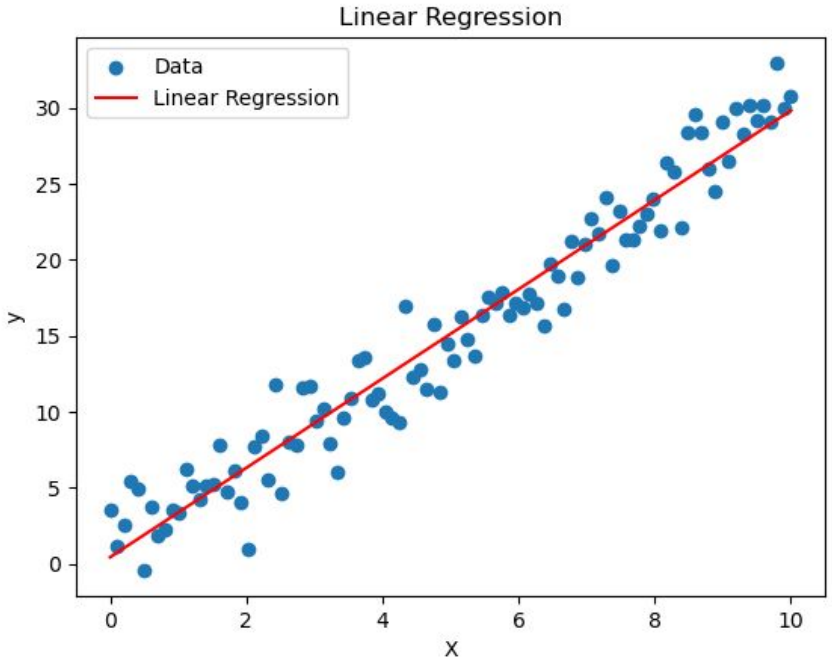
Multivariate regression

Jolla Kullgren
Department of Chemistry - Ångström

Multivariate linear regression



- Linear regression refer to the process of finding a linear relation between predictor variables ($x_{j,i}$) and response variables, y_i. For one set predictor variables we can write:

$$\text{minimize} \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_i))^2$$

- By differentiating the sum of squared residuals with respect to each coefficient and set the derivatives equal to zero we obtain the normal equations. Which, in the general case, can be written as:

$$\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{y}$$

where **X** collect the predictor variables and **y** the response variables.

A simple python example using statsmodels (input):

```
import statsmodels.api as sm
import numpy as np
import itertools as iter

# Generate a set of predictor variables
span = np.linspace(0, 10, 100)
x=list( iter.combinations_with_replacement(span,2))
x=np.array(x)

# Generate the corresponding responce variables
y = 3 * x[:,0] + np.random.randn(len(x[:,1])) * 5 + 2*x[:,1]

# Perform least square fit
est = sm.OLS(y,x).fit()
print(est.summary())
```

```
==============================================================================
=
```

The output is shown on the next slide.

A simple python example using statsmodels (output):

OLS Regression Results

Dep. Variable:	y	R-squared (uncentered):	0.964
Model:	OLS	Adj. R-squared (uncentered):	0.964
Method:	Least Squares	F-statistic:	6.714e+04
Date:	Thu, 15 Jun 2023	Prob (F-statistic):	0.00
Time:	11:06:11	Log-Likelihood:	-15233.
No. Observations:	5050	AIC:	3.047e+04
Df Residuals:	5048	BIC:	3.048e+04
Df Model:	2		
Covariance Type:	nonrobust		

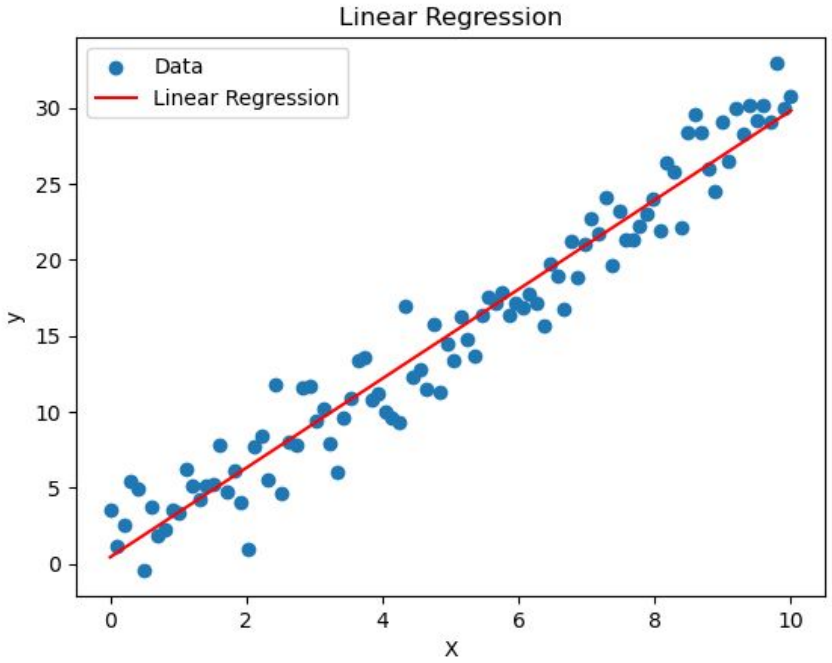
	coef	std err	t	P>|t|	[0.025	0.975]
x1	2.9933	0.034	89.186	0.000	2.927	3.059
x2	1.9928	0.019	102.502	0.000	1.955	2.031

