Data Analysis and Machine Learning: Linear Regression and more Advanced Regression Analysis

Morten Hjorth-Jensen[1,2]

Department of Physics, University of Oslo[1]

Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University[2]

Sep 7, 2018

Regression analysis, overarching aims

Regression modeling deals with the description of the sampling distribution of a given random variable y varies as function of another variable or a set of such variables $\hat{x} = [x_0, x_1, \dots, x_p]^T$. The first variable is called the **dependent**, the **outcome** or the **response** variable while the set of variables $\hat{x}$ is called the independent variable, or the predictor variable or the explanatory variable.

A regression model aims at finding a likelihood function $p(y|\hat{x})$, that is the conditional distribution for y with a given $\hat{x}$. The estimation of $p(y|\hat{x})$ is made using a data set with

- n cases $i = 0, 1, 2, \dots, n-1$
- Response (dependent or outcome) variable y_i with $i = 0, 1, 2, \dots, n-1$
- p Explanatory (independent or predictor) variables $\hat{x}_i = [x_{i0}, x_{i1}, \dots, x_{ip}]$ with $i = 0, 1, 2, \dots, n-1$

The goal of the regression analysis is to extract/exploit relationship between y_i and $\hat{x}_i$ in or to infer causal dependencies, approximations to the likelihood functions, functional relationships

Regression analysis, overarching aims II

Consider an experiment in which p characteristics of n samples are measured. The data from this experiment are denoted $\mathbf{X}$, with $\mathbf{X}$ as above. The matrix $\mathbf{X}$ is called the *design matrix*. Additional information of the samples is available in the form of $\mathbf{Y}$ (also as above). The variable $\mathbf{Y}$ is generally referred to as the *response variable*. The aim of regression analysis is to explain $\mathbf{Y}$ in terms of $\mathbf{X}$ through a functional relationship like $Y_i = f(\mathbf{X}_{i,*})$. When no prior knowledge on the form of $f(\cdot)$ is available, it is common to assume a linear relationship between $\mathbf{X}$ and $\mathbf{Y}$. This assumption gives rise to the *linear regression model* where $\beta = (\beta_1, \dots, \beta_p)^\top$ is the *regression parameter*. The parameter β_j, $j = 1, \dots, p$, represents the effect size of covariate j on the response. That is, for each unit change in covariate j (while keeping the other covariates fixed) the observed change in the response is equal to β_j.

General linear models

Before we proceed let us study a case from linear algebra where we aim at fitting a set of data $\hat{y} = [y_0, y_1, \dots, y_{n-1}]$. We could think of these data as a result of an experiment or a complicated numerical experiment. These data are functions of a series of variables $\hat{x} = [x_0, x_1, \dots, x_{n-1}]$, that is $y_i = y(x_i)$ with $i = 0, 1, 2, \dots, n-1$. The variables x_i could represent physical quantities like time, temperature, position etc. We assume that $y(x)$ is a smooth function.

Since obtaining these data points may not be trivial, we want to use these data to fit a function which can allow us to make predictions for values of y which are not in the present set. The perhaps simplest approach is to assume we can parametrize our function in terms of a polynomial of degree $n-1$ with n points, that is

$$y = y(x) \rightarrow y(x_i) = \tilde{y}_i + \epsilon_i = \sum_{j=0}^{n-1} \beta_i x_i^j + \epsilon_i,$$

where ϵ_i is the error in our approximation.

Rewriting the fitting procedure as a linear algebra problem

For every set of values y_i, x_i we have thus the corresponding set of equations

$$\begin{aligned}
y_0 &= \beta_0 + \beta_1 x_0^1 + \beta_2 x_0^2 + \dots + \beta_{n-1} x_0^{n-1} + \epsilon_0 \\
y_1 &= \beta_0 + \beta_1 x_1^1 + \beta_2 x_1^2 + \dots + \beta_{n-1} x_1^{n-1} + \epsilon_1 \\
y_2 &= \beta_0 + \beta_1 x_2^1 + \beta_2 x_2^2 + \dots + \beta_{n-1} x_2^{n-1} + \epsilon_2 \\
&\dots\dots \\
y_{n-1} &= \beta_0 + \beta_1 x_{n-1}^1 + \beta_2 x_{n-1}^2 + \dots + \beta_1 x_{n-1}^{n-1} + \epsilon_{n-1}.
\end{aligned}$$

Rewriting the fitting procedure as a linear algebra problem, follows

Defining the vectors

$$\hat{y} = [y_0, y_1, y_2, \dots, y_{n-1}]^T,$$

and

$$\hat{\beta} = [\beta_0, \beta_1, \beta_2, \dots, \beta_{n-1}]^T,$$

and

$$\hat{\epsilon} = [\epsilon_0, \epsilon_1, \epsilon_2, \dots, \epsilon_{n-1}]^T,$$

and the matrix

$$\hat{X} = \begin{bmatrix}
1 & x_0^1 & x_0^2 & \dots & \dots & x_0^{n-1} \\
1 & x_1^1 & x_1^2 & \dots & \dots & x_1^{n-1} \\
1 & x_2^1 & x_2^2 & \dots & \dots & x_2^{n-1} \\
\dots & \dots & \dots & \dots & \dots & \dots \\
1 & x_{n-1}^1 & x_{n-1}^2 & \dots & \dots & x_{n-1}^{n-1}
\end{bmatrix}$$

we can rewrite our equations as

Generalizing the fitting procedure as a linear algebra problem

We are obviously not limited to the above polynomial. We could replace the various powers of x with elements of Fourier series, that is, instead of x_i^j we could have $\cos{(jx_i)}$ or $\sin{(jx_i)}$, or time series or other orthogonal functions. For every set of values y_i, x_i we can then generalize the equations to

$$
\begin{aligned}
y_0 &= \beta_0 x_{00} + \beta_1 x_{01} + \beta_2 x_{02} + \dots + \beta_{n-1} x_{0n-1} + \epsilon_0 \\
y_1 &= \beta_0 x_{10} + \beta_1 x_{11} + \beta_2 x_{12} + \dots + \beta_{n-1} x_{1n-1} + \epsilon_1 \\
y_2 &= \beta_0 x_{20} + \beta_1 x_{21} + \beta_2 x_{22} + \dots + \beta_{n-1} x_{2n-1} + \epsilon_2 \\
&\dots\dots \\
y_i &= \beta_0 x_{i0} + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_{n-1} x_{in-1} + \epsilon_i \\
&\dots\dots \\
y_{n-1} &= \beta_0 x_{n-1,0} + \beta_1 x_{n-1,2} + \beta_2 x_{n-1,2} + \dots + \beta_1 x_{n-1,n-1} + \epsilon_{n-1}.
\end{aligned}
$$

