Artificial Neural Networks Lab

MLP for Regression

Lab 3

Dr. Konstantinos Karampidis

MLP for Regression problems

MLP's are also suitable for regression prediction problems where a real-valued quantity is predicted given a set of inputs. Data are often provided in a tabular format, such in a CSV file or a spreadsheet.

MLP for Regression problems

`sklearn.neural_network.MLPRegressor`(*hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*)

- **hidden_layer_sizes** *array-like of shape(n_layers - 2,), default=(100,)*The ith element represents the number of neurons in the ith hidden layer.
- **Activation** *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*Activation function for the hidden layer.
  - 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
  - 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
  - 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
  - 'relu', the rectified linear unit function, returns f(x) = max(0, x)

Solver *{'lbfgs', 'sgd', 'adam'}, default='adam'* The solver for weight optimization.

- 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- 'sgd' refers to stochastic gradient descent.
- 'adam' refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

Note: The default solver 'adam' works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, 'lbfgs' can converge faster and perform better.

MLP for Regression problems

learning_rate *{'constant', 'invscaling', 'adaptive'}, default='constant'*Learning rate schedule for weight updates.

- 'constant' is a constant learning rate given by 'learning_rate_init'.
- 'invscaling' gradually decreases the learning rate learning_rate_ at each time step 't' using an inverse scaling exponent of 'power_t'. effective_learning_rate = learning_rate_init / pow(t, power_t)
- 'adaptive' keeps the learning rate constant to 'learning_rate_init' as long as training loss keeps decreasing. Each time two consecutive epochs fail to decrease training loss by at least tol, or fail to increase validation score by at least tol if 'early_stopping' is on, the current learning rate is divided by 5.

Only used when solver='sgd'.

- **batch_size** *int, default='auto'*Size of minibatches for stochastic optimizers. If the solver is 'lbfgs', the classifier will not use minibatch. When set to "auto", batch_size=min(200, n_samples).
- **early_stopping** *bool, default=False*Whether to use early stopping to terminate training when validation score is not improving. If set to true, it will automatically set aside 10% of training data as validation and terminate training when validation score is not improving by at least tol for n_iter_no_change consecutive epochs. Only effective when solver='sgd' or 'adam'.
- **max_iter** *int, default=200*Maximum number of iterations. The solver iterates until convergence (determined by 'tol') or this number of iterations. For stochastic solvers ('sgd', 'adam'), note that this determines the number of epochs (how many times each data point will be used), not the number of gradient steps.
- **random_state** *int, RandomState instance, default=None*Determines random number generation for weights and bias initialization, train-test split if early stopping is used, and batch sampling when solver='sgd' or 'adam'. Pass an int for reproducible results across multiple function calls.

MLP for Regression problems

Example

For this example, the dataset *House Sales in King County*, *USA* from the statistic platform [“Kaggle”]() was used. Download it from eclass

```
# Loading the libraries and the data
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.neural_network import MLPRegressor

from sklearn import metrics

from sklearn.model_selection import GridSearchCV
df = pd.read_csv('house_prices.txt')
df = df.drop(['id', 'date', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'long'], axis=1)
```

MLP for Regression problems

```
# Data pre-processing
x = df.drop('price', axis=1)
y = df['price']

# Split dataset

trainX, testX, trainY, testY = train_test_split(x, y, test_size = 0.2)
sc=StandardScaler()

# Scaling
scaler = sc.fit(trainX)
trainX_scaled = scaler.transform(trainX)
testX_scaled = scaler.transform(testX)
```

