Data Science

Practical Sessions (1)

15 December 2020

Mirna El Ghosh

Purpose

- Implement a machine learning model on a regression problem using ***RidgeRegression*** and ***sklearn***.

What is Regression?

- Consists of a set of Machine Learning methods that allow us to predict a continuous outcome variable (y) based on the value of one or multiple predictor variable (x).
- **Goal**: to build a mathematical equation that defines (y) as a function of (x).
- **Example**: salaries for employees (y), age of employees (x)

  $y = ax + b$, (*a* and *b* coefficients to be defined).

  (linear regression)

Penalized Regression: Ridge and Lasso

- The **penalized regression** allowing to create a linear regression model that is penalized, for having too many variables in the model, by adding a **constraint** in the equation.
- The consequence of imposing this penalty, is to **reduce the coefficient values towards zero**.
- The **regularization** requires the selection of a tuning parameter (**lambda**) that determines the amount of regularization.
- Ridge regression is highly affected by the **scale** of the **predictors**.
- It is better to **standardize** (i.e., scale) the predictors **before applying the ridge regression**, so that all the **predictors are on the same scale**.
- Ridge regression shrinks the coefficients towards zero, but it will not set any of them exactly to zero.

Penalized Regression: Ridge and Lasso

- The lasso regression is an alternative that overcomes this drawback.
- Lasso: **reduced set of predictors**.
- Cross-validation methods can be used for identifying which of these two techniques is better on a particular data set.

Data set

- Boston houses.
- Description of houses by 13 features + prices.
- **Goal**: to predict the price of a house given its description.

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Import the dataset

1. Import the dataset from **sklearn** and check the documentation

```
from sklearn.datasets import load_boston
help(load_boston)
```

Explore the dataset

- to explore the dataset using dataframe

```
import pandas as pd
boston=load_boston()
data=pd.DataFrame(boston.data)
data.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Explore the dataset

- adding features names to the dataframe

```
data.columns = boston.feature_names
data.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Explore the dataset

- All the dataset

```
data
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

Explore the dataset

- Target values

```
boston.target
```

```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
       15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
       13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
       21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
       35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
       19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
       20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
       23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
       33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
       21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
       20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
       23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
       15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
       17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
       25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
       23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
       32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
       34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
       20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
       26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
       31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
       22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
```

Import the dataset

2. import the dataset into two variables, namely `X` and `y`.

`X` will contains the description of data, `y` the outputs associated to each data.

Check the dimensions of the two variables.

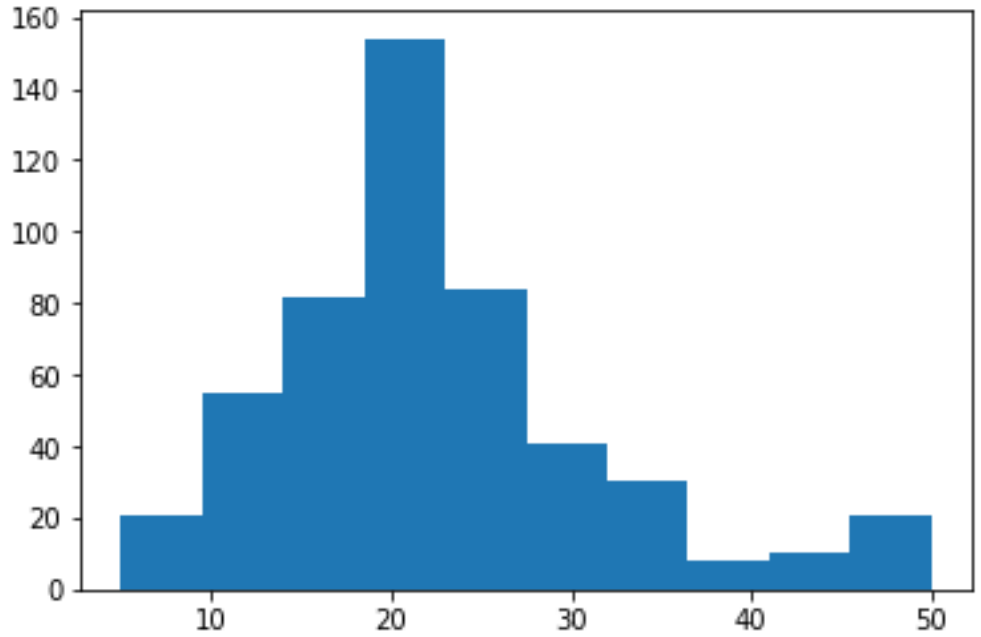
```
#load data
X, y = load_boston(return_X_y=True)
print(f"Shape of X : {X.shape}")
print(f"Shape of y : {y.shape}")
```