Generalizing the fitting procedure as a linear algebra problem

We redefine in turn the matrix $\hat{X}$ as

$$
\hat{X} = \begin{bmatrix}
x_{00} & x_{01} & x_{02} & \dots & \dots & x_{0,n-1} \\
x_{10} & x_{11} & x_{12} & \dots & \dots & x_{1,n-1} \\
x_{20} & x_{21} & x_{22} & \dots & \dots & x_{2,n-1} \\
\dots & \dots & \dots & \dots & \dots & \dots \\
x_{n-1,0} & x_{n-1,1} & x_{n-1,2} & \dots & \dots & x_{n-1,n-1}
\end{bmatrix}
$$

and without loss of generality we rewrite again our equations as

$$
\hat{y} = \hat{X}\hat{\beta} + \hat{\epsilon}.
$$

The left-hand side of this equation forms know. Our error vector $\hat{\epsilon}$ and the parameter vector $\hat{\beta}$ are our unknow quantities. How can we obtain the optimal set of β_i values?

Optimizing our parameters

We have defined the matrix $\hat{X}$

$$
\begin{aligned}
y_0 &= \beta_0 x_{00} + \beta_1 x_{01} + \beta_2 x_{02} + \dots + \beta_{n-1} x_{0n-1} + \epsilon_0 \\
y_1 &= \beta_0 x_{10} + \beta_1 x_{11} + \beta_2 x_{12} + \dots + \beta_{n-1} x_{1n-1} + \epsilon_1 \\
y_2 &= \beta_0 x_{20} + \beta_1 x_{21} + \beta_2 x_{22} + \dots + \beta_{n-1} x_{2n-1} + \epsilon_1 \\
&\dots\dots \\
y_i &= \beta_0 x_{i0} + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_{n-1} x_{in-1} + \epsilon_1 \\
&\dots\dots \\
y_{n-1} &= \beta_0 x_{n-1,0} + \beta_1 x_{n-1,2} + \beta_2 x_{n-1,2} + \dots + \beta_1 x_{n-1,n-1} + \epsilon_{n-1}.
\end{aligned}
$$

Optimizing our parameters, more details

We well use this matrix to define the approximation $\hat{\tilde{y}}$ via the unknown quantity $\hat{\beta}$ as

$$
\hat{\tilde{y}} = \hat{X}\hat{\beta},
$$

and in order to find the optimal parameters β_i instead of solving the above linear algebra problem, we define a function which gives a measure of the spread between the values y_i (which represent hopefully the exact values) and the parametrized values $\tilde{y}_i$, namely

$$
Q(\hat{\beta}) = \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2 = \left(\hat{y} - \hat{\tilde{y}}\right)^T \left(\hat{y} - \hat{\tilde{y}}\right),
$$

or using the matrix $\hat{X}$ as

$$
Q(\hat{\beta}) = \left(\hat{y} - \hat{X}\hat{\beta}\right)^T \left(\hat{y} - \hat{X}\hat{\beta}\right).
$$

Interpretations and optimizing our parameters

The function

$$
Q(\hat{\beta}) = \left(\hat{y} - \hat{X}\hat{\beta}\right)^T \left(\hat{y} - \hat{X}\hat{\beta}\right),
$$

can be linked to the variance of the quantity y_i if we interpret the latter as the mean value of for example a numerical experiment. When linking below with the maximum likelihood approach below, we will indeed interpret y_i as a mean value

$$
y_i = \langle y_i \rangle = \beta_0 x_{i,0} + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \dots + \beta_{n-1} x_{i,n-1} + \epsilon_i,
$$

where $\langle y_i \rangle$ is the mean value. Keep in mind also that till now we have treated y_i as the exact value. Normally, the response (dependent or outcome) variable y_i the outcome of a numerical experiment or another type of experiment and is thus only an approximation to the true value. It is then always accompanied by an error estimate, often limited to a statistical error estimate given by the standard deviation discussed earlier. In the discussion here we will treat y_i as our exact value for the response variable.

Interpretations and optimizing our parameters

We can rewrite

$$
\frac{\partial Q(\hat{\beta})}{\partial \hat{\beta}} = 0 = \hat{X}^T \left(\hat{y} - \hat{X}\hat{\beta}\right),
$$

as

$$
\hat{X}^T\hat{y} = \hat{X}^T\hat{X}\hat{\beta},
$$

and if the matrix $\hat{X}^T\hat{X}$ is invertible we have the solution

$$
\hat{\beta} = \left(\hat{X}^T\hat{X}\right)^{-1}\hat{X}^T\hat{y}.
$$

Interpretations and optimizing our parameters

The residuals $\hat{\epsilon}$ are in turn given by

$$\hat{\epsilon} = \hat{y} - \hat{\tilde{y}} = \hat{y} - \hat{X}\hat{\beta},$$

and with

$$\hat{X}^T\left(\hat{y} - \hat{X}\hat{\beta}\right) = 0,$$

we have

$$\hat{X}^T\hat{\epsilon} = \hat{X}^T\left(\hat{y} - \hat{X}\hat{\beta}\right) = 0,$$

meaning that the solution for $\hat{\beta}$ is the one which minimizes the residuals. Later we will link this with the maximum likelihood approach.

The χ^2 function

Normally, the response (dependent or outcome) variable y_i the outcome of a numerical experiment or another type of experiment and is thus only an approximation to the true value. It is then always accompanied by an error estimate, often limited to a statistical error estimate given by the standard deviation discussed earlier. In the discussion here we will treat y_i as our exact value for the response variable.

