mlp

April 10, 2022

1 Lab 3 : Ludovic Mocquais et Mariétou Sarr

4IRB2

2 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.

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

[2]:
```
# Preprocessing: scale input data
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
```

[3]:
```
# 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)
```

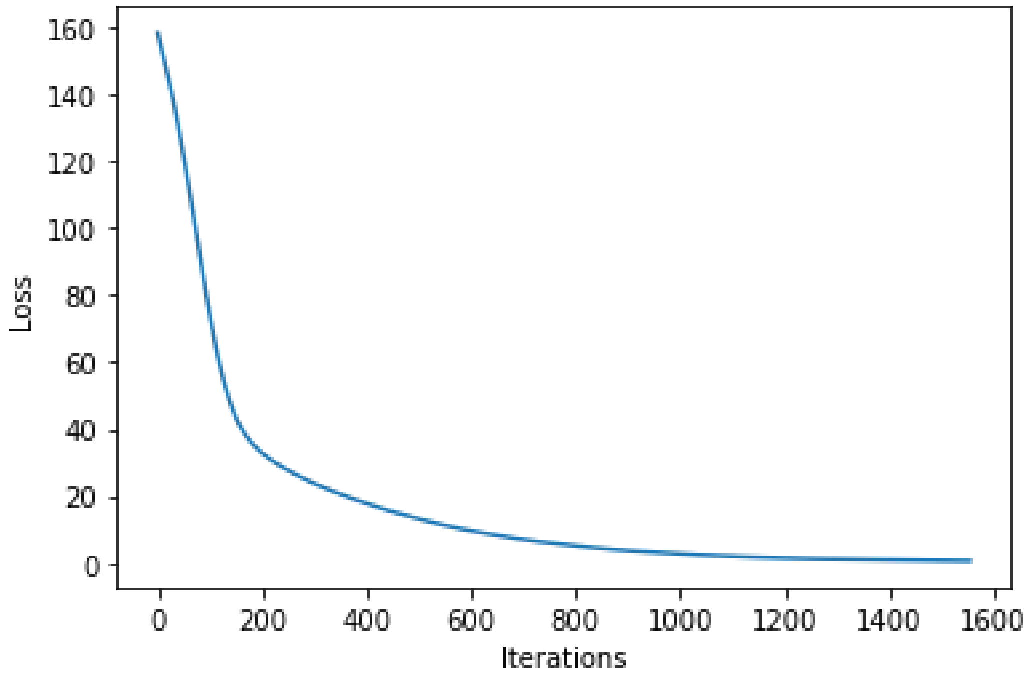
[4]:
```
# Define a multi-layer perceptron (MLP) network for regression
from sklearn.neural_network import MLPRegressor
mlp = MLPRegressor(max_iter=3000, random_state=1) # define the model, with␣
 ↪default params
mlp.fit(x_train, y_train) # train the MLP
```

[4]:
```
MLPRegressor(max_iter=3000, random_state=1)
```

[5]:
```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")
```

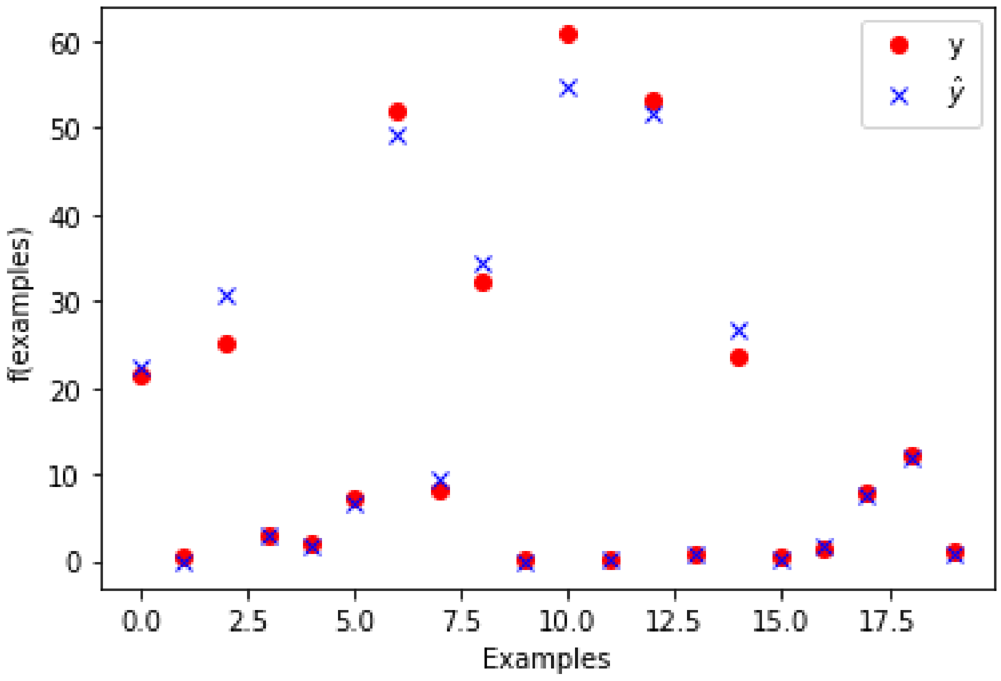
```
Train score:  0.9940765369322633
Test score:   0.9899773031580283
```

