N-fold Cross-Validation

The **Leave-one-out Cross-Validation** method (or N-fold Cross-Validation) is a ressampling and data partitioning method which aims at estimating the prediction error of a model.

In the context of regressions models, the prediction error $\hat{\theta}_{pe}$ is defined as follows:

$$\hat{\theta}_{pe} = E[e_i^2] = E\left[(y_i - \hat{y}_i)^2\right] = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Then the **Root Mean Squared Error** (RMSE) is simply

$$RMSE = \sqrt{\hat{\theta}_{pe}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$$

N-fold Cross-Validation algorithm

To implement the leave-one-out cross-validation method for regression models, the general procedure is the following:

For i in $1, ..., n$

1. Compute $\hat{y}_i^{(-i)}$ the predictions from the model fitted on all observations except for the i^{th} observation.

2. Compute the errors $e_i = y_i - \hat{y}_i^{(-i)}$

3. Compute the RMSE as $\sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$

Working example

Example: We generate $n = 300$ artificial data points (x_i, y_i) from the following process:

$$x_i \sim N(5, 3)$$
$$y_i = x_i^2 + \epsilon_i \qquad \text{with } \epsilon_i \sim N(0, 8)$$

Suppose that we wish to fit the two following models:

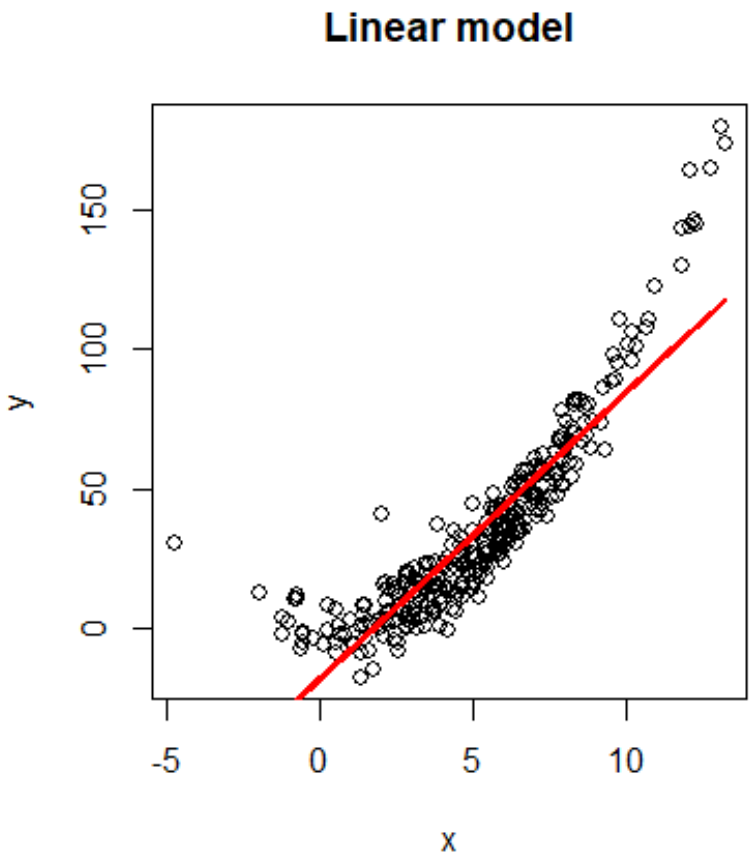
$$\text{Model 1:} \qquad y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$
$$\text{Model 2:} \qquad y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$$

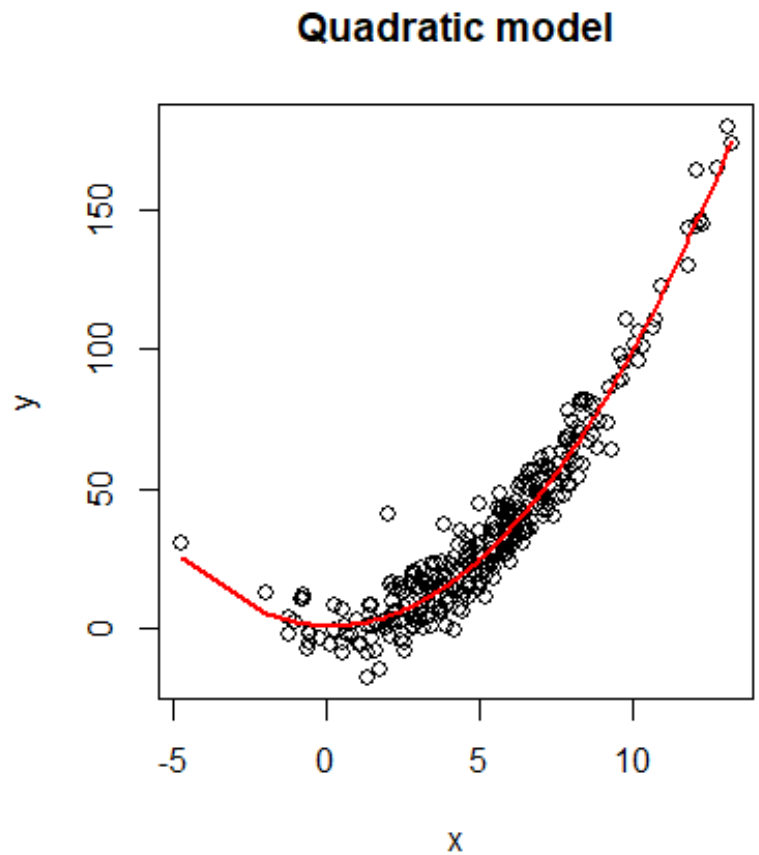
We will use the Leave-one-out Cross-Validation method in R and Python to select the best model, that is the model with the lowest RMSE.