Introducing the standard deviation σ_i for each measurement y_i, we define now the χ^2 function as

$$\chi^2(\hat{\beta}) = \sum_{i=0}^{n-1}\frac{(y_i - \tilde{y}_i)^2}{\sigma_i^2} = \left(\hat{y} - \hat{\tilde{y}}\right)^T\frac{1}{\hat{\Sigma}^2}\left(\hat{y} - \hat{\tilde{y}}\right),$$

where the matrix $\hat{\Sigma}$ is a diagonal matrix with σ_i as matrix elements.

The χ^2 function

In order to find the parameters β_i we will then minimize the spread of $\chi^2(\hat{\beta})$ by requiring

$$\frac{\partial \chi^2(\hat{\beta})}{\partial \beta_j} = \frac{\partial}{\partial \beta_j}\left[\sum_{i=0}^{n-1}\left(\frac{y_i - \beta_0 x_{i,0} - \beta_1 x_{i,1} - \beta_2 x_{i,2} - \dots - \beta_{n-1}x_{i,n-1}}{\sigma_i}\right)^2\right]$$

which results in

$$\frac{\partial \chi^2(\hat{\beta})}{\partial \beta_j} = -2\left[\sum_{i=0}^{n-1}\frac{x_{ij}}{\sigma_i}\left(\frac{y_i - \beta_0 x_{i,0} - \beta_1 x_{i,1} - \beta_2 x_{i,2} - \dots - \beta_{n-1}x_{i,n-1}}{\sigma_i}\right)\right]$$

or in a matrix-vector form as

$$\frac{\partial \chi^2(\hat{\beta})}{\partial \hat{\beta}} = 0 = \hat{A}^T\left(\hat{b} - \hat{A}\hat{\beta}\right).$$

where we have defined the matrix $\hat{A} = \hat{X}/\hat{\Sigma}$ with matrix elements $a_{ij} = x_{ij}/\sigma_i$ and the vector $\hat{b}$ with elements $b_i = y_i/\sigma_i$.

The χ^2 function

We can rewrite

$$\frac{\partial \chi^2(\hat{\beta})}{\partial \hat{\beta}} = 0 = \hat{A}^T\left(\hat{b} - \hat{A}\hat{\beta}\right),$$

as

$$\hat{A}^T\hat{b} = \hat{A}^T\hat{A}\hat{\beta},$$

and if the matrix $\hat{A}^T\hat{A}$ is invertible we have the solution

$$\hat{\beta} = \left(\hat{A}^T\hat{A}\right)^{-1}\hat{A}^T\hat{b}.$$

The χ^2 function

If we then introduce the matrix

$$\hat{H} = \left(\hat{A}^T\hat{A}\right)^{-1},$$

we have then the following expression for the parameters β_j (the matrix elements of $\hat{H}$ are h_{ij})

$$\beta_j = \sum_{k=0}^{p-1}h_{jk}\sum_{i=0}^{n-1}\frac{y_i}{\sigma_i}\frac{x_{ik}}{\sigma_i} = \sum_{k=0}^{p-1}h_{jk}\sum_{i=0}^{n-1}b_i a_{ik}$$

We state without proof the expression for the uncertainty in the parameters β_j as (we leave this as an exercise)

$$\sigma^2(\beta_j) = \sum_{i=0}^{n-1}\sigma_i^2\left(\frac{\partial \beta_j}{\partial y_i}\right)^2,$$

resulting in

The χ^2 function

The first step here is to approximate the function y with a first-order polynomial, that is we write

$$y = y(x) \to y(x_i) \approx \beta_0 + \beta_1 x_i.$$

By computing the derivatives of χ^2 with respect to β_0 and β_1 show that these are given by

$$\frac{\partial \chi^2(\hat{\beta})}{\partial \beta_0} = -2\left[\sum_{i=0}^{n-1}\left(\frac{y_i - \beta_0 - \beta_1 x_i}{\sigma_i^2}\right)\right] = 0,$$

and

$$\frac{\partial \chi^2(\hat{\beta})}{\partial \beta_0} = -2\left[\sum_{i=0}^{n-1}x_i\left(\frac{y_i - \beta_0 - \beta_1 x_i}{\sigma_i^2}\right)\right] = 0.$$

The χ^2 function

For a linear fit we don't need to invert a matrix!! Defining

$$\gamma = \sum_{i=0}^{n-1} \frac{1}{\sigma_i^2},$$

$$\gamma_x = \sum_{i=0}^{n-1} \frac{x_i}{\sigma_i^2},$$

$$\gamma_y = \sum_{i=0}^{n-1} \left(\frac{y_i}{\sigma_i^2}\right),$$

$$\gamma_{xx} = \sum_{i=0}^{n-1} \frac{x_i x_i}{\sigma_i^2},$$

$$\gamma_{xy} = \sum_{i=0}^{n-1} \frac{y_i x_i}{\sigma_i^2},$$

we obtain

Simple regression model

We are now ready to write our first program which aims at solving the above linear regression equations. We start with data we have produced ourselves, in this case normally distributed random numbers along the $x$-axis. These numbers define then the value of a function $y(x) = 4 + 3x + N(0, 1)$. Thereafter we order the $x$ values and employ our linear regression algorithm to set up the best fit. Here we find it useful to use the numpy function $c\_$ arrays where arrays are stacked along their last axis after being upgraded to at least two dimensions with ones post-pended to the shape. The following examples help in understanding what happens

```
import numpy as np
print(np.c_[np.array([1,2,3]), np.array([4,5,6])])
print(np.c_[np.array([[1,2,3]]), 0, 0, np.array([[4,5,6]])])

# Importing various packages
from random import random, seed
import numpy as np
import matplotlib.pyplot as plt

x = 2*np.random.rand(100,1)
y = 4+3*x+np.random.randn(100,1)
```

