svr & mlpnn regression

November 10, 2021

1 SVR & MLPNN Regression Exercise

The target of this assignment is to design a regression system to predict boston housing prices. The regression algorithms should contain support vector regression and MLPNN.

1.1 Table of Contents

- 1-Packages
- 2-Load the Dataset
- 3-Support Vector Regression
- 4-MLPNN Regression

1.2 1 - Packages

First import all the packages needed during this assignment.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVR
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error,mean_squared_error
from sklearn.neural_network import MLPRegressor

%matplotlib inline

%load_ext autoreload
%autoreload 2
```

1.3 2 - Load the Dataset

[2]:
```
%%capture --no-display

boston = load_boston()
X = boston.data
y = boston.target

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=10)
```

```
y_train = np.array(y_train).reshape(-1,)
y_test = np.array(y_test).reshape(-1,)
```

[3]:
```
# As a sanity check, print out the size of the training and test data
print('Training data shape: ', x_train.shape)
print('Training labels shape: ', y_train.shape)
print('Test data shape: ', x_test.shape)
print('Test labels shape: ', y_test.shape)
```

```
Training data shape:  (404, 13)
Training labels shape:  (404,)
Test data shape:  (102, 13)
Test labels shape:  (102,)
```

1.4 3 - Support Vector Regression

The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only minor differences. The main idea behind SVR is to decide a decision boundary distance from the original hyperplane such that data points closest to the hyperplane or the support vectors are within that boundary line.

Optimization objective:

$$\begin{aligned} \min \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) \\ \text{subject to} \quad & y_i - wx_i - b \leq \varepsilon + \xi_i \\ & wx_i + b - y_i \leq \varepsilon + \xi_i^* \\ & \xi_i, \xi_i^* \geq 0 \end{aligned}$$

Mapping function:

$$K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2)$$
$$y = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) \cdot K(x_i, x) + b$$

In this section, we use *Radial basis function (rbf)* as kernel function to build our SVM. The tolerence factor C is set to 10^2 and the γ in rbf is set to 0.1.

[4]:
```
# Training
model = SVR(kernel='rbf', C=1e2, gamma=0.1)
model.fit(x_train, y_train)
```

[4]:
```
SVR(C=100.0, gamma=0.1)
```

[5]:
```
# Evaluation
pred = model.predict(x_test)
mae = mean_absolute_error(y_test, pred)
mse = mean_squared_error(y_test, pred)
```

```python
print('MAE ', mae)
print('MSE ', mse)
```

```
MAE  6.996249978087986
MSE  101.36142683921726
```

1.5 4 - MLPNN Regression

In this section, we would build a three-layer MLPNN using `sklearn`. *ReLU* is selected as activation funciton for each hidden layer and Adam optimizer is applied in the training stage.

The mapping function of the network can be describe as:

$$
\begin{aligned}
O_1 &= ReLU(w_1^T X + b_1) \\
O_2 &= ReLU(w_2^T X + b_2) \\
O_3 &= ReLU(w_3^T X + b_3)
\end{aligned}
$$

where O_1, O_2, O_3 is the output of layer 1, layer 2 and layer 3 respectively.

We use *Mean Squared Error (MSE)* as loss function, which is defined as:

$$
\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (f(x) - y)^2
$$

where $f(x)$ is the output of MLPNN and y is the corresponding label.

[6]:
```python
model = MLPRegressor(
    hidden_layer_sizes=(50, 25),
    activation='relu',
    solver='adam'
)
model.fit(x_train, y_train)
```

[6]:
```
MLPRegressor(hidden_layer_sizes=(50, 25))
```

[7]:
```python
# Evaluation
pred = model.predict(x_test)
mae = mean_absolute_error(y_test, pred)
mse = mean_squared_error(y_test, pred)
print('MAE ', mae)
print('MSE ', mse)
```

```
MAE  6.137284501473741
MSE  89.44981470868586
```