```
Shape of X : (506, 13)
Shape of y : (506,)
```

Data Analysis

- plot an histogram of `y` values to check its distribution. You can have a look at `***hist***` function of **matplotlib.pyplot** module

```
import matplotlib.pyplot as plt
plt.hist(y);
```



Data Analysis

- Check the 13 features included in X. Compute **mean** and standard deviation (**std**) for each feature.

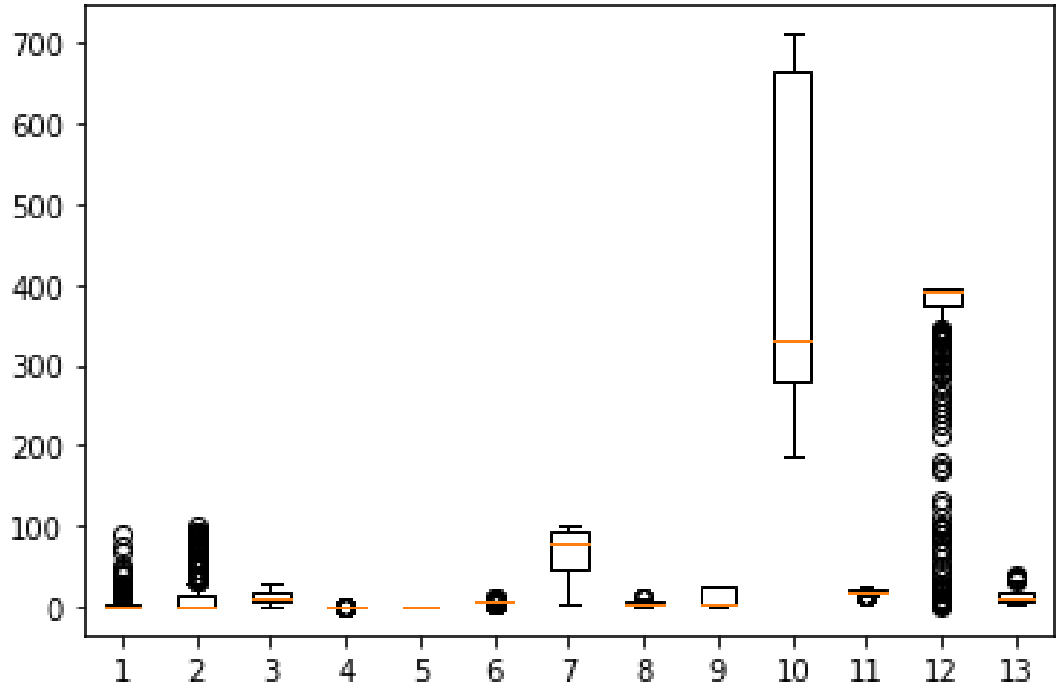
```
import numpy as np
means = np.mean(X,axis=0)
std = np.std(X,axis=0)
for m,s in zip(means,std):
    print(f"{m:.2f},{s:.2f}")
```

```
3.61,8.59
11.36,23.30
11.14,6.85
0.07,0.25
0.55,0.12
6.28,0.70
68.57,28.12
3.80,2.10
9.55,8.70
408.24,168.37
18.46,2.16
356.67,91.20
12.65,7.13
```

Data Analysis

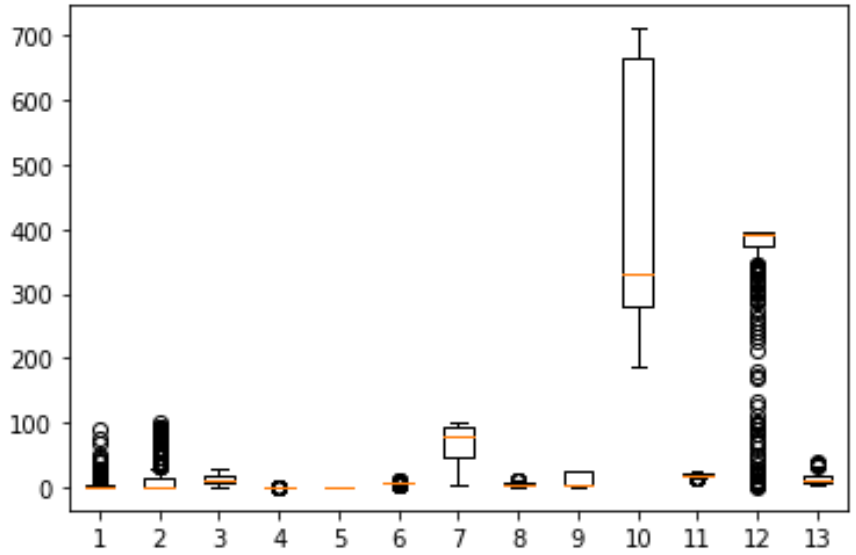
- Check this information using boxplot.

