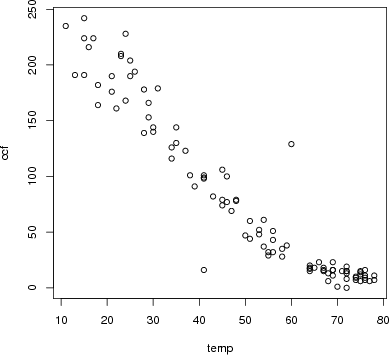
StartR Curve-Fitting and Linear Algebra

At first glance, the terms “linear” and “curve” might seem contradictory. Lines are straight, curves are not.

The word linear really refers to “linear combination,” not “straight line.” As you will see, you can construct complicated curves by taking linear combinations of functions, and use the linear algebra projection operation to match these curves as closely as possible to data. That process of matching is called “fitting.”

To illustrate, the data in the file "utilities.csv" records the average temperature each month (in degrees F) as well as the monthly natural gas usage (in cubic feet, ccf). There is, as you might expect, a strong relationship between the two.

   u = read.csv("utilities.csv")  
   plot(ccf ~ temp, data=u)



Many different sorts of functions might be used to represent these data. One of the simplest and most commonly used in modeling is a straight-line function. In terms of linear algebra, this is a linear combination of the functions f1(T) = 1 and f2(T) = T. Conventionally, of course, the straight-line function is written f(T) = b + mT. (Perhaps you prefer to write it this way: f(x) = mx + b. Same thing.) This conventional notation is merely naming the scalars as m and b that will participate in the linear combination. To find the numerical scalars that best match the data — to “fit the function” to the data — can be done with the linear algebra project( ) operator.

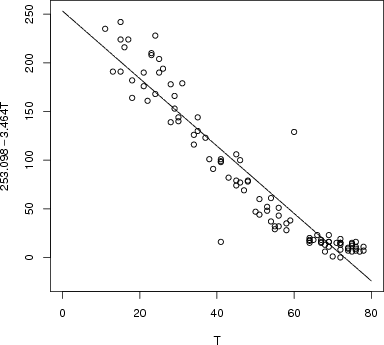
   project(ccf ~ temp + 1, data=u)

  (Intercept) 253.098208  
  temp         -3.464251

The project( ) operator gives the values of the scalars. The best fitting function itself is built by using these scalar values to combine the functions involved.

plot(u$temp,u$ccf) # The data

abline(a = 253.098, b = -3.464, xlim=range(0,80),col="blue") # The line



You can add other functions into the mix easily. For instance, you might think that sqrt(T) works in there somehow. Try it out!

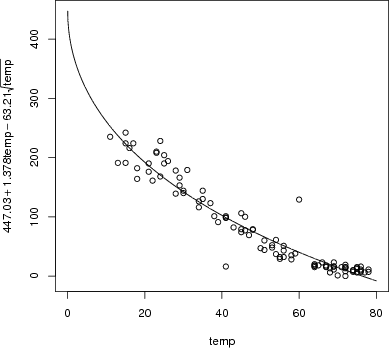
   project(ccf ~ temp  + sqrt(temp) + 1, data=u)

  (Intercept) 447.029273  
  temp          1.377666   
  sqrt(temp)  -63.208025

To get the best fit, use:

plot(u$temp,u$ccf)

lines(lowess(u$temp,u$ccf))



You can also plot a specific curve that you have an expression for (rather than the best fit) via

plot(u$temp,u$ccf)

curve(447.03 + 1.378\*x - 63.21\*sqrt(x), add=T)

Note that the **curve** function requires you to use the variable x, regardless of the names of your axes.

Understanding the mathematics of projection is important for using it, but focus for a moment on the notation being used to direct the computer to carry out the linear algebra notation.

The project( ) operator takes a series of vectors. When fitting a function to data, these vectors are coming from a data set and so the command must refer to the names of the quantities as they appear in the data set, e.g., ccf or temp. You’re allowed to perform operations on those quantities, for instance the sqrt in the above example, to create a new vector. The ~ is used to separate out the “target” vector from the set of one or more vectors onto which the projection is being made. In traditional mathematical notation, this operation would be written as an equation involving a matrix A composed of a set of vectors ⃗v ,⃗v ,...,⃗v )
 1  2     p= A, a target vector b, and the set of unknown coefficients x. The equation that connects these quantities is written A ⋅x ≈
 b. In this notation, the process of “solving” for xis implicit. The computer notation rearranges this to

x = project(⃗b ~ ⃗v1 + ⃗v2 + ...+ ⃗vp).


Once you’ve done the projection and found the coefficients, you can construct the corresponding mathematical function by using the coefficients in a mathematical expression to create a function. As with all functions, the names you use for the arguments are a matter of personal choice, although it’s sensible to use names that remind you of what’s being represented by the function.

The choice of what vectors to use in the projection is yours: part of the modeler’s art.