Simple regression model, now using **scikit-learn**

We can repeat the above algorithm using **scikit-learn** as follows

```
# Importing various packages
from random import random, seed
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

x = 2*np.random.rand(100,1)
y = 4+3*x+np.random.randn(100,1)
linreg = LinearRegression()
linreg.fit(x,y)
xnew = np.array([[0],[2]])
ypredict = linreg.predict(xnew)

plt.plot(xnew, ypredict, "r-")
plt.plot(x, y ,'ro')
plt.axis([0,2.0,0, 15.0])
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
plt.title(r'Random numbers ')
plt.show()
```

Simple linear regression model using **scikit-learn**

We start with perhaps our simplest possible example, using **scikit-learn** to perform linear regression analysis on a data set produced by us. What follows is a simple Python code where we have defined function $y$ in terms of the variable $x$. Both are defined as vectors of dimension 1×100. The entries to the vector $\hat{x}$ are given by random numbers generated with a uniform distribution with entries $x_i \in [0, 1]$ (more about probability distribution functions later). These values are then used to define a function $y(x)$ (tabulated again as a vector) with a linear dependence on $x$ plus a random noise added via the normal distribution.

The Numpy functions are imported used the **import numpy as np** statement and the random number generator for the uniform distribution is called using the function **np.random.rand()**, where we specificy that we want 100 random variables. Using Numpy we define automatically an array with the specified number of elements, 100 in our case. With the Numpy function **randn()** we can compute random numbers with the normal distribution (mean value μ equal to zero and variance σ^2 set to one) and produce the

Simple linear regression model

This example serves several aims. It allows us to demonstrate several aspects of data analysis and later machine learning algorithms. The immediate visualization shows that our linear fit is not impressive. It goes through the data points, but there are many outliers which are not reproduced by our linear regression. We could now play around with this small program and change for example the factor in front of $x$ and the normal distribution. Try to change the function $y$ to

$$y = 10x + 0.01 \times N(0, 1),$$

where $x$ is defined as before.

Less noise

Does the fit look better? Indeed, by reducing the role of the normal distribution we see immediately that our linear prediction seemingly reproduces better the training set. However, this testing 'by the eye' is obviouly not satisfactory in the long run. Here we have only defined the training data and our model, and have not discussed a more rigorous approach to the **cost** function.

How to study our fits

We need more rigorous criteria in defining whether we have succeeded or not in modeling our training data. You will be surprised to see that many scientists seldomly venture beyond this 'by the eye' approach. A standard approach for the *cost* function is the so-called χ^2 function

$$
\chi^2 = \frac{1}{n}\sum_{i=0}^{n-1}\frac{(y_i-\tilde{y}_i)^2}{\sigma_i^2},
$$

where σ_i^2 is the variance (to be defined later) of the entry y_i. We may not know the explicit value of σ_i^2, it serves however the aim of scaling the equations and make the cost function dimensionless.

Minimizing the cost function

Minimizing the cost function is a central aspect of our discussions to come. Finding its minima as function of the model parameters (α and β in our case) will be a recurring theme in these series of lectures. Essentially all machine learning algorithms we will discuss center around the minimization of the chosen cost function. This depends in turn on our specific model for describing the data, a typical situation in supervised learning. Automatizing the search for the minima of the cost function is a central ingredient in all algorithms. Typical methods which are employed are various variants of **gradient** methods. These will be discussed in more detail later. Again, you'll be surprised to hear that many practitioners minimize the above function ''by the eye', popularly dubbed as 'chi by the eye'. That is, change a parameter and see (visually and numerically) that the χ^2 function becomes smaller.

Relative error

There are many ways to define the cost function. A simpler approach is to look at the relative difference between the training data and the predicted data, that is we define the relative error as

$$
\epsilon_{\mathrm{relative}} = \frac{|\hat{y}-\hat{\tilde{y}}|}{|\hat{y}|}.
$$

We can modify easily the above Python code and plot the relative error instead

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

x = np.random.rand(100,1)
y = 5*x+0.01*np.random.randn(100,1)
linreg = LinearRegression()
linreg.fit(x,y)
ypredict = linreg.predict(x)

plt.plot(x, np.abs(ypredict-y)/abs(y), "ro")
plt.axis([0,1.0,0.0, 0.5])
plt.xlabel(r'$x$')
plt.ylabel(r'$\epsilon_{\mathrm{relative}}$')
plt.title(r'Relative error')
```

The richness of **scikit-learn**

As mentioned above, **scikit-learn** has an impressive functionality. We can for example extract the values of α and β and their error estimates, or the variance and standard deviation and many other properties from the statistical data analysis.

Here we show an example of the functionality of scikit-learn.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_squared

x = np.random.rand(100,1)
y = 2.0+ 5*x+0.5*np.random.randn(100,1)
linreg = LinearRegression()
linreg.fit(x,y)
ypredict = linreg.predict(x)
print('The intercept alpha: \n', linreg.intercept_)
print('Coefficient beta : \n', linreg.coef_)
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y, ypredict))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y, ypredict))
# Mean squared log error
print('Mean squared log error: %.2f' % mean_squared_log_error(y, ypred
# Mean absolute error
```

Functions in **scikit-learn**

The function **coef** gives us the parameter β of our fit while **intercept** yields α. Depending on the constant in front of the normal distribution, we get values near or far from $alpha = 2$ and $\beta = 5$. Try to play around with different parameters in front of the normal distribution. The function **meansquarederror** gives us the mean square error, a risk metric corresponding to the expected value of the squared (quadratic) error or loss defined as

$$
MSE(\hat{y},\hat{\tilde{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2,
$$

The smaller the value, the better the fit. Ideally we would like to have an MSE equal zero. The attentive reader has probably recognized this function as being similar to the χ^2 function defined above.

Other functions in **scikit-learn**

The **r2score** function computes R^2, the coefficient of determination. It provides a measure of how well future samples are likely to be predicted by the model. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of $\hat{y}$, disregarding the input features, would get a R^2 score of 0.0.

