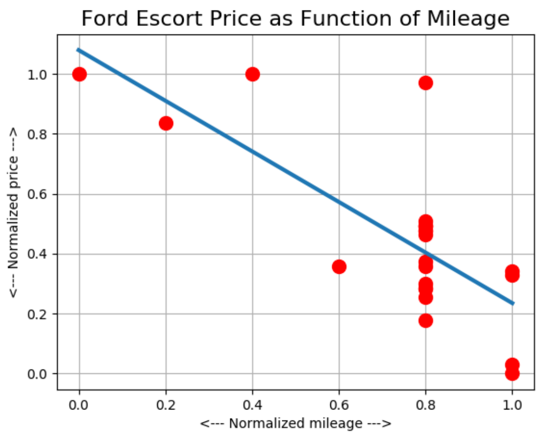
Linear Regression

MATH1900: Machine Learning

Location: http://people.sc.fsu.edu/~jburkardt/classes/ml_2019/linear_regression/linear_regression.pdf



Find coefficients for an approximating formula to data.

Linear Regression Problem

Given n sets of data, define a formula that minimizes the least squares approximation error .

1 Simple linear regression problem

In the previous class, we looked at a problem in which we wanted to fit $n = 23$ Ford Escort mileage (x) and price (y) data values with a simple linear formula:

$$y = b + m * x$$

We first normalized our data so that $0 \leq x, y \leq 1$. Then we used gradient descent to estimate the values of b and m:

```
import numpy as np
from gradient_descent2 import gradient_descent2
bm0 = np.array ( [ 0.5, 0.0 ] )
r = 0.01
dxtol = 0.001
dftol = 0.001
itmax = 1000
bm, it = gradient_descent2 ( ford_f, ford_df, bm0, r, dxtol, dftol, itmax )
```

Listing 1: Calling gradient_descent2 for the Ford data.

This calculation came up with values of b and m for the formula:

$$y = 1.04232 - 0.796815 * x$$

Since we have 23 sets of data, and a straight line could only exactly match two such sets, we must expect that our formula is only an approximation. For each data index i, we therefore have an error:

$$e_i = y_i - 1.04232 + 0.796815 * x_i$$

We choose as our error function the sum of the squares of these individual errors:

$$E = \sum_{i=1}^{23} e_i^2$$

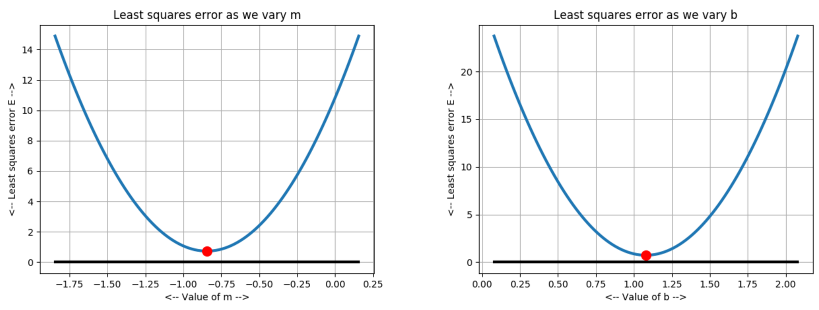
The fact that we say that b and m should be the "best" values is equivalent to the following statement:

> *For all possible values of b and m in the formula $y = b + m * x$, the values we are seeking should result in the least possible value of the sum-of-squares error E.*

2 Features of the sum-of-squares error E

When we say that b and m are the best values, that means that choosing any other values will make the error worse, at least when measured using our sum-of-squares error.

To verify this, let's look at what happens if we choose other values. A simple check is to hold one variable fixed, and vary the other. We get two plots, both of which look like parabolas, and both of which are minimized when we use the numbers we computed by gradient descent:



Our values of m and b are the best for minimizing the error.

The fact that both graphs are parabolas is not an accident. In fact, if we think of the error as a function $E(b, m)$ where both b and m are free to vary, then a plot would display a 3d shape that looks like a cup whose bottom is at the optimal values. This suggests that we could solve our linear regression problem in a different, simpler way.

The error function $E(b, m)$ is a kind of parabolic shape. It is a quadratic in each of the variables b and m. Calculus tells us that this function will be minimized when the partial derivatives $\frac{\partial E}{\partial b}$ and $\frac{\partial E}{\partial m}$ are zero.

