Machine Learning and Applications: Linear Regression

Alessio Benavoli

CSIS
University of Limerick

Regression

Many problems we find in science/engineering/business are of the following form.

Output:

- We have a continuous variable (i.e., variable represented using real numbers).
- We call this variable the dependent, predicted, or outcome variable.

Input:

- We want to model how this dependent variable depends on one or more other variables
- that we call independent, predictor, or input variables.

The independent variables can be either **continuous** or **categorical** or mixed.
The dependent variable can only be **continuous**.

These type of problems are called regression problems, we will initially focus on **linear regression**.

Simple or Multiple linear regression

If we have only one independent variable we may use a *simple linear regression* model.

If we have more than one independent variable then we may apply a *multiple linear regression* model.

Example

A farmer wants to decide if the use of a fertilizer increases the crop productivity. He knows that other factors affect crop productivity such as rain and soil salinity.

Output:

- crop productivity (continuous)

Inputs:

- rain (continuous)
- soil salinity (continuous)
- fertilizer yes/no (categorical)

This is an example of multiple linear regression.

Single linear regression

Let's begin to learn how to build linear models.

You may already be familiar with the following equation:

$$y = \beta\, x + \alpha$$

It says there is a linear relation between the variable x and the variable y.

The slope parameter ($\beta \in \mathbb{R}$) can be interpreted as the change in the variable y per unit change in the variable x.

The other parameter is the intercept ($\alpha \in \mathbb{R}$) and tells us the value of y when $x = 0$.

Graphically, the intercept is the point where the line intercepts the y-axis.

Unknowns

Given the choice of a linear relationship between x and y, to complete our model we need to specify the values of the parameters

$$\alpha, \beta$$

Machine learning (and statistics) deals with the problem of **estimating** these values from data:

- in linear regression "data" (the observations) is a list of pairs of values of the independent and dependent variables, that is $\mathcal{D} = \{(x_i, y_i) : \; i = 1, \ldots, N\}$

This is the meaning of **learning**, that is learning a model from data.

In the case of linear regression, the model is a line and to learn a line we need to learn the value of the intercept and slope.

A concrete example

Assume you are on a train, moving at constant speed on a straight railway from city A to city B (without intermediate stops). We want to predict the arrival time at the station B to see if we can make a tight bus connection.

We do not know the train speed and we assume acceleration/deceleration are instantaneous.

How can we compute that?

In order to do that, we need to use a law of physics:

$$t = f(s)$$

where s is space (distance along the straight railway) and t is travel time.

We do not remember the law of physics, (in other words we do not the function $f(\cdot)$). We do not have internet connection, but we have our watch that we can use to measure time. Along the railway there are regularly placed *marking stones* that mark some distance (sort of milestones).

Linearity hypothesis

Let's make this hypothesis:

$$t = \beta\, s + \alpha$$

the unknown physics law is linear. We can measure (s, t) and, therefore, we can derive the parameters α, β.

We can write this relationship as

$$y = \beta\, x + \alpha$$

where $x = s, y = t$.

Determining the parameters of a line

Assume we have two pairs of observations (s, t):

$$\mathcal{D} = \{(1, 3.4), (2, 5.9)\}$$

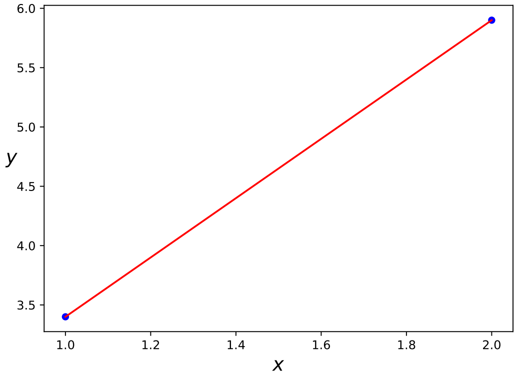
where 1 (in $(1, 3.4)$) means the first marking stone and the time measure is *minutes*.
We want to find the line (α, β) that **fits** the data.

We do it by solving the following system of equations:

$$\begin{aligned} 3.4 &= \beta\, 1 + \alpha \\ 5.9 &= \beta\, 2 + \alpha \end{aligned}$$

whose solution is $\alpha = 0.9$ and $\beta = 2.5$.