Omnibus:	0.346	Durbin-Watson:	2.004
Prob(Omnibus):	0.841	Jarque-Bera (JB):	0.348
Skew:	0.020	Prob(JB):	0.840
Kurtosis:	2.996	Cond. No.	4.33

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

We get plenty printed out! We only consider what is circled in red now and return to some of the other parts later.

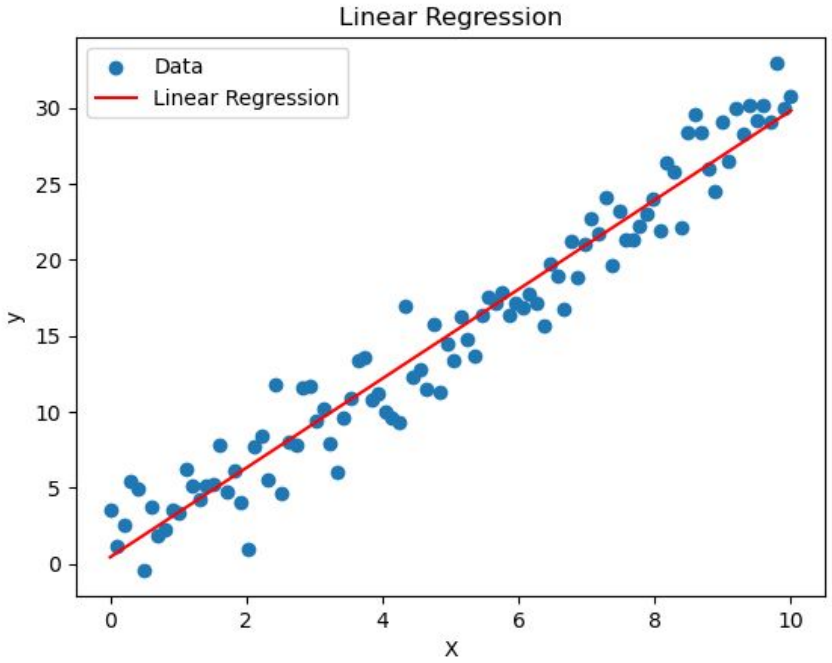
Goodness-of-fit



There are multiple measures for the goodness-of-fit. The most common are perhaps:

- R-squared (coefficient of determination)
- Adjusted R-squared
- F-test

Goodness-of-fit (R-squared)



R-squared (coefficient of determination) represents the proportion of the variance in the dependent variable (response variable) that can be explained by the independent variables (predictor variables) in a regression model.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

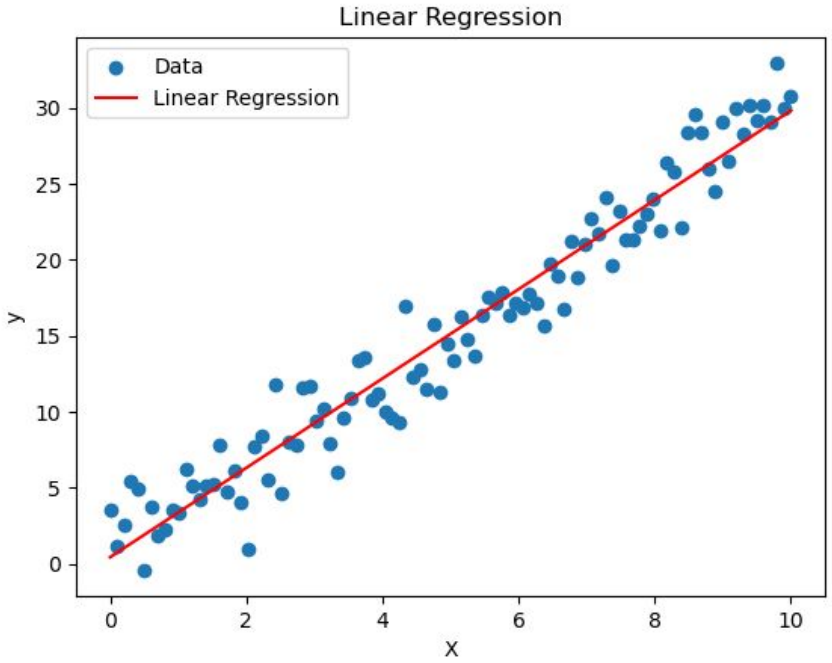
y_i : observed y-value

$\hat{y}_i$: predicted y-value

$\bar{y}$: mean of y-values

UPPSALA UNIVERSITET

Goodness-of-fit (adjusted R-squared)