Because E is a parabola, we have a linear system of equations to solve:

$$\begin{aligned}
E &= \sum_{i=1}^{n} (y_i - b - m * x_i)^2 \\
\frac{\partial E}{\partial b} &= -2 * \sum_{i=1}^{n} y_i - b - m * x_i \\
\frac{\partial E}{\partial m} &= -2 * \sum_{i=1}^{n} (y_i - b - m * x_i) * x_i
\end{aligned}$$

which gives us the linear system:

$$\sum_{i=1}^{n} \begin{pmatrix} 1 & x_i \\ x_i & x_i^2 \end{pmatrix} \begin{pmatrix} b \\ m \end{pmatrix} = \sum_{i=1}^{n} \begin{pmatrix} y_i \\ x_i y_i \end{pmatrix}$$

To simplify things, consider the following matrix:

$$A = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ ... & ... \\ 1 & x_n \end{pmatrix}$$

Then our linear system actually has the form

$$A' \, A \begin{pmatrix} b \\ m \end{pmatrix} = A' \, y$$

This system is known as the **normal equations**. The matrix A is easy to define; it turns out that, in general, the matrix $A'A$ will be square and nonsingular,and hence we can solve the linear system using a standard linear equation solver.

3 Example: Normal equations for the Ford data

We have two unknowns, b and m, and we have 23 data values, so our matrix A will have shape 23×2. The first column is all 1's, the second column is the x values (mileage). Our right hand side is the y values (price).

To set up our linear system, we need to compute $A'A$ as our system matrix, and $A' * y$ as our system right hand side. Here we go:

```python
def ford_normal ( ):
  import numpy as np
  from ford_data import ford_data

  x, y = ford_data ( )
  n = len ( x )

# Create matrix A
  A = np.zeros ( [ n, 2 ] )
  A[:,0] = 1;
  A[:,1] = x;

# Create system matrix A'A and right hand side A'y
```

```
  AtA = np.matmul ( A.transpose(), A )
  Aty = np.matmul ( A.transpose(), y )

#  Solve AtA * bm = Aty
  bm = np.linalg.solve ( AtA, Aty )
```

Listing 2: ford_normal.py

```
ford_normal:
  Use normal equations to estimate best b and m.

  Estimating model parameters:
  x = normalized mileage
  y = normalized price
  Seek relationship y = b + m * x
  b = 1.07923
  m = -0.844404
  Total error is 0.727866
```

Listing 3: Output from ford_normal.py

Notice that the values of b and m are somewhat different from those we computed for the gradient descent method. There is an explanation for this difference. To start with, how can we decide which of these two solutions is better? What could explain why the other solution is not as good?

4 Exercise: The China Data

The file *china_data.txt*, contains 12 pairs *year* (our x variable) and *income* (our y variable). We are looking for a relationship of the form

$$income = b + m * year$$

that allows us to "explain" the relatonship between these variables, and to predict income values are other years. Our task is to set up the linear regression system, solve it, and report the solution.

Download the data file *china_data.txt* and then try the following commands:

```
  import numpy as np

  data = np.loadtxt ( 'china_data.txt' )
  year = data[:,0]
  income = data[:,1]
  n = len ( year )

  A = np.zeros ( [ n, 2 ] )
  A[:,0] = 1.0
  A[:,1] = year

  AtA = np.matmul ( A.transpose(), A )
  Aty = np.matmul ( A.transpose(), income )

  bm = np.linalg.solve ( AtA, Aty )

  print ( 'Best formula is income = ', bm[0], ' + 'year * ', bm[1] )
```

Listing 4: Set up for China data problem

5 Example: Least squares solver for the Ford data

You may have noticed that we want to approximate the solution to the rectangular set of equations A*x = y, and that to do so using the normal equations, we always have to compute the transpose matrix A'

and multiply through to get a square system. In a better world, you might think that some function would automatically take care of those details for you.

In fact, there is a better approach entirely, called a linear least squares solver. This lets you specify your problem using the original data, and automatically works out a way to solve the system so that the sum-of-squares error is minimized. The linear least squares solver is designed to solve **any** rectangular system of linear equations $Ax = b$, choosing the solution that minimizes the sum-of-squares of the error.

The function that does this in `numpy` is `np.linalg.lstsq()`. Let's redo the Ford problem using this idea:

```
def ford_lstsq ( ):
  import numpy as np
  from ford_data import ford_data

  x, y = ford_data ( )
  n = len ( x )

# Create matrix A
  A = np.zeros ( [ n, 2 ] )
  A[:,0] = 1;
  A[:,1] = x;

#  Solve the rectangular linear system.
  bm, _, _, _ = np.linalg.lstsq ( A, y )
```

Listing 5: ford_lstsq.py

The three underscores are there because the function returns 4 items of output, but we will ignore the last 3 of them.

```
ford_lstsq:
  Use least squares solver to estimate best b and m.

  Estimating model parameters:
  x = normalized mileage
  y = normalized price
  Seek relationship y = b + m * x
  b = 1.07923
  m = -0.844404
  Total error is 0.727866
```

Listing 6: Output from ford_lstsq.py

We get the same optimal values for b and m that the normal equations approach gave us.