Solution



Three points

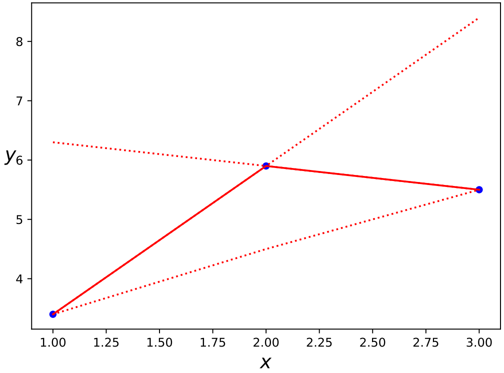
$$\mathcal{D} = \{(1, 3.4), (2, 5.9), (3, 5.5)\}$$

We need to solve:

$$\begin{aligned} 3.4 &= \beta\, 1 + \alpha \\ 5.9 &= \beta\, 2 + \alpha \\ 5.5 &= \beta\, 3 + \alpha \end{aligned}$$

This system of equations does not have solutions!

No Solution

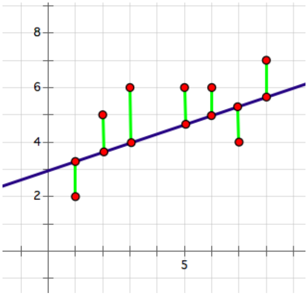


What does it mean?

1. Does it mean the physics law is not linear?
2. Are my measurements wrong?

The accuracy of our time-measure is quite bad (we stop the watch when we see the marking stone, sometimes we miss the marking stone,...) so we need to account for that.

LS method



https://en.wikipedia.org/wiki/File:Least_Squares.png

Least squares method aims to find the parameters α, β by solving the optimisation problem:

$$\min_{\alpha,\beta} \sum_{i=1}^{N} (y_i - x_i\,\beta + \alpha)^2$$

This is the line that *fits* the data (it is “close” to the data) without going through the points. This allows us to account for the measurement noise.

Our example

$$\arg\min_{\alpha,\beta} \sum_{i=1}^{N} (y_i - x_i\,\beta + \alpha)^2$$

we have that $N = 3$ observations and

$$\sum_{i=1}^{3} (y_i - x_i\,\beta + \alpha)^2 = (3.4 - \beta - \alpha)^2 + (5.5 - 2\beta - \alpha)^2 + (5.5 - 3\beta - \alpha)^2$$

To compute the minimum, we need to compute the derivatives w.r.t. α, β, set them equal to zero and solve for α, β. Try it by yourself!

Matrix form

A fast way to do that is to put the problem in matrix form. We define the matrices

$$H = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \quad \theta = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

and observe that

$$\sum_{i=1}^{N} (y_i - x_i\,\beta - \alpha)^2 = (y - H\theta)^T (y - H\theta)$$

Note that ^{T} denotes transpose. Verify it!

Least squares solution

We need to solve

$$\hat{\theta} := \arg\min_{\theta} (y - H\theta)^T (y - H\theta)$$

the solution is

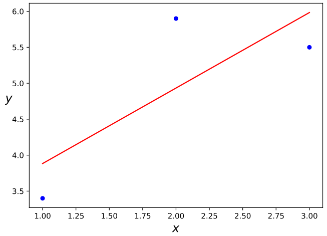
$$\hat{\theta} = (H^T H)^{-1} H^T y$$

the first component of $\hat{\theta}$ is the LS estimate of the intercept and the second of the slope.

Our problem, $N = 3$

$$H = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}, \quad y = \begin{bmatrix} 3.4 \\ 5.5 \\ 5.5 \end{bmatrix}, \quad \theta = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

$\hat{\theta} = [2.83, 1.05]$

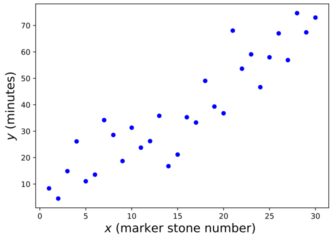


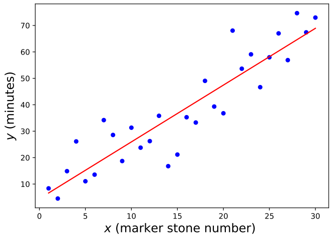
Python

```
#this code implements leastsquares
def leastsquares(x,y):
    X = np.hstack([np.ones((len(x),1)),x.reshape(-1,1)])
    yc = y.reshape(-1,1) #column vector
    M = np.linalg.inv(np.matmul(X.T,X))
    theta = np.matmul(M, np.matmul(X.T,y) )
    return theta

x = np.array([1,2,3])
y_obs = np.array([3.4,5.9,5.5])
theta=leastsquares(x,y_obs)
print(theta)
>>[2.83,1.05]
```