If $\tilde{\hat{y}}_i$ is the predicted value of the $i-th$ sample and y_i is the corresponding true value, then the score R^2 is defined as

$$
R^2(\hat{y},\tilde{\hat{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i-\bar{y})^2},
$$

where we have defined the mean value of $\hat{y}$ as

$$
\bar{y} = \frac{1}{n}\sum_{i=0}^{n-1}y_i.
$$

The mean absolute error and other functions in **scikit-learn**

Another quantity will meet again in our discussions of regression analysis is mean absolute error (MAE), a risk metric corresponding to the expected value of the absolute error loss or what we call the $l1$-norm loss. In our discussion above we presented the relative error. The MAE is defined as follows

$$\text{MAE}(\hat{y}, \hat{\tilde{y}}) = \frac{1}{n} \sum_{i=0}^{n-1} |y_i - \tilde{y}_i| .$$

Finally we present the squared logarithmic (quadratic) error

$$\text{MSLE}(\hat{y}, \hat{\tilde{y}}) = \frac{1}{n} \sum_{i=0}^{n-1} (\log_e(1 + y_i) - \log_e(1 + \tilde{y}_i))^2,$$

where $\log_e(x)$ stands for the natural logarithm of $x$. This error estimate is best to use when targets having exponential growth, such as population counts, average sales of a commodity over a span of years etc.

Cubic polynomial in **scikit-learn**

We will discuss in more detail these and other functions in the various lectures. We conclude this part with another example. Instead of a linear $x$-dependence we study now a cubic polynomial and use the polynomial regression analysis tools of scikit-learn. Add description of the various python commands.

```
import matplotlib.pyplot as plt
import numpy as np
import random
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression

x=np.linspace(0.02,0.98,200)
noise = np.asarray(random.sample((range(200)),200))
y=x**3*noise
yn=x**3*100
poly3 = PolynomialFeatures(degree=3)
X = poly3.fit_transform(x[:,np.newaxis])
clf3 = LinearRegression()
clf3.fit(X,y)

Xplot=poly3.fit_transform(x[:,np.newaxis])
poly3_plot=plt.plot(x, clf3.predict(Xplot), label='Cubic Fit')
plt.plot(x,yn, color='red', label="True Cubic")
```

Polynomial Regression

```
# Importing various packages
from math import exp, sqrt
from random import random, seed
import numpy as np
import matplotlib.pyplot as plt

m = 100
x = 2*np.random.rand(m,1)+4.
y = 4+3*x*x+ +x-np.random.randn(m,1)

xb = np.c_[np.ones((m,1)), x]
theta = np.linalg.inv(xb.T.dot(xb)).dot(xb.T).dot(y)
xnew = np.array([[0],[2]])
xbnew = np.c_[np.ones((2,1)), xnew]
ypredict = xbnew.dot(theta)

plt.plot(xnew, ypredict, "r-")
plt.plot(x, y ,'ro')
plt.axis([0,2.0,0, 15.0])
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
plt.title(r'Random numbers ')
plt.show()
```

Linking the regression analysis with a statistical interpretation

Before we proceed, and to link with our discussions of Bayesian statistics to come, it is useful the derive the standard regression analysis equations using a statistical interpretation. This allows us also to derive quantities like the variance and other expectation values in a rather straightforward way.

It is assumed that $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ and the ε_i are independent, i.e.:

$$\text{Cov}(\varepsilon_{i_1}, \varepsilon_{i_2}) = \left\{ \begin{array}{lcc} \sigma^2 & \text{if} & i_1 = i_2, \\ 0 & \text{if} & i_1 \neq i_2. \end{array} \right.$$

The randomness of ε_i implies that $\mathbf{Y}_i$ is also a random variable. In particular, $\mathbf{Y}_i$ is normally distributed, because $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ and $\mathbf{X}_{i,\ast} \, \beta$ is a non-random scalar. To specify the parameters of the distribution of $\mathbf{Y}_i$ we need to calculate its first two moments.

Expectation value and variance

Its expectation equals:

$$\mathbb{E}(Y_i) = \mathbb{E}(\mathbf{X}_{i,\ast} \, \beta) + \mathbb{E}(\varepsilon_i) \quad = \quad \mathbf{X}_{i,\ast} \, \beta,$$

while its variance is

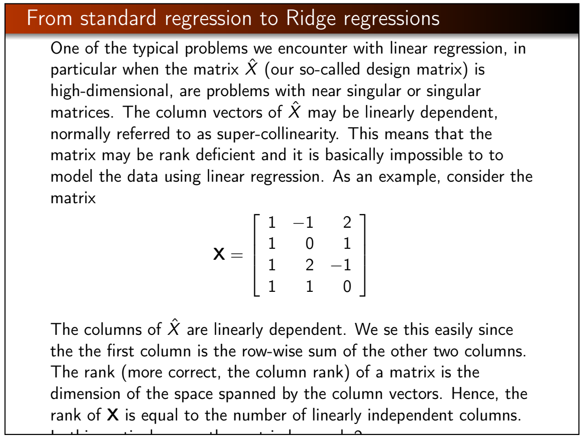
$$\begin{aligned} \text{Var}(Y_i) = \mathbb{E}\{[Y_i - \mathbb{E}(Y_i)]^2\} & = \mathbb{E}(Y_i^2) - [\mathbb{E}(Y_i)]^2 \\ = \mathbb{E}[(\mathbf{X}_{i,\ast} \, \beta + \varepsilon_i)^2] - (\mathbf{X}_{i,\ast} \, \beta)^2 & \\ = \mathbb{E}[(\mathbf{X}_{i,\ast} \, \beta)^2 + 2 \varepsilon_i \mathbf{X}_{i,\ast} \, \beta + \varepsilon_i^2] - (\mathbf{X}_{i,\ast} \, \beta)^2 & \\ = (\mathbf{X}_{i,\ast} \, \beta)^2 + 2 \mathbb{E}(\varepsilon_i) \mathbf{X}_{i,\ast} \, \beta + \mathbb{E}(\varepsilon_i^2) - (\mathbf{X}_{i,\ast} \, \beta)^2 & \\ = \mathbb{E}(\varepsilon_i^2) \quad = \quad \text{Var}(\varepsilon_i) & = \sigma^2. \end{aligned}$$

Hence, $Y_i \sim \mathcal{N}(\mathbf{X}_{i,\ast} \, \beta, \sigma^2)$.