```
plt.boxplot(X);
```



Data Analysis

- What do you conclude for this first step?
- **Answer**: features are not on a relatively similar scale or close to normally distributed (mixture of scales). we need to normalize each feature.



Machine Learning

- Before starting to learn a model, we have to use a clear and as much as possible unbiased protocol to evaluate our model. This protocol starts by **splitting the data into two sets**. The **first one** will be used to **learn** the parameters of our model, the **second one** will be used to **evaluate** the performance of our learning step.

Machine Learning

- Split the data into two subsets using **train_test_split** method from **sklearn.model_selection.**

```
#Split data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)
```

Normalization: preprocess data

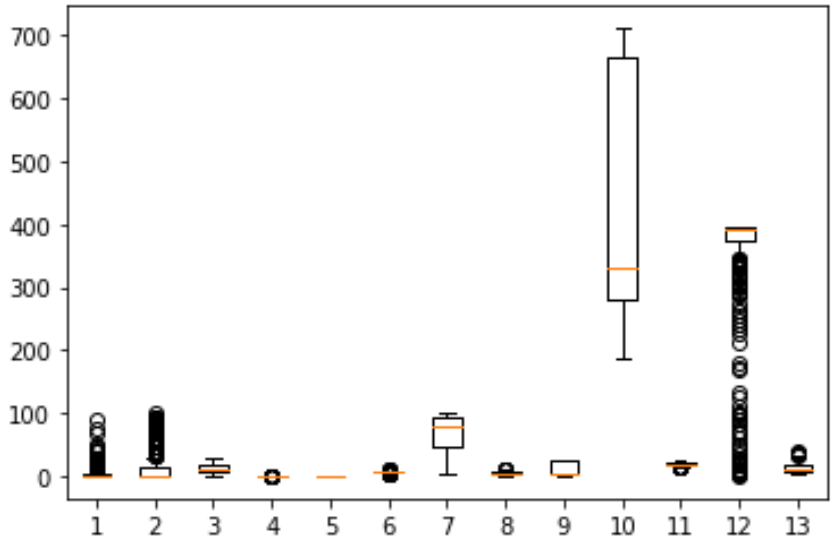
- To normalize the data according to the observation made previously, compute **X_train_norm** and **X_test_norm** corresponding normalized data to **X_train** and **X_test**.

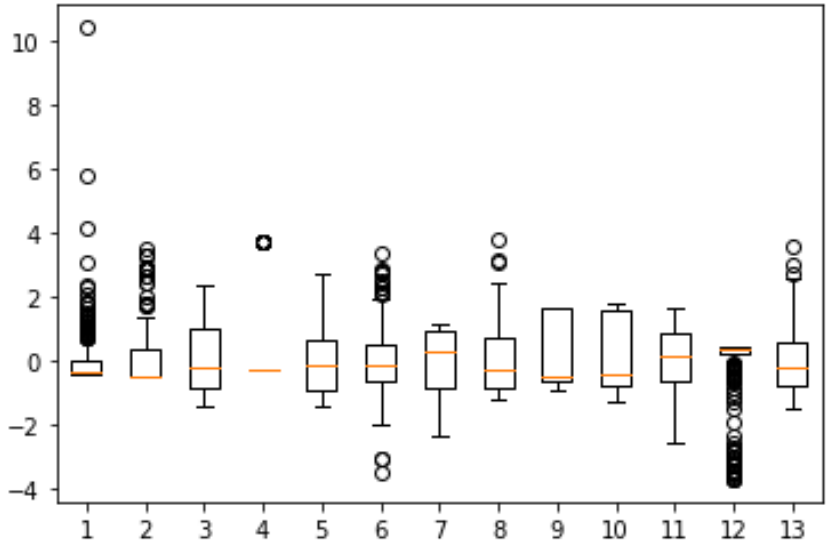
Normalization: standard scaling

- **Standardization** or **Z-score normalization:** consists of subtracting the **mean** and divide by the **standard deviation.**
- **Scale data:** change the range of the values
- **Algorithms perform better when features are on a relatively similar scale.**