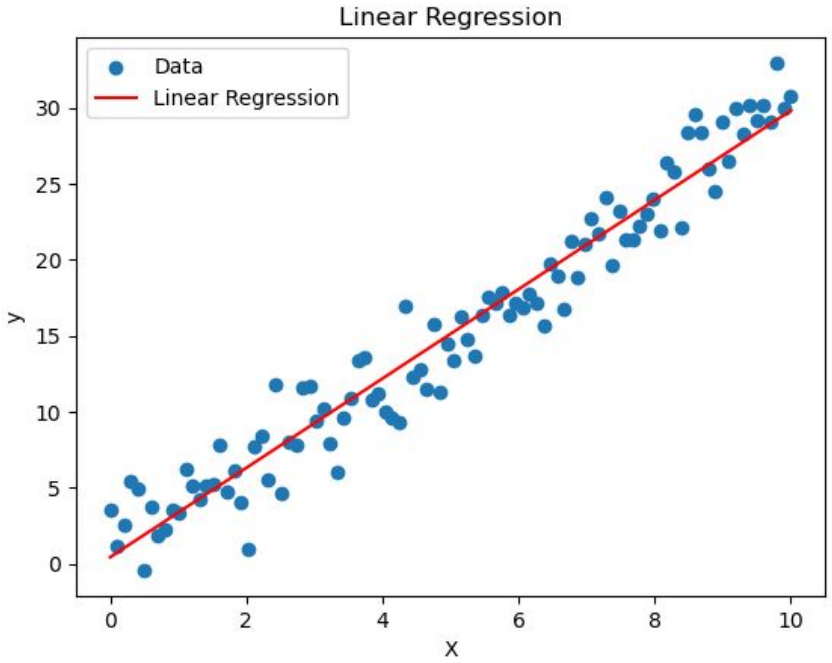


Adjusted R-squared is a modified version of R-squared that takes into account the number of predictor variables in the regression model. It provides a more reliable measure of the model's goodness-of-fit, especially when comparing models with different numbers of predictors.

$$\text{Adjusted } R^2 = 1 - \left(\frac{n-1}{n-k-1}\right)\left(1 - R^2\right)$$

In the formula, *n* is the number of observations and *k* is the number of predictor variables.

Goodness-of-fit (F-test)

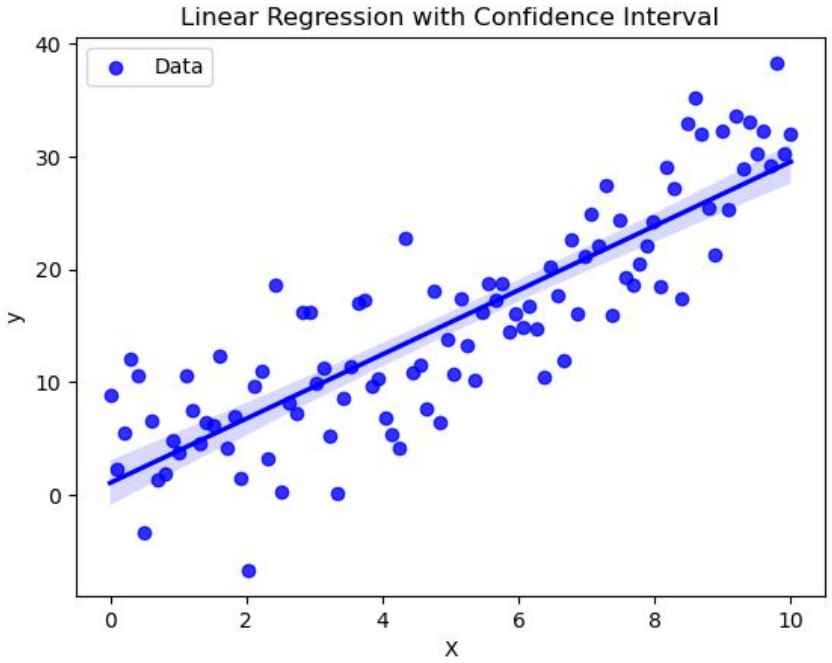


We ca use F-test statistical test to assess the overall significance of the regression model. It determines whether the relationship between the independent variables and the dependent variable, as captured by the model, is statistically significant.

$$F = \frac{\left(\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2\right)/k}{\left(\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right)/(n-k-1)}$$

