DATA11002 Introduction to Machine Learning

Kai Puolamäki

4 November 2020

Announcements

- *Exercise Set 1 (E1)* submission DL will be on **8 November**
    - please use, e.g., R or Jupyter notebook for Exercise Set 1-3 answers! (you can easily combine include text, mathematical formulas, code, and plots with them)
    - Please take this course on R Markdown, if necessary!
- *Term project groups* of 1-3 students will be formed after **9 November**
    - you can **form groups of 1-3 students** on your own by **8 November** (you can now submit your groups in Moodle!)
    - if you don't form a group on your own you will be assigned to groups of (mostly) 3 students in random after 9 November
    - you can be assigned to a term project group **only after passing E0**

Contents of lectures 3-4

- Reminder about generalization error and supervised learning
- Linear regression
- Estimating model parameters
- Estimating loss: evaluation of model performance
- Controlling the model complexity

Reminder about generalization error and supervised learning

Summary of the setting

- *Task* is what an end user actually wants to do.
- *Computational problem* is a (hopefully general) mathematical definition of the task
    - usually includes some *performance measure* (to be optimised somehow)
- *Model* is a (hopefully good) solution to the computational problem.
- *Data* consists of objects in the domain the user is interested in, with perhaps some additional information attached.
- *Machine learning algorithm* produces a model based on data.
- *Features* are how we represent the objects in the domain.

What we learned about generalisation

- *Underfitting*: model too simple or too much data (poor generalization)
- *Overfitting*: model too complex or too little data (poor generalization)
- *Inductive bias*: choice of hypothesis space incorporates assumptions (generalization is impossible without any assumptions)
- Lets now look at *curve fitting* (*regression*):

Regression

Regression is like classification, except that y is not a discrete variable but a real number.

Example: ordinary least squares (OLS) linear regression

x and y are both real numbers.

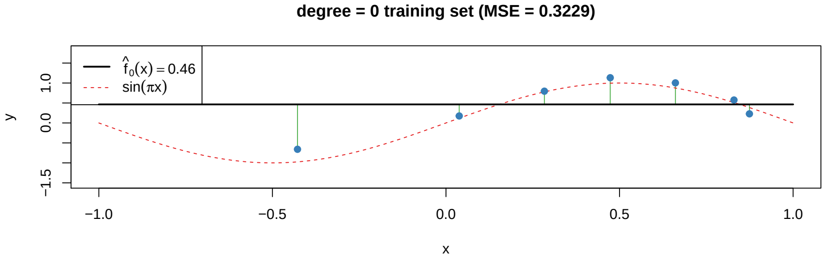
$$D = \{(x_i, y_i)\}_{i=1}^{n}$$

$$\hat{f}_k(X) = \sum_{p=0}^{k} w_{kp} x^p$$

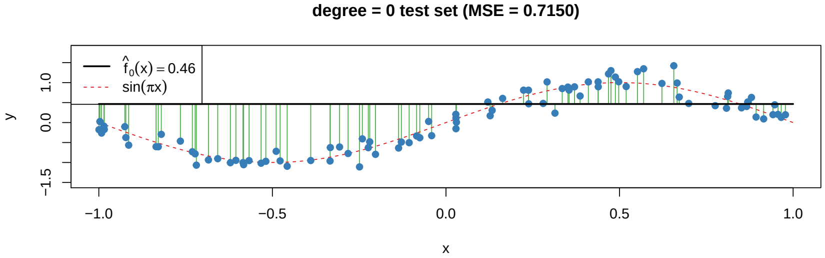
$$\mathrm{MSE} = \sum_{t=1}^{n} \left(y_i - \hat{f}_k(x_i)\right)^2 / n$$

(MSE = mean squared error)