```
#Scale data
#Mean and std must be evaluated on train to avoid bias
import numpy as np
mu_train = np.mean(X_train,0)
sigma_train = np.std(X_train,0)
X_train_norm = (X_train - mu_train)/sigma_train
plt.boxplot(X_train_norm);

#apply scale to X_test
X_test_norm = (X_test - mu_train)/sigma_train
```

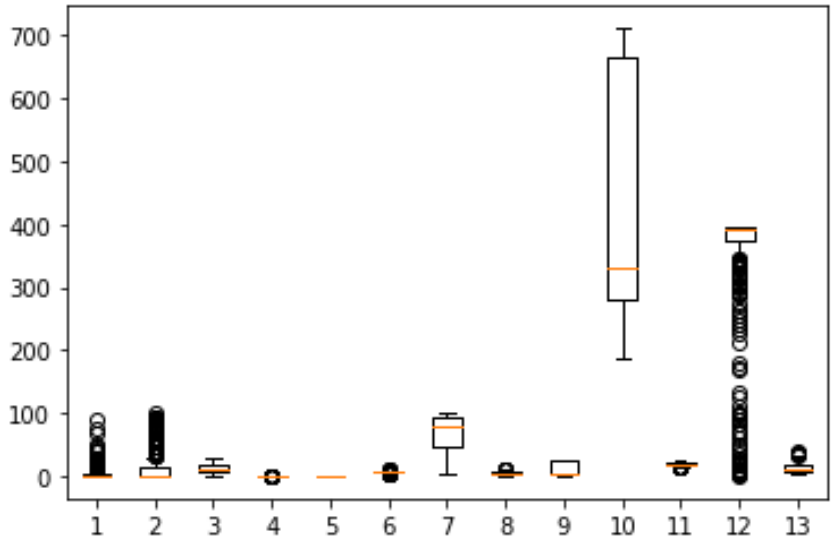


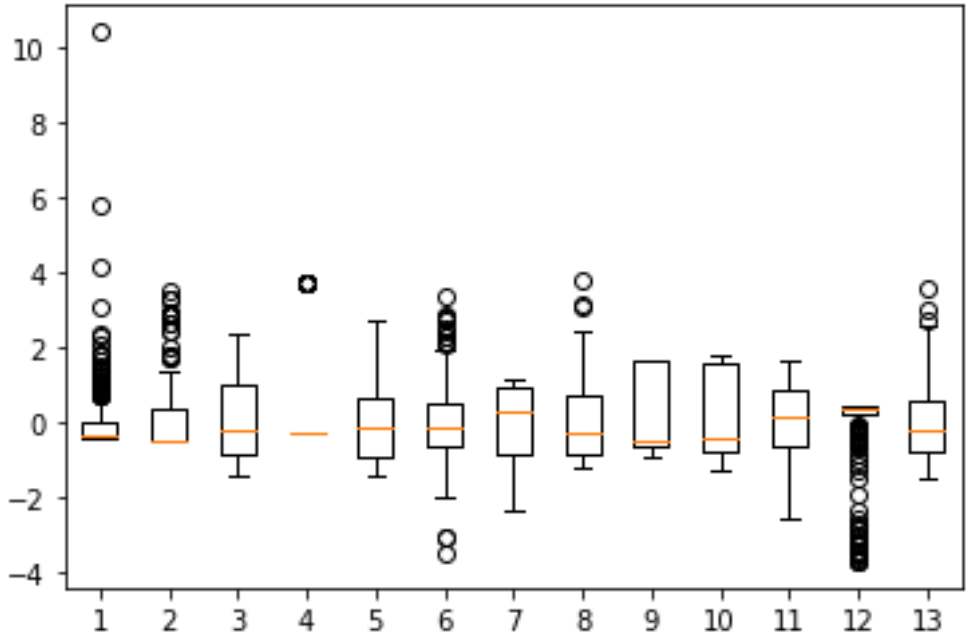


Normalization: StandardScaler

- scale the features such that the distribution is centered around 0 (mean value) and standard deviation of 1.
- **Initial fitting on the training set.**
- **Apply the transformation on the test set.**

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_norm = scaler.fit_transform(X_train)
X_test_norm = scaler.transform(X_test)
plt.boxplot(X_train_norm);
```





