mlp

May 12, 2022

1 Introduction

The objective of this lab is to dive into particular kind of neural network: the *Multi-Layer Perceptron* (MLP).

To start, let us take the dataset from the previous lab (hydrodynamics of sailing boats) and use scikit-learn to train a MLP instead of our hand-made single perceptron. The code below is already complete and is meant to give you an idea of how to construct an MLP with scikit-learn. You can execute it, taking the time to understand the idea behind each cell.

[3]:
```
# Importing the dataset
import numpy as np
dataset = np.genfromtxt("yacht_hydrodynamics.data", delimiter='')
X = dataset[:, :-1]
Y = dataset[:, -1]
```

[4]:
```
# Preprocessing: scale input data
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
```

[5]:
```
# Split dataset into training and test set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y,random_state=1,␣
↪test_size = 0.20)
```

[6]:
```
# Define a multi-layer perceptron (MLP) network for regression
from sklearn.neural_network import MLPRegressor
mlp = MLPRegressor(max_iter=3000, random_state=1) # define the model, with␣
↪default params
mlp.fit(x_train, y_train) # train the MLP
```

[6]:
```
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
             beta_2=0.999, early_stopping=False, epsilon=1e-08,
             hidden_layer_sizes=(100,), learning_rate='constant',
             learning_rate_init=0.001, max_fun=15000, max_iter=3000,
             momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
             power_t=0.5, random_state=1, shuffle=True, solver='adam',
```

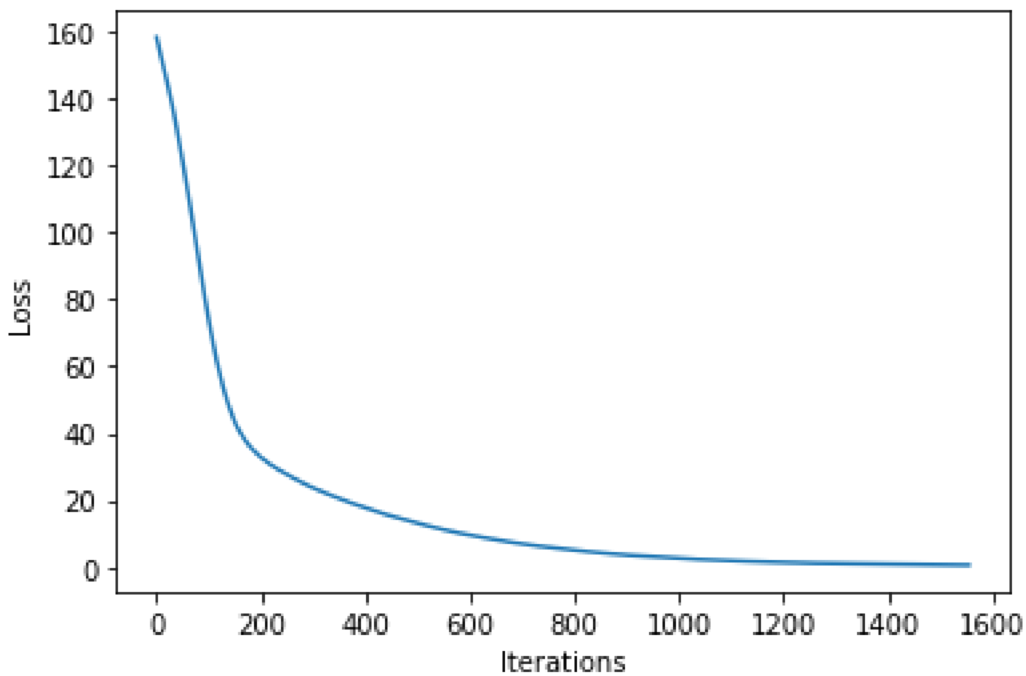
```
                    tol=0.0001, validation_fraction=0.1, verbose=False,
                    warm_start=False)
```

[7]:
```python
# Evaluate the model
from matplotlib import pyplot as plt

print('Train score: ', mlp.score(x_train, y_train))
print('Test score:  ', mlp.score(x_test, y_test))
plt.plot(mlp.loss_curve_)
plt.xlabel("Iterations")
plt.ylabel("Loss")
```