The singular value decompostion

A general $m \times n$ matrix $\hat{A}$ can be written in terms of a diagonal matrix $\hat{D}$ of dimensionality $n \times n$ and two orthognal matrices $\hat{U}$ and $\hat{V}$, where the first has dimensionality $m \times m$ and the last dimensionality $n \times n$. We have then

$$\hat{A} = \hat{U} \hat{D} \hat{V}^T$$

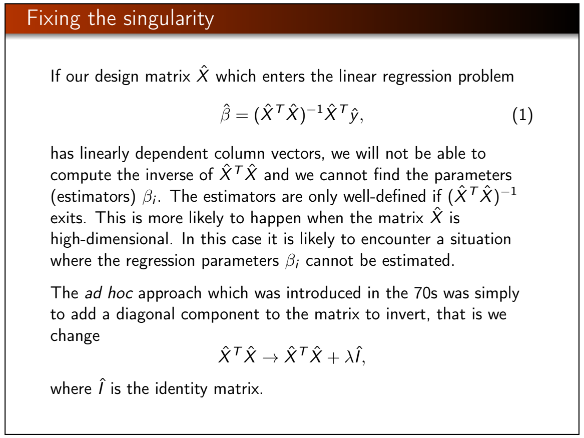


From standard regression to Ridge regressions

One of the typical problems we encounter with linear regression, in particular when the matrix $\hat{X}$ (our so-called design matrix) is high-dimensional, are problems with near singular or singular matrices. The column vectors of $\hat{X}$ may be linearly dependent, normally referred to as super-collinearity. This means that the matrix may be rank deficient and it is basically impossible to to model the data using linear regression. As an example, consider the matrix

$$\mathbf{X} = \begin{bmatrix} 1 & -1 & 2 \\ 1 & 0 & 1 \\ 1 & 2 & -1 \\ 1 & 1 & 0 \end{bmatrix}$$

The columns of $\hat{X}$ are linearly dependent. We se this easily since the the first column is the row-wise sum of the other two columns. The rank (more correct, the column rank) of a matrix is the dimension of the space spanned by the column vectors. Hence, the rank of $\mathbf{X}$ is equal to the number of linearly independent columns. In this particular case the matrix has rank 2.



Fixing the singularity

If our design matrix $\hat{X}$ which enters the linear regression problem

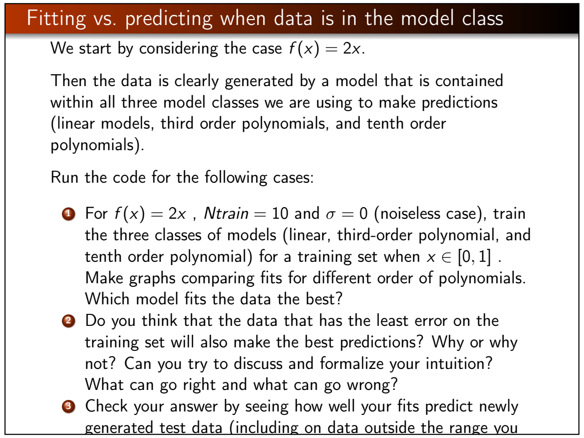
$$\hat{\beta} = (\hat{X}^T\hat{X})^{-1}\hat{X}^T\hat{y}, \tag{1}$$

has linearly dependent column vectors, we will not be able to compute the inverse of $\hat{X}^T\hat{X}$ and we cannot find the parameters (estimators) β_i. The estimators are only well-defined if $(\hat{X}^T\hat{X})^{-1}$ exits. This is more likely to happen when the matrix $\hat{X}$ is high-dimensional. In this case it is likely to encounter a situation where the regression parameters β_i cannot be estimated.

The *ad hoc* approach which was introduced in the 70s was simply to add a diagonal component to the matrix to invert, that is we change

$$\hat{X}^T\hat{X} \rightarrow \hat{X}^T\hat{X} + \lambda\hat{I},$$

where $\hat{I}$ is the identity matrix.



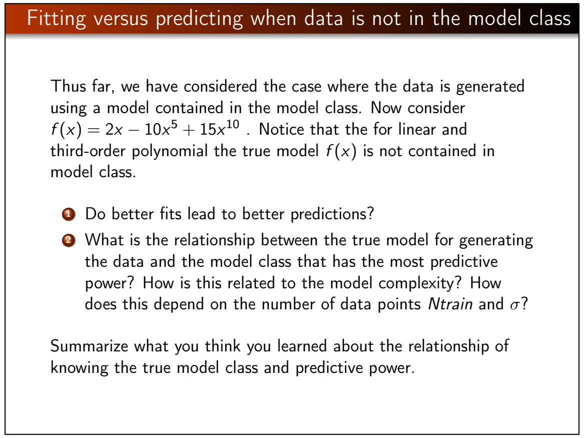
Fitting vs. predicting when data is in the model class

We start by considering the case $f(x) = 2x$.

Then the data is clearly generated by a model that is contained within all three model classes we are using to make predictions (linear models, third order polynomials, and tenth order polynomials).

Run the code for the following cases:

1. For $f(x) = 2x$, $Ntrain = 10$ and $\sigma = 0$ (noiseless case), train the three classes of models (linear, third-order polynomial, and tenth order polynomial) for a training set when $x \in [0, 1]$. Make graphs comparing fits for different order of polynomials. Which model fits the data the best?
2. Do you think that the data that has the least error on the training set will also make the best predictions? Why or why not? Can you try to discuss and formalize your intuition? What can go right and what can go wrong?
3. Check your answer by seeing how well your fits predict newly generated test data (including on data outside the range you

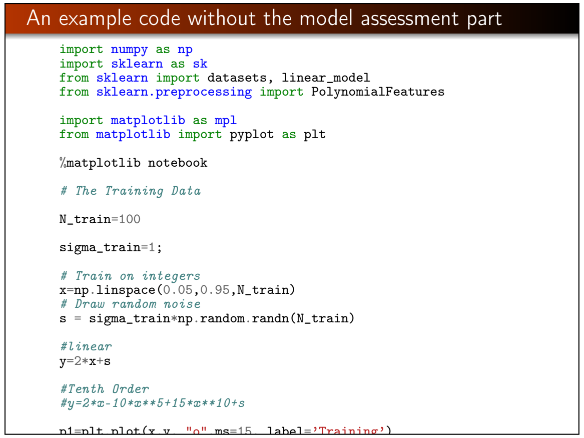


Fitting versus predicting when data is not in the model class

Thus far, we have considered the case where the data is generated using a model contained in the model class. Now consider $f(x) = 2x - 10x^5 + 15x^{10}$. Notice that the for linear and third-order polynomial the true model $f(x)$ is not contained in model class.