N=30 observations



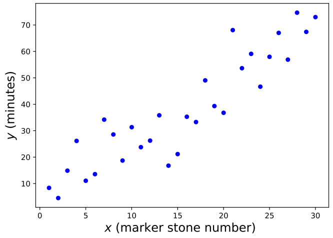


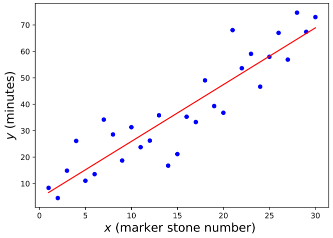
$\hat{\theta} = [4.47, 2.15]$

```
#You can get the same result in Python
#using the implementation of linear regression in sklearn
from sklearn.linear_model import LinearRegression
lreg = LinearRegression(fit_intercept=True)
res = lreg.fit(x.reshape(-1,1), y_obs)

print([res.intercept_, res.coef_[0]])
>>[4.465180322042002, 2.148603272224278]
```

N=30 observations





$\hat{\theta} = [4.47, 2.15]$

The previous code is equivalent to

```
X = np.hstack([np.ones((len(x),1)),x.reshape(-1,1)])
lreg = LinearRegression(fit_intercept=False)
res = lreg.fit(X, y_obs)
print(res.coef_)
>>[4.465180322042002, 2.1486032722224278]
```

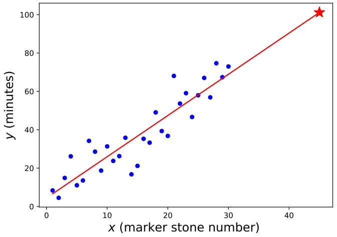
Prediction

Now we can predict the time of arrival at the station B (45th marker stone):

$$y_{pred} = [1, x_{pred}]\,\hat{\theta}$$

where $x_{pred} = 45$ in this example. Plugging in the LS estimate $\hat{\theta}$:

$$y_{pred} \approx 4.465 + 2.15 \cdot 45 = 101.15 \ \ (minutes)$$



Done, now we have a prediction of the TOA of the train.

Prediction in Python

For the first sklearn implementation, we can do predictions as

```
 #in sklearn we can do it as follows
ypred = lreg.predict(np.array([[45]]))
print(ypred)
>>101.15232757
```

For the second sklearn implementation, we can do predictions as

```
 #in sklearn we can do it as follows
ypred = lreg.predict(np.array([[1,45]]))
print(ypred)
>>101.15232757
```

Machine Learning

- $\mathcal{D} = \{(x_i, y_i) : \ i = 1, \ldots, N\}$ is called *training set*, because we use it to learn the parameters (train) of the model;
- the points x_{pred} are called *test points*.

Usually we do not know the value of the y corresponding to x_{pred}, the goal of ML is to use the trained model to provide a prediction of the y corresponding to x_{pred}.

General ML Recipe

Let's denote with

$$y = f(x, \theta)$$

a general ML algorithm that takes as input x and outputs y, with θ denoting its parameters.

Training phase, it uses the data $\mathcal{D} = \{(x_i, y_i) : \; i = 1, \ldots, N\}$:

$$\hat{\theta} := \arg\min_{\theta} \sum_{i=1}^{N} L(y_i, f(x_i, \theta))$$

where $L(\cdot, \cdot)$ is called loss function.

Prediction:

$$\hat{y} = f(x_{pred}, \hat{\theta})$$

This is a general formulation: it holds for linear regression but also for neural networks, deep neural networks and any other standard ML algorithm.

Traditional Linear Regression

In case of Linear Regression we have

$$y = f(x, \theta) = \beta\, x + \alpha$$

it takes as input x and outputs y, with $\theta = [\alpha, \beta]$ denoting its parameters.

