



Nom et Prénom : BATTACHE Nassim, DJOUAD Iskander.

Master 1 OIVM - Optique Image Vision MultiMedia

Apprentissage supervisé

Régression linéaire et non linéaire.

Abstract :

Ce rapport résume la méthode et les étapes à utiliser pour trouver une fonction cout minimum qui nous permet de séparer le mieux nos données avec la bibliothèque SKLEARN.

Introduction

Principes fondamentaux de l'apprentissage supervisé

Outils et méthodes

AFD

AMC

Conclusion

1.Introduction :

Le machine Learning consiste à donner à une machine la capacité d'apprendre sans le programmer de façon explicite.

Apprentissage supervisé consiste à montrer à la machine des exemples x, y et elle lui demander de trouver l'association qui relie le x avec le y qui consiste à trouver une fonction F tel que $y=F(x)$, ces exemples sont sous forme de DATASET

2.Principes fondamentaux de l'apprentissage supervisé :

2.1.DATASET :

En apprentissage supervisé notre dataset contient toujours deux types variables la première la TARGET variable la deuxième et les FEATURES (les facteurs)

2.1.1.la TARGET variable :

C'est notre objectif, c'est ce qu'on veut que la machine apprenne à prédire par exemple (un prix d'appartement, la bourse ...)

2.1.2.les FEATURES (les facteurs) :

Les facteurs qui viennent influencer les valeurs de y qui est la TARGET

2.2.MODEL :

Une deuxième notion fondamentale dans l'apprentissage superviser est le MODEL, en effet à partir du dataset on va essayer de trouver un modèle soit linéaire soit polynomiale ou autres ou on aura des paramètres qui seront fixer par les entraînements de la machine

2.3.Fonction cout :

Est une notion très importante représente les erreurs de la prédiction de notre modèle le but est d'avoir un bon modèle, c'est un modèle qui nous donne de petites erreurs.

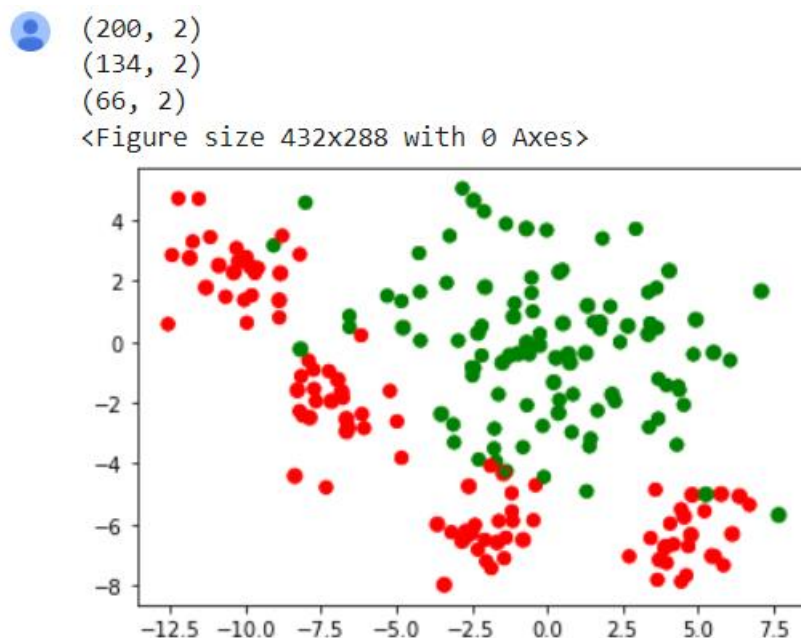
algorithme d'apprentissage (algorithme de minimisation):

le machine Learning va développer une stratégie qui cherche à trouver quelles sont les paramètres qui minimise la fonction cout , qui minimise l'ensemble de nos erreurs .

Outils et méthodes :

1-La première étape consiste à générer les données à partir de lois normales bidimensionnelles.

On copie le code du programme donné dans la feuille du TP1 et puis on visualise nos données.



Dans un deuxième temps on génère un découpage entre données d'apprentissage et données test.

On prend la taille des données test égale à 33% de nombre total de données de notre dataset

Dans notre cas ici la dataset contient 200 échantillons de jeu de données d'apprentissage . 134 pour l'entraînement du model . 66 pour le test de validation du model.