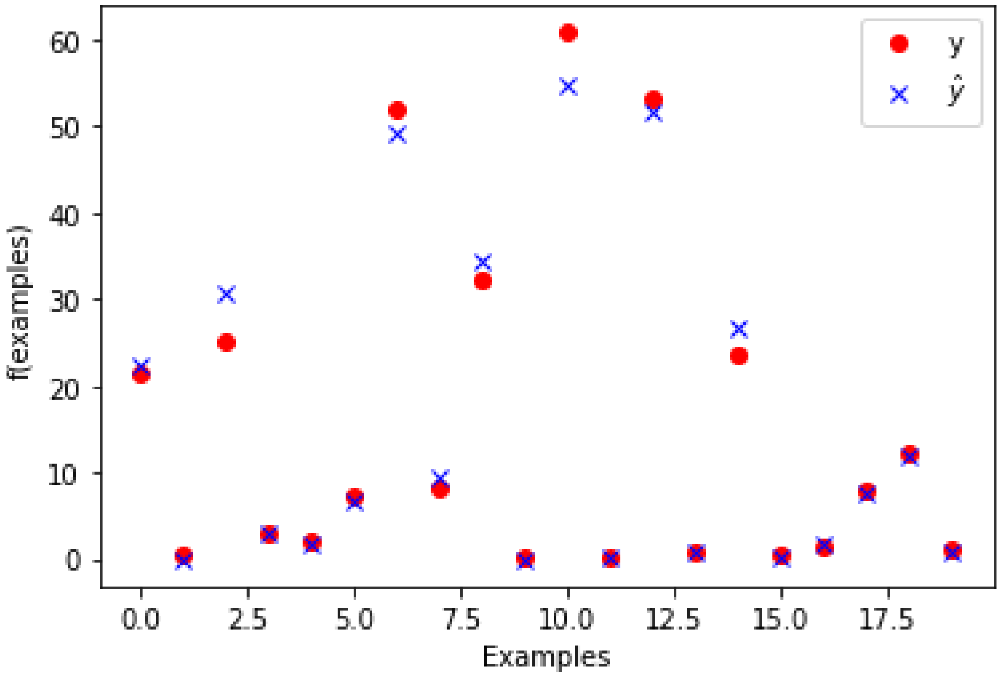
```
Train score:  0.9940765369322632
Test score:   0.9899773031580282
```

[7]:
```
Text(0, 0.5, 'Loss')
```



[8]:
```python
# Plot the results
num_samples_to_plot = 20
plt.plot(y_test[0:num_samples_to_plot], 'ro', label='y')
yw = mlp.predict(x_test)
plt.plot(yw[0:num_samples_to_plot], 'bx', label='$\hat{y}$')
plt.legend()
plt.xlabel("Examples")
plt.ylabel("f(examples)")
```

[8]: Text(0, 0.5, 'f(examples)')



1.0.1 Analyzing the network

Many details of the network are currently hidden as default parameters.

Using the documentation of the MLPRegressor, answer the following questions.

- What is the structure of the network?
- What it is the algorithm used for training? Is there algorithm available that we mentioned during the courses?
- How does the training algorithm decide to stop the training?

-Il y a 100 neurones dans la couche cachée numéro 0 par défaut.

-L'algorithme utilisé par défaut est l'optimiseur basé sur le gradient stochastique. L'algorithme vu en cours est l'algorithme de gradient stochastique (SGD) qui correspond à l'argument solver='sgd'. Un optimiseur basé sur la méthode quasi-newton est aussi disponible.

-Si le nombre maximal d'itération est atteint alors l'algorithme de training s'arrête. Si l'option early-stopping est mise à True, une partie des données d'entraînement est utilisé pour faire des tests qui, s'ils ne s'améliorent pas assez vite, déclenchent l'arrêt de l'algorithme.

2 Onto a more challenging dataset: house prices

For the rest of this lab, we will use the (more challenging) California Housing Prices dataset.

[9]:
```
# clean all previously defined variables for the sailing boats
%reset -f
```

[1]:
```
"""Import the required modules"""
from sklearn.datasets import fetch_california_housing
import pandas as pd

num_samples = 2000 # only use the first N samples to limit training time

cal_housing = fetch_california_housing()
X = pd.DataFrame(cal_housing.data,columns=cal_housing.feature_names)[:
↪num_samples]
y = cal_housing.target[:num_samples]

X.head(10) # print the first 10 values
```

[1]:
```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85
5  4.0368      52.0  4.761658   1.103627       413.0  2.139896     37.85
6  3.6591      52.0  4.931907   0.951362      1094.0  2.128405     37.84
7  3.1200      52.0  4.797527   1.061824      1157.0  1.788253     37.84
8  2.0804      42.0  4.294118   1.117647      1206.0  2.026891     37.84
9  3.6912      52.0  4.970588   0.990196      1551.0  2.172269     37.84

   Longitude
0    -122.23
1    -122.22
2    -122.24
3    -122.25
4    -122.25
5    -122.25
6    -122.25
7    -122.25
8    -122.26
9    -122.25
```

Note that each row of the dataset represents a **group of houses** (one district). The `target` variable denotes the average house value in units of 100.000 USD. Median Income is per 10.000 USD.