Training phase, it uses the data $\mathcal{D} = \{(x_i, y_i) :\ i = 1, \ldots, N\}$:

$$\hat{\theta} := \arg\min_{\theta} \sum_{i=1}^{N} L(y_i, f(x_i, \theta))$$

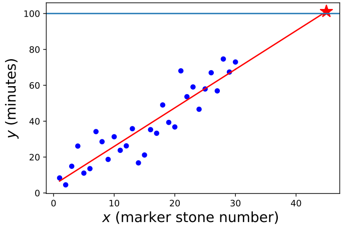
where $L(y_i, f(x_i, \theta)) = (y_i - \beta\, x - \alpha)^2$ is called square-loss function.

Prediction:

$$\hat{y} = \hat{\beta}\, x_{pred} + \hat{\alpha}$$

Will we arrive on time?

Let's assume that the bus leaves at the 100th minute (horizontal line)



It seems that we cannot do it!

However, we are not taking into account that our observations are noisy. Maybe we will arrive on time. How can we account for the uncertainty?

we will answer this question later on.

Back to the train

What if the law of physics that describes the relation between speed and time is not linear ?

$$t = f(s) \quad \text{but} \quad f(s) \neq \alpha + \beta s$$

May it be quadratic, $\alpha + \beta s + \gamma s^2$?

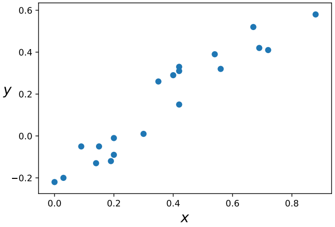
We do not remember the law, can we use the data we collected to answer this question ?

Model Comparison: Linear or Quadratic?

Our question is:

what better describes the train motion: a linear (M_1) or quadratic (M_2) curve?

$$M_1: \quad y = \theta_0 + \theta_1 x$$
$$M_2: \quad y = \theta_0 + \theta_1 x + \theta_2 x^2$$



NonLinear function, but the model is linear in the parameters

This a nonlinear function of x

$$y = \theta_0 + \theta_1 x + \theta_2 x^2$$

but it is linear in θ_i. We can still use the Least Squares methods to learn θ_i from data.

Matrix form: we have already seen the linear case, let's see the quadratic case

$$H = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \\ 1 & x_N & x_N^2 \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

and observe that

$$\sum_{i=1}^{N} (y_i - \theta_0 - \theta_1 x_i - \theta_2 x_i^2)^2 = (y - H\theta)^T (y - H\theta)$$

Least squares solution

We need to solve

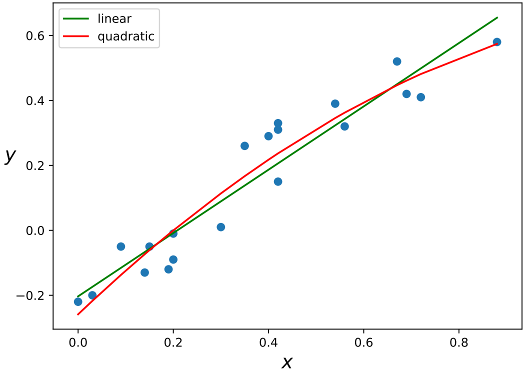
$$\hat{\theta} := \arg\min_{\theta} (y - H\theta)^T (y - H\theta)$$

the solution is

$$\hat{\theta} = (H^T H)^{-1} H^T y$$

This holds no matter the degree of the polynomial because the matrix form is general!

Solution



Note that, the fitting error is in general expressed as an average:

$$MSE = \frac{1}{n}(y - H\hat{\theta})^T(y - H\hat{\theta})$$

and is called *Mean Squared Error* (MSE).

Model Comparison

The **Question** in model Comparison is:

how can we quantify the difference between these models and decide which model better describes our data?

One common mistake is to assume that we can select between models via the value of the fitting error

$$\frac{1}{n}(y - H_{M_1}\hat{\theta}_{M_1})^T(y - H_{M_1}\hat{\theta}_{M_1}) \lesseqgtr \frac{1}{n}(y - H_{M_2}\hat{\theta}_{M_2})^T(y - H_{M_2}\hat{\theta}_{M_2})$$

The two errors are:

M_1 : 0.113

M_2 : 0.094

M_2 will be always better. The reason is that the class of functions M_2 includes the class M_1:

$$y = \theta_0 + \theta_1 x + (\theta_2 x^2)$$

so M_2 can always reach the same minimum as M_1 by simply selecting $\theta_2 = 0$.

