Devoir_1_Pratique_6390A

October 20, 2022

IFT 6390A Fondements de l'apprentissage machine

Devoir 1 - Pratique - Rapport

MAMACHE Hamed Nazim

Importation des bibliothèques nécessaires au TP, ainsi que des fonctions implémentées dans solution.py

[5]:
```
from solution import *
import numpy as np
import matplotlib.pyplot as plt
from random import gauss
from tqdm import tqdm
import time
```

Importation du dataset banknote

[6]:
```
banknote = np.genfromtxt("data_banknote_authentication.txt" , delimiter = ",")
```

1 Question 5

Calcul des taux d'erreurs et des temps de calcul pour chaque paramètre h et σ de chacune des deux méthodes

[ ]:
```
def get_errors(banknote):
    train, validation, test = split_dataset(banknote)
    x_train = train[:,:-1]
    y_train = train[:,-1]
    x_val = validation[:,:-1]
    y_val = validation[:,-1]

    error_rate = ErrorRate(x_train, y_train, x_val, y_val)
    h = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0]
    sigma = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0]

    hard_parzen_error = []
    soft_parzen_error = []

    for i in h:
```

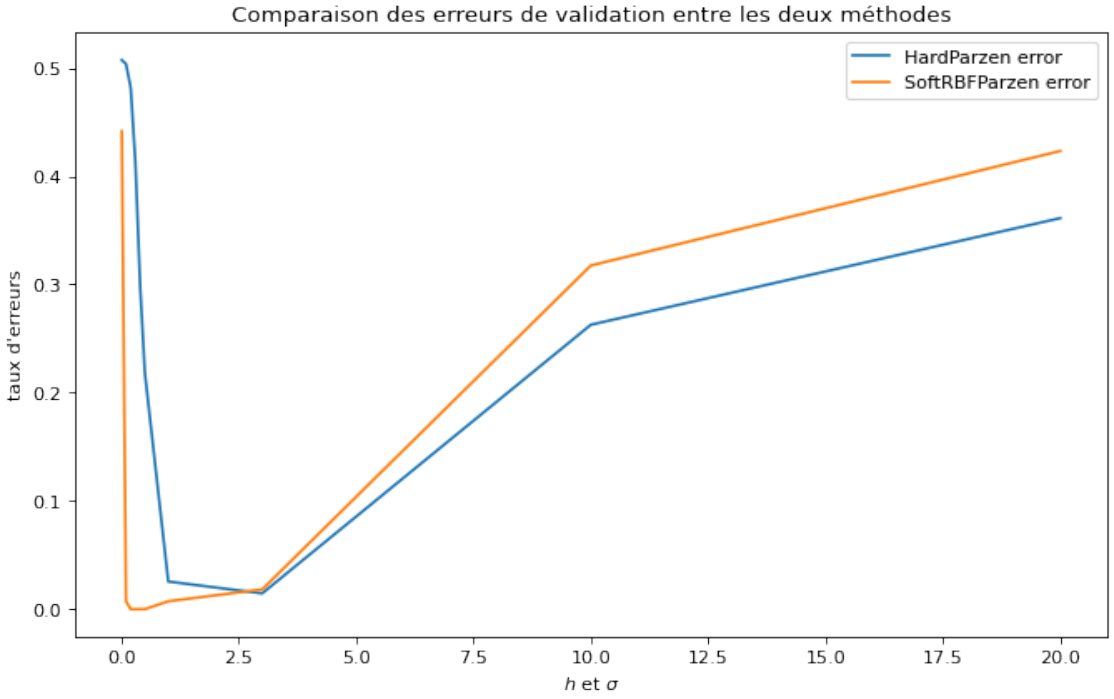
```
        hard_parzen_error.append(error_rate.hard_parzen(i))
    for i in sigma:
        soft_parzen_error.append(error_rate.soft_parzen(i))

    return hard_parzen_error, soft_parzen_error
```

Courbe obtenue :

[ ]:
```
hard_parzen_error, soft_parzen_error = get_errors(banknote)
```

[ ]:
```
h = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0]
sigma = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0]
```