The resulting F-value is compared against the critical F-value from the F-distribution with degrees of freedom (p, n- k-1)

Standard error in the regression coefficients



- The standard error for a regression coefficient is given by:

$$SE(\hat{\beta}_1) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{(n-2)\sum_{i=1}^{n}(x_i - \bar{x})^2}}$$

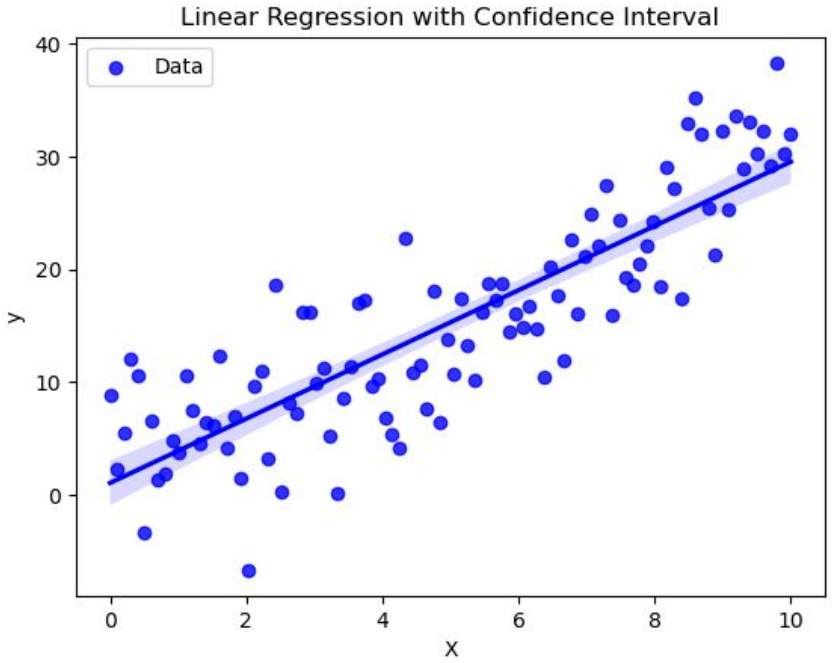
y_i : observed y-value

$\hat{y}_i$: predicted y-value

x_i : observed x-value

$\bar{x}$: mean of x-values

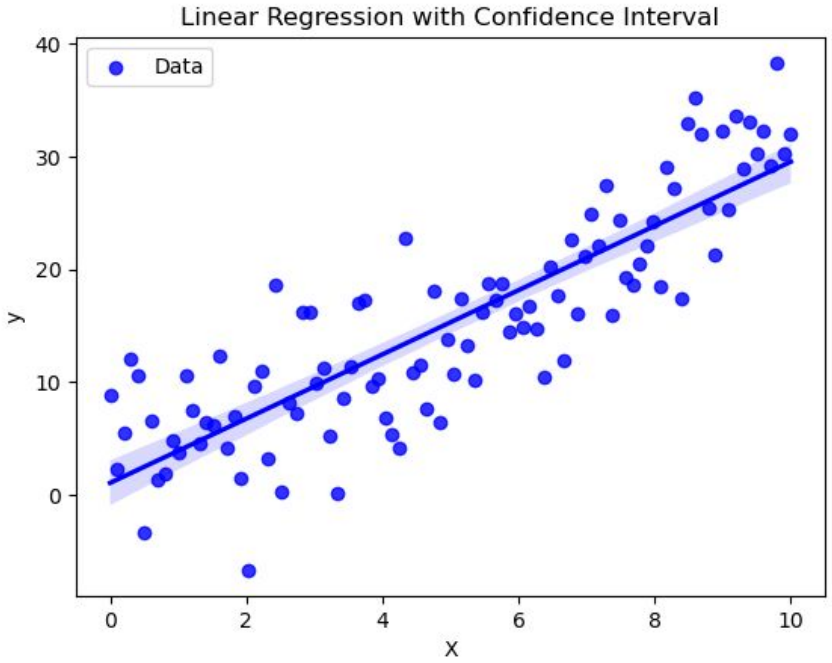
Significance in the regression coefficients



- The significance of the coefficient can tested with a t-test:

$$t^{test} = \frac{\hat{\beta}_1}{SE(\hat{\beta}_1)}$$

Reducing the model (t-test way)



