Rapport TME Régression

Master ANDROÏDE

Binôme : Rida Taleb, Achraf Jday
Encadrant : Olivier Sigaud

Study part 1

Does your train(self, x_data, y_data) function provide exactly the same results as the train_from_stats(self, x_data, y_data) function? Call both functions 2000 times. Is one faster than the other?

La fonction **train** prend les mesures x_data et y_data pour effectuer la méthode des moindres carrés linéaires et retourne le modèle optimal donné par la formule :

$$\boldsymbol{\theta}^* = (\bar{\mathbf{x}}^\intercal \bar{\mathbf{x}})^{-1} \bar{\mathbf{x}}^\intercal \mathbf{y}.$$

avec la matrice de Gram $\bar{\mathbf{X}}$ **:**

$$\bar{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} & 1 \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,d} & 1 \end{pmatrix}$$

Pour tester ces codes, nous utilisons la classe **SampleGenerator** pour générer des échantillons (un point entre [0, 1] pour un échantillon).
échantillon). Nous calculons également la somme des erreurs entre la fonction latente et chaque point de données afin de comparer les différences entre les performances des fonctions.

Nous obtenons les résultats suivants :

```
LLS time: 0.00027119999998603816
LLS error: 0.7942569005641469
```

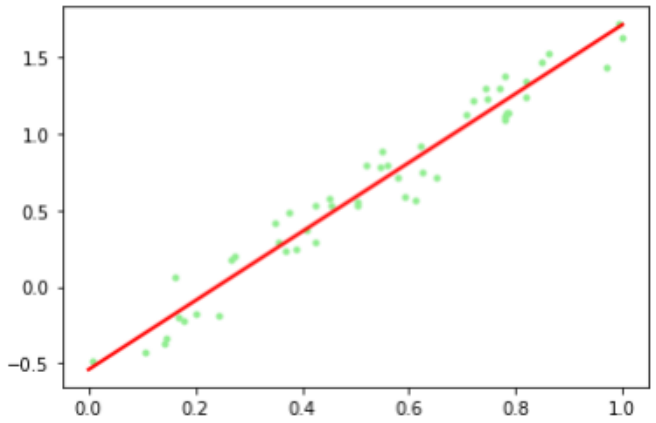


Figure 1: LLS from our function ***train***

```
LLS from scipy stats: 0.0004591000000000048967
LLS error from scipy stats: 0.7942569005641469
```

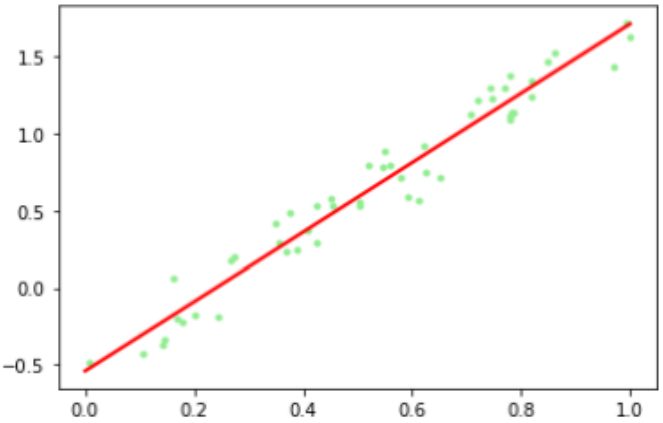


Figure 2: LLS from the function provided by Scipy

Nous modifions également les codes concernant l'estimation de la durée du processus d'une fonction en utilisant la fonction perf_counter afin de calculer plus précisément

lorsque cette durée est très courte (autour de 0). D'après les deux figures, nous observons que nous avons la même erreur. Cependant, la fonction **train** est plus rapide que la **LLS** fournie par Scipy mais cela n'est pas significatif car la taille de l'échantillon est trop petite. Nous testons également avec une grande taille de lot (par exemple 10 000) mais parfois le LLS de Scipy est plus rapide. Nous ne pouvons donc pas conclure quoi que ce soit à ce sujet.

