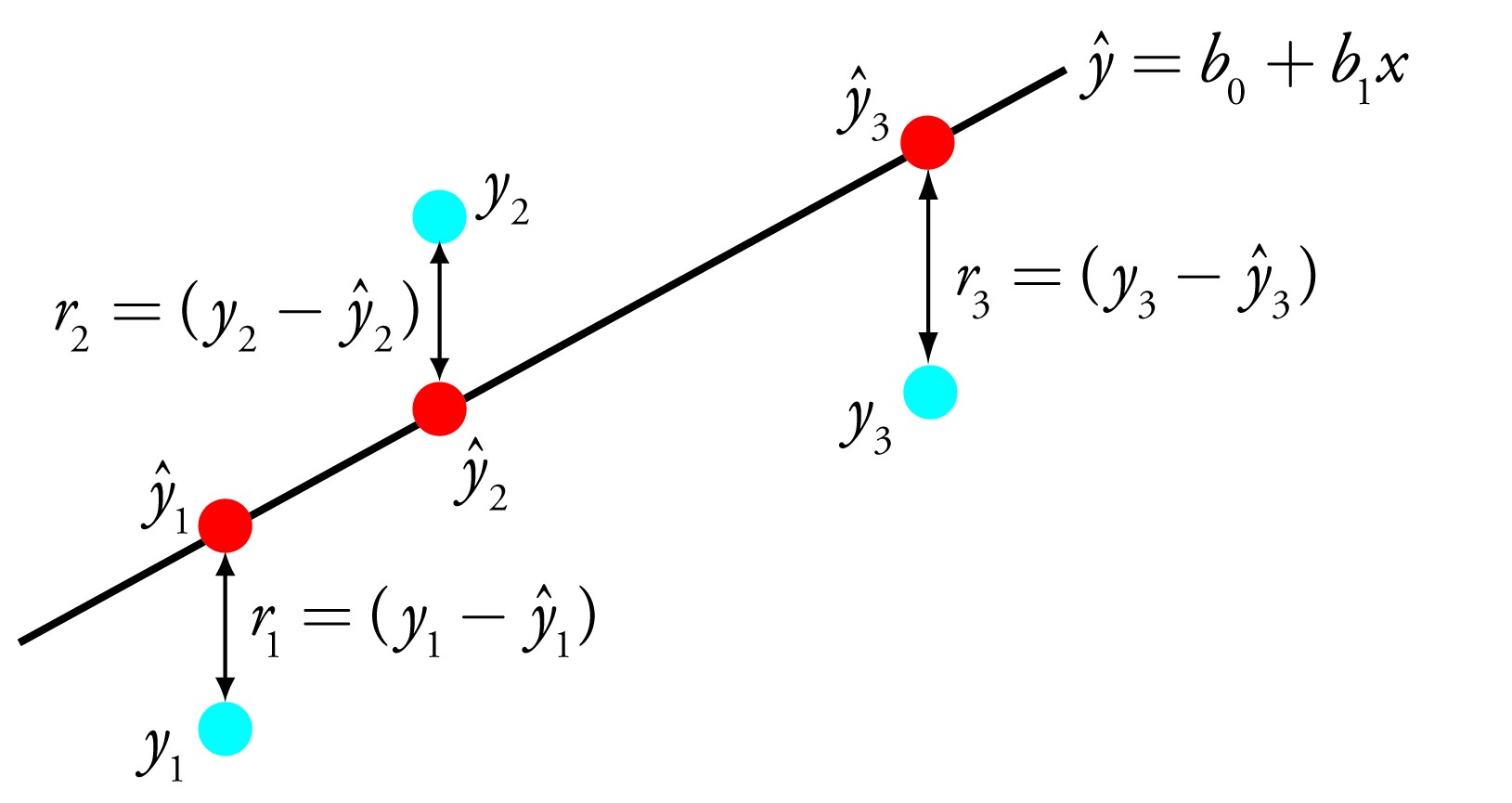
# Prediction Through Regression

## Linear Regression

When we say that the variable can be predicted by the variable , what we are saying is that if we know the value of , we can also know the value of . This means that is some deterministic function of . In many problems, we assume a linear relationship between and .

Frequently, we infer this relationship between and from a set of data points , as shown below.



If we suspect that the relationship between and is linear, there remains many ways to draw a straight line through the data points. However, we assert that these straight lines are not all equally good, and there is one best straight line that is better than all other straight lines. For example, this best straight line can be the one that minimizes the sum of square errors .

Sometimes the best straight line is still not good enough (for example, if is a curve). To check how good a straight line fits the data points, we look at the *coefficient of determination*

where

is the mean of the dependent variable . The closer is to 1, the better the straight line fit.

## Linear Regression in Python

To see how we can perform linear regression in Python, we first import numpy and matplotlib. Then,

x = np.arange(10)

y = np.zeros(10)

for n in range(10):

y[n] = x[n] + 0.5\*np.random.randn()

plt.plot(x, y, ‘o’)

plt.show()



The simplest way to find the best straight line is to use the polyfit() function in numpy, i.e.

np.polyfit(x, y, 1)

which gives array([ 1.01361714, 0.16539255]) as the output. The first element in this output array is the slope , whereas the second element in this output array is the intercept .