MLP for Regression problems

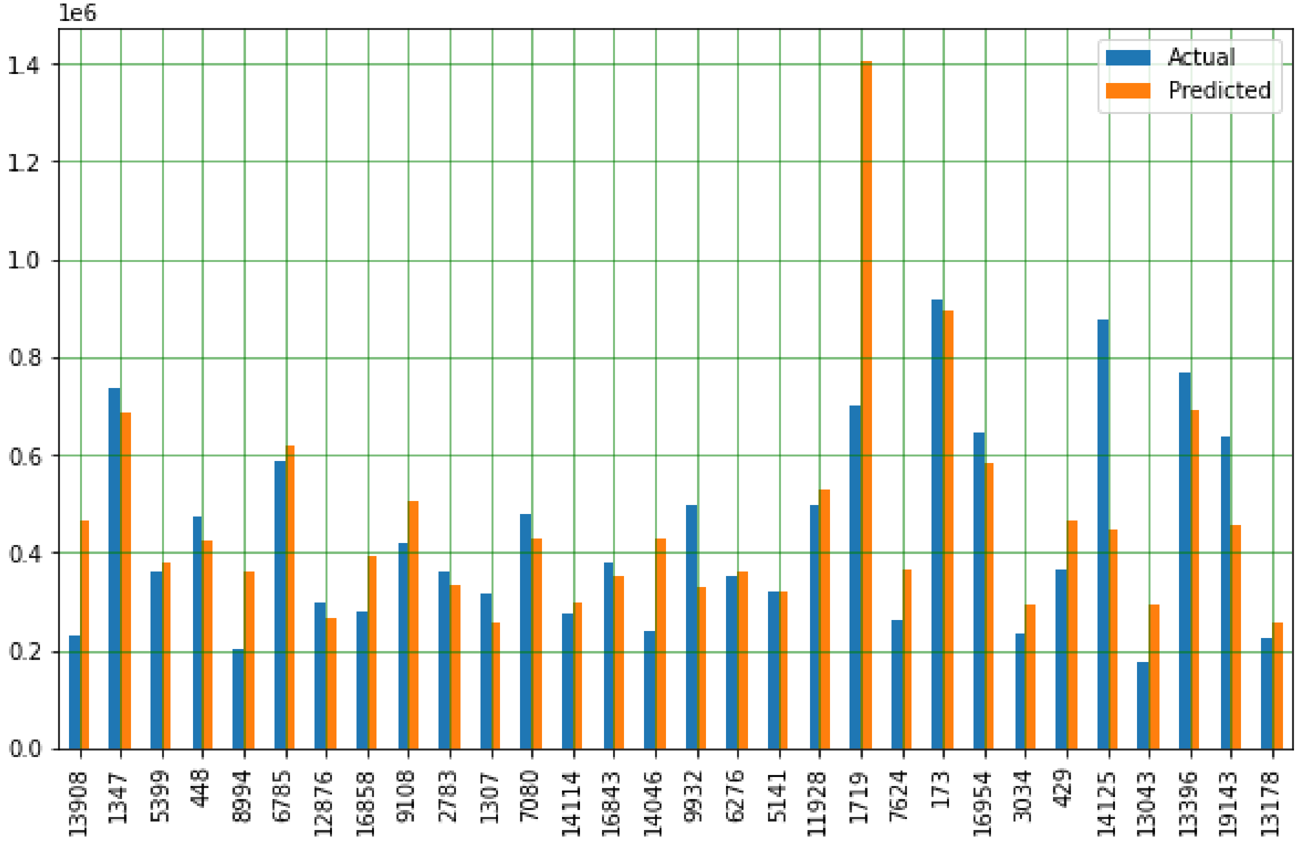
```
# MLPRegressor
mlp_reg = MLPRegressor(hidden_layer_sizes=(150,100,50),
                       max_iter = 300,activation = 'relu',
                       solver = 'adam')

mlp_reg.fit(trainX_scaled, trainY)

# Model Evaluation
y_pred = mlp_reg.predict(testX_scaled)
df_temp = pd.DataFrame({'Actual': testY, 'Predicted': y_pred})
df_temp.head()
```

MLP for Regression problems

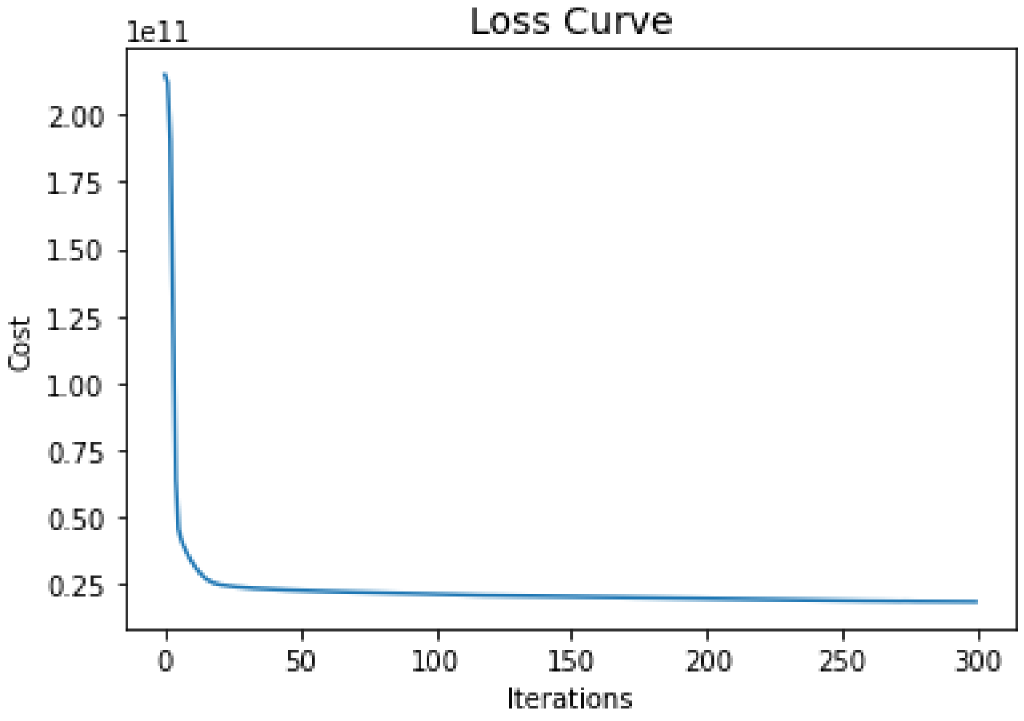
```
# Model Evaluation
df_temp = df_temp.head(30)
df_temp.plot(kind='bar',figsize=(10,6))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



MLP for Regression problems