Study part 2

For a batch of 50 points, study with the *train_regularized(self,x_data,y_data,coef)* function how the residuals degrade as you increase the value of coef. A good idea would be to make a picture with coef in the x axis and the residuals in the y axis, and then to comment it.

La fonction **train_regularized** prend les mesures x_data et y_data pour exécuter la méthode de régression Ridge et renvoie le modèle optimal donné par la formule :

$$\theta^* = (\lambda I + \bar{X}^T \bar{X})^{-1} \bar{X}^T y$$

où le facteur de régularisation $\lambda \geq 0$, I est la matrice d'identité, et $\bar{X}$ est la même matrice de Gram que dans **la Study part 1**.

Nous obtenons le résultat suivant :

```
regularized LLS : 0.00015529999999941424
regularized LLS error: 1.1629852120697082
```

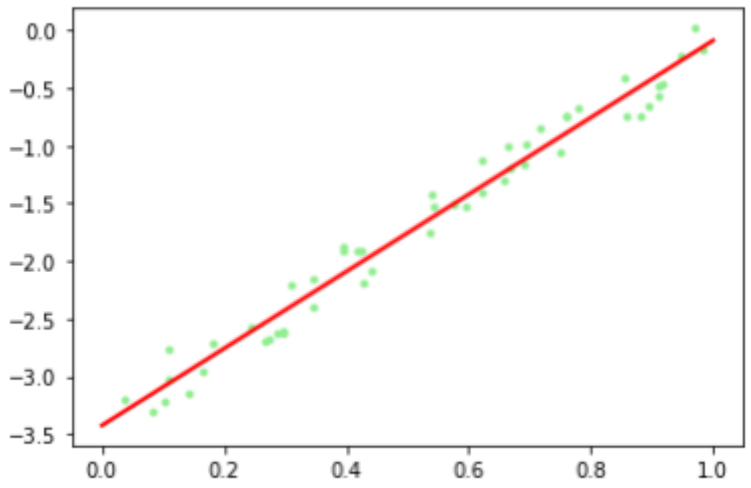


Figure 3: RLLS for 50 data points

Nous observons que la somme de l'erreur entre la fonction latente et chaque point de données est plus grande que l'erreur **LLS** car cette fois-ci, nous optimisons avec des poids plus faibles afin d'éviter les singularités potentielles lorsque nous inversons les matrices.

Afin d'étudier comment les résidus se dégradent lorsque nous augmentons la valeur de coef, nous traçons l'évolution de la somme des erreurs entre la fonction latente et chaque point de données en fonction de la valeur de coef.

Nous obtenons le résultat suivant :

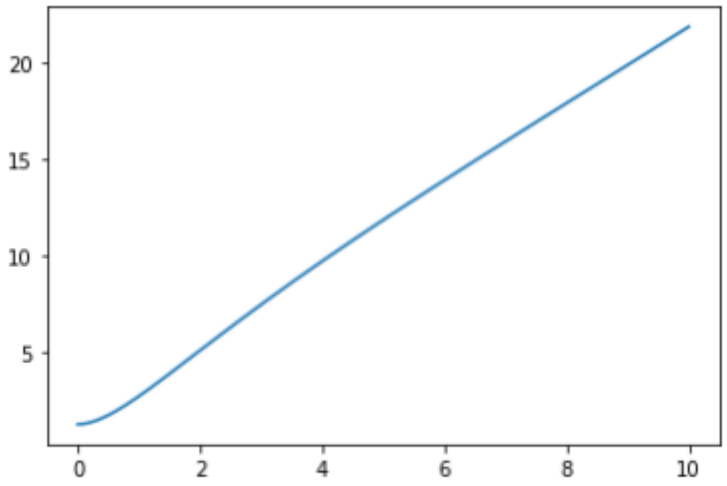


Figure 4: The evolution of the RLLS error depending on `coef` *for 50 points*

Nous observons que les résidus se dégradent linéairement lorsque nous augmentons la valeur de coef.

Study part 3

Study the evolution of the error as a function of the number of features. A good idea would be to make a picture with the number of features in the x axis and the residuals in the y axis, and then to comment it. Have a look at the model when the number of features is large (e.g. 30). What is happening?

