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]

May 30, 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,

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

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

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

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

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

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

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 $\hat{X}$

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

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.

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()
```

Correlations and the quality of our results

In order to test the quality of our fit, there are several measures which can be implemented. One is the so-called correlation function defined as

$$\mathrm{Corr}(X, Y) = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

Another quantity is the autocorrelation function we discussed in our chapter on statistical analysis. Let us now try to assess the quality of our fit by studying various measures.

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

$$\hat{H} = \hat{A}^T\hat{A},$$

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

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

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

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

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

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

and

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

The χ^2 function

We define then

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

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

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

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

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

and show that

Simple regression model with gradient descent

Add info about the equations, play around with different learning rates

```python
# Importing various packages
from math import exp, sqrt
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)

xb = np.c_[np.ones((100,1)), x]
theta_linreg = np.linalg.inv(xb.T.dot(xb)).dot(xb.T).dot(y)
print(theta_linreg)
theta = np.random.randn(2,1)

eta = 0.1
Niterations = 1000
m = 100

for iter in range(Niterations):
    gradients = 2.0/m*xb.T.dot(xb.dot(theta)-y)
    theta -= eta*gradients

print(theta)
xnew = np.array([[0],[2]])
```

Simple regression model with stochastic gradient descent

Add info about the equations, play around with different learning rates

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

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

xb = np.c_[np.ones((100,1)), x]
theta_linreg = np.linalg.inv(xb.T.dot(xb)).dot(xb.T).dot(y)
print(theta_linreg)
sgdreg = SGDRegressor(n_iter = 50, penalty=None, eta0=0.1)
sgdreg.fit(x,y.ravel())
print(sgdreg.intercept_, sgdreg.coef_)
```

Polynomial Regression

```python
# 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()
```

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)
```

The singular value decompostion

How can we use the singular value decomposition to find the parameters β_j? More details will come. We first note that a general $m\times n$ matrix $\hat{A}$ can be written in terms of a diagonal matrix $\hat{\Sigma}$ of dimensionality $n\times n$ and two orthognal matrices $\hat{U}$ and $\hat{V}$, where the first has dimensionality $m\times n$ and the last dimensionality $n\times n$. We have then

$$\hat{A} = \hat{U}\hat{\Sigma}\hat{V}$$