Complex models have smaller fitting error

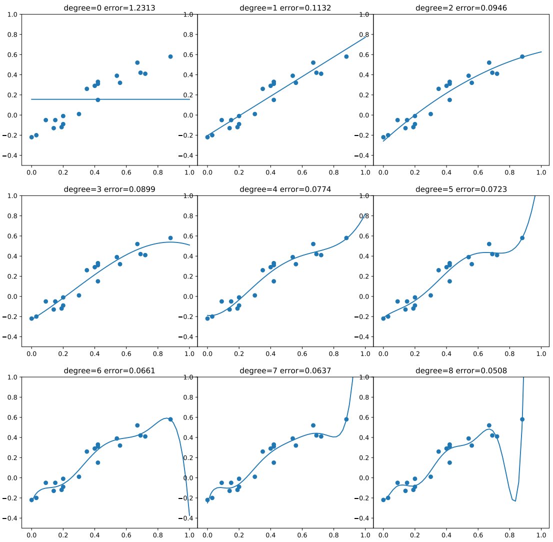
The quadratic model yields a lower error, but this does not necessarily mean it is the better model!

The problem is that the quadratic model has more degrees of freedom than the linear model, and thus will always give an equal or smaller MSE (or, equivalently, an equal or larger maximum likelihood), regardless of the data!

This trend holds generally: as you increase model complexity, the fitting error will (almost) always decrease!

Let's take a look at the fitting error for a series of polynomial fits (linear, quadratic, cubic, quartic, etc.).

Complex models have smaller fitting error



Overfitting

We have seen that the fitting error always decreases as we increase the degree of the polynomial. Looking at the figures, we see how this happens: while the degree-8 polynomial certainly leads to a smaller error, it achieves this by **overfitting** the data.

overfitting occurs when a function is too closely fit to a limited set of data points.

overfitting the model generally takes the form of making an overly complex model to explain noise in the data.

The intuition may be clear but these definitions are not very formal: what does "too closely" mean? what does "explaining noise" mean?

Underfitting

Underfitting occurs when a ML algorithm cannot capture the underlying trend of the data.

Intuitively, underfitting occurs when the model does not fit the data well enough.

Underfitting is often a result of an excessively simple model.

Number of observations

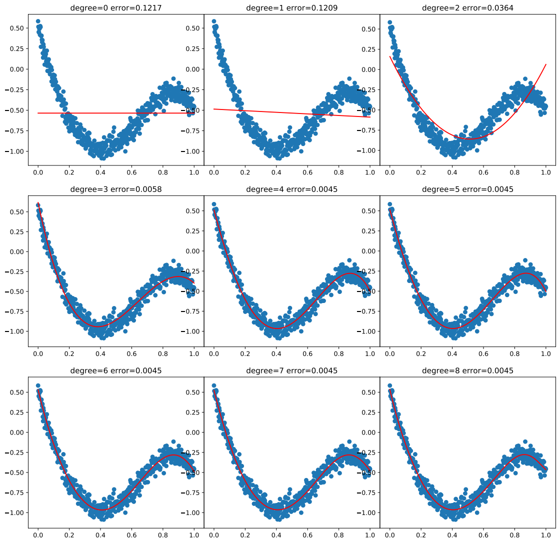
The problem of “overfitting” in general gets better at the increase of the number of observations.

Note that we are considering nested models, that is the polynomials of degree n include the class of polynomials of degree n-1 (to prove that, it is enough you put to zero the coefficient of the monomial with the highest degree).

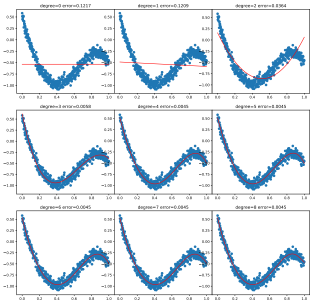
Therefore, we can recover lower degree polynomial models using higher degree polynomial models.

Let’s generate some data from a polynomial model of degree 4.