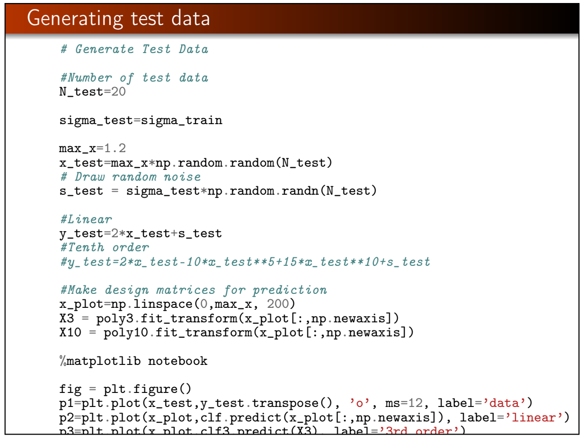
1. Do better fits lead to better predictions?
2. What is the relationship between the true model for generating the data and the model class that has the most predictive power? How is this related to the model complexity? How does this depend on the number of data points $Ntrain$ and σ?

Summarize what you think you learned about the relationship of knowing the true model class and predictive power.



An example code without the model assessment part

```
import numpy as np
import sklearn as sk
from sklearn import datasets, linear_model
from sklearn.preprocessing import PolynomialFeatures

import matplotlib as mpl
from matplotlib import pyplot as plt

%matplotlib notebook

# The Training Data

N_train=100

sigma_train=1;

# Train on integers
x=np.linspace(0.05,0.95,N_train)
# Draw random noise
s = sigma_train*np.random.randn(N_train)

#linear
y=2*x+s

#Tenth Order
#y=2*x-10*x**5+15*x**10+s

p1=plt.plot(x,y, "o",ms=15, label='Training')
```



Generating test data

```
# Generate Test Data

#Number of test data
N_test=20

sigma_test=sigma_train

max_x=1.2
x_test=max_x*np.random.random(N_test)
# Draw random noise
s_test = sigma_test*np.random.randn(N_test)

#Linear
y_test=2*x_test+s_test
#Tenth order
#y_test=2*x_test-10*x_test**5+15*x_test**10+s_test

#Make design matrices for prediction
x_plot=np.linspace(0,max_x, 200)
X3 = poly3.fit_transform(x_plot[:,np.newaxis])
X10 = poly10.fit_transform(x_plot[:,np.newaxis])

%matplotlib notebook

fig = plt.figure()
p1=plt.plot(x_test,y_test.transpose(), 'o', ms=12, label='data')
p2=plt.plot(x_plot,clf.predict(x_plot[:,np.newaxis]), label='linear')
p3=plt.plot(x_plot,clf3.predict(X3), label='3rd order')
```

How can we effectively evaluate the various models?

In Ridge regression and the subsequent discussion of its properties the bias or penalty parameter is considered known or 'given'. In practice, it is unknown and the user needs to make an informed decision on its value. How do we do that? Much of the same considerations apply to the Lasso method.

Code examples for Ridge and Lasso Regression

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

#creating data with random noise
x=np.arange(50)

delta=np.random.uniform(-2.5,2.5, size=(50))
np.random.shuffle(delta)
y =0.5*x+5+delta

#arranging data into 2x50 matrix
a=np.array(x) #inputs
b=np.array(y) #outputs

#Split into training and test
X_train=a[:37, np.newaxis]
X_test=a[37:, np.newaxis]
y_train=b[:37]
y_test=b[37:]

print ("X_train: ", X_train.shape)
print ("y_train: ", y_train.shape)
print ("X_test: ", X_test.shape)
print ("y_test: ", y_test.shape)
```

A second-order polynomial with Ridge and Lasso

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score

np.random.seed(4155)

n_samples = 100

x = np.random.rand(n_samples,1)
y = 5*x*x + 0.1*np.random.rand(n_samples,1)

# Centering  x and y.
x_ = x - np.mean(x)
y_ = y - np.mean(y) # beta_0 = mean(y)

X = np.c_[np.ones((n_samples,1)), x, x**2]
X_ = np.c_[x_, x_**2]


### 1.
lmb_values = [1e-4, 1e-3, 1e-2, 10, 1e2, 1e4]
num_values = len(lmb_values)

## Ridge-regression of centered and not centered data
beta_ridge = np.zeros((3,num_values))
beta_ridge_centered = np.zeros((3,num_values))
```

Resampling methods

Resampling methods are an indispensable tool in modern statistics. They involve repeatedly drawing samples from a training set and refitting a model of interest on each sample in order to obtain additional information about the fitted model. For example, in order to estimate the variability of a linear regression fit, we can repeatedly draw different samples from the training data, fit a linear regression to each new sample, and then examine the extent to which the resulting fits differ. Such an approach may allow us to obtain information that would not be available from fitting the model only once using the original training sample.

Resampling approaches can be computationally expensive

Resampling approaches can be computationally expensive, because they involve fitting the same statistical method multiple times using different subsets of the training data. However, due to recent advances in computing power, the computational requirements of resampling methods generally are not prohibitive. In this chapter, we discuss two of the most commonly used resampling methods, cross-validation and the bootstrap. Both methods are important tools in the practical application of many statistical learning procedures. For example, cross-validation can be used to estimate the test error associated with a given statistical learning method in order to evaluate its performance, or to select the appropriate level of flexibility. The process of evaluating a model's performance is known as model assessment, whereas the process of selecting the proper level of flexibility for a model is known as model selection. The bootstrap is widely used.