Polynomial degree 0

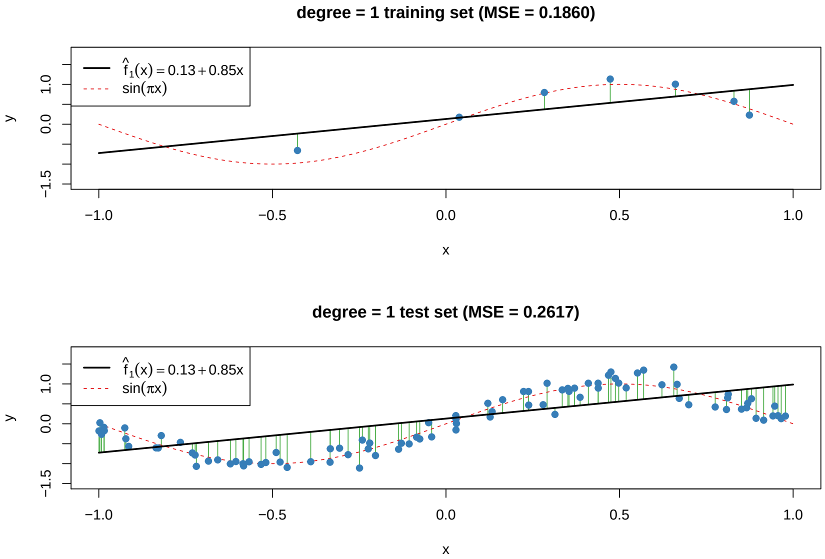


degree = 0 training set (MSE = 0.3229)

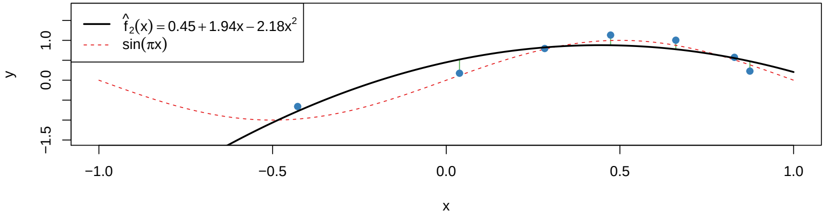


degree = 0 test set (MSE = 0.7150)

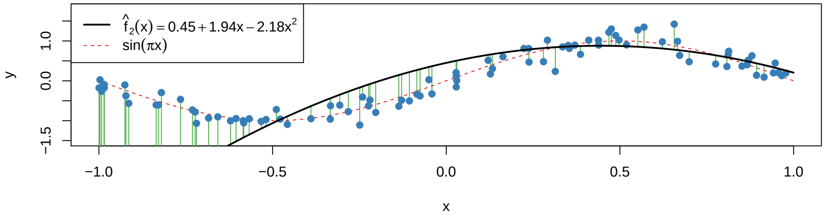
Polynomial degree 1



Polynomial degree 2

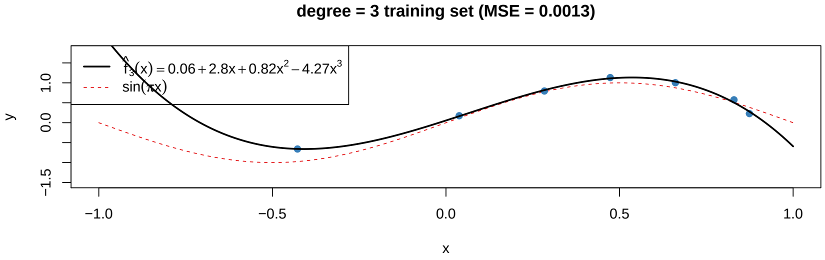


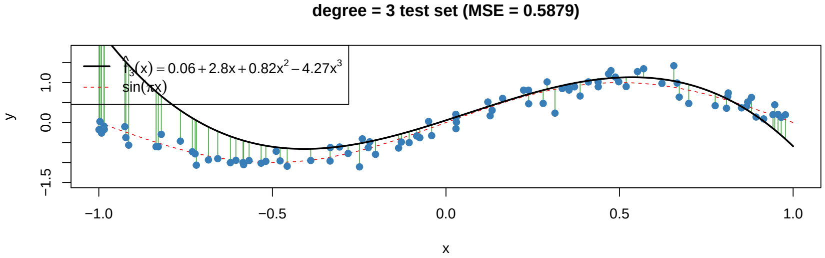
degree = 2 training set (MSE = 0.0446)