2.0.1 Extracting a subpart of the dataset for testing

- Split the dataset between a training set (75%) and a test set (25%)

Please use the conventional names `X_train`, `X_test`, `y_train` and `y_test`.

```
[2]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,random_state=1,␣
 ↪test_size = 0.25)
```

2.0.2 Scaling the input data

A step of **scaling** of the data is often useful to ensure that all input data centered on 0 and with a fixed variance.

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance). The function `StandardScaler` from `sklearn.preprocessing` computes the standard score of a sample as:

`z = (x - u) / s`

where `u` is the mean of the training samples, and `s` is the standard deviation of the training samples.

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using transform.

- Apply the standard scaler to both the training dataset (`X_train`) and the test dataset (`X_test`).
- Make sure that **exactly the same transformation** is applied to both datasets.

Documentation of standard scaler in scikit learn

```
[3]: from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
print(sc.fit(X_train))#Les deux datasets subissent la même transformation car u␣
 ↪et s sont calculés pour le dataset
                      #d'entraînement et sont utilisés aussi pour la␣
 ↪transformation du dataset de test
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

2.1 Overfitting

In this part, we are only interested in maximizing the **train score**, i.e., having the network memorize the training examples as well as possible.

- Propose a parameterization of the network (shape and learning parameters) that will maximize the train score (without considering the test score).

While doing this, you should (1) remain within two minutes of training time, and (2) obtain a score that is greater than 0.90.

- Is the **test** score substantially smaller than the **train** score (indicator of overfitting) ?
- Explain how the parameters you chose allow the learned model to overfit.

[12]:
```
from sklearn.neural_network import MLPRegressor
import time
#On ajoute  autant de couches de neurones que possible, tout en gardant un␣
 ↪temps d'apprentissage inférieur à deux minutes.
#Ajouter des neurones et des couches cachées permet d'accroître la capacité␣
 ↪d'apprentissage.
#C'est le fait de trop apprendre (et donc d'apprendre le bruit) qui constitue␣
 ↪l'overfitting.
mlp = MLPRegressor(hidden_layer_sizes=(1000,1000,100,), max_iter=5000,␣
 ↪random_state=1)
start=time.time()
mlp.fit(X_train, y_train)
stop=time.time()
print('Train score: ', mlp.score(X_train, y_train))
print('Test score:  ', mlp.score(X_test, y_test))
print('Training time: ', str(stop-start)) #affiche la durée d'apprentissage en␣
 ↪secondes
```

```
Train score:  0.9291922380616646
Test score:   0.7707780930171036
Training time:  70.7204840183258
```

On voit en effet que le train score est très bon alors que le test score l'est beaucoup moins (92% par rapport à 77%) car le réseaux est trop spécialisé pour avoir des résultats corrects.

2.2 Hyperparameter tuning

In this section, we are now interested in maximizing the ability of the network to predict the value of unseen examples, i.e., maximizing the **test** score. You should experiment with the possible parameters of the network in order to obtain a good test score, ideally with a small learning time.

Parameters to vary:

- number and size of the hidden layers
- activation function
- stopping conditions
- maximum number of iterations
- initial learning rate value

Results to present for the tested configurations:

- Train/test score
- training time

Present in a table the various parameters tested and the associated results. You can find in the last cell of the notebook a code snippet that will allow you to plot tables from python structure. Be methodical in the way your run your experiments and collect data. For each run, you should record the parameters and results into an external data structure.

(Note that, while we encourage you to explore the solution space manually, there are existing methods in scikit-learn and other learning framework to automate this step as well, e.g., GridSearchCV)

[6]:

```python
import numpy as np
import pandas as pd
import random
import time

#On crée ici des listes de taille 6 représentant les différents paramètres à␣
↪donner à la fonction MLPRegressor
hidden_layer_sizes = [(100,), (10,2), (100, 100,), (100, 10), (1000,1000), (10,␣
↪10)]
max_iter = [5000, 1500, 800, 100, 500, 100]
initial_learning_rate = [0.001, 0.005, 0.0001, 0.1, 0.002, 0.0001]
tol = [0.0001, 0.0001, 0.001, 0.001, 0.01, 0.0005]
early_stopping = [True, False, True, False, True, False]
activation_function = ['identity', 'logistic', 'tanh', 'relu', 'logistic',␣
↪'relu']
data = []


for loop in range(100):
    #Pour chaque itération, une combinaison aléatoire de paramètres est testée
    i, j, k, l, m, n = random.randint(0, 5), random.randint(0, 5), random.
↪randint(0, 5),\
    random.randint(0, 5), random.randint(0, 5), random.randint(0, 5)

    mlp = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes[i],␣
↪max_iter=max_iter[j],
                            early_stopping=early_stopping[k],
                            learning_rate_init=initial_learning_rate[l],␣
↪tol=tol[m],
                            activation=activation_function[n], random_state=1
    )
    start=time.time()
    mlp.fit(X_train, y_train)
    stop=time.time()
    data.append({
            'activation': activation_function[n],
            'learning_rate_init': initial_learning_rate[l],
            'hidden_layer_sizes': hidden_layer_sizes[i],
            'max_iter': str(max_iter[j]),
            'early_stopping' : str(early_stopping[k]),
            'test_score': str(mlp.score(X_test, y_test)),
            'train_score': str(mlp.score(X_train, y_train)),
            'tol': str(tol[m]),
            'learning time':str(stop-start)
    })
    print(f"Test n°{loop} over.")
```

```python
table = pd.DataFrame.from_dict(data)
table = table.replace(np.nan, '-')
table = table.sort_values(by='test_score', ascending=False)
#On affiche les dix meilleurs résultats
table.head(10)
```

```
Test n°0 over.
Test n°1 over.
Test n°2 over.
Test n°3 over.
Test n°4 over.
Test n°5 over.
Test n°6 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Test n°7 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Test n°8 over.
Test n°9 over.
Test n°10 over.
Test n°11 over.
Test n°12 over.
Test n°13 over.
Test n°14 over.
Test n°15 over.
Test n°16 over.
Test n°17 over.
Test n°18 over.
Test n°19 over.
Test n°20 over.
Test n°21 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Test n°22 over.
Test n°23 over.
Test n°24 over.
Test n°25 over.
Test n°26 over.
Test n°27 over.
Test n°28 over.
Test n°29 over.
Test n°30 over.
Test n°31 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Test n°32 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Test n°33 over.
Test n°34 over.
Test n°35 over.
Test n°36 over.
Test n°37 over.
Test n°38 over.
Test n°39 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Test n°40 over.
Test n°41 over.
Test n°42 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Test n°43 over.
Test n°44 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°45 over.
Test n°46 over.
Test n°47 over.
Test n°48 over.
Test n°49 over.
Test n°50 over.
Test n°51 over.
Test n°52 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°53 over.
Test n°54 over.
Test n°55 over.
Test n°56 over.
Test n°57 over.
Test n°58 over.
Test n°59 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°60 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°61 over.
Test n°62 over.
Test n°63 over.
Test n°64 over.
Test n°65 over.
Test n°66 over.
Test n°67 over.
Test n°68 over.
```

```
Test n°69 over.
Test n°70 over.
Test n°71 over.
Test n°72 over.
Test n°73 over.
Test n°74 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°75 over.
Test n°76 over.
Test n°77 over.
Test n°78 over.
Test n°79 over.
Test n°80 over.
Test n°81 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°82 over.
Test n°83 over.
Test n°84 over.
Test n°85 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°86 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°87 over.
Test n°88 over.
Test n°89 over.
Test n°90 over.
Test n°91 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Test n°92 over.
Test n°93 over.
Test n°94 over.
Test n°95 over.
Test n°96 over.
Test n°97 over.
Test n°98 over.
Test n°99 over.
```

[6]:

```
   activation  learning_rate_init hidden_layer_sizes max_iter early_stopping  \
4        relu              0.0001       (1000, 1000)     1500           True
83       relu              0.0010       (1000, 1000)      800           True
27       relu              0.0010          (100, 10)      100          False
67   logistic              0.0050            (10, 2)     1500           True
85       relu              0.0050           (10, 10)      500          False
44   logistic              0.1000            (10, 2)      100           True
95   logistic              0.1000             (100,)     1500           True
26       relu              0.0010          (100, 10)      500          False
70       relu              0.0020          (100, 10)     1500          False
97       tanh              0.0050           (10, 10)      800          False

             test_score          train_score     tol        learning time
4     0.817929270815983   0.8450802509222831  0.0005    117.1925060749054
83   0.8174955377053713   0.8390217734635804   0.001    36.72002387046814
27   0.8154664798217646   0.7965854503725822  0.0005    0.910881757736206
67    0.815321158112775   0.8052801532529619  0.0001   1.4996156692504883
85   0.8140906295298296   0.7796998961171722  0.0005   0.3196103572845459
44   0.8139547173424885   0.8035999145716243  0.0001  0.30989861488342285
95   0.8134720833350176   0.7884261745508402    0.01  0.16295886039733887
26   0.8090207737830764   0.7742562083461876   0.001  0.48030734062194824
70    0.807944108681685     0.76327077080489    0.01   0.2318248748779297
97   0.8076400050077982   0.8224694198275926  0.0001   0.7591807842254639
```

[14]:

```python
# Code snippet to display a nice table in jupyter notebooks  (remove from␣
↪report)
import numpy as np
import pandas as pd
data = []
data.append({'activation': 'relu', 'max_iter': '500', 'test_score': 0.87})
data.append({'activation': 'tanh', 'max_iter': '200', 'early_stopping': False,␣
↪'test_score': 0.91})
```

```
table = pd.DataFrame.from_dict(data)
table = table.replace(np.nan, '-')
table = table.sort_values(by='test_score', ascending=False)
table
```

[14]:

	activation	max_iter	test_score	early_stopping
1	tanh	200	0.91	False
0	relu	500	0.87	-

2.3 Evaluation

- From your experiments, what seems to be the best model (i.e. set of parameters) for predicting the value of a house?

Unless you used cross-validation, you have probably used the "test" set to select the best model among the ones you experimented with. Since your model is the one that worked best on the "test" set, your selection is *biased*.

In all rigor the original dataset should be split in three:

- the **training set**, on which each model is trained
- the **validation set**, that is used to pick the best parameters of the model
- the **test set**, on which we evaluate the final model

Evaluate the score of your algorithm on a test set that was not used for training nor for model selection.

Le meilleur modèle semble être : activation: relu, hidden layer size: (1000,1000), early_stopping: true, tol:0.0001 ou 0.0005, learning rate init:0.0001.

[7]:
```
#On divise cette fois le dataset par trois (la moitié du dataset correspond au␣
↪set d'entraînement,
#le set de validation et celui de test correspondent chacun à un quart du set␣
↪oiginel).
X_train, X_test, y_train, y_test = train_test_split(X, y,random_state=1,␣
↪test_size = 0.25)
X_train, X_valid, y_train, y_valid = train_test_split(X_train,␣
↪y_train,random_state=1, test_size = 0.25)
print(sc.fit(X_train))
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
X_valid=sc.transform(X_valid)

for loop in range(100):

    i, j, k, l, m, n = random.randint(0, 5), random.randint(0, 5), random.
↪randint(0, 5),\
    random.randint(0, 5), random.randint(0, 5), random.randint(0, 5)
```

```
    mlp = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes[i],␣
↪max_iter=max_iter[j],
                           early_stopping=early_stopping[k],
                           learning_rate_init=initial_learning_rate[l],␣
↪tol=tol[m],
                           activation=activation_function[n], random_state=1
    )
    start=time.time()
    mlp.fit(X_train, y_train)
    stop=time.time()
    data.append({
            'activation': activation_function[n],
            'learning_rate_init': initial_learning_rate[l],
            'hidden_layer_sizes': hidden_layer_sizes[i],
            'max_iter': str(max_iter[j]),
            'early_stopping' : str(early_stopping[k]),
         #On calcule cette fois le test score à partir du dataset de validation␣
↪afin de déterminer les paramètres optimaux.
            'test_score': str(mlp.score(X_valid, y_valid)),
            'train_score': str(mlp.score(X_train, y_train)),
            'tol': str(tol[m]),
            'learning time':str(stop-start)
    })
    print(f"Test n°{loop} over.")


table = pd.DataFrame.from_dict(data)
table = table.replace(np.nan, '-')
table = table.sort_values(by='test_score', ascending=False)
table.head(10)
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
Test n°0 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Test n°1 over.
Test n°2 over.
Test n°3 over.
Test n°4 over.
Test n°5 over.
Test n°6 over.
Test n°7 over.
Test n°8 over.
```

