Deep Learning: Mastering Neural Networks

DL-ENG-002 1.2 Assignment: Regression Model Optimization

Daniel Nascimento
danielsdn0725@gmail.com

General data

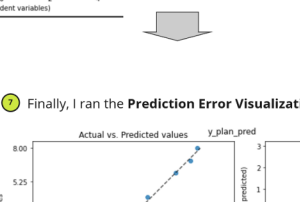
I created a simple python program - just for add hoc calculations and plotting.

The **scikit-learn** machine learning library was the best choice, after all, it's tools and modules can give many possible approaches for an assignment like this.

It was necessary to update the environment in order to run the **PredictionErrorDisplay** module.

```
!pip install --upgrade scikit-learn
```

In the spreadsheet, I set the plot parameters to show the trend line and display **R2.**
It has also highlighted the line's equation $y = 2.47 + 1.18X$



X	Y
1	4
-2	3
3	6
4.5	8
0	2
-4	-3
-1	-2
4	7

Principal modules

- LinearRegression
- mean_squared_error
- r2_score
- PredictionErrorDisplay
- cross_val_predict

Test 1 - Using the basic way of the linear regression function library

1. First I ran the module's linear regression: **LinearRegression**

```
print(X)
[[ 1. ]
 [-2. ]
 [ 3. ]
 [ 4.5]
 [ 0. ]
 [-4. ]
 [-1. ]
 [ 4. ]
 [-1. ]]
```

```
print(y) # True values of y data
[[ 4. ]
 [ 3. ]
 [ 6. ]
 [ 8. ]
 [ 2. ]
 [-3. ]
 [-2. ]
 [ 7. ]
 [ 2.5]]
```

```
#linear curve
y_pred1=lr.predict(X)

y_pred2=lr.fit(X,y, sample_weight=None) ### fit original

#lr.fit(X,y_pred, sample_weight=None)

#y_pred = cross_val_predict(lr, X, y, cv=4)
```

2. I named **y_plan_pred**, the $\hat{y}$. Those are the predicted values: these

```
[17] y_plan_pred=lr.predict(X)
```

```
lr.fit(X,y, sample_weight=None) ### fit original
```

```
print(y_plan_pred)
[[ 3.64457071]
 [ 0.1104798 ]
 [ 6.00063131]
 [ 7.76767677]
 [ 2.4665404 ]
 [-2.24558081]
 [ 1.2885101 ]
 [ 7.17866162]
 [ 1.2885101 ]]
```

3. To calculate the Loss, I utilized the **MSE model** (mean squared error) using the **mean_squared_error** function.

Result: 2.403374017957351

```
# Mean square Error for y_pred0
from sklearn.metrics import mean_squared_error

mean_squared_error(y, y_plan_pred)

2.403374017957351
```

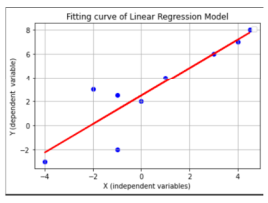
4. For R^2, I ran the **r2_score** function
Result: 0.8089565304665894

```
# R2 Score for the split model
r2_score(y, y_plan_pred, multioutput='variance_weighted')

0.8089565304665894
```

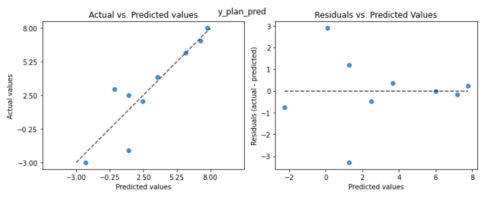
5. I printed the data and composed the line's equation through the obtained coefficients and scores.

```
Coeficieentes e scores do modelo da planilha (minimos quadrados) sem otimização

R2 score:  0.8089565304665894

Interceptação de y:  2.466540404040404
Slope da reta:  [1.1780303]

Equação da Reta: y = 2.47 + 1.18X
```

6. Sequentially, I plotted the **fitting line**.



7. Finally, I ran the **Prediction Error Visualization** of a regression model.



Test 2 - Trying to optimize the model with cross_val_predict

1st Iteration

1. Generate cross-validated estimates for each input data point.
The data is split according to the **cv** parameter. Each sample belongs to exactly one test set, and its prediction is computed with an estimator fitted on the corresponding training set.

```
[34] # the fitting function     optimization with cost function J(0)

     lr = LinearRegression()

     # cross_val_predict *** k-fold cross-validation
     y_pred = cross_val_predict(lr, X, y, cv=6)

     lr.fit(X,y, sample_weight=None) ### fit original
```