[5]:
```
Text(0, 0.5, 'Loss')
```



[6]:
```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)")
```

[6]:
```
Text(0, 0.5, 'f(examples)')
```



2.0.1 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? The network is a multi-layer perceptron wwith hidden layers which we can choose se number and the activation function.
- What it is the algorithm used for training? Is there algorithm available that we mentioned during the courses? The training algorithm is the stochastic gradient descent. Yes it was mentionned during the courses.
- How does the training algorithm decides to stop the training? We have tol that is the minimum desired improvement of the loss or score for consecutive iterations. The training stops when when there is no change.

3 Onto a more challenging dataset: house prices

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

[7]:
```
# clean all previously defined variables for the sailing boats
%reset -f
```

[8]:
```
"""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
```

[8]:

```
   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.

3.0.1 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`.

[9]:

```
# TODO
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)
```

3.0.2 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

[10]:
```
# TODO
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(x_train)
X_test = sc.fit_transform(x_test)
```

3.1 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) ?

  Yes because with a good train score we tend to overfit

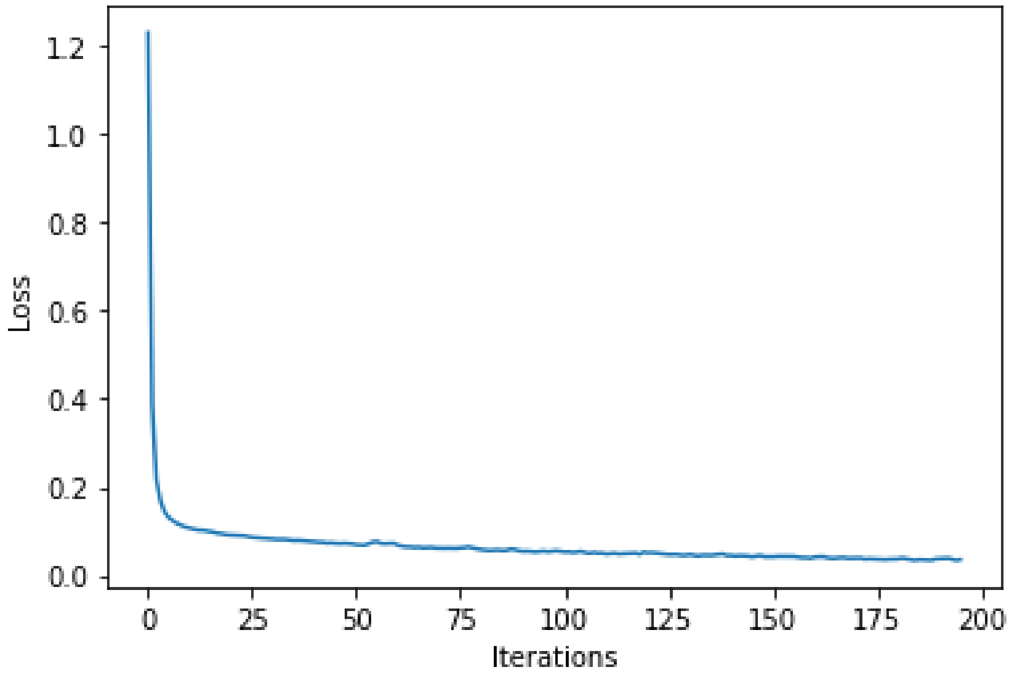
- Explain how the parameters you chose allow the learned model to overfit.

  To increase the train score, we can start by adding mode hidden layers to predicate better the training value of the houses and delete the validation fraction so that all of the set is used for training. We can also decrease the tolerance, increase the number of iterations of the algorithm to get a more precise results.