Puis on visualise le score d'apprentissage et de test

```
Le score sur le jeu d'apprentissage est de :0.8656716417910447
Le score sur le jeu de test est de : 0.8787878787878788
```

2-regression linéaire (AFD) :

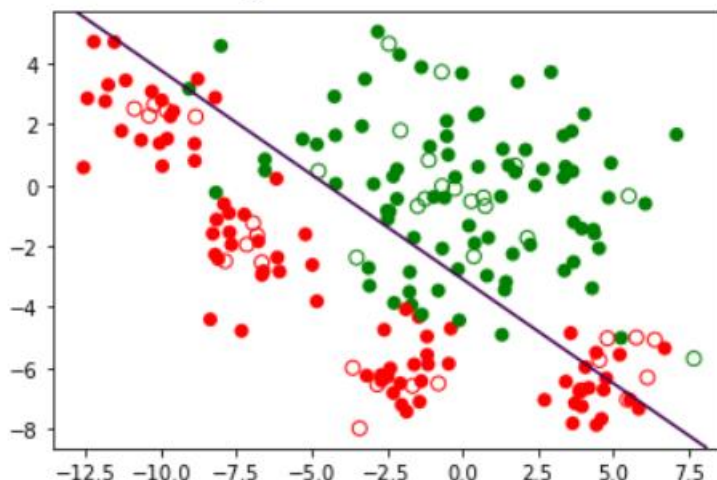
On considère un modèle de régression de la forme

$$y_i = f(x_i) + \varepsilon_i$$

où les erreurs sont centrées et la fonction f est supposée régulière : existence de dérivées jusqu'à un certain ordre. Dans ce contexte, de nombreux estimateurs de f ont été proposés. Ils conduisent souvent à des résultats assez voisins, le point le plus sensible étant le choix de λ .

Dans cette partie on fixe notre fonction à 0.5, c'est-à-dire $F(x)=0.5$.

Le score sur le jeu d'apprentissage est de : 0.875
Le score sur le jeu de test est de : 0.85



On prend la taille des données test égale à 20% de nombre total de données de notre dataset

On génère d'autres découpages apprentissage / test (avec une même valeur $\text{test_size}=0.33$ et 0.20) et examinez la variabilité des résultats


On change le pourcentage de test de 33 % à 20% en remarque que Le score sur le jeu d'apprentissage est de : 87.5% (à augmenter 1.5% par rapport à l'ancien test size qui a donnée environ 86.6 %)

remarque : à chaque fois qu'on diminue le nombre de données test, on remarque une augmentation du risque de surapprentissage (overfitting)

3- régression non linéaire (réseau de neurone connecté) (PMC) :

Classificateur Perceptron multicouche.

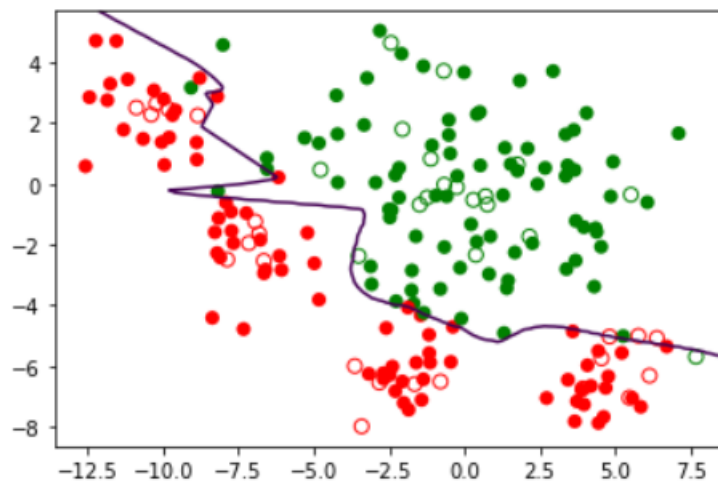
Ce modèle optimise la fonction log-loss en utilisant LBFGS ou descente de gradient stochastique.

 Le score en train est 0.98125
Le score en test est 0.925
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data:

<https://scikit-learn.org/stable/modules/preprocessing.html>

```
self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self  
<matplotlib.contour.QuadContourSet at 0x7f15e45c5410>
```