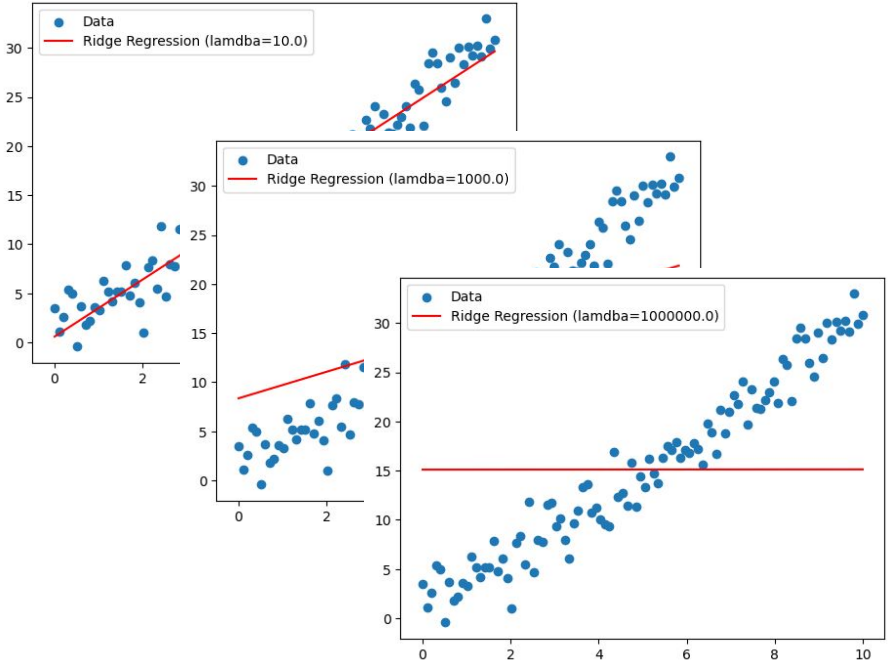
1. Perform regression using all (remaining) predictor variables.
2. Compute the significance of the regression coefficients.
3. Remove insignificant coefficients.
4. Repeat steps 1-3 until all coefficients are significant.

A simple python example using statsmodels (output):

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared (uncentered):                   0.964
Model:                            OLS   Adj. R-squared (uncentered):              0.964
Method:                 Least Squares   F-statistic:                          6.714e+04
Date:                Thu, 15 Jun 2023   Prob (F-statistic):                        0.00
Time:                        11:06:11   Log-Likelihood:                         -15233.
No. Observations:                5050   AIC:                                  3.047e+04
Df Residuals:                    5048   BIC:                                  3.048e+04
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             2.9933      0.034     89.186      0.000       2.927       3.059
x2             1.9928      0.019    102.502      0.000       1.955       2.031
==============================================================================
Omnibus:                        0.346   Durbin-Watson:                   2.004
Prob(Omnibus):                  0.841   Jarque-Bera (JB):                0.348
Skew:                           0.020   Prob(JB):                        0.840
Kurtosis:                       2.996   Cond. No.                         4.33
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Ridge regression



- An alternative way of reducing model complexity is to use so-called ridge regression (RR).
- We add a penalty term to our objective which is proportional to the square sum of the coefficients.

$$\text{Minimize:} \quad \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}\beta_j^2$$

- This way we damp-out small coefficients.

A simple python example using sklearn:

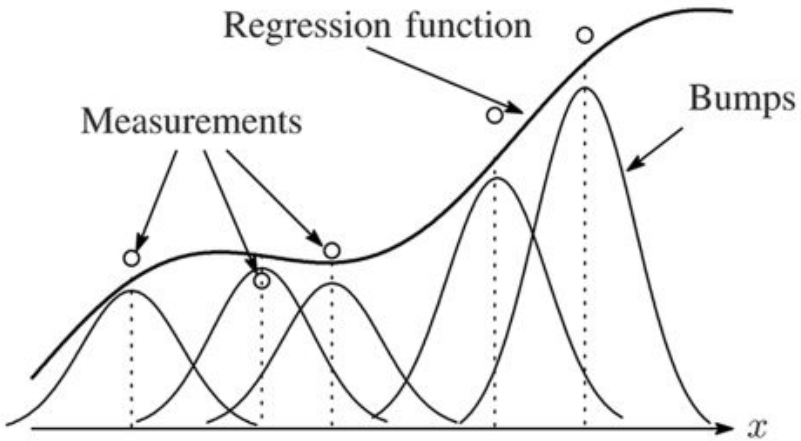
```
import numpy as np
from sklearn.linear_model import Ridge

# Generate a set of predictor variables
span = np.linspace(0, 10, 100)
x=list( iter.combinations_with_replacement(span,2))
x=np.array(x)

# Generate the corresponding responce variables
y = 3 * x[:,0] + np.random.randn(len(x[:,1])) * 5 + 2*x[:,1]

# Ridge regression
lamb=1.0
ridge_reg = Ridge(alpha=lamb)  # You can adjust the regularization parameter
(alpha) as needed
ridge_reg.fit(x, y)
ridge_reg_pred = ridge_reg.predict(x)
```

Kernel Ridge Regression (KRR)