Data Normalized: Learn first model

- Use the **Ridge** class from **sklearn.linear_model** which implements the ridge regression scheme.
  1. Learn a first model using **Ridge** class and **X_train_norm**.
  2. What is the default λ defined by the **Ridge** class? (*the lambda term can be configured via the "alpha" argument when defining the class. The default value is 1.0)*

```
#First Ridge witout scaling and cv
from sklearn.linear_model import Ridge
import numpy as np

model = Ridge()
print(f"Default lambda value is {model.alpha}")
print(model)
```

```
Default lambda value is 1.0
Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Data Normalized: Learn first model

Train your model on X_train_norm and predict the same data. Compute the MAE (Mean Absolute Error)

```
model.fit(X_train_norm,y_train)
y_hat_train = model.predict(X_train_norm)
mae_train = np.mean([np.abs(y_train[i] - y_hat_train[i]) for i in range(0,len(y_train))])
print(f"MAE on train set : {mae_train:.2f}")
```

```
MAE on train set : 3.21
```

MAE is the sum of **absolute** differences between target and predicted variables.

Data Normalized: Learn first model

Predict on X_test_norm. Compute the MAE. What do you observe?

```
y_hat_test = model.predict(X_test_norm)
mae_test = np.mean([np.abs(y_test[i] - y_hat_test[i]) for i in range(0,len(y_test))])
print(f"MAE on test set : {mae_test:.2f}") # The error on test set is higher -> generalization
```

```
MAE on test set : 3.39
```

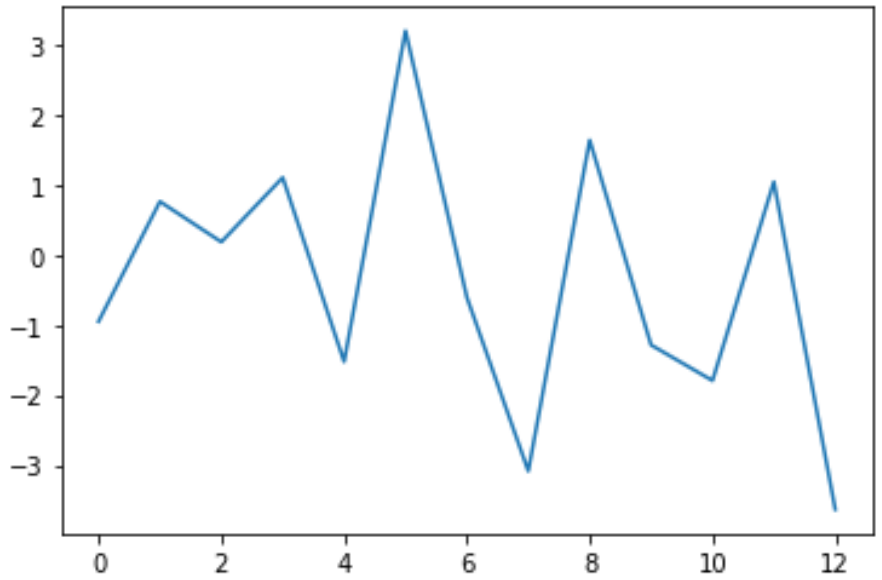
.

Model Parameters: Coefficients

Print and plot the coefficients using the instance of the Ridge class

```
print(model.coef_);
plt.plot(model.coef_);
w1 = model.coef_
```

```
[-0.94906125  0.76584764  0.18813391  1.10903048 -1.52340139  3.2066708
 -0.60482224 -3.08756672  1.64356908 -1.28523236 -1.7940949   1.04717232
 -3.63711199]
```



Model Parameters: Coefficients

Change the lambda to 1000. Retrain your model. Recompute the prediction error. What do you observe? (in sklearn λ is denoted as α)

