challenge

November 23, 2023

1 Challenge: Méthodes de Régression Avancées

Presented by: Omar ALLOUCH

[31]:
```
import pandas as pd
import numpy as np
import tensorflow as tf

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import RidgeCV, LassoCV, ElasticNet, ElasticNetCV
```

1.1 Import and first steps

[32]:
```
# Import the dataset
data = pd.read_csv('data.txt', sep=" ")
display(data.head())
```

```
         y      X1      X2      X3      X4      X5      X6      X7      X8  \
1  -1.3819 -0.3304  0.2842 -0.2587 -0.1855 -0.0179  0.1950 -1.6062 -0.5463
2  -4.3784 -1.3468  0.5185 -0.3685 -0.5757 -0.8651  0.3696 -0.8717 -1.3216
3  -4.2931  0.1663  0.5063 -0.3393 -1.2111 -0.2620  0.7885  0.1536  0.8682
4  13.2109 -0.0805 -0.3233 -0.4649 -0.5250  0.5482  0.5116  0.5214 -0.2874
5   5.1431  0.0711 -0.0750  0.9774  1.0988  1.1078 -1.2257  0.2114  0.3370

       X9  …    X191    X192    X193    X194    X195    X196    X197  \
1 -1.6920  … -0.1669  0.7056  0.6023 -0.5360  0.1092  0.1914  0.1673
2 -1.4954  … -0.4493 -1.5861 -2.6679 -1.5648 -0.1526 -0.5994 -0.2909
3 -0.2265  … -1.5629  0.0824 -0.2451  1.0143  0.4988  0.5196 -0.8992
4 -1.3224  … -0.9284  0.2909  0.1849  0.6542 -0.8008  0.2323 -0.6862
5 -1.4028  … -0.0233  0.1827  0.5129 -0.0415  1.8795 -0.4297 -1.2061

     X198    X199    X200
1 -1.2968  0.7273 -0.5954
2 -0.1614 -1.1927 -0.8196
3 -0.1279  0.0966 -0.6236
4 -0.8753  0.6655 -0.0076
5 -0.9366 -0.4383  0.4647
```

```
[5 rows x 201 columns]
```

[33]:
```python
X, y = data.drop('y', axis=1), data['y']
```

[34]:
```python
# Calculate the standard deviation of y as reference for the RMSE
y_std = np.std(y)
print("Standard deviation of y: ", y_std)
print("This acts as a reference for the RMSE, it's like predicting the mean of␣
 ↪y")
```

```
Standard deviation of y:  8.681768399451578
This acts as a reference for the RMSE, it's like predicting the mean of y
```

The main objective is to implement the best regression model that minimizes RMSE

The model should be trained on the training set and evaluated on the test set

The RMSE on the test set should be reported

1.2 Data preprocessing

[35]:
```python
# Standardize the data
scaler = StandardScaler()
X = scaler.fit_transform(X) # We reduce X only, not y
```

1.3 Regression models

1.3.1 Regular linear regression

We won't bother with trying the (regular) linear model, because knowing the shape of the data we are confident that such a model won't perform well, so we'll skip it.

1.3.2 Ridge and Lasso cross-validation

[36]:
```python
alphas = np.logspace(-10, 10, 21)

ridge_cv = RidgeCV(alphas, fit_intercept=False, cv=5,␣
 ↪scoring='neg_mean_squared_error')
lasso_cv = LassoCV(fit_intercept=False, cv=5)

# Fit the models
ridge_cv.fit(X, y)
lasso_cv.fit(X, y)

# Evaluate the models
ridge_cv_rmse = np.sqrt(-ridge_cv.best_score_)
lasso_cv_rmse = np.sqrt(np.min(lasso_cv.mse_path_))
```

```
# To get the RMSE of Lasso we take the square root of the minimum of the␣
↪mse_path_ (which is the mean squared error for each alpha)

# Best alpha
ridge_cv_alpha = ridge_cv.alpha_
lasso_cv_alpha = lasso_cv.alpha_

# Print the results
print("Ridge best alpha: ", ridge_cv_alpha)
print("Ridge RMSE: ", ridge_cv_rmse)
print("Lasso best alpha: ", lasso_cv_alpha)
print("Lasso RMSE: ", lasso_cv_rmse)
```

```
Ridge best alpha:  100.0
Ridge RMSE:  4.958307641667428
Lasso best alpha:  0.38562239545698485
Lasso RMSE:  4.006943725249836
```

As expected, Lasso performed very well. We'll give other models a shot before predicting with the Lasso model.