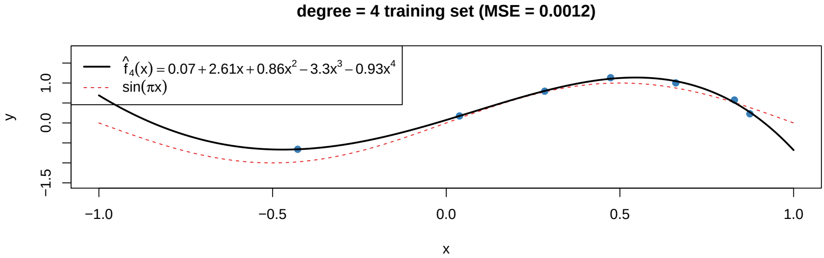
degree = 2 test set (MSE = 1.2808)

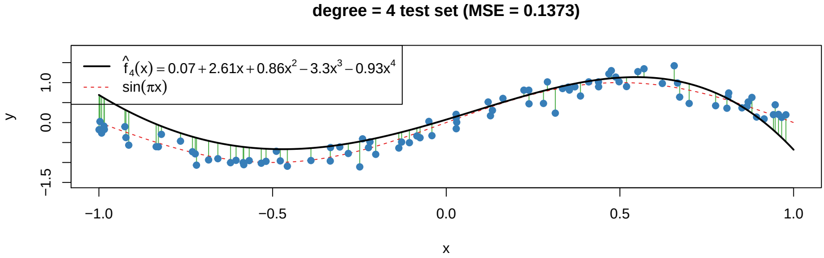
Polynomial degree 3



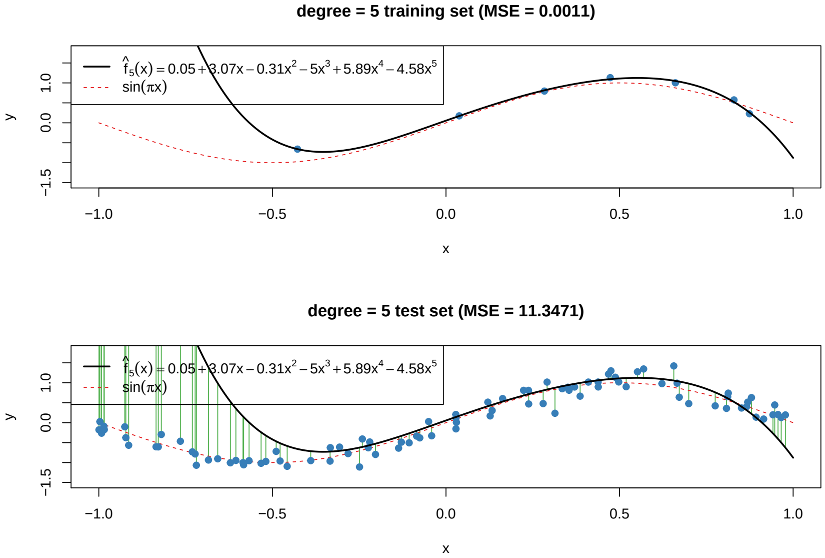


Polynomial degree 4



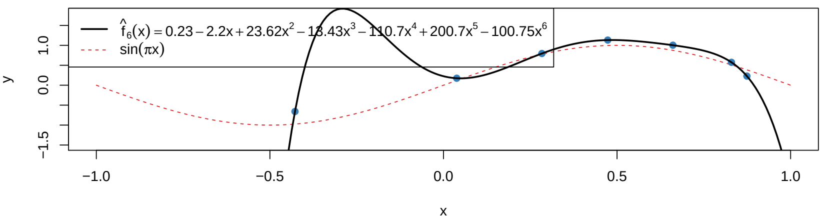


Polynomial degree 5

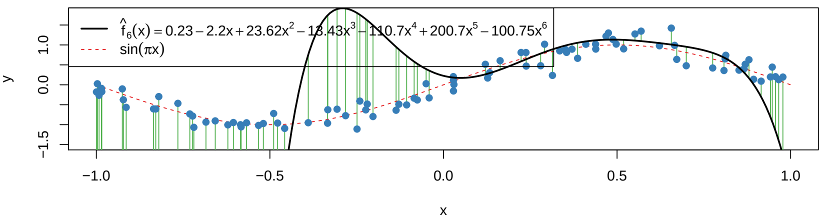


Polynomial degree 6

degree = 6 training set (MSE = 0.0000)



degree = 6 test set (MSE = 9586.9679)