```
[11]: # TODO
from sklearn.neural_network import MLPRegressor
from matplotlib import pyplot as plt


mlp = MLPRegressor(hidden_layer_sizes=(100,100,100), tol = 1e-8,
                   learning_rate='adaptive',validation_fraction=0,
                    max_iter = 2500)


mlp.fit(X_train, y_train) # train the MLP

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:  0.9143311098389366
Test score:   0.3194622548310331
```

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



3.2 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)

[12]:
```
# TODO
from time import time
data = []

def test_conf(conf) :

    t = time()
    mlp = MLPRegressor(**conf)
    mlp.fit(X_train, y_train) # train the MLP
    conf["Time"] = time()-t

    conf["Train score"]=mlp.score(X_train, y_train)
    conf["Test score"]= mlp.score(X_test, y_test)

    data.append(conf)
    return mlp


conf_d = {"hidden_layer_sizes" : (100,), "activation" : 'relu', "tol" :0.0001,␣
 ↪"early_stopping" : False, "max_iter":200, "learning_rate_init":0.001}
test_conf(conf_d)
```

```
/usr/local/insa/anaconda/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
```

```
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

```
[12]: MLPRegressor()
```

```
[13]: conf1 = {"hidden_layer_sizes" : (100,100,100, 100), "activation" : 'relu',␣
       ↪"tol" :0.0001, "early_stopping" : False, "max_iter":200,␣
       ↪"learning_rate_init":0.001}
      test_conf(conf1)
```

```
[13]: MLPRegressor(hidden_layer_sizes=(100, 100, 100, 100))
```

```
[14]: conf2 = {"hidden_layer_sizes" : (10,), "activation" : 'relu', "tol" :0.0001,␣
       ↪"early_stopping" : False, "max_iter":200, "learning_rate_init":0.001}
      test_conf(conf2)
```

```
/usr/local/insa/anaconda/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

```
[14]: MLPRegressor(hidden_layer_sizes=(10,))
```

```
[15]: conf3 = {"hidden_layer_sizes" : (100,), "activation" : 'identity', "tol" :0.
       ↪0001, "early_stopping" : False, "max_iter":200, "learning_rate_init":0.001}
      test_conf(conf3)
```

```
[15]: MLPRegressor(activation='identity')
```

```
[16]: conf4 = {"hidden_layer_sizes" : (100,), "activation" : 'logistic', "tol" :0.
       ↪0001, "early_stopping" : False, "max_iter":200, "learning_rate_init":0.001}
      test_conf(conf4)
```

```
/usr/local/insa/anaconda/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

```
[16]: MLPRegressor(activation='logistic')
```

```
[17]: conf5 = {"hidden_layer_sizes" : (100,), "activation" : 'tanh', "tol" :0.0001,␣
       ↪"early_stopping" : False, "max_iter":200, "learning_rate_init":0.001}
      test_conf(conf5)
```

```
/usr/local/insa/anaconda/lib/python3.8/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[17]:
```
MLPRegressor(activation='tanh')
```

[18]:
```
conf6 = {"hidden_layer_sizes" : (100,), "activation" : 'relu', "tol" :0.01,␣
 ↪"early_stopping" : False, "max_iter":200, "learning_rate_init":0.001}
test_conf(conf6)
```

[18]:
```
MLPRegressor(tol=0.01)
```

[19]:
```
conf7 = {"hidden_layer_sizes" : (100,), "activation" : 'relu', "tol" :0.0001,␣
 ↪"early_stopping" : True, "max_iter":200, "learning_rate_init":0.001}
test_conf(conf7)
```

[19]:
```
MLPRegressor(early_stopping=True)
```

[20]:
```
conf8 = {"hidden_layer_sizes" : (100,), "activation" : 'relu', "tol" :0.0001,␣
 ↪"early_stopping" : False, "max_iter":2000, "learning_rate_init":0.001}
test_conf(conf8)
```

[20]:
```
MLPRegressor(max_iter=2000)
```

[21]:
```
conf9 = {"hidden_layer_sizes" : (100,), "activation" : 'relu', "tol" :0.0001,␣
 ↪"early_stopping" : False, "max_iter":200, "learning_rate_init":0.1}
final_mlp = test_conf(conf9)
```

[22]:
```
#The best configuration is one that gives us the best score for both training␣
 ↪and testing set
#We should then compromise on the number of layers and the maximum of iterations