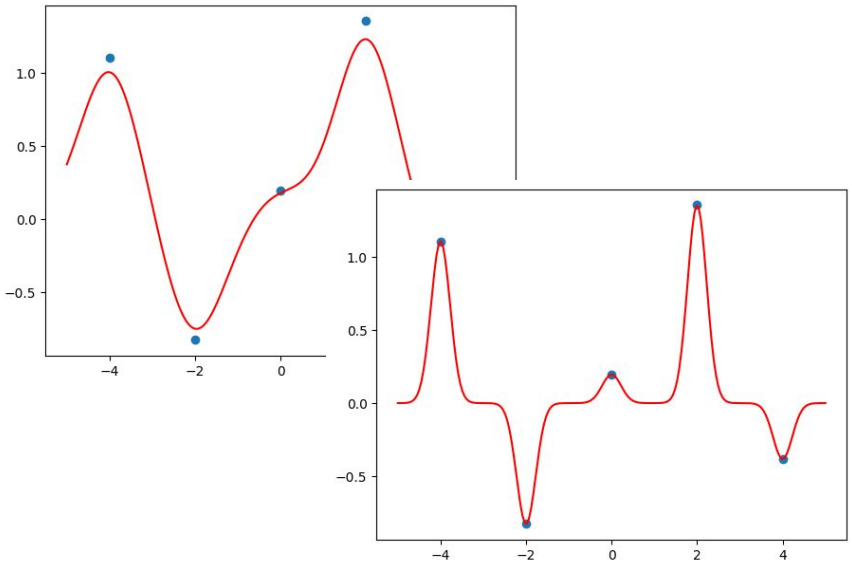


(a) Kernel regression

IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 16, NO. 2, FEBRUARY 2007

- In kernel ridge regression we produce a model directly from our measurements.
- The model return y-values for any input value **x** by comparing the already known measurements.
- Comparison is made using a similarity kernel. The kernel can be seen as a generalized way of measure the distance between all existing data-points and our input value **x**.

Kernel Ridge Regression (KRR)

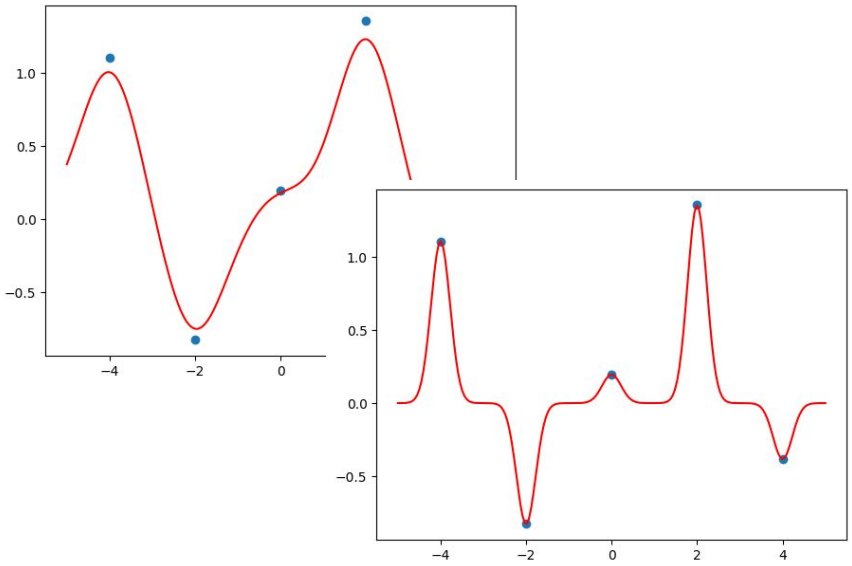


Two examples of KRR models. Top: A farsighted model. Bottom: A nearsighted model.

- In kernel ridge regression we produce a model directly from our measurements.
- The model return y-values for any input value **x** by comparing the already known measurements.
- Comparison is made using a similarity kernel. The kernel can be seen as a generalized way of measure the distance between all existing data-points and our input value **x**.
- The models can be more or less nearsighted as illustrated in figures to the left. The first is farsighted and the second nearsighted.

UPPSALA UNIVERSITET

Kernel Ridge Regression (KRR)