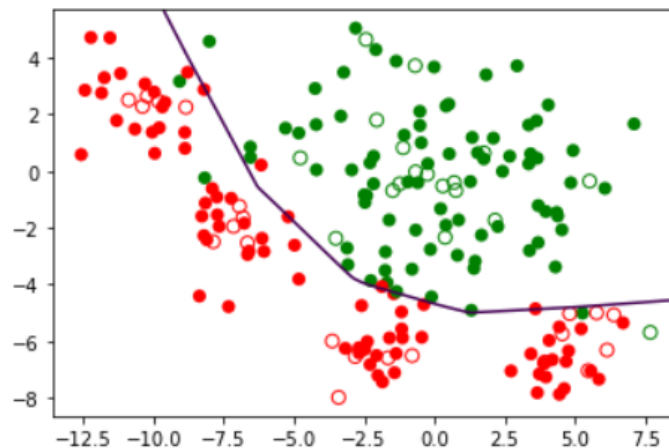


Ici on remarque bien que le score d'apprentissage et de test changent, on a un pourcentage d'entraînement et de 98.12% et celui de test et de 92.5%.



```
Le score en train est 0.9625
Le score en test est 0.975
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mu
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as
<https://scikit-learn.org/stable/modules/preprocessing.html>
self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.ma
<matplotlib.contour.QuadContourSet at 0x7f15e4532b50>



avec Le taux d'apprentissage initial égale à 1. Il contrôle la taille du pas dans la mise à jour des poids.


Utilisé uniquement lorsque solver='sgd' (ou adama dans le cas où on a des coches de convolution dans un CNN) on obtient le score d'entrainement de est de 0.9625 => 96.25 % et le score en test est 0.975 => 97.75 % tan dit que avec alpha = 1e-5 : en a Le score en train est 0.98125 =>98.125 % Le score en test est 0.925 => 92.25 % or avec alpha égale a 1 on obtient un meilleure score de test mais un score inférieur de score d'entrainement dans ce cas avec alpha = 1 on obtient un score plus fort avec un score de out layers égale a 1.25% qui est une estimation acceptable vue que il est délicat d'avoir 100 de réussite .

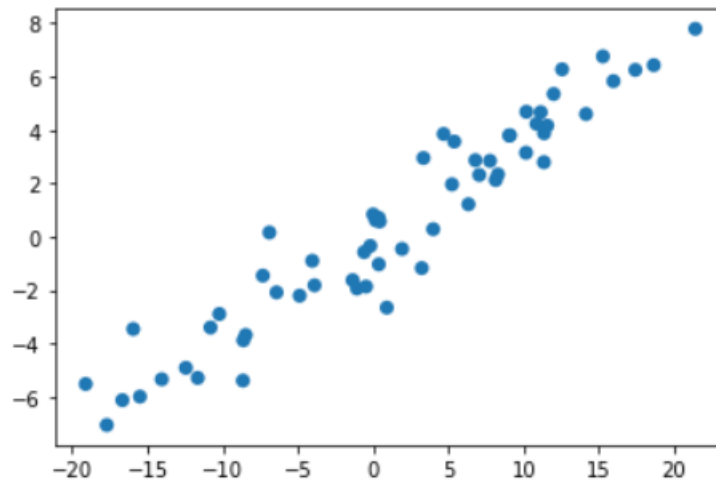
2) Avec les autres découpages apprentissage / test on avait aussi un petit taux de out layers de 1.3% mais le taux de score d'entrainement et de test de validation sont inferieur de 10%.

4-Données unidimensionnels :

Régression linéaire


Nous générons des données à partir d'une loi normale unidimensionnelle.

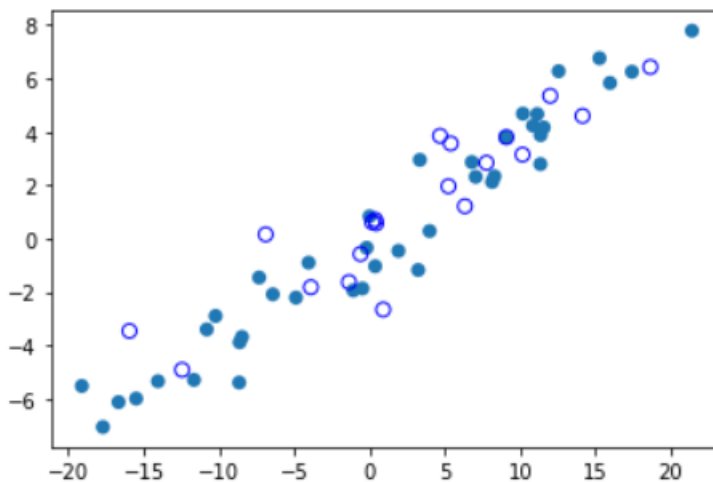
 <matplotlib.collections.PathCollection at 0x7f15e44b1d50>



Puis de la même façon on va génère un découpage entre données d'apprentissage et données test.

On prend la taille des données test égale à 33% de nombre total de données de notre dataset

 <matplotlib.collections.PathCollection at 0x7f15e4422dd0>



Puis on entraine notre modèle

Coefficient de détermination R^2 en train : 0.9420403323870808

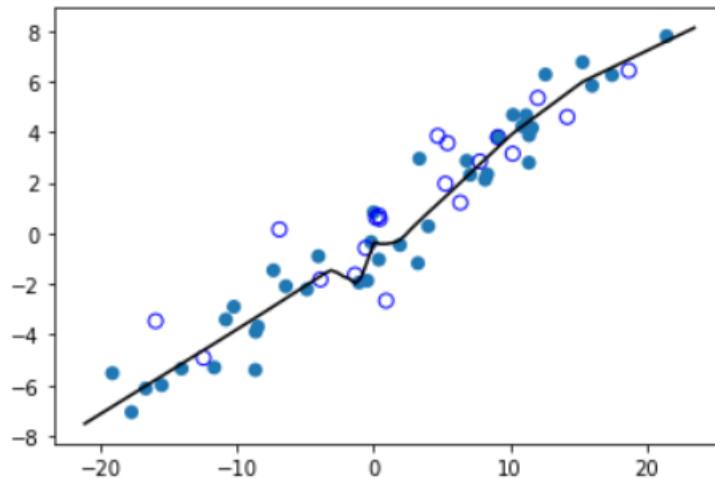
Coefficient de détermination R^2 en test : 0.7936655902477017

Régression non linéaire

Voici le résultat avec le taux d'apprentissage initial $\alpha = 1e-5$.

Le coefficient R^2 de train est 0.9528927649301804
Le coefficient R^2 de test est 0.7913850574884655
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data:
<https://scikit-learn.org/stable/modules/preprocessing.html>
self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self
[<matplotlib.lines.Line2D at 0x7f15e9eb8310>]