6 Exercise: Do Younger Players Get the Money?

The file *basketball_data.txt* contains 30 records. Each record stores values of 5 quantities for a player: number, height (centimeters), weight(pounds), sponsorship($), and age(years). It seems like young players are getting the most sponsorship dollars. To see if this is true, let's explore the possibility of a linear relationship of the form:

$$sponsorship = b + m * age$$

where age will play the role of x and sponsorship the role of y.

Download the data file *basketball_data.txt* and then try the following commands:

```
  import numpy as np

  data = np.loadtxt ( 'basketball_data.txt' )
  sponsor = data[:,3]
  age = data[:,4]
  n = len ( age )

  A = np.zeros ( [ n, 2 ] )
  A[:,0] = 1.0
  A[:,1] = age

  bm, _, _, _ = np.linalg.lstsq ( A, sponsor )

  print ( 'Best formula is sponsor = ', bm[0], ' + ', bm[1] * age' )
```

Listing 7: Set up for basketball data problem

Does your model suggest that sponsorship decreases with age?

7 Example: Studying several variables

Linear regression can be used for more complicated formulas than just $y = b + m * x$. We can ask for a good quadratic model, such as $y = c_0 + c_1 * x + c_2 * x^2$. If we have several x factors that might affect the value of the y quantity, we can ask for a formula such as $y = c_0 + c_1 * x_1 + c_2 * x_2$. Let us consider an example of this problem.

The file *insurance_data.txt* includes 1338 records about medical insurance. Each record lists, for a given person, 7 factors: **age, sex, bmi, kid, smo, reg, bil**. We seek a linear formula for the medical charges `bil`, based on the values of the first five quantities. (We ignore `reg`, which records the region of the country.)

$$bil \approx c_0 + c_1\,age + c_2\,sex + c_3\,bmi + c_4\,kid + c_5\,smo$$

```
import numpy as np
data = np.loadtxt ( 'insurance_data.txt' )

age = data[:,0]
sex = data[:,1]
bmi = data[:,2]
kid = data[:,3]
smo = data[:,4]
bil = data[:,6]

n = len ( age )

A = np.zeros ( [ n, 6 ] )
A[:,0] = np.ones ( n )
A[:,1] = age
A[:,2] = sex
A[:,3] = bmi
A[:,4] = kid
A[:,5] = smo

c, _, _, _ = np.linalg.lstsq ( A, bil, rcond = None )

print ( '  c0       = %g' % ( c[0] ) )
print ( '  c1 (age) = %g' % ( c[1] ) )
print ( '  c2 (sex) = %g' % ( c[2] ) )
print ( '  c3 (bmi) = %g' % ( c[3] ) )
print ( '  c4 (kid) = %g' % ( c[4] ) )
print ( '  c5 (smo) = %g' % ( c[5] ) )

e = np.sum ( ( bil - c[0] - c[1] * age - c[2] * sex - c[3] * bmi - c[4] * kid - c[5] * smo )
    **2 )
print ( '  Total sum-of-squares error is %g' % ( e ) )
```

Listing 8: Analyze insurance data.

```
c0       = -12052.5
c1 (age) = 257.735
c2 (sex) = -128.64
c3 (bmi) = 322.364
c4 (kid) = 474.411
c5 (smo) = 23823.4

Total sum-of-squares error is 4.9073e+10
```

Listing 9: Medical data model coefficients.

The magnitude and sign of each coefficient tells us something about the imporance of each factor. We can also experiment by looking at how good this model is for predicting a medical bill. For a patient with data age=19, sex=1, bmi=24.6, kid=1, smo=0, bil=1837.23, our formula predicts a bill of $1,120.43, off by $716.80. A patient with age=34, sex=0, bmi=31.92, kid=1,smo=1 had a bill of $37,701, and our formula predicted $31,298.20. If you have any experience of medical billing, it should be no surprise that the data is rather difficult to model accurately, but at least we have a ball park estimate.

8 Computing Assignment #6

The file *homes_data.txt* stores 50 records of home sales. Each record stores 9 separate variables.

- 0: **sel**, the selling price ($)
- 1: **ask**, asking price ($)
- 2: **liv**, living area (square feet)
- 3: **rms**, rooms (#)
- 4: **bed**, bedrooms (#)
- 5: **bat**, bathrooms (#)
- 6: **age**, age (years)
- 7: **lot**, lot size (acres)
- 8: **tax**, taxes ($)

We'd like to be able to predict the selling price based on the values of `liv`, `rms`, `age`, and `lot`, that is, a formula

$$sel \approx c_0 + c_1\, liv + c_2\, rms + c_3\, age + c_4\, lot$$

Write a Python program called *hw6.py* which

1. reads the data file,
2. computes the coefficients,
3. prints each coefficient,
4. prints the predicted selling price of a house with `liv` = 1800, `rms` = 8, `age` = 25, `lot` = 0.5

Email a copy of your program to Dr Schneier **mhs64@pitt.edu** before Wednesday, 23 October.