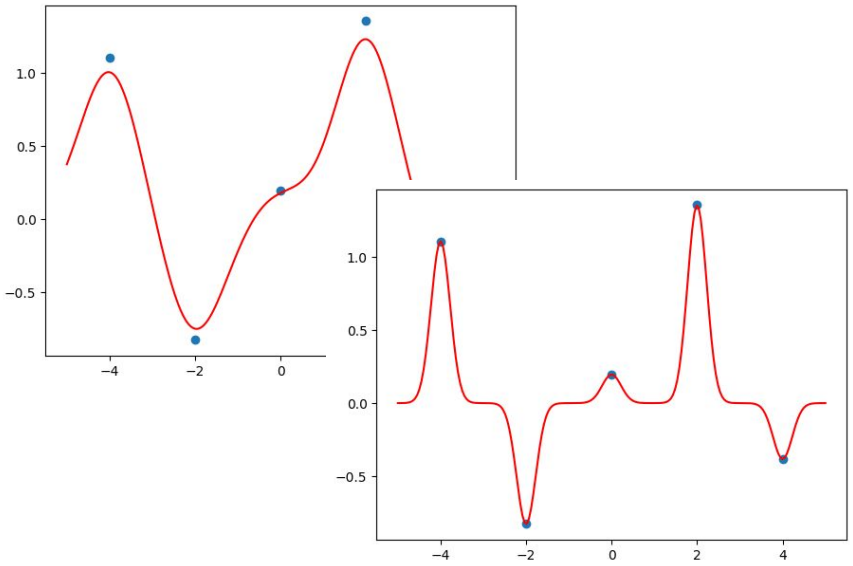


Two examples of KRR models. Top: A farsighted model. Bottom: A nearsighted model.

- In kernel ridge regression we produce a model directly from our measurements.
- The model return y-values for any input value **x** by comparing the already known measurements.
- Comparison is made using a similarity kernel. The kernel can be seen as a generalized way of measure the distance between all existing data-points and our input value **x**.
- The models can be more or less nearsighted as illustrated in figures to the left. The first is farsighted and the second nearsighted.

UPPSALA UNIVERSITET

Kernel Ridge Regression (KRR)



Two examples of KRR models. Top: A farsighted model. Bottom: A nearsighted model.

- The mathematical machinery is as follows: "Kernel", e.g:

$$K_{ij} = k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

"Ridge Regression Coefficients"

$$\alpha = (K + \lambda I)^{-1} y$$

- The actual curve (predictions) is generated from:

$$\hat{y}_{\text{new}} = \sum_{i=1}^{N} \alpha_i k(x_{\text{new}}, x_i)$$

UPPSALA UNIVERSITET

A simple python example using sklearn:

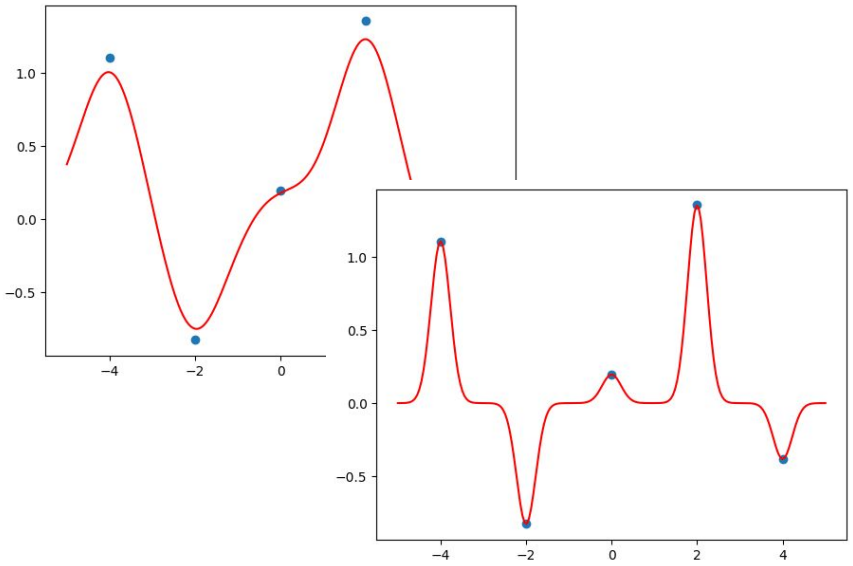
```
import numpy as np
from sklearn.kernel_ridge import KernelRidge


# Generate a set of predictor variables
span = np.linspace(0, 10, 100)
x=list( iter.combinations_with_replacement(span,2))
x=np.array(x)

# Generate the corresponding responce variables
y = 3 * x[:,0] + np.random.randn(len(x[:,1])) * 5 + 2*x[:,1]

# Kernel ridge regression with RBF
gamma=1.0
kernel_ridge_reg = KernelRidge(alpha=0.1, kernel='rbf', gamma=gamma)
kernel_ridge_reg.fit(X, y)
kernel_ridge_reg_pred = kernel_ridge_reg.predict(X)
```

Kernel Ridge Regression (KRR) - standardizing



Two examples of KRR models. Top: A farsighted model. Bottom: A nearsighted model.

- In multivariate KRR, if the ranges for the different independent variables are very different, our “distance measure” becomes skewed.
- One way to remedy this problem is to standardize the x-values.
  - **The procedure**: Center to the mean and component wise scale to unit variance.