```
Test n°9 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°10 over.
Test n°11 over.
Test n°12 over.
Test n°13 over.
Test n°14 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°15 over.
Test n°16 over.
Test n°17 over.
Test n°18 over.
Test n°19 over.
Test n°20 over.
Test n°21 over.
Test n°22 over.
Test n°23 over.
Test n°24 over.
Test n°25 over.
Test n°26 over.
Test n°27 over.
Test n°28 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°29 over.
Test n°30 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (800) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°31 over.
Test n°32 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Test n°33 over.
Test n°34 over.
Test n°35 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Test n°36 over.
Test n°37 over.
Test n°38 over.
Test n°39 over.
Test n°40 over.
Test n°41 over.
Test n°42 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Test n°43 over.
Test n°44 over.
Test n°45 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Test n°46 over.
Test n°47 over.
Test n°48 over.
Test n°49 over.
Test n°50 over.
Test n°51 over.
Test n°52 over.
Test n°53 over.
```

```
C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
```

```
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°54 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°55 over.
Test n°56 over.
Test n°57 over.
Test n°58 over.
Test n°59 over.
Test n°60 over.
Test n°61 over.
Test n°62 over.
Test n°63 over.
Test n°64 over.
Test n°65 over.
Test n°66 over.
Test n°67 over.
Test n°68 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°69 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°70 over.
Test n°71 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°72 over.
Test n°73 over.
Test n°74 over.
```

```
Test n°75 over.
Test n°76 over.
Test n°77 over.
Test n°78 over.
Test n°79 over.
Test n°80 over.
Test n°81 over.
Test n°82 over.
Test n°83 over.
Test n°84 over.
Test n°85 over.
Test n°86 over.
Test n°87 over.
Test n°88 over.
Test n°89 over.
Test n°90 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°91 over.
Test n°92 over.
Test n°93 over.
Test n°94 over.

C:\Users\felix\anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:571:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

Test n°95 over.
Test n°96 over.
Test n°97 over.
Test n°98 over.
Test n°99 over.
```

[7]:

```
   activation  learning_rate_init hidden_layer_sizes max_iter early_stopping  \
4        relu              0.0001       (1000, 1000)     1500           True
83       relu              0.0010       (1000, 1000)      800           True
27       relu              0.0010          (100, 10)      100          False
67   logistic              0.0050            (10, 2)     1500           True
85       relu              0.0050           (10, 10)      500          False
44   logistic              0.1000            (10, 2)      100           True
95   logistic              0.1000             (100,)     1500           True
26       relu              0.0010          (100, 10)      500          False
```

```
70       relu                0.0020          (100, 10)     1500           False
97       tanh                0.0050           (10, 10)      800           False

            test_score           train_score      tol         learning time
4    0.817929270815983   0.8450802509222831   0.0005    117.1925060749054
83  0.8174955377053713   0.8390217734635804    0.001     36.72002387046814
27  0.8154664798217646   0.7965854503725822   0.0005     0.910881757736206
67   0.815321158112775   0.8052801532529619   0.0001    1.4996156692504883
85  0.8140906295298296   0.7796998961171722   0.0005    0.3196103572845459
44  0.8139547173424885   0.8035999145716243   0.0001   0.30989861488342285
95  0.8134720833350176   0.7884261745508402     0.01   0.16295886039733887
26  0.8090207737830764   0.7742562083461876    0.001   0.48030734062194824
70   0.807944108681685     0.76327077080489     0.01    0.2318248748779297
97  0.8076400050077982   0.8224694198275926   0.0001    0.7591807842254639
```

[8]:

```python
#On crée un multi-layer perceptron avec les paramètres donnant le plus haut␣
↪score pour le dataset de vaildation.
mlp = MLPRegressor(hidden_layer_sizes=(1000,1000), max_iter=1500,
                            early_stopping=True,
                            learning_rate_init=0.0001, tol=0.0005,
                            activation='relu', random_state=1)
start=time.time()
mlp.fit(X_train, y_train)
stop=time.time()
print('Train score: ', mlp.score(X_train, y_train))
print('Test score:  ', mlp.score(X_test, y_test))
print('Training time: ', str(stop-start))
```

```
Train score:  0.7995726932952246
Test score:   0.8151386237877918
Training time:  49.99225831031799
```

Le test score est ici satisfaisant, il est néanmoins plus bas que celui correspondant au dataset de validation. Ce qui est logique puisque on a calculé les paramètres optimaux pour le dataset de validation, et non de test