La fonction **train_ls** prend les mesures x_data et y_data pour exécuter la méthode des moindres carrés en utilisant des réseaux de fonctions de base radiales et renvoie le modèle optimal donné par la formule suivante :

$$\theta^* = (\bar{G}^T \bar{G})^{-1} \bar{G}^T y$$

avec la matrice de Gram $\bar{G}$:

$$\bar{G} = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_E(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_E(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \cdots & \phi_E(x_N) \end{pmatrix}$$

Nous observons que X = $\overline{G}^{T}$ car la fonction phi_output renvoie un vecteur vertical pour un point de données. Pour trouver les valeurs du nombre de features permettant d'obtenir de bons résultats, nous avons testé différentes valeurs de nb_features allant de 10 à la taille du lot, puis nous avons obtenu les graphiques.

Voici quelques exemples de graphiques :

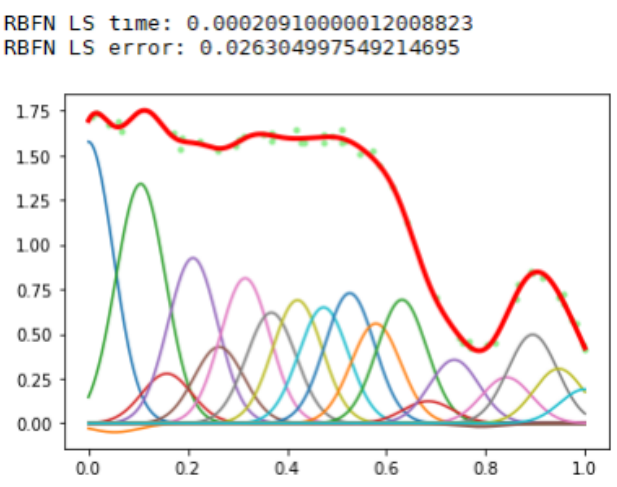


Figure 5: RBFN for 50 data points and 20 features (good results)

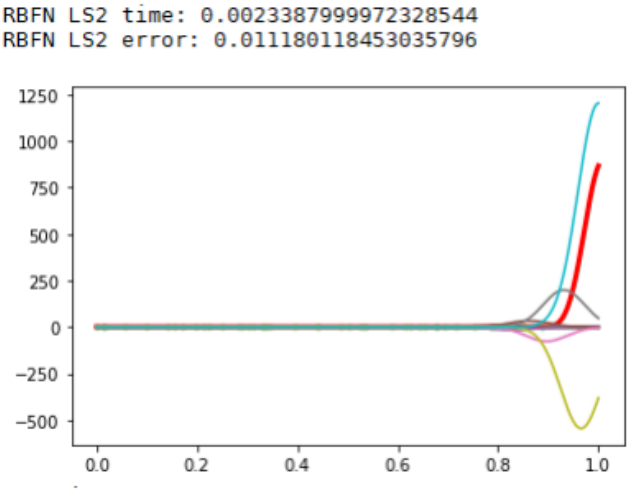


Figure 6: RBFN for 50 data points and 30 features (bad results)

Nous observons qu'obtenir de bons résultats en sélectionnant un nombre approprié de features signifie que les poids associés aux fonctions gaussiennes ne soient pas trop grands par rapport à nos points de données (c'est le cas de la figure 5). Dans le cas de la figure 6, les poids associés aux fonctions gaussiennes sont trop grands donc ces grandes features sont néfastes puisqu'elles influencent la fonction latente à dévier vers des prédictions erronées (dans la figure 6, nous avons des prédictions erronées pour x $\in$ [0.9, 1]). Par conséquent, nous concluons que le nombre de features conduisant à de bons résultats est compris entre 10 et 20.

Study part 4

By varying the number of features, the number of samples and the amount of noise in the data generator, compare both recursive variants (with and without the Sherman-Morrison formula) and gradient descent. Which is the most precise? The fastest? Using graphical displays where you are varying the above parameters is strongly encouraged. To change the amount of noise in the generated data, look in the sample_generator.py file.

Afin de comparer ces 3 méthodes, nous avons fixé maxIter = 1000, nb_features = 50 et alpha = 0.5. Ces valeurs nous permettent de trouver les mauvais résultats afin de vérifier facilement les faiblesses de ces 3 méthodes.