[ ]:
```
fig, ax = plt.subplots(figsize=(10, 6), dpi=80)
ax.plot(h, hard_parzen_error, label="HardParzen error")
ax.plot(sigma, soft_parzen_error, label="SoftRBFParzen error")
ax.set_xlabel(r'$h$ et $\sigma$')
ax.set_ylabel('taux d\'erreurs')
ax.legend()
plt.title('Comparaison des erreurs de validation entre les deux méthodes')
plt.show()
```



Commentaire : On observe sur la courbe ci-dessus que pour σ et h inférieurs à 3, la méthode Soft RBF Parzen semble présenter de meilleurs résultats, en comparaison à la méthode Hard Parzen

(avec notamment des taux d'erreurs plus petits et très proches de 0). La méthode de Soft RBF Parzen a un taux d'erreur très faible pour un σ très faible.

Lorsque $\sigma = h = 2.5$, on observe une intersection des courbes d'erreurs : Hard Parzen et Soft RBF Parzen présentent les mêmes résultats en terme de taux d'erreurs et semblent équivalents.

Cependant, pour σ et h supérieurs à 3 (soit supérieurs ou égaux à 10), la méthode Hard Parzen présente de meilleurs résultats que celle de Soft RBF Parzen.

Pour des paramètres supérieurs à 3, il ne semble pas intéressant de considérer de tels paramètres, puisque le taux d'erreur semble s'acroître considérablement après ce seuil. En effet, la valeur minimale de taux d'erreur des deux méthodes semblent avoir été atteintes préalablement, pour des valeurs de σ et h plus petites.

2 Question 7

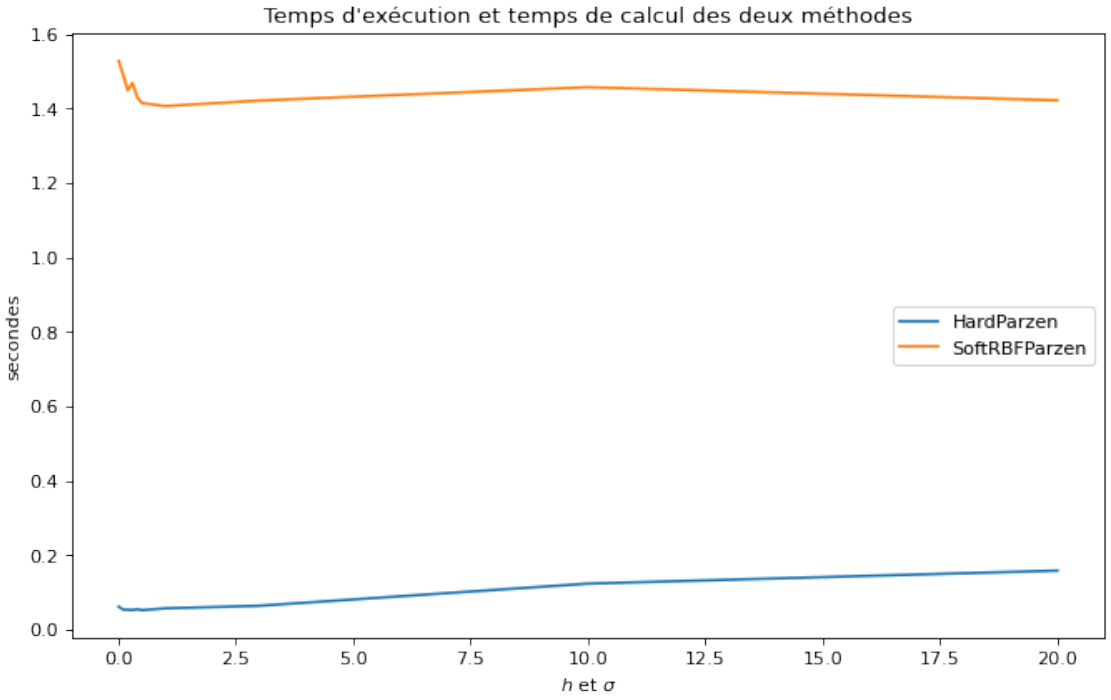
Discussion sur la complexité temporelle (temps de calcul) des deux méthodes