```
# Model Evaluation
print('Mean Absolute Error:', metrics.mean_absolute_error(testY, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(testY, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(testY, y_pred)))
```

```
# Model Evaluation
plt.plot(mlp_reg.loss_curve_)
plt.title("Loss Curve", fontsize=14)
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.show()
```



MLP for Regression problems

Now let's try to fine tune our neural network. That is, to change the number of neurons in the hidden layers, the number of epochs, the optimizer , the learning rate etc.

```
# Hyper Parameter Tuning
param_grid = {
    'hidden_layer_sizes': [(150,100,50), (120,80,40), (100,50,30)],
    'max_iter': [50, 100],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
```

The grid search provided by GridSearchCV exhaustively generates candidates from a grid of parameter values specified with the param_grid parameter.

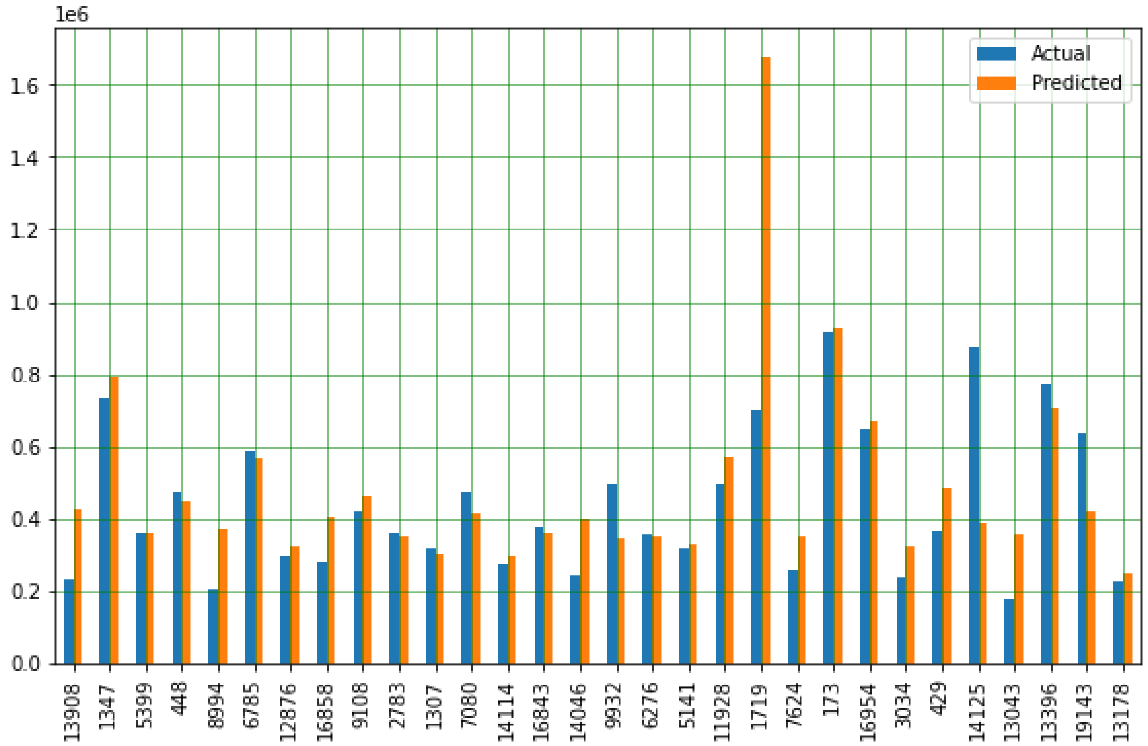
```
grid = GridSearchCV(mlp_reg, param_grid, n_jobs= -1, cv=5)
grid.fit(trainX_scaled, trainY)