1.3.3 Elastic NET regression

[37]:
```
elastic = ElasticNet()

# Grid search
l1_ratio = np.linspace(0.01, 1, 99)
elastic_cv = ElasticNetCV(alphas=alphas, l1_ratio=l1_ratio,␣
↪fit_intercept=False, cv=5)

# Fit the model
elastic_cv.fit(X, y)

# Evaluate the model
elastic_cv_rmse = np.sqrt(np.mean(elastic_cv.mse_path_))
print("Elastic Net RMSE: ", elastic_cv_rmse)
```

```
/home/omar/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:628: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 2.563e+00, tolerance: 6.103e-01
  model = cd_fast.enet_coordinate_descent(
/home/omar/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:628: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 4.019e+00, tolerance: 6.103e-01
```

```
  model = cd_fast.enet_coordinate_descent(
```

```
Elastic Net RMSE:  7.525899571239208
```

```
/home/omar/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:628: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 6.297e-01, tolerance: 6.160e-01
  model = cd_fast.enet_coordinate_descent(
```

Among the family of "regression" models, Lasso was the best with an RMSE of 4.0069. We'll use it to do the prediction, but before that let's try to improve it with PCA.

1.4 PCA

[38]:
```
# Fit the PCA
pca = PCA()
pca.fit(X)

# Transform the data
X_pca = pca.transform(X)

# Run lasso on the transformed data
lasso_cv.fit(X_pca, y)

# Evaluate the model
lasso_cv_rmse_pca = np.sqrt(np.min(lasso_cv.mse_path_))
print("Lasso RMSE with PCA: ", lasso_cv_rmse_pca)
```

```
Lasso RMSE with PCA:  3.871935092055523
```

It's indeed better in terms of RMSE.

1.5 Neural Network

Let's give the neural network a shot.

[39]:
```
# Run the neural network 5 times and save the RMSE
rmse = []
for i in range(5):
    # Build the model
    nn = tf.keras.Sequential([
        tf.keras.layers.Dense(64, activation='relu', input_shape=[X.shape[1]]),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(1)
    ])

    # Compile the model
    nn.compile(
```

```
        optimizer='adam',
        loss='mse',
        metrics=['mse'],
    )

    # Train the model
    nn.fit(
        X, y,
        validation_split=0.2,
        batch_size=32,
        epochs=100,
        verbose=0
    )

    # Evaluate the model
    rmse.append(np.sqrt(nn.evaluate(X, y)[0]))

# Calculate the mean RMSE
nn_rmse = np.mean(rmse)
print("Neural Network RMSE: ", nn_rmse)
```

```
4/4 [==============================] - 0s 13ms/step - loss: 3.8998 - mse: 3.8998
4/4 [==============================] - 0s 22ms/step - loss: 4.4020 - mse: 4.4020
4/4 [==============================] - 0s 11ms/step - loss: 5.3568 - mse: 5.3568
4/4 [==============================] - 0s 3ms/step - loss: 4.9051 - mse: 4.9051
4/4 [==============================] - 0s 11ms/step - loss: 4.1845 - mse: 4.1845
Neural Network RMSE:  2.129541934340325
```

Although the RMSE seems to be way better, I wouldn't trust the neural network in our case because of the low number of data points, which would highly mean that it's overfitting.

1.6 Predicting on Xtest

1.6.1 Lasso

[40]:
```
# Apply on the test set
test = pd.read_csv('Xtest.txt', sep=" ")

# Standardize the data
scaler = StandardScaler()
test = scaler.fit_transform(test)

# Apply PCA
test = pca.transform(test)

# Predict the values with
y_pred = lasso_cv.predict(test)
```

```python
# Set precision to 4 decimals
y_pred = np.around(y_pred, decimals=4)

# Save the predictions
np.savetxt("ALLOUCH.txt", y_pred, delimiter=" ", fmt="%s")
```