Code for scatterplots and fit in R

```r
# 1. generate artificial data
set.seed(2023)
n <- 300
x <- rnorm(n = n, mean = 5, sd = 3)
y <- x^2 + rnorm(n = n, mean = 0, sd = 8)
data = data.frame(x = x, y = y)

# 2. plot the models that we want to fit
par(mfrow = c(1,2), pty = "s")

model1 <- lm(y ~ x)
plot(x = x, y = y, main = 'Linear model', cex = 1.1, pch = 1, lwd = 1.2)
yhat1 <- model1$coef[1] + model1$coef[2] * x
lines(x, yhat1, lw = 2.5, col = 'red')

model2 <- lm(y ~ x + I(x^2))
plot(x = x, y = y, main = 'Quadratic model', cex = 1.1, pch = 1, lwd = 1.2)
yhat2 <- model2$coef[1] + model2$coef[2] * x  + model2$coef[3] * x^2
lines(x[order(x)], yhat2[order(x)], lw = 2.5, col = 'red')
```

Plots in R

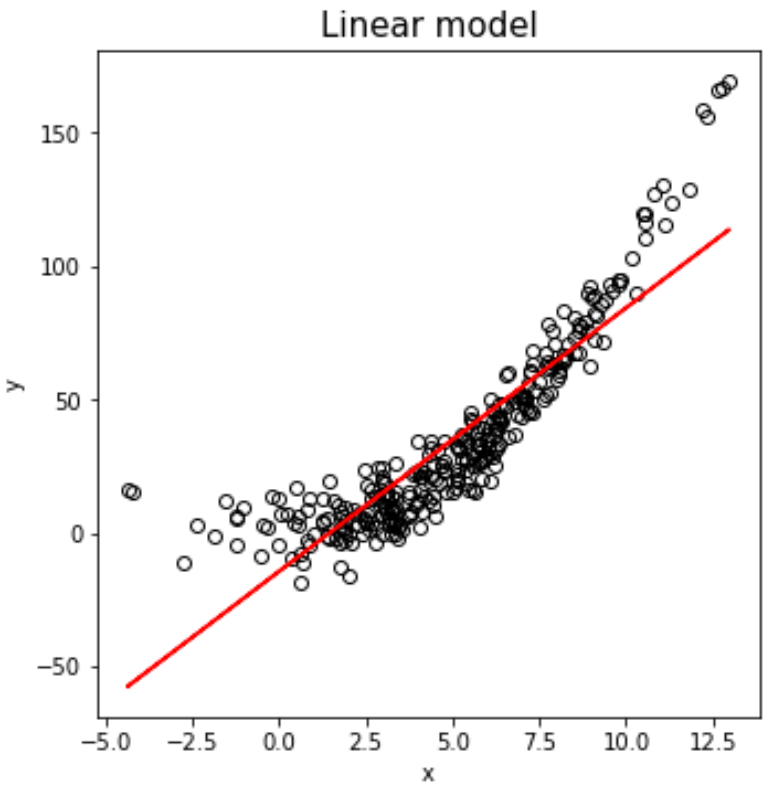

Linear model
y
150
100
50
0
-5
0
5
10
x

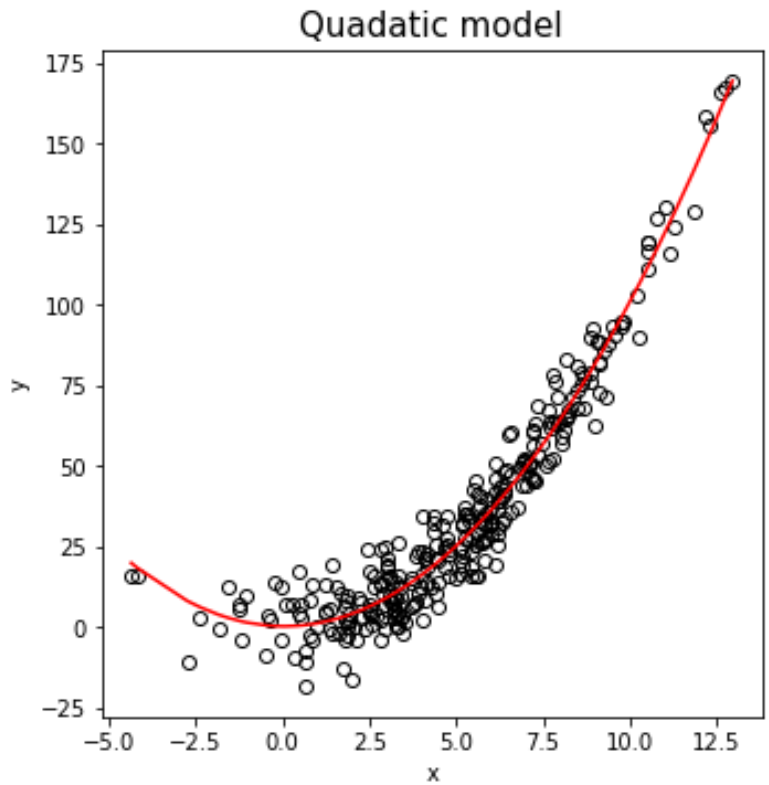

Quadratic model
y
150
100
50
0
-5
0
5
10
x

Code for scatterplots and fit in Python

```
# 1. generate artificial data
import numpy as np
np.random.seed(2023) ; n = 300
x = np.random.normal(loc = 5, scale = 3, size = n)
y = x**2 + np.random.normal(loc = 0, scale = 8, size = n)

# 2. plot the models that we want to fit
import matplotlib.pyplot as plt

plt.subplot(1, 2, 1)
plt.plot(x, y, 'o', fillstyle = 'none', color = 'black')
plt.title('Linear model', fontsize = 15)
plt.xlabel("x") ; plt.ylabel("y")
beta1, beta0 = np.polyfit(x, y, 1) ; yhat1 = beta0 + beta1*x
plt.plot(x, yhat1, color = 'red')

plt.subplot(1, 2, 2)
plt.plot(x, y, 'o', fillstyle = 'none', color = 'black')
plt.xlabel('x') ; plt.ylabel('y') ; plt.title('Quadatic model', fontsize = 15)
beta2, beta1, beta0 = np.polyfit(x, y, 2) ; yhat2 = beta0 + beta1*x + beta2*(x
    **2)
orders = np.argsort(x.ravel())
plt.plot(x[orders], yhat2[orders], color = 'red')
```

Plots in Python





N-fold Cross-Validation in R

```r
# 3. Cross-Validation
# fit the models on leave-one-out samples

pred.cv.mod1 <- pred.cv.mod2 <- numeric(n)

for(i in 1:n) {

  # quadratic model
  mod1 = lm(y ~ x, subset = -i)
  pred.cv.mod1[i] = predict(mod1, data[i,])

  # quadratic model
  mod2 = lm(y ~ x + I(x^2), subset = -i)