print(grid.best_params_)
```

MLP for Regression problems

```
grid_predictions = grid.predict(testX_scaled)
df_temp2 = pd.DataFrame({'Actual': testY, 'Predicted': grid_predictions})
df_temp2.head()

df_temp2 = df_temp2.head(30)
df_temp2.plot(kind='bar',figsize=(10,6))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```




```
print('Mean Absolute Error:', metrics.mean_absolute_error(testY, grid_predictions))
print('Mean Squared Error:', metrics.mean_squared_error(testY, grid_predictions))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(testY, grid_predictions)))
```

Evaluation Metrics in Regression Analysis

1. R^2

The value R^2 tells us how much variance in the outcome variable can be explained by the predictors.

```
print('R²: ' + str(lm.score(trainX_scaled, trainY)))
```

Evaluation Metrics in Regression Analysis

2. Mean Absolute Error (MAE)

MAE stands for Mean Absolute Error. Each residual is calculated for each data point, taking only the absolute value of each point. This prevents positive and negative residuals from not canceling each other out. Then the average of all residuals is taken.

Example

Actual Costs:		Predicted Costs:	
Room 1:	300	Room 1:	330
Room 2:	350	Room 2:	320
Room 3:	400	Room 3:	440
Room 4:	600	Room 4:	520

Example

Error = Actual Costs – Predicted Costs

Actual Costs:		Predicted Costs:		Error:
Room 1:	300	Room 1:	330	-30
Room 2:	350	Room 2:	320	30
Room 3:	400	Room 3:	440	-40
Room 4:	600	Room 4:	520	80

The absolute values are summed up and we calculate the mean value of the absolute error values.

Actual Costs:		Predicted Costs:		Error:
Room 1:	300	Room 1:	330	-30
Room 2:	350	Room 2:	320	30
Room 3:	400	Room 3:	440	-40
Room 4:	600	Room 4:	520	80

180 / 4 = 45

In this example, we can say that our model predictions are off by about €45

```
print('Mean Absolute Error:', metrics.mean_absolute_error(testY, y_pred))
```

Evaluation Metrics in Regression Analysis

3. Mean Squared Error (MSE)

MAE stands for Mean Squared Error. It is like the Mean Absolute Error, but differs from it in that it squares the difference before summing instead of just taking the absolute value

Since the difference is squared, the MSE will almost always be greater than the MAE. For this reason, the MSE cannot be directly compared to the MAE. Only the error metric of our model can be compared to that of a competing model. The effect of the quadratic term in the MSE equation is most apparent when there are outliers in the data set. Whereas in MAE this residual contributes proportionally to the total error, in MSE the error grows quadratically. Thus, existing outliers contribute to a much higher total error in MSE than they would in MAE.

```
print('Mean Squared Error:', metrics.mean_squared_error(testY, y_pred))
```

Evaluation Metrics in Regression Analysis

4. Root Mean Squared Error (RMSE)

RMSE stands for Root Mean Squared Error, and this is the square root of the MSE. We know that the MSE is squared. Thus, its units do not match those of the original output. To convert the error metric back to similar units, the RMSE is used. This simplifies the interpretation again. Both MSE and RMSE are affected by outliers. Their common goal is to measure how large the residuals are distributed. Their values lie in the range between zero and positive infinity.

```
print('Root Mean Squared Error:',
metrics.mean_squared_error(testY, y_pred,
squared=False))
```

Evaluation Metrics in Regression Analysis

5. Mean Absolute Percentage Error (MAPE)

MAPE stands for Mean Absolute Percentage Error and is the percentage equivalent of MAE.
MAPE indicates how far the predictions of the model used deviate on average from the corresponding outputs. Both MAPE and MAE are accompanied by a clear interpretation, as percentages are easier to conceptualize for most people.
Both MAPE and MAE are robust to the effects of outliers. This is due to the use of absolute values.

```
print('Mean Absolute Percentage Error:',
metrics.mean_absolute_percentage_error(testY,
y_pred))
```

Exercise

Download Concrete_Data.xls from UCI repository https://archive.ics.uci.edu/dataset/165/concrete+compressive+strength and build a MLP Regressor to predict the compressive strength of concrete.

You may have to install the xlrd library to read .xls files. Open anaconda prompt and write: pip install xlrd

Ex

Thank you !!!