IFT 3395 Fondements de l'apprentissage machine
Prof. Ioannis Mitliagkas

Devoir 1 - 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. A 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 sur 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 sur 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 - Pratique - 3395` pour le code et `Devoir 1 - Pratique - Rapport - 3395` 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 jeu de données Iris. Celui-ci contient 150 points (un par rangée), chacun avec 4 attributs (les 4 premières colonnes) et un label dans $\{1, 2, 3\}$ (la 5ième colonne). Il est recommandé de le télécharger ici puis de tester votre code en l'important comme ceci :

```
import numpy as np
iris = np.genfromtxt('iris.txt')
```

Quand le gabarit de réponse dans solution.py contient `iris` comme argument, vous pouvez assumer que cet argument est le jeu de données en format numpy. Votre fonction doit utiliser cet argument afin de faire les calculs, et non une version du jeu de données que vous auriez chargé vous-même. Les premières 4 colonnes indiquent, donc, les attributs et la dernière colonne indique le label.

1. [4 points] Écrivez des fonctions qui prennent le jeu de données en entrée et retournent les statistiques suivantes:

   (a) Q1.feature_means : Un tableau contenant les moyennes de chaque attribut, pour tous les exemplaires dans le jeu de données. Faites attention à l'ordre des attributs.
   e.g. : Q1.feature_means(iris) = $[\mu_1, \mu_2, \mu_3, \mu_4]$

   (b) Q1.empirical_covariance : Une matrice 4×4 correspondant à la matrice de covariance empirique des attributs sur le jeu de données entier.

   (c) Q1.feature_means_class_1 : Un tableau contenant les moyennes de chaque attribut, mais seulement pour les points dont le label est 1. Les classes possibles dans le jeu de données iris sont 1, 2 et 3.
   e.g. : Q1.feature_means_class_1(iris) = $[\mu_1, \mu_2, \mu_3, \mu_4]$

   (d) Q1.empirical_covariance_matrix_class_1 : Une matrice 4×4 correspondant à la matrice de covariance empirique des attributs sur l'ensemble des points dont le label est 1.

2. [2 points] Implémentez la méthode de Parzen avec une fenêtre discrète de paramètre h. Utilisez la distance L_1 (Manhattan) sur les attributs

du jeu de données:

$$d_{L_1}(p, X) = \sum_{i=1}^{n} |p - X_i|$$

où nous comparrons le point $p \in \mathbb{R}^4$ contre un ensemble de points arbitraire $X \in \mathbb{R}^{n \times 4}$. Votre solution doit avoir le comportement suivant:
f = HardParzen(h) initialise une instance de l'algorithme avec le paramètre h
f.fit(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 representés par des entiers, mais le nombre de classes dans Y peut varier.
f.predict(X_test) calcule les labels prédits et les retourne 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 l'entraînement 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. [10 points] Implémentez la méthode de Parzen avec une fenêtre continue. Nous utiliserons comme kernel une fonction à base radiale (RBF) (aussi connue comme kernel *gaussien*) avec paramètre σ. Utilisez la distance L_1 (Manhattan) sur les attributs du jeu de données. 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. [7 points] Implémentez une fonction **split_dataset** qui sépare le jeu de données iris de la façon suivante:

   - Un ensemble d'entrainement 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 élement du tuple est une matrice à 5 colonnes (les 4 attributs et les labels, gardés dans le même ordre).

5. [15 points] Implémentez deux fonctions **ErrorRate.hard_parzen** et **ErrorRate.soft_parzen** qui calculent le taux d'erreur (i.e. la proportion de mauvaises classifications) des algorithms 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'algorithm HardParzen 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 courbes :

   (a) l'erreur de classification de Hard Parzen sur l'ensemble de validation du jeu de données iris, 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 du jeu de données iris, 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. [6 points] 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 (parmis celles proposées à la question 5) qui minimise l'erreur de Hard Parzen sur l'ensemble de validation, et σ^* est le paramètre (parmis 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. [6 points] Ajoutez à votre rapport une discussion sur la complexité temporelle (temps de calcul) de ces deux méthodes. Comment évolue-t-elle pour chacune des méthodes lorsque h ou σ changent? Pourquoi?

8. [bonus points] 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. $project\_data(X_{n,4}, A_{4,2}) = X^{proj}_{n,2}$

9. [bonus points] De même que dans la question 5, calculez les erreurs de validation du Hard Parzen entrainé sur 500 projections aléatoires de l'ensemble d'entrainement, 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 4 variables indépendantes tirées aléatoirement d'une gaussienne de moyenne 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 le même graphe 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 égales à $0.2\times$ la déviation standard.
Comment vos résultats se comparent-ils aux précédents ?