If we capture the output of polyfit using

pfit = np.polyfit(x, y, 1)

we can draw the best-fit straight line on top of the data points using

xx = np.linspace(0, 10, 100)

yy = np.polyval(pfit, xx)

plt.plot(x, y, ‘o’)

plt.plot(xx, yy, ‘r’)

plt.xlabel(‘x’)

plt.ylabel(‘y’)

plt.show()



Unfortunately, numpy’s polyfit() function does not give us the coefficient of determination . Therefore, we resort to the linregress() function in scipy.stats, by:

from scipy.stats import linregress

slope, intercept, r\_value, p\_value, std\_err = linregress(x,y)

to find slope = 1.0136171445402056, intercept = 0.16539255060594815, r\_value = 0.98471473045478686, p\_value = 2.3446720000470375e-07, and std\_err = 0.063387572003469478.

From these output values, we can compute the coefficient of determination to be . This is very close to 1, and agrees with our visual inspection that the straight line fit is good.

## Linear Regression for Variable Selection

Now let us consider how we can use linear regression to select between three variables, , such that and are informative, but is not.

x = np.concatenate((3 + 0.3\*np.random.randn(5), -2 + 0.2\*np.random.randn(5)))

y = np.concatenate((5 + 0.3\*np.random.randn(5), -1 + 0.25\*np.random.randn(5)))

z = 3\*np.random.rand(10)

When we plot against , we find that



and when we fit a best straight line (red) through the data,

slope, intercept, r\_value, p\_value, std\_err = linregress(x,y)

we find slope = 1.2957242848291199, intercept = 1.1987594980335672, and most importantly, r\_value = 0.9980483876570726. This means that the variable strongly predicts the variable .

Conversely, if we reverse the role of and ,

slope, intercept, r\_value, p\_value, std\_err = linregress(y,x)

we find that slope = 0.76875967809482559, intercept = -0.91900317605692561, and most importantly, r\_value = 0.9980483876570726. This means that the variable also strongly predicts the variable .

On the other hand, for and ,

slope, intercept, r\_value, p\_value, std\_err = linregress(x,z)

we find that slope = 0.32156352266923877, intercept = -1.1273946008564484. But most importantly, r\_value = 0.39055107436649783, which is significantly lower than 1. What we then say is that the variable does not predict the variable .

Conversely, we can try to use the variable does not predict the variables and , to obtain the r\_values 0.39055107436649783 and 0.37233382695559247 respectively.

Therefore, we can conclude that the variables and are strongly predictive of each other, but not of the variable , while the variable cannot predict the variables and . If we organize these results into a matrix, where the row represents the first variable and the column represents the second variable, we would have

R = np.zeros((3,3))

R[0,0] = 1.0

R[1,1] = 1.0

R[2,2] = 1.0

slope, intercept, r\_value, p\_value, std\_err = linregress(x,y)

R[0,1] = r\_value

slope, intercept, r\_value, p\_value, std\_err = linregress(x,z)

R[0,2] = r\_value

slope, intercept, r\_value, p\_value, std\_err = linregress(y,x)

R[1,0] = r\_value

slope, intercept, r\_value, p\_value, std\_err = linregress(y,z)

R[1,2] = r\_value

slope, intercept, r\_value, p\_value, std\_err = linregress(z,x)

R[2,0] = r\_value

slope, intercept, r\_value, p\_value, std\_err = linregress(z,y)

R[2,1] = r\_value

plt.matshow(R)

plt.xticks([0,1,2],['x','y','z'])

plt.yticks([0,1,2],['x','y','z'])

plt.colorbar()

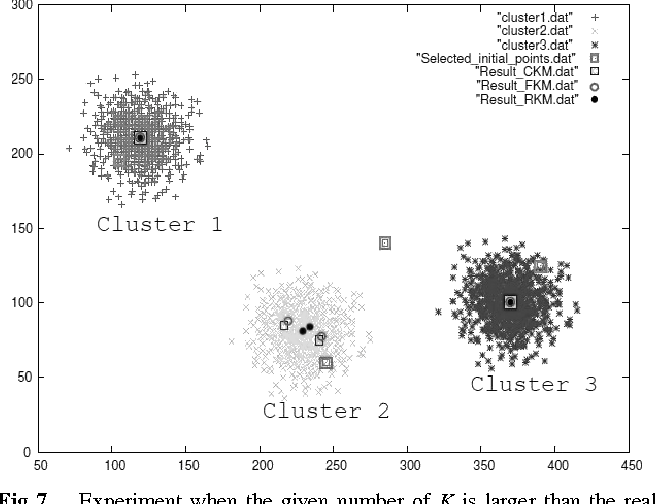
plt.show()



This confirms that the variables and are informative, but the variable is not.

## Problem with Simple Linear Regression

The simple linear regression approach to identify informative variables is too simple. A way that this can fail is when there are three clusters of points, like that shown below.



In this case, there is no straight line that we can fit through all three clusters.

We can of course try to fit a curve through all through clusters, but once we go beyond a simple linear regression, it is difficult to justify simple curves to use for the fit.

Because of these difficulties, we are tempted to think about ‘prediction’ without any ‘fits’.

Looking at the figure above, we see that cluster 1 falls between 70 and 160 along the first variable, and between 160 and 250 along the second variable, whereas cluster 2 falls between 180 and 280 along the first variable, and between 40 and 120 along the second variable. Finally, cluster 3 falls between 330 and 420 along the first variable, and between 60 and 150 along the second variable.