Number of observations



Number of observations

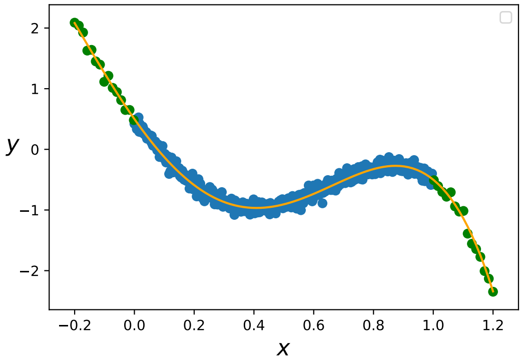


$$\theta_{true} = [0.5, -7, 6.5, 8, -8.5]$$

$$\hat{\theta} = [0.54, -8.79, 30.92, -140.44, 463.32, -838.61, 837.2, -437.48, 92.84]$$

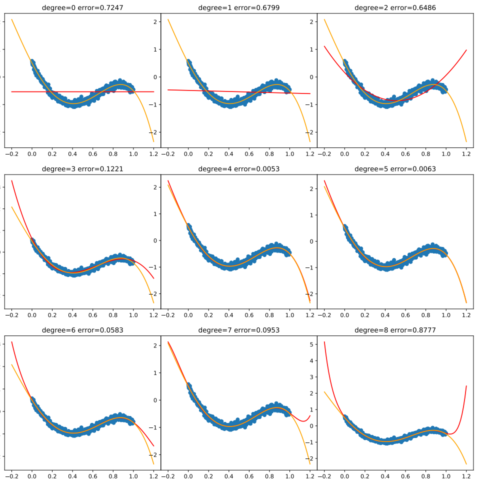
Note that, contrarily what we could think, the parameters $\hat{\theta}_5, \hat{\theta}_6, \hat{\theta}_7, \hat{\theta}_8$ that multiply the monomials x^5, x^6, x^7, x^8 are in general not zero.
So we cannot always use the values of $\hat{\theta}_5, \hat{\theta}_6, \hat{\theta}_7, \hat{\theta}_8$ to understand that the true model has a lower degree.

Generalisation Error



Generalisation Error: it is a measure of how accurately an algorithm is able to predict outcome values for previously unseen data.
When this data lie outside the region that includes the training data, we talk about extrapolation error (see figure).

Extrapolation Error



We have computed the error with respect to the green observations (see previous figure).

Cross-validation

Although overfitting gets better at the increase of the number of observations, it is still present.

Moreover, when the model we are using is misspecified (for example we are using a polynomial model and the "true" model that generates the data is sinusoidal), then the fitting error is not a good measure of the ability of our model to describe the data (even with a lot of observations).

Cross-validation

The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to overcome problems like overfitting and to give an insight on how the model will generalize to an unseen dataset.

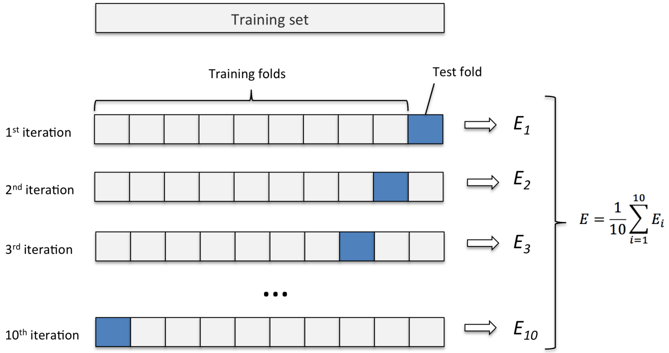
k-fold cross-validation: the original sample is randomly partitioned into k equal sized subsamples.

Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k 1 subsamples are used as training data.

The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data.

The k results can then be averaged to produce a single error measure.

10-fold Cross-validation



Note that data are randomly shuffled before creating the 10-folds.

2-fold Cross-validation

For example, setting $k = 2$ results in 2-fold cross-validation.

In 2-fold cross-validation, we randomly shuffle the dataset into two sets D_1 and D_2, so that both sets are equal size (this is usually implemented by shuffling the data array and then splitting it in two).

We then train on D_1 and validate on D_2, followed by training on D_2 and validating on D_1.

When k = n (the number of observations), the k-fold cross-validation is the **leave-one-out** cross-validation.