Les résultats sont les suivants :

```
RBFN Incr time: 0.36092430005919596
RBFN Incr error: 1.051887533402606
```

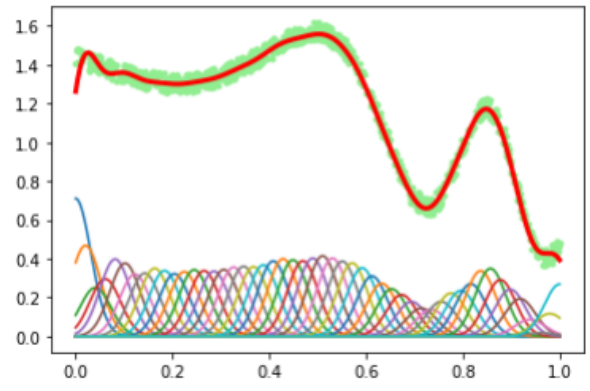


Figure 13: RBFN with RLS and the Sherman-Morrison formula

```
RBFN Incr time: 0.8874569000181509
RBFN Incr error: 0.7907511598622227
```

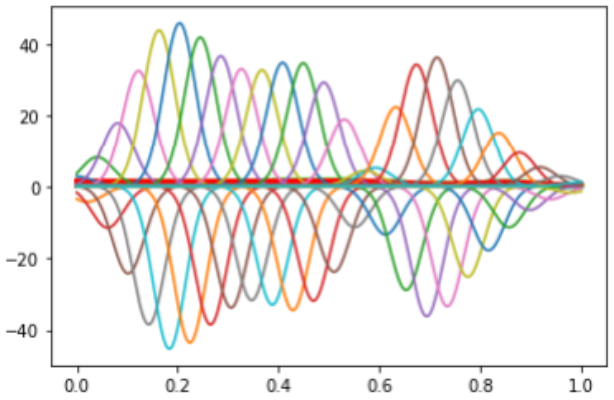


Figure 12: RBFN with RLS

```
RBFN Incr time: 0.10446920001595572
RBFN Incr error: 2.433876414661369
```

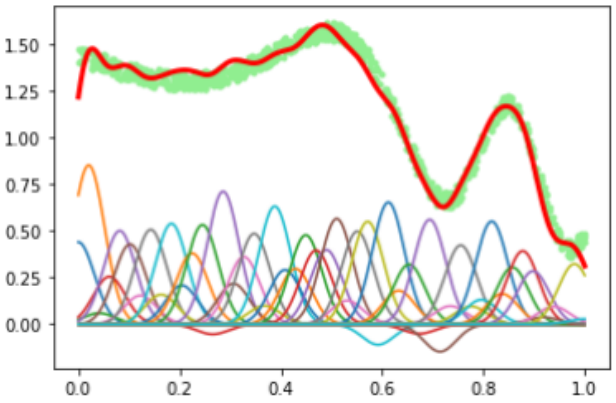


Figure 11: RBFN with gradient descent

On observe que la méthode la plus rapide reste la descente de gradient mais qu'elle présente l'erreur la plus élevée alors que la méthode la plus précise est la méthode classique des moindres carrés récursifs. Cependant, celle-ci sur-ajuste les points de données (poids très importants associés à certaines features). Enfin, la méthode des moindres carrés récursifs avec la formule de Sherman-Morrison se situe entre les deux méthodes précédentes sans sur-ajustement et parvient à obtenir une erreur raisonnable.