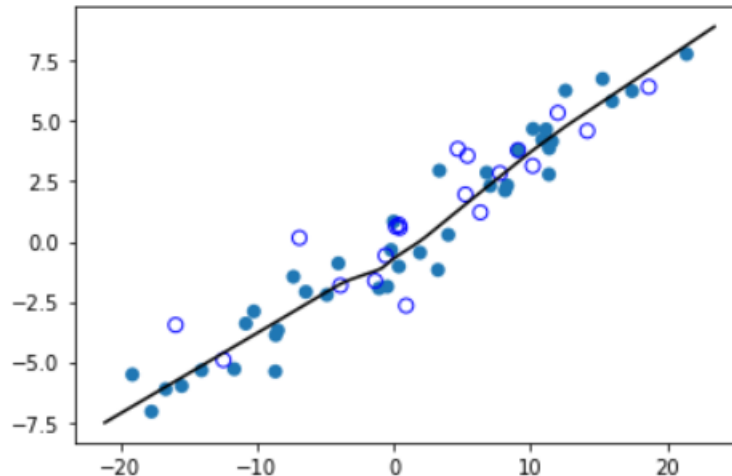
L'évaluation de prédiction de ce modèle est :

MSE test: 1.846
MSE train: 0.981

le résultat avec le taux d'apprentissage initial $\alpha = 1$

Le coefficient R^2 de train est 0.9492621630442571
 Le coefficient R^2 de test est 0.7897251726310307
 /usr/local/lib/python3.7/dist-packages/sklearn/neural_network,
 STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data:
<https://scikit-learn.org/stable/modules/preprocessing.html>
 self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.n_iter_)
 [<matplotlib.lines.Line2D at 0x7f15e44784d0>]



avec le taux d'apprentissage initial $\alpha = 1$.

Le coefficient R^2 de train est 0.95282989980058 => 95.28 %

Le coefficient R^2 de test est 0.7897251726310307 => 78.9725 % => avec un taux d'erreur de 0.865 tan dit qu'avec $\alpha = 1e-5$: Le coefficient R^2 de train est 0.9486136149161364 => 94.86% .

Le coefficient R^2 de test est 0.7913850574884655 => 79.13% . avec un taux d'erreur de 0.8645

les deux résultats en un taux d'erreur presque similaire mais avec α égale a 1 $\exp(-5)$ on a un coefficient de R^2 validation plus fort .

avec α égale à 1 on a un coefficient d'entraînement légèrement plus fort .

Conclusion :

- L'objectif est d'obtenir le modèle qui présente la meilleure généralisation (et non la plus faible erreur d'entraînement).
- La moyenne des erreurs d'entraînement de la régression linéaire est élevée, la capacité est insuffisante pour le problème où les classes ne sont pas linéairement séparables. Les modèles linéaires ne peuvent pas approcher suffisamment le meilleur modèle.
- Pour les PMC avec $\alpha=10^{-5}$, la moyenne des erreurs d'entraînement est très faible, donc l'erreur est potentiellement faible. En revanche, la moyenne des erreurs de test est bien plus élevée que la moyenne des erreurs d'entraînement et la variance des erreurs de test est élevée (supérieure à celle avec $\alpha=1$), ce qui indique que l'erreur de prédiction est élevée.
- L'erreur d'entraînement est excessivement optimiste comme estimateur de l'erreur de généralisation .
- Pour la famille définie par les PMC avec $\alpha=1$, la moyenne des erreurs de test est comparativement faible, ce qui indique une somme assez faible entre erreur d'approximation et erreur d'estimation. La généralisation est meilleure que celle obtenue avec les deux autres familles. Enfin, la moyenne des erreurs de test est assez faible et proche de la moyenne des erreurs d'apprentissage.
- Pour minimiser l'erreur de généralisation, il faut chercher le bon compromis entre la minimisation de l'erreur de test et la minimisation de l'erreur d'entraînement.