[2]:
```
def get_time(banknote):
    train, validation, test = split_dataset(banknote)
    x_train = train[:,:-1]
    y_train = train[:,-1]
    x_val = validation[:,:-1]
    y_val = validation[:,-1]

    time_hard_parzen = []
    time_soft_parzen = []

    for i in h:
        hp = HardParzen(i)
        hp.train(x_train, y_train)
        start_time = time.time()
        y_pred = hp.compute_predictions(x_val)
        end_time = time.time() - start_time
        time_hard_parzen.append(end_time)
    for i in sigma:
        sp = SoftRBFParzen(i)
        sp.train(x_train, y_train)
        start_time = time.time()
        y_pred = sp.compute_predictions(x_val)
        end_time = time.time() - start_time
        time_soft_parzen.append(end_time)

    return time_hard_parzen, time_soft_parzen
```

Avec les paramètres h et σ entre 0.01 et 20.0, on obtient les graphiques suivants:

[14]:
```
h = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0]
sigma = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0]
```

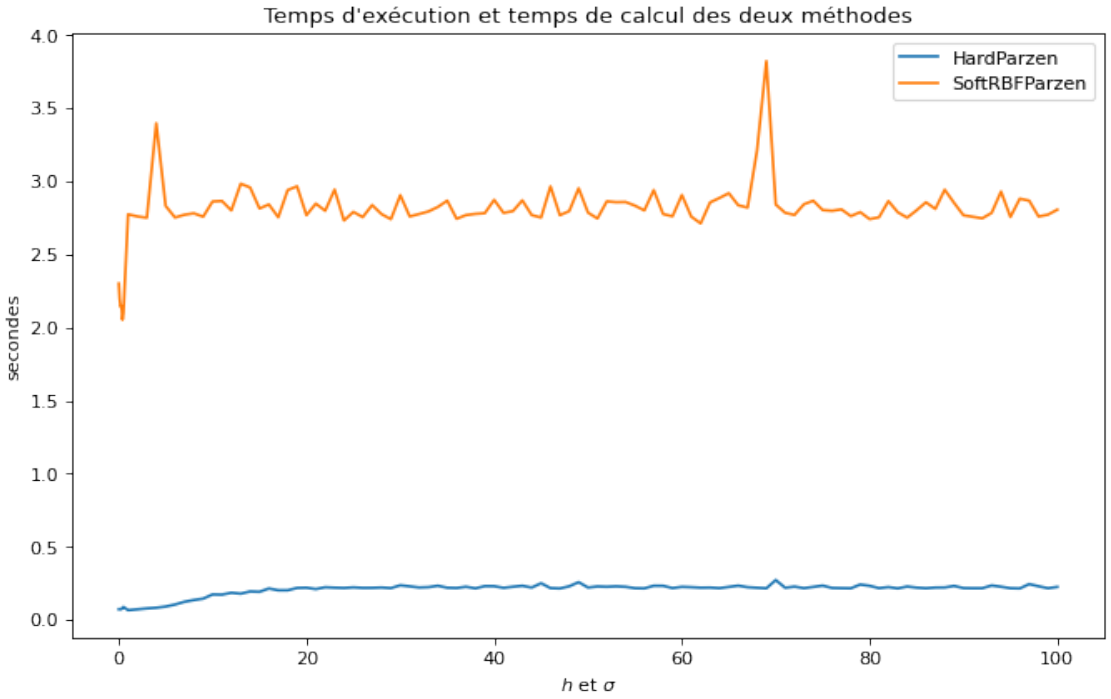
```
time_hard_parzen, time_soft_parzen = get_time(banknote)

fig, ax = plt.subplots(figsize=(10, 6), dpi=80)
ax.plot(h, time_hard_parzen, label="HardParzen")
ax.plot(sigma, time_soft_parzen, label="SoftRBFParzen")
ax.set_xlabel(r'$h$ et $\sigma$')
ax.set_ylabel('secondes')
ax.legend()
plt.title('Temps d\'exécution et temps de calcul des deux méthodes')
plt.show()
```