Log-likelihood

A popular strategy is to choose a penalty parameter that yields a good but parsimonious model. Information criteria measure the balance between model fit and model complexity. One possibility is Aikaike's information criterion (AIC). The AIC measures model fit by the log-likelihood and model complexity is measured by the number of parameters used by the model. The number of model parameters in regular regression simply corresponds to the number of covariates in the model. Or, by the degrees of freedom consumed by the model, which is equivalent to the trace of the hat matrix. For ridge regression it thus seems natural to define model complexity analogously by the trace of the ridge hat matrix. This yields the AIC for the linear regression model with ridge estimates:

$$
\begin{aligned}
\mathrm{AIC}(\lambda) &= 2\,p - 2\log(\hat{L}) \\
&= 2\,\mathrm{tr}[\mathbf{H}(\lambda)] - 2\log\{L[\hat{\beta}(\lambda), \hat{\sigma}^2(\lambda)]\} \\
&= 2\sum^{p} \frac{d_{jj}^2}{d_{jj}^2 + \lambda} + 2n\,\log[\sqrt{2\,\pi}\,\hat{\sigma}(\lambda)] + \frac{1}{\hat{\sigma}^2(\lambda)}\sum^{n}[y_i - \mathbf{X}_{i,*}\,\hat{\beta}(\lambda)]^2
\end{aligned}
$$

Cross-validation

Instead of choosing the penalty parameter to balance model fit with model complexity, cross-validation requires it (i.e. the penalty parameter) to yield a model with good prediction performance. Commonly, this performance is evaluated on novel data. Novel data need not be easy to come by and one has to make do with the data at hand. The setting of 'original' and novel data is then mimicked by sample splitting: the data set is divided into two (groups of samples). One of these two data sets, called the *training set*, plays the role of 'original' data on which the model is built. The second of these data sets, called the *test set*, plays the role of the 'novel' data and is used to evaluate the prediction performance (often operationalized as the log-likelihood or the prediction error or its square or the R2 score) of the model built on the training data set. This procedure (model building and prediction evaluation on training and test set, respectively) is done for a collection of possible penalty parameter choices. The penalty parameter that yields the model with the best prediction performance is to be preferred. The thus obtained performance evaluation depends on the actual split of the data set. To remove this dependence the

Computationally expensive

The validation set approach is conceptually simple and is easy to implement. But it has two potential drawbacks:

- The validation estimate of the test error rate can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set.
- In the validation approach, only a subset of the observations, those that are included in the training set rather than in the validation set are used to fit the model. Since statistical methods tend to perform worse when trained on fewer observations, this suggests that the validation set error rate may tend to overestimate the test error rate for the model fit on the entire data set.

Various steps in cross-validation

When the repetitive splitting of the data set is done randomly, samples may accidently end up in a fast majority of the splits in either training or test set. Such samples may have an unbalanced influence on either model building or prediction evaluation. To avoid this k-fold cross-validation structures the data splitting. The samples are divided into k more or less equally sized exhaustive and mutually exclusive subsets. In turn (at each split) one of these subsets plays the role of the test set while the union of the remaining subsets constitutes the training set. Such a splitting warrants a balanced representation of each sample in both training and test set over the splits. Still the division into the k subsets involves a degree of randomness. This may be fully excluded when choosing $k = n$. This particular case is referred to as leave-one-out cross-validation (LOOCV).

How to set up the cross-validation for Ridge and/or Lasso

1. Define a range of interest for the penalty parameter.
2. Divide the data set into training and test set comprising samples $\{1, \ldots, n\} \setminus i$ and $\{i\}$, respectively.
3. Fit the linear regression model by means of ridge estimation for each λ in the grid using the training set, and the corresponding estimate of the error variance $\hat{\sigma}^2_{-i}(\lambda)$, as

$$\hat{\beta}_{-i}(\lambda) = (\hat{X}^{\top}_{-i,*}\hat{X}_{-i,*} + \lambda\hat{I}_{pp})^{-1}\hat{X}^{\top}_{-i,*}\hat{y}_{-i}$$

1. Evaluate the prediction performance of these models on the test set by $\log\{L[y_i, \hat{X}_{i,*}; \hat{\beta}_{-i}(\lambda), \hat{\sigma}^2_{-i}(\lambda)]\}$. Or, by the prediction error $|y_i - \hat{X}_{i,*}\hat{\beta}_{-i}(\lambda)|$, the relative error, the error squared or the R2 score function.
2. Repeat steps 1) to 3) such that each sample plays the role of the test set once.
3. Average the prediction performances of the test sets at each grid point of the penalty bias/parameter by computing the *cross-validated log-likelihood*. It is an estimate of the

Predicted Residual Error Sum of Squares

Another approach in the LOOCV scheme is to the use the so-called Predicted Residual Error Sum of Squares (PRESS).
We can define the optimal penalty parameter to minimize

$$\lambda_{\mathrm{opt}} = \arg\min_{\lambda} \frac{1}{n}\sum_{i=1}^{n}[y_i - \hat{X}_{i,*}\hat{\beta}_{-i}(\lambda)]^2.$$

The LOOCV prediction performance can be expressed analytically in terms of the known quantities derived from the design matrix and the parameters β.

Bootstrap

Bootstrapping is a nonparametric approach to statistical inference that substitutes computation for more traditional distributional assumptions and asymptotic results. Bootstrapping offers a number of advantages:

1. The bootstrap is quite general, although there are some cases in which it fails.
2. Because it does not require distributional assumptions (such as normally distributed errors), the bootstrap can provide more accurate inferences when the data are not well behaved or when the sample size is small.
3. It is possible to apply the bootstrap to statistics with sampling distributions that are difficult to derive, even asymptotically.
4. It is relatively simple to apply the bootstrap to complex data-collection plans (such as stratified and clustered samples).