Study part 5

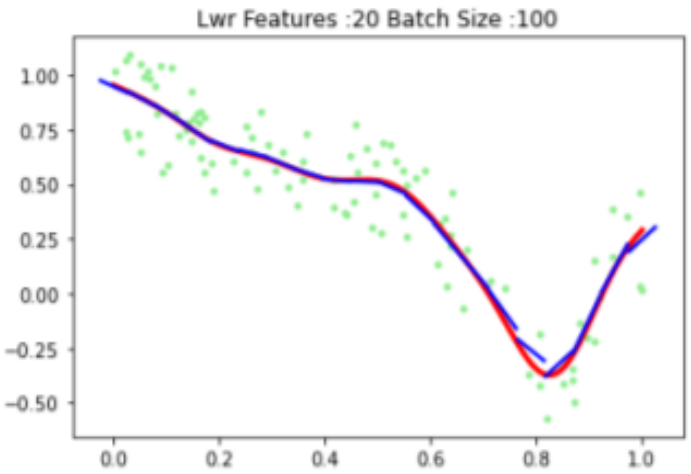
Using RBFNs, comment on the main differences between incremental and batch methods. What are their main advantages and disadvantages? Explain how you would choose between an incremental and a batch method, depending on the context.

Si nous supposons que les hyperparamètres sont correctement réglés, les principales différences entre les méthodes batch et incrémentale sont les suivantes : la méthode batch est plus précise que la méthode incrémentale mais plus coûteuse en termes de calcul lorsque la taille du lot est importante, tandis que la méthode incrémentale est efficace en termes de calcul avec un lot important mais moins précise que la méthode batch. Ainsi, la méthode batch est stable et coûteuse alors que la méthode incrémentale est instable mais efficace en termes de calcul. Cependant, comme nous l'avons vu dans les questions précédentes, nous pouvons modifier un peu la méthode incrémentale pour obtenir une meilleure performance afin d'être plus ou moins stable. Par conséquent, si nous n'accordons pas correctement les hyperparamètres, il est préférable d'utiliser les méthodes incrémentales avec les moindres carrés récursifs et la formule de Sherman-Morrison plutôt que la méthode batch afin d'éviter les phénomènes de surajustement. Sinon, lorsque nous réglons correctement les hyper-paramètres, si le lot est petit, il est préférable d'utiliser la méthode batch et si le lot est grand, il est préférable d'utiliser la méthode incrémentale.

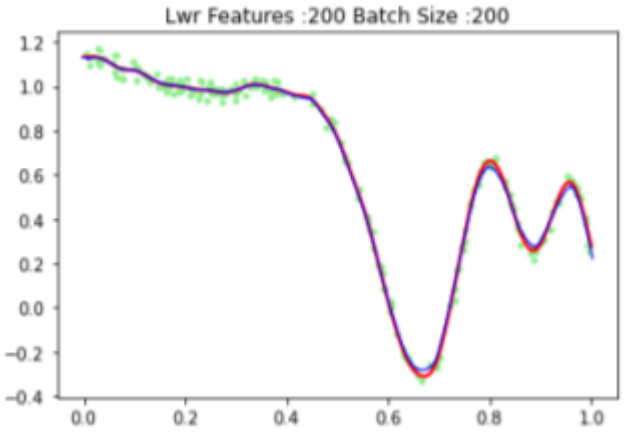
Study part 6

Study the impact of *nb_features* on the accuracy of LWR. As usual, make a drawing where you measure the residuals as a function of the number of features.

LWR est plus résistant au bruit des données que les RBFN et moins précis lorsque le nombre de caractéristiques est petit mais si ce nombre est grand, LWR est moins enclin à l'overfitting que les autres méthodes.
On peut voir sur cette image ci-dessous, avec 20 features, 200 données et un peu de bruit (de l'échantillon, sigma égal à 0,5), que le LWR a réussi à bien apprendre à partir de données avec un bruit élevé.



Ensuite, nous avons testé sur un grand nombre de caractéristiques pour voir la différence entre RBFNs et LWRs, donc avec 200 features, 200 données et un peu de bruit (de l'échantillon échantillon, sigma égal à 0.1)



LWR est beaucoup plus lent que tous les RBFN puisqu'il faut environ 0,13s pour le calculer dans les mêmes conditions que les RBFN.

En conclusion, les LWR donnent de meilleurs résultats dans la plupart des cas.

Study part 7

- comment on the difference of results between the batch version, the incremental version and the incremental version with mini-batches
- by measuring the loss function, study the evolution of the final loss and the training time depending on the size of the minibatch, for a constant budget of iterations
- add a training set, a validation set and a test set, and study overfitting

Nous n'avons pas réussi à faire fonctionner cette partie du code.