tirées aléatoirement d'une distribution gaussienne de centre 0 et de variance 1.

Vous pouvez par exemple représenter ces erreurs de validation dans une matrice 500×10 , avec une rangée par projection aléatoire et une colonne par valeur de h .

Faites la même chose pour Soft RBF Parzen, pour

$$\sigma \in \{0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0\}$$

Tracez et incluez dans votre rapport sur la même courbe les valeurs moyennes de l'erreur de validation (moyenner sur toutes les projections aléatoires) pour chaque valeur de h et σ , avec des intervalles d'erreur (sous forme graphique de barres) de longueur égale à $0.2 \times$ la déviation standard.

Comment vos résultats se comparent-ils aux précédents ?