Summary of performance

degree	MSE (train)	MSE (test)
0	0.3229	0.7150
1	0.1860	0.2617
2	0.0446	1.2808
3	0.0013	0.5879
4	0.0012	**0.1373**
5	0.0011	11.347
6	**0.0000**	9586.9

- *Underfitting*: model too inflexible / too much data (poor generalization)
- *Overfitting*: model too flexible / too little data (poor generalization)
- *Inductive bias*: choice of hypothesis space incorporates assumptions (generalization is impossible without any assumptions!)

Supervised learning terminology

- $x = (x_1, \ldots, x_p)$: *input variables*, or *features*, *predictors*, *covariates*, or *independent variables*.
- y: *output variable*, *response*, *dependent variable*, or *class label* (in classification)
- For now on, we focus on the following regression setting:

  $$y = g(x) + \epsilon,$$

  where g is some unknown (possibly horribly complicated) function, x is sampled from some arbitrary distribution, and ϵ is the *error term* that is sampled independently from a distribution that satisfies $E[\epsilon] = 0$ and $E[\epsilon^2] = \sigma^2$ for some σ^2. (This defines our sampling distribution $(x, y) \sim F$!)
- **Informal objective:** *learn* $\hat{f}$ that is an *estimate* of g.
- **Formal objective:** find $\hat{f}$ that minimizes loss on new data.

Linear regression

One-dimensional ordinary least squares (OLS) linear regression

- $f(x) = \beta_0 + \beta_1 x$.
  - β_0 is the *intercept*, β_1 is the *slope*.
- Minimise empirical loss $E_{(x,y)\sim F}\left[L(\hat{y}, y)\right] \approx L_{emp} = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2/n$.
- We can solve equations $\partial L_{emp}/\partial \beta_0 = \partial L_{emp}/\partial \beta_1 = 0$ analytically and obtain the following expressions:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})/n}{\sum_{i=1}^{n} (x_i - \overline{x})^2/n} = \frac{\hat{\sigma}_{xy}}{\hat{\sigma}_{xx}},$$

$$\hat{\beta}_0 = \overline{y} - \hat{\beta}_1 \overline{x},$$

where $\overline{x} = \sum_{i=1}^{n} x_i/n$ and $\overline{y} = \sum_{i=1}^{n} y_i/n$.

OLS linear regression

- Useful trick (makes equations simpler):
  - include to data a dummy input variable of all ones. This will be the intercept
  - i.e., if you have p dimensional data $x_i = (x_{i1}, \dots, x_{ip}) \in \mathbb{R}^p$ replace it by $p+1$ dimensional data $x_i' = (1, x_{i1}, \dots, x_{ip}) \in \mathbb{R}^{p+1}$!
  - This makes *affine* function $f(x) = \beta^T x + \beta_0$ *linear* $f(x') = \beta'^T x'$, where $\beta' = (\beta_0, \beta_1, \dots, \beta_p)$.
- From now on we therefore mostly present everything for linear functions (instead of affine)
  - In practice this means adding a column of ones to the data matrix

Multivariate OLS linear regression

- Matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ has n instances x_i as its rows and $y \in \mathbb{R}^n$ contains the corresponding output variables y_i.
- Terminology: $\mathbf{X}$ is the *design matrix*
- Assume that the data has been generated by

  $$y = \mathbf{X}\beta + \epsilon,$$

  where the *residual* $\epsilon \in \mathbb{R}^n$ are i.i.d. variables with zero mean and variance of σ^2.
- *Optimization problem:* Find $\hat{\beta}$ such that loss $L(\hat{\beta}) = \hat{\epsilon}^T \hat{\epsilon}$ is minimised, where $\hat{\epsilon} = y - \mathbf{X}\hat{\beta}$.