LOO

The leave-one-out cross-validation is important in linear regression because we can compute the "generalisation error" in closed form (without training the algorithm n times). [1]

Define

$$\begin{aligned}\hat{\theta} &= (H^T H)^{-1} H^T y\\ R &= H(H^T H)^{-1} H^T\\ e_i &= y_i - H_i \hat{\theta}\end{aligned}$$

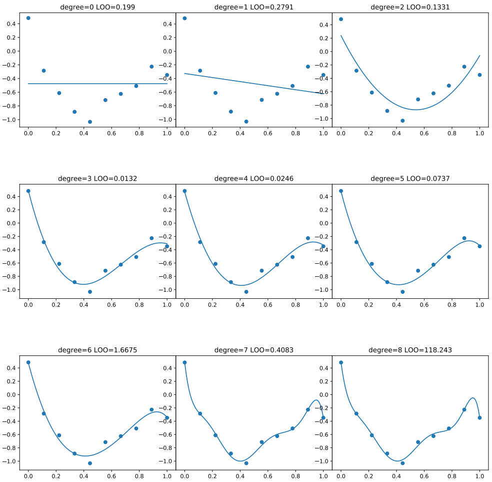
where H_i is the i-th row of H.
If the diagonal values of R are denoted by $r_1, \ldots, r_N$, then the LOO CV statistic can be computed using

$$CV = \frac{1}{N} \sum_{i=1}^{N} (e_i/(1 - r_i))^2$$

[1] `https://robjhyndman.com/hyndsight/loocv-linear-models/`

LOO

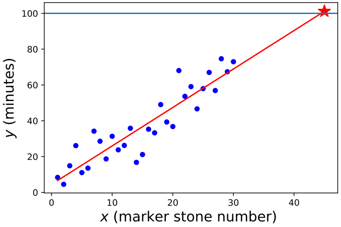
Let's generate some data from a polynomial model of degree 4.



The best model is now degree 4!

Will we arrive on time?

Let's assume that the bus leaves at the 100th minute (horizontal line)



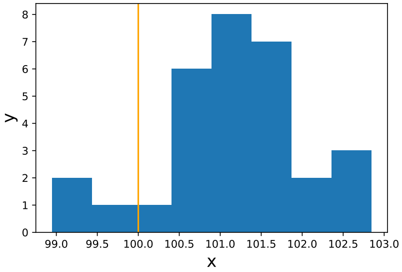
It seems that we cannot do it!

Can we use LOO to answer this question?

```
from sklearn.model_selection import LeaveOneOut
X = np.hstack([np.ones((len(x),1)),x.reshape(-1,1)])
xpred=45
loo = LeaveOneOut()
Ypred=[]
for train_index, test_index in loo.split(X):
    X_train = X[train_index]
    y_train= y_obs[train_index]
    lreg = LinearRegression(fit_intercept=False)
    res=lreg.fit(X_train, y_train)
    Ypred.append(res.predict(np.array([[1,xpred]])))

plt.hist(np.array(Ypred),bins='auto')
plt.axvline(100,color='orange')
plt.xlabel("x",fontsize=16);
plt.ylabel("y",fontsize=16);
print(``% of TOA<=100'',len(np.where(np.array(Ypred)<=100)[0])/len(
                                                    Ypred))
```

Can we use LOO to answer this question?



In 10% of the cases, we will arrive on time.

We can reject with confidence at least 89% the hypothesis that we will be on time.
We cannot reject with confidence at least 90% the hypothesis that we will be on time.
We **cannot reject with confidence at least 95%** the hypothesis that we will be on time.
We **cannot reject with confidence 99%** the hypothesis that we will be on time.

Bootstrap

This way of assessing the model is also called **Bootstrap**.
In practice, we are generating datasets by removing one observation and for each dataset we compute the prediction of our model.

This approach has several limitations:

1. Should we remove more than one observation?
2. What is the meaning of the histogram?

Answers:

1. Bootstrapping is not very principled, because it depends on the way we decide to perform it.
2. The histogram does not tell us the probability that we will take our bus at the minute 100 (that is the probability that TOA<100). It tells us that, under our bootstrapping method, there are cases where y_{pred} is lower than 100. In other words, if we had seen all observations apart from a certain one, then our prediction would have been less than 100

There is a more principled way to model uncertainty in ML, but we won't see it in this module.