A simple python example using sklearn:

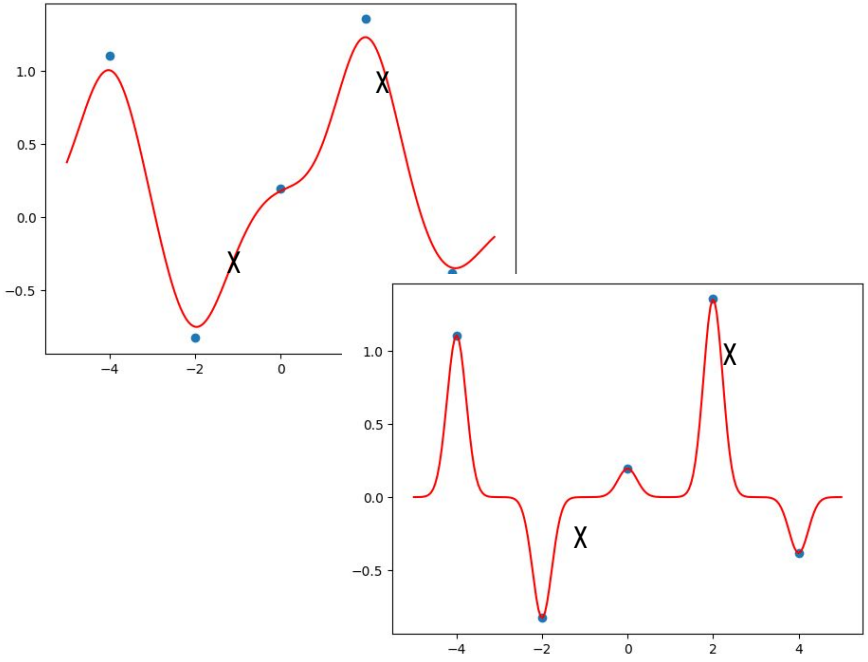
```
import numpy as np
from sklearn.kernel_ridge import KernelRidge
from sklearn.preprocessing import scale

# Generate a set of predictor variables
span = np.linspace(0, 10, 100)
x=list( iter.combinations_with_replacement(span,2))
x=np.array(x)
standardised_x = scale(x)

# Generate the corresponding responce variables
y = 3 * x[:,0] + np.random.randn(len(x[:,1])) * 5 + 2*x[:,1]

# Kernel ridge regression with RBF
gamma=1.0
kernel_ridge_reg = KernelRidge(alpha=0.1, kernel='rbf', gamma=gamma)
kernel_ridge_reg.fit(standardised_x, y)
kernel_ridge_reg_pred = kernel_ridge_reg.predict(standardised_x)
```

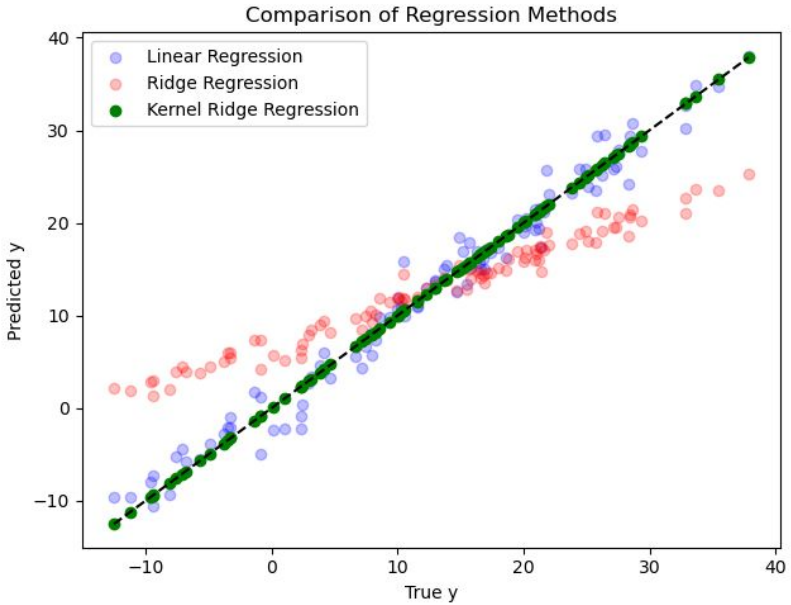
Training and validation



- When fitting somewhat complex models it is often useful to perform some sort of validation.
- A popular way of doing such validation is to split the data-set 80/20 into a training-set and validation-set.
  - Fitting is done on 80% of the data
  - Validation is done on the remaining 20% of the data
- In the figure to the left, “x” is used to indicate validation-points. In this example, the first model is better at describing the points in the validation-set and would therefore be the preferred model.

UPPSALA UNIVERSITET

Training and validation



- A informative way of illustrating the quality of the fitted model is to make correlation plot.
- In the correlation plot we typically use the y-value from the validation-set on the “x-axis” and the predicted y-value from our fitted model on the “y-axis”.
- A “perfect” model would have all points on the line y=x.

Now, let’s play!