Multivariate OLS linear regression

- Can be solved, e.g., by using derivatives, by setting $\partial L / \partial \beta_j = 0$.
- Denote by $\mathbf{A}^{-1}$ the inverse of square matrix $\mathbf{A}$, i.e., $\mathbf{A}\mathbf{A}^{-1} = \mathbf{1}$. The solution is then $\hat{\hat{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y$.
- If columns of $\mathbf{X}$ are linearly independent $\mathbf{X}^T\mathbf{X}$ is of full rank and has an inverse
  - True for $n \geq p$ except for degenerate special cases
- $\mathbf{X}^T\mathbf{X} \in \mathbb{R}^{p \times p}$, inverting takes $O(p^3)$ time [or $O(p^{2.373})$ if we nit-pick]
  - May be unfeasible for very high-dimensional problems
  - You should usually not deal with matrices directly but use OLS software libraries instead (e.g., `lm` in R)

Nonlinear models by transforming the input

- *Linear* regression can be used to fit models which are *nonlinear* functios of the input
- Example: For fitting a degree 3 polynomial

$$f(x_i) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3$$

create the input matrix

$$\mathbf{X} = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}.$$

Nonlinear models by transforming the input

- We can also explicitly include *interaction terms*, as in

$$f(x_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i1} x_{i2}$$

  using the following matrix

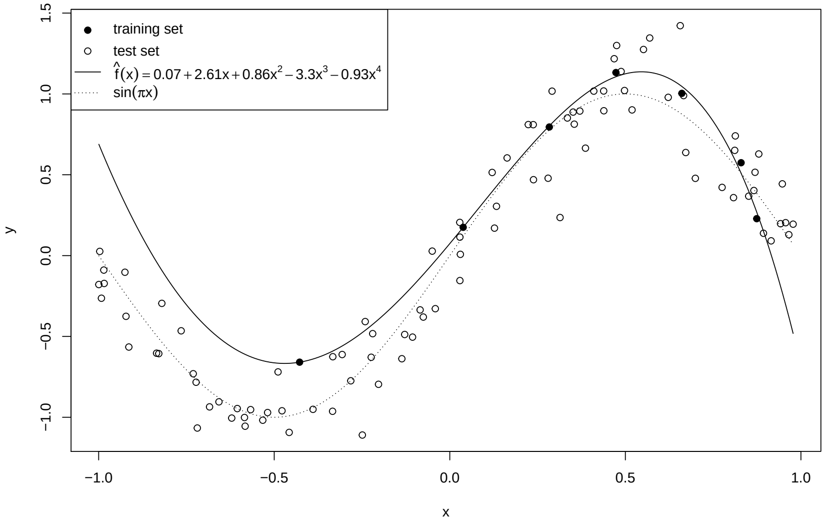
$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & x_{11}x_{12} \\ 1 & x_{21} & x_{22} & x_{21}x_{22} \\ 1 & x_{31} & x_{32} & x_{31}x_{32} \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}.$$

Polynomial degree 4

```
data.fit <- lm(y ~ poly(x,4,raw=TRUE),data)
summary(data.fit)
```

```
##
## Call:
## lm(formula = y ~ poly(x, 4, raw = TRUE), data = data)
##
## Residuals:
##          1          2          3          4          5          6          7
##  6.483e-02 -4.073e-02  1.062e-04 -4.290e-02 -7.160e-03 -2.867e-05  2.588e-02
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               0.0742     0.0755   0.983   0.4294
## poly(x, 4, raw = TRUE)1   2.6125     0.4729   5.524   0.0312 *
## poly(x, 4, raw = TRUE)2   0.8572     0.3923   2.185   0.1605
## poly(x, 4, raw = TRUE)3  -3.2982     2.4058  -1.371   0.3040
## poly(x, 4, raw = TRUE)4  -0.9259     2.2496  -0.412   0.7206
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0649 on 2 degrees of freedom
## Multiple R-squared:  0.9963, Adjusted R-squared:  0.9888
## F-statistic: 133.7 on 4 and 2 DF,  p-value: 0.007439
```