Avec les paramètres h et σ entre 0.01 et 100.0, on obtient les graphiques suivants:

[ ]:
```
h = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5] + [i for i in np.arange(1, 101, 1)]
sigma = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5] + [i for i in np.arange(1, 101, 1)]

time_hard_parzen, time_soft_parzen = get_time(banknote)

fig, ax = plt.subplots(figsize=(10, 6), dpi=80)
ax.plot(h, time_hard_parzen, label="HardParzen")
ax.plot(sigma, time_soft_parzen, label="SoftRBFParzen")
ax.set_xlabel(r'$h$ et $\sigma$')
ax.set_ylabel('secondes')
```

```
ax.legend()
plt.title('Temps d\'exécution et temps de calcul des deux méthodes')
plt.show()
```



Commentaire : On observe ici que les temps d'exécution des deux méthodes ne varie pas beaucoup (pour chacune d'elles) : il est quasi-constant. De plus le temps d'éxecution de Soft RBF Parzen semble beaucoup plus grand que celui de la méthode Hard Parzen. En effet, le calcul de la loi normale utilisée dans la méthode Soft RBF Parzen semble être le facteur ajoutant du temps de calcul pour celle-ci.

De plus, la constance en temps de calcul pour chacune des méthode s'explique par le fait que le choix des paramètres n'influe en rien la complexité des méthodes.

3 Question 9

Calcul des erreurs de validation du Hard Parzen et du Soft Parzen entrainé sur 500 projections aléatoires de l'ensemble d'entrainement :

[ ]:
```
def simulation(banknote):
    HardP_validation_error = []
    SoftP_validation_error = []
    train, validation, test = split_dataset(banknote)
    for i in tqdm(range(500)):
        A = np.array([[gauss(0,1) for x in range(2)] for y in range(4)])
        x_train = random_projections(train[:,:-1],A)
```

```
        y_train = train[:,-1]
        x_val = random_projections(validation[:,:-1],A)
        y_val = validation[:,-1]

        error_rate = ErrorRate(x_train, y_train, x_val, y_val)
        h = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0]
        sigma = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0]

        hard_parzen_error = []
        soft_parzen_error = []
        for j in h:
            hard_parzen_error.append(error_rate.hard_parzen(j))
        for j in sigma:
            soft_parzen_error.append(error_rate.soft_parzen(j))

        HardP_validation_error.append(hard_parzen_error)
        SoftP_validation_error.append(soft_parzen_error)

    return HardP_validation_error, SoftP_validation_error
```

Stockage des deux matrices de taille 500×10 dans des .json

[ ]:
```
import json
import os

if os.path.exists('HardP_validation_error.json') and os.path.
 ↪exists('SoftP_validation_error.json'):
    with open('HardP_validation_error.json', 'r') as f:
        HardP_validation_error = json.load(f)
    with open('SoftP_validation_error.json', 'r') as f:
        SoftP_validation_error = json.load(f)
else:
    HardP_validation_error, SoftP_validation_error = simulation(banknote)
    with open('HardP_validation_error.json', 'w') as f:
        json.dump(HardP_validation_error, f)
    with open('SoftP_validation_error.json', 'w') as f:
        json.dump(SoftP_validation_error, f)
```

[ ]:
```
print(np.shape(HardP_validation_error))
print(np.shape(SoftP_validation_error))
```

```
(500, 10)
(500, 10)
```