best_conf = {"hidden_layer_sizes" : (100,25), "activation" : 'relu', "tol" :0.
 ↪0001, "early_stopping" : True,
             "learning_rate" : 'adaptive',"validation_fraction" : 0.1,␣
 ↪"max_iter":500, "learning_rate_init":0.001 }
test_conf(best_conf)
```

[22]:
```
MLPRegressor(early_stopping=True, hidden_layer_sizes=(100, 25),
             learning_rate='adaptive', max_iter=500)
```

[23]:
```
import numpy as np

table = pd.DataFrame.from_dict(data)
table = table.replace(np.nan, '-')
```

```python
table = table.sort_values(by='Test score', ascending=False)
table
```

[23]:

```
        hidden_layer_sizes activation     tol  early_stopping  max_iter  \
10               (100, 25)       relu  0.0001            True       500
7                   (100,)       relu  0.0001            True       200
0                   (100,)       relu  0.0001           False       200
6                   (100,)       relu  0.0100           False       200
9                   (100,)       relu  0.0001           False       200
5                   (100,)       tanh  0.0001           False       200
4                   (100,)   logistic  0.0001           False       200
8                   (100,)       relu  0.0001           False      2000
3                   (100,)   identity  0.0001           False       200
2                    (10,)       relu  0.0001           False       200
1     (100, 100, 100, 100)       relu  0.0001           False       200

    learning_rate_init       Time  Train score  Test score learning_rate  \
10               0.001  13.593365     0.800659    0.796612      adaptive
7                0.001  18.896147     0.797888    0.785586             -
0                0.001  25.887275     0.818538    0.781142             -
6                0.001   2.804348     0.739251    0.767593             -
9                0.100   4.988372     0.822721    0.765393             -
5                0.001  35.992563     0.788781    0.753041             -
4                0.001  28.594653     0.740026    0.752026             -
8                0.001  48.092507     0.838026    0.737605             -
3                0.001   4.758420     0.688997    0.725396             -
2                0.001   0.832452     0.736469    0.687174             -
1                0.001  36.901365     0.902037    0.566535             -

   validation_fraction
10                 0.1
7                    -
0                    -
6                    -
9                    -
5                    -
4                    -
8                    -
3                    -
2                    -
1                    -
```

3.3 Evaluation

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

    * the higher the number and size of hidden layer the smaller the test score, the best the train score and the higher the computation time
    * the relu activation function give the best test and train score, followed by the logistic activation function but that one takes more time to compute
    * less iterations give the best test score, worst train score and best computation time
    * early_stopping set to True is good to optimize the computation time
    * higher tolerance gives the quickest computation time but decreases the test and train score
    * higher initial learning rate increases the train score but decreases the test score
    * validation fraction helps with overfitting so it improves the test score
    * learning rate optimize time
  - Our best config would be : **best_conf = {"hidden_layer_sizes" : (100,50,), "activation" : 'relu', "tol" :0.0001, "early_stopping" : True, "learning_rate" : 'adaptive',"validation_fraction" : 0.2, "max_iter":500, "learning_rate_init":0.001}**

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.

[24]:
```
x_test_new = pd.DataFrame(cal_housing.data,columns=cal_housing.
 ↪feature_names)[2000:2500]
Y = cal_housing.target[2000:2500]
X_test_new = sc.fit_transform(x_test_new)
print("Test score 1= "+ str(final_mlp.score(X_test_new, Y)))

x_test_new = pd.DataFrame(cal_housing.data,columns=cal_housing.
 ↪feature_names)[3000:3500]
Y = cal_housing.target[3000:3500]
X_test_new = sc.fit_transform(x_test_new)
print("Test score 2= "+ str(final_mlp.score(X_test_new, Y)))

x_test_new = pd.DataFrame(cal_housing.data,columns=cal_housing.
 ↪feature_names)[4000:4500]
Y = cal_housing.target[4000:4500]
X_test_new = sc.fit_transform(x_test_new)
print("Test score 3= "+ str(final_mlp.score(X_test_new, Y)))
```

```
Test score 1= -9.00692735001358
Test score 2= -0.3160968532235562
Test score 3= -0.10173092242689696
```

We can see that with data outside of our training or testing set, the test score are terrible. This is because our algorithm is biased and our training set is small.

```
[ ]:
```