Polynomial degree 4



Estimating model parameters

Estimators and their key properties

- In machine learning we try to compute unknown quantities (denoted here by β) based on observed data by using statistical *estimators* (denoted here by $\hat{\beta}$)
- Examples:
  - estimating regression coefficients $\hat{\beta}$ by OLS linear regression, “true” value being β
  - estimating loss by empirical loss $L_{emp} = \sum_{i=1}^{n} L(y_i, \hat{f}(x_i))/n$, “true” loss being $E_{(x,y)\sim F}[L(y, \hat{f}(x))]$.
- Important properties of estimators:
  - $\mathrm{bias} = E[\hat{\beta}] - \beta$.
    - The estimator is *unbiased* if $\mathrm{bias} = 0$
  - $\mathrm{variance} = E[(\hat{\beta} - E(\hat{\beta}))^2]$.
  - $\mathrm{MSE} = E[(\hat{\beta} - \beta)^2] = \mathrm{bias}^2 + \mathrm{variance}$.

Consistent estimators

Estimator is *consistent* if it produces a true value at the limit of infinite data.

Many of the estimates we would like to use in machine learning are consistent. This means that if we have very large data then the "statistical learning" will just be "learning". For example, the empirical estimate of loss is typically consistent estimator of the true loss (at the loss function $L()$ is well-behaving etc.), i.e.,

$$\lim_{n\to\infty} L_{emp} = \lim_{n\to\infty} \sum_{i=1}^{n} L(y_i, \hat{y}_i)/n = E_{(x,y)\sim F}\left[L(y, \hat{y})\right].$$

Then machine learning problem reduces just to optimising the empirical loss, without need for any further statistical considerations.

Estimating mean and empirical loss of the mean estimate

- Consider problem of estimating mean of n data points $y_1, \ldots, y_n$.
- Assume the data comes from a fixed but unknown distribution with "true" mean $\beta = E[y]$ and variance $\sigma^2 = E[(y-\beta)^2]$.
- Estimate the mean by empirical mean: $\hat{\beta} = \sum_{i=1}^{n} y_i / n$
  - The estimator is unbiased: $E[\hat{\beta}] = \sum_{i=1}^{n} E[y_i]/n = \beta$ (because y_i comes from F which has mean of β, i.e., $E[y_i] = \beta$)
- Estimate MSE $\sigma^2 = E[(y-\beta)^2]$ by empirical loss: $L_{emp} = \sum_{i=1}^{n} (y_i - \hat{\beta})^2 / n$
  - The estimator is biased: $E[L_{emp}] = E[\sum_{i=1}^{n} (y_i - \hat{\beta})^2 / n] = (1 - 1/n)\sigma^2$.
  - $\mathrm{bias} = E[L_{emp}] - \sigma^2 = -\sigma^2/n$.
  - More generally: empirical loss tends to underestimate the generalisation loss!

Parametric estimation of confidence interval

The most important special case is given by t statistics. The t-statistics is given by the value of the statistics (e.g., estimate $\hat{\beta}$ normalised by its standard deviation), i.e.,

$$t = \hat{\beta}/\mathrm{sd}(\hat{\beta}),$$

where

$$\mathrm{sd}(\hat{\beta}) = \sigma/\sqrt{n}.$$

If the data is normally distributed then the 95% confidence interval is given by $\pm 1.96 \times \mathrm{sd}(\hat{\beta})$, corresponding to $-1.96 \leq t \leq 1.96$.