  pred.cv.mod2[i] = predict(mod2, data[i,])

}
```

N-fold Cross-Validation in Python

```
# 3. Cross-Validation
# fit the models on leave-one-out samples

import pandas as pd

data = pd.DataFrame({'x': x, 'y': y})
xn = data['x'].values.reshape(-1,1); yn = data['y'].values.reshape(-1,1)

from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import LeaveOneOut, cross_val_score

loocv = LeaveOneOut()

# linear model
mod1 = PolynomialFeatures(degree = 1, include_bias = False).fit_transform(xn)
mod11 = LinearRegression().fit(mod1, yn)
scoresmod1 = cross_val_score(mod11, mod1, yn,
                             scoring = 'neg_mean_squared_error',
                             cv = loocv)
# quadratic model
mod2 = PolynomialFeatures(degree = 2, include_bias = False).fit_transform(xn)
mod22 = LinearRegression().fit(mod2, yn)
scoresmod2 = cross_val_score(mod22,  mod2, yn,
                             scoring = 'neg_mean_squared_error',
                             cv = loocv)
```

Conclusion from R and Python outputs

In R:

```r
MSE1 = (1/n) * sum((y - pred.cv.mod1)^2) # theta_hat_pe
MSE2 = (1/n) * sum((y - pred.cv.mod2)^2) # theta_hat_pe

# Root Mean Squared Error (RMSE)
sqrt(c(MSE1, MSE2))
# [1] 15.68599  7.99332
```

In Python

```python
# Root Mean Squared Error (RMSE)
import statistics
import math

RMSE1 = math.sqrt(statistics.mean(abs(scoresmod1)))
RMSE2 = math.sqrt(statistics.mean(abs(scoresmod2)))
[RMSE1, RMSE2]
# [16.169293289892607, 7.873829105930071]
```

The second model (Quadratic) has the lowest RMSE and thus is prefered (to the Linear) in both cases. It was expected since the fit as described by the red line or curve on the graphs decribe the generated data more accurately.

Further reading and code

Rizzo, M.L. (2019). Statistical Computing with R, Second Edition (2nd ed.). Chapman and Hall/CRC.
https://doi.org/10.1201/9780429192760

The R Project for Statistical Computing:
https://www.r-project.org/

Python:
https://www.python.org/

Accessing R and Python code:
https://github.com/JRigh/LOOCV-Cross-validation-for-regression