The **cv** parameter determines the split strategy for cross validation

2. I set the **cv=6** (6-fold cross validation) parameter.

I named **y_pred**, the predicted values $\hat{y}$

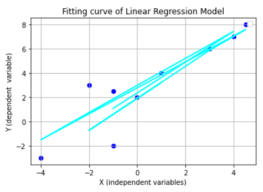
```
print(y_pred) #y_pred = cross_val_predict(lr, X, y, cv=6)
[[ 3.21      ]
 [-0.73      ]
 [ 5.86363636]
 [ 7.58522727]
 [ 2.77129472]
 [-1.51149915]
 [ 1.84711965]
 [ 7.2540395 ]
 [ 1.08271787]]
```

3. To calculate the Loss, I utilized the **MSE model** (mean squared error) using the **mean_squared_error** function.
Value: 3.8235240051501416

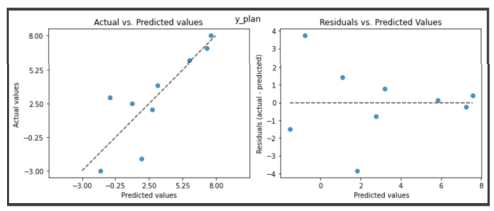
```
# Mean square Error #y_pred = cross_val_predict(lr, X, y, cv=6)
from sklearn.metrics import mean_squared_error

mean_squared_error(y, y_pred)

3.8235240051501416
```

4. For R^2, I ran the **r2_score** function
Value: 0.6960692400224128

```
# R2 Score #y_pred = cross_val_predict(lr, X, y, cv=6)
r2_score(y, y_pred, multioutput='variance_weighted')

0.6960692400224128
```



5. **Oddly, the plot of the line for the **y_pred** values got a totally unexpected shape.



6. Finally, I ran the **Prediction Error Visualization** of a regression model.

2nd iteration

1. I set the **cv=none** (default 5-fold cross validation) parameter.

```
# cross_val_predict *** k-fold cross-validation requires at least one
# k-fold cross-validation requires at least one
y_pred = cross_val_predict(lr, X, y)
```

```
print(y_pred) #after y_pred = cross_val_predict(lr, X, y)
[[ 3.21      ]
 [-0.73      ]
 [ 5.86363636]
 [ 7.58522727]
 [ 2.77129472]
 [-1.51149915]
 [ 1.86290323]
 [ 7.42741935]
 [ 1.08271787]]
```

2. To calculate the Loss, I utilized the **MSE model** (mean squared error) using the **mean_squared_error** function.
Value: 3.850173227804849

```
# Mean square Error for the original Model (with no split in Y)
from sklearn.metrics import mean_squared_error

mean_squared_error(y, y_pred)

3.850173227804849
```

3. For R^2, I ran the **r2_score** function
Value: 0.6939509014208118

```
# R2 Score #after y_pred = cross_val_predict(lr, X, y)
r2_score(y, y_pred, multioutput='variance_weighted')

0.6939509014208118
```

3rd iteration

1. I set the **cv=4** (4-fold cross validation) parameter.

```
# cross_val_predict *** k-fold cross-validation requires at least one
# k-fold cross-validation requires at least one
y_pred = cross_val_predict(lr, X, y, cv=4)
```

```
print(y_pred) #after cross_val_predict(lr, X, y, cv=4)
[[ 3.17815192]
 [-0.73805795]
 [ 5.7889585 ]
 [ 7.703125  ]
 [ 2.5       ]
 [-0.02094972]
 [ 2.59078212]
 [ 7.19074334]
 [ 1.08976157]]
```

2. To calculate the Loss, I utilized the **MSE model** (mean squared error) using the **mean_squared_error** function.
Value: 5.222920322935 11

```
# Mean square Error #after cross_val_predict(lr, X, y, cv=4)
from sklearn.metrics import mean_squared_error

mean_squared_error(y, y_pred)

5.222929032293511
```

3. For R^2, I ran the **r2_score** function
Value: 0.5848309601415366

```
# R2 Score #after cross_val_predict(lr, X, y, cv=4)
r2_score(y, y_pred, multioutput='variance_weighted')

0.5848309601415366
```

Conclusions

1. The best results were obtained in **test 1**, where I ran only the linear regression function.
2. The results of **Test 1** are identical to the values in the excel spreadsheet.
3. When performing **Test 2**, seeking some optimization, the results were lower than **Test 1**.
4. In **Test 2**, performing **cross-validation** with K-folders, the results (I supposed) was compromised because we were using a very small dataset (only 9 observations).
5. Even though, changing the **cv** parameters of the **cross_val_predict** function, there were no improvements, but worse values (due to the size of the dataset as well)
6. I believe that, depending on the size of the dataset, some optimization would be possible through the use of **MSE** (Mean Squared Error) + **gradient descent**. If this institution considers it necessary, I can create an analysis using this scenario.