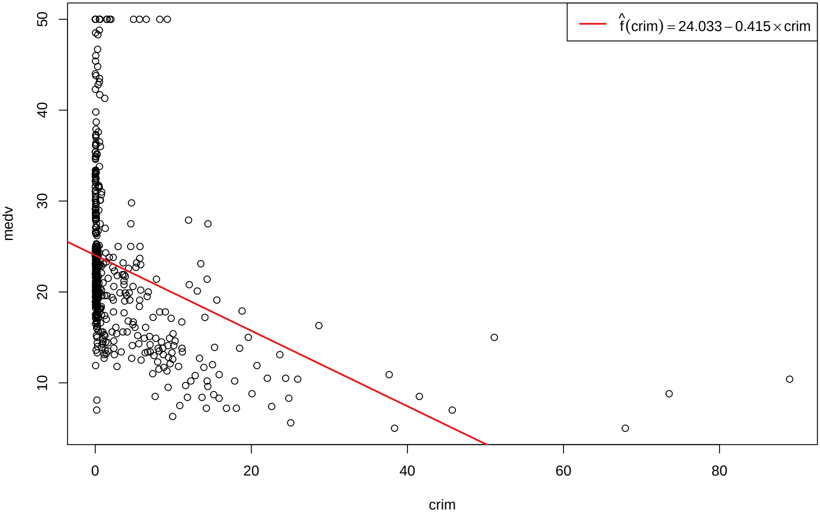
Caution: be extra careful in interpreting your parameter estimates if the underlying assumptions are not obeyed (e.g., variables heteroscedastic, residuals ϵ_i correlated, g clearly non-linear, outliers).

Boston housing data

```r
library(MASS)
boston.fit <- lm(medv ~ crim, data = Boston)
summary(boston.fit)
```

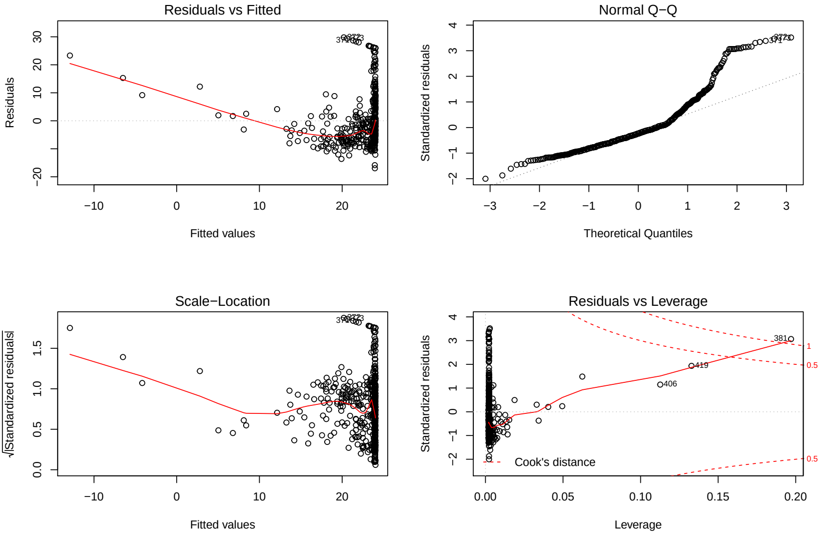
```
##
## Call:
## lm(formula = medv ~ crim, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -16.957  -5.449  -2.007   2.512  29.800
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 24.03311    0.40914   58.74   <2e-16 ***
## crim        -0.41519    0.04389   -9.46   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.484 on 504 degrees of freedom
## Multiple R-squared:  0.1508, Adjusted R-squared:  0.1491
## F-statistic: 89.49 on 1 and 504 DF,  p-value: < 2.2e-16
```

Boston housing data



Boston housing data: diagnostic plots

```r
plot(boston.fit)
```



Residuals vs Fitted
Residuals
30
20
10
0
−20
−10
0
10
20
Fitted values
371
372
373
Normal Q−Q
Standardized residuals
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
Theoretical Quantiles
371
372
373
Scale−Location
√|Standardized residuals|
1.5
1.0
0.5
0.0
−10
0
10
20
Fitted values
371
372
373
Residuals vs Leverage
Standardized residuals
4
3
2
1
0
−1
−2
0.00
0.05
0.10
0.15
0.20
Leverage
381
419
406
1
0.5
0.5
Cook's distance