```
#regularization effect
model = Ridge(alpha=1000)
model.fit(X_train_norm,y_train)
y_hat_train = model.predict(X_train_norm)
y_hat_test = model.predict(X_test_norm)

mae_train = np.mean([np.abs(y_train[i] - y_hat_train[i]) for i in range(0,len(y_train))])
mae_test = np.mean([np.abs(y_test[i] - y_hat_test[i]) for i in range(0,len(y_test))])
print(f"MAE on train set : {mae_train:.2f}")
print(f"MAE on test set : {mae_test:.2f}")
#Errors are much higher, and test set error is lower than train set error : We are "under"fitting
```

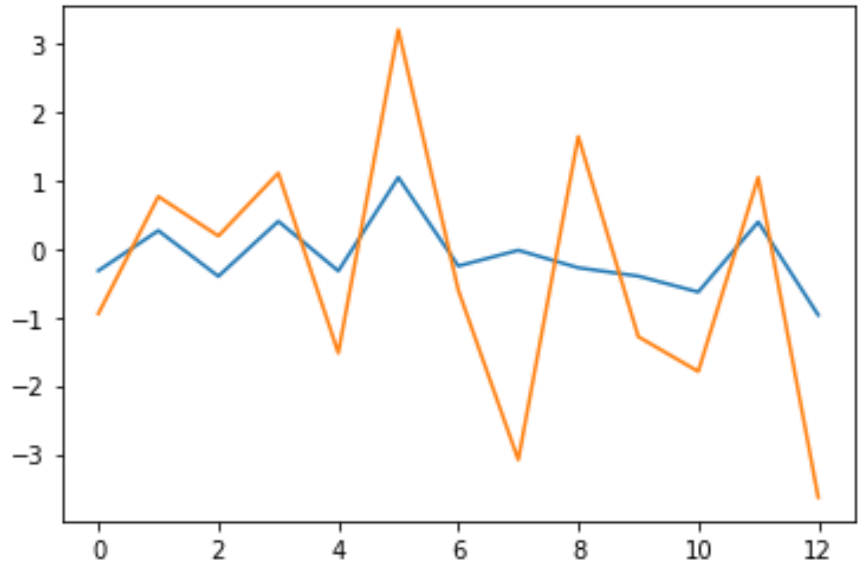
```
MAE on train set : 4.87
MAE on test set : 4.53
```

Model Parameters: Coefficients

Check the coefficients and compare them (calculating **linalg.norm**) when lambda=1. What do you conclude?

```
plt.plot(model.coef_)
plt.plot(w1)
print(f"||w|| when lambda = 1 : {np.linalg.norm(w1):.2f}")
print(f"||w|| when lambda = 1000 : {np.linalg.norm(model.coef_):.2f}")
#we observe the penality of ||w||_2^2. For lambda=1000 the norm is mush lower.
```

```
||w|| when lambda = 1 : 6.87
||w|| when lambda = 1000 : 1.87
```



Finding the best λ

λ has a strong effect on the model performance, there is a need to find the best value according to the present dataset.

1. Try different values and evaluate the performance.

Use a log scale of values between 10^{-3} to 10^{4} , i.e. `alphas = np.logspace(-3,4,10)` .

```
mae_train = []
mae_test = []
coefs = []
alphas = np.logspace(-3,4,10)
for alpha in alphas:
    model = Ridge(alpha=alpha)

    model.fit(X_train_norm,y_train)

    y_hat_train = model.predict(X_train_norm)
    y_hat_test = model.predict(X_test_norm)

    coefs.append(model.coef_)

    mae_train.append(np.mean([np.abs(y_train[i] - y_hat_train[i]) for i in range(0,len(y_train))]))
    mae_test.append(np.mean([np.abs(y_test[i] - y_hat_test[i]) for i in range(0,len(y_test))]))

plt.plot(mae_train)
plt.plot(mae_test)
plt.legend(["MAE on train", "MAE on test"]);
```

Finding the best λ

What is the best lambda λ?

we may choose the one with lowest error on test set since we are interested by the performance on unseen data. However, by doing so, we "cheat" since we use test set to tune our *hyperparameters*. This is a problem we solve by using cross validation later

Finding the best λ

What is the best lambda λ?

we may choose the one with lowest error on test set since we are interested by the performance on unseen data. However, by doing so, we "cheat" since we use test set to tune our *hyperparameters*. This is a problem we solve by using cross validation later

Plot the different values of $\mathbf{W}$ parameters for each λ values. Discuss.

```
#Regularization path
plt.plot(alphas,coefs);
plt.xscale("log")

#As we increase lambda, the norm of w vanishes and all parameters tends to a 0 value.
```