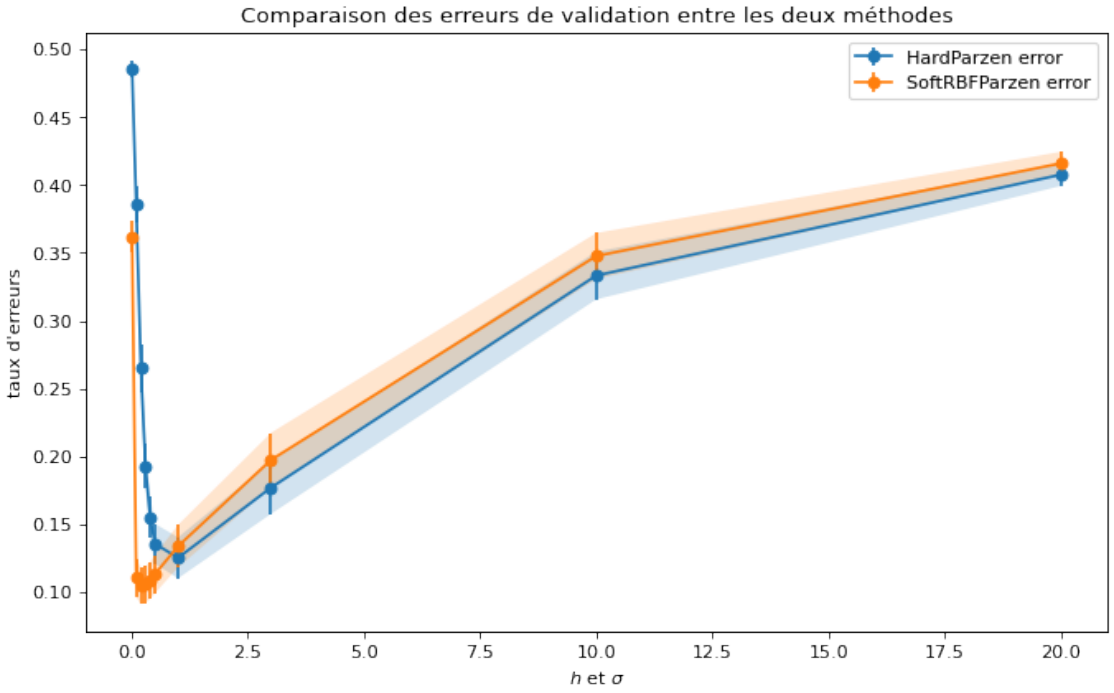
Calcul des valeurs moyennes des erreurs de validation, et des écarts-types

[ ]:
```
mean_HP = np.mean(HardP_validation_error, axis = 0)
mean_SP = np.mean(SoftP_validation_error, axis = 0)
```

[ ]:
```
std_HP = np.std(HardP_validation_error, axis = 0)
std_SP = np.std(SoftP_validation_error, axis = 0)
```

Courbe obtenue :

[ ]:
```
h = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0]
sigma = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0, 3.0, 10.0, 20.0]
error_HP = 0.2 * std_HP
error_SP = 0.2 * std_SP
fig, ax = plt.subplots(figsize=(10, 6), dpi=80)
ax.errorbar(h, mean_HP, label="HardParzen error", yerr = 0.2 * std_HP, fmt='-o')
ax.errorbar(sigma, mean_SP, label="SoftRBFParzen error", yerr = 0.2 * std_SP,␣
↪fmt='-o')
plt.fill_between(h, mean_HP-error_HP, mean_HP+error_HP,alpha=0.2)
plt.fill_between(sigma, mean_SP-error_SP, mean_SP+error_SP,alpha=0.2)
ax.set_xlabel(r'$h$ et $\sigma$')
ax.set_ylabel('taux d\'erreurs')
ax.legend()
plt.title('Comparaison des erreurs de validation entre les deux méthodes')
plt.show()
```



Commentaire : Nos résultats semblent être cohérentes avec les résultats obtenus précédemment. En effet, l'allure des courbes restent les mêmes.

Cependant, cette fois-ci les taux d'erreurs minimales sont supérieurs aux taux d'erreurs minimales

obtenus précedemment (0.10 taux d'erreurs contre environ 0.0 obtenu précedemment).

De plus, les paramètres σ et h, à partir des quels les taux d'erreurs ne s'améliorent plus, sont plus petits.