Throughout the natural and social sciences, a very important and widely used technique is to use multiple variables in a projection. To illustrate, look at the data in "used-hondas.csv" on the prices of used Honda automobiles.

   hondas = read.csv("used-hondas.csv")  
   head(hondas)

    Price Year Mileage Location Color Age  
  1 20746 2006   18394  St.Paul  Grey   1   
  2 19787 2007       8  St.Paul Black   0  
  3 17987 2005   39998  St.Paul  Grey   2   
  4 17588 2004   35882  St.Paul Black   3  
  5 16987 2004   25306  St.Paul  Grey   3   
  6 16987 2005   33399  St.Paul Black   2

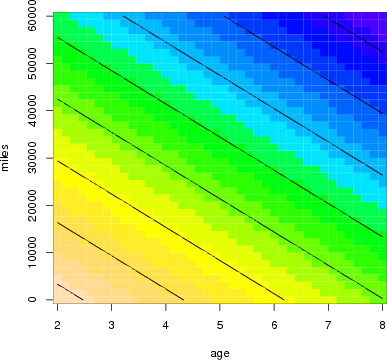
As you can see, the data set includes the variables Price, Age, and Mileage. It seems reasonable to think that price will depend both on the mileage and age of the car. Here’s a very simple model that uses both variables:

   project(Price ~ Age + Mileage, data=hondas)

  (Intercept)  2.133049e+04   
  Age         -5.382931e+02  
  Mileage     -7.668922e-02

You can plot that out as a mathematical function:

   plotFun(21330-5.383e2\*age-7.669e-2\*miles ~ age & miles,   
       agelim=range(2,8), mileslim=range(0,60000))

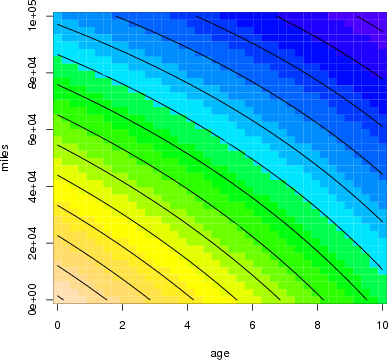


A somewhat more sophisticated model might include what’s called an “interaction” between age and mileage, recognizing that the effect of age might be different depending on mileage.

   project(Price ~ Age + Mileage + Age\*Mileage, data=hondas)

  (Intercept)  2.213744e+04  
  Age         -7.494928e+02   
  Mileage     -9.413962e-02  
  Age:Mileage  3.450033e-03

   plotFun( 22137-7.495e2\*age-9.414e-2\*miles + 3.450e-3\*age\*miles ~ age & miles,   
       agelim=range(0,10), mileslim=range(0,100000))



**EXERCISES**

Exercise 1: Fitting Polynomials Most college students take a course in algebra that includes a lot about polynomials, and polynomials are very often used in modeling. (Probably, they are used more often than they should be. And algebra teachers might be disappointed to hear that the most important polynomials models are low-order ones, e.g., f(x,y) = a + bx + cy + dxy rather than being cubics or quartics, etc.) Fitting a polynomial to data is a matter of linear algebra: constructing the appropriate vectors to represent the various powers. For example, here’s how to fit a quadratic model to the ccf versus temp variables in the "utilities.csv" data file:

   u = read.csv("utilities.csv")  
   project(ccf ~ 1 + temp + I(temp^2), data=u)

  (Intercept) 317.58743630   
  temp         -6.85301947  
  I(temp^2)     0.03609138

You may wonder, what is the I( ) for? It turns out that there are different notations for statistics and mathematics, and that the ^ has a subtly different meaning in R formulas than simple exponentiation. The I( ) tells the software to take the exponentiation literally in a mathematical sense.

The coefficients tell us that the best-fitting quadratic model of ccf versus temp is:

   plotFun(317.587 - 6.853\*T + 0.0361\*T^2 ~ T, T=range(20,80))

To find the value of this model at a given temperature, just evaluate the function. (And note that ccfQuad( ) was defined with an input variable T.)

   ccfQuad(T=72)

  [1] 11.3134

(a)

Fit a 3rd-order polynomial of ccf versus temp to the utilities data. What is the value of this model for a temperature of 32 degrees?

 87  103  128  142  143  168  184

(b)

Fit a 4th-order polynomial of ccf versus temp to the utilities data. What is the value of this model for a temperature of 32 degrees?

 87  103  128  140  143  168  184

(c)

Make a plot of the difference between the 3rd- and 4th-order models over a temperature range from 20 to 60 degrees. What’s the biggest difference (in absolute value) between the outputs of the two models?

|  |  |
| --- | --- |
| A | About 1 ccf. |
| B | About 4 ccf. |
| C | About 8 ccf. |
| D | About 1 degree F. |
| E | About 4 degrees F. |
| F | About 8 degress F. |
|  |  |

Exercise 2: Multiple Regression. In 1980, the magazine Consumer Reports studied 1978-79 model cars to explore how different factors influence fuel economy. The measurement included fuel efficiency in miles-per-gallon, curb weight in pounds, engine power in horsepower, and number of cylinders. These variables are included in the file "cardata.csv".

   cars = read.csv("cardata.csv")  
   head(cars)

     mpg  pounds horsepower cylinders tons constant   
  1 16.9 3967.60        155         8  2.0        1   
  2 15.5 3689.14        142         8  1.8        1   
  3 19.2 3280.55        125         8  1.6        1   
  4 18.5 3585.40        150         8  1.8        1   
  5 30.0 1961.05         68         4  1.0        1   
  6 27.5 2329.60         95         4  1.2        1

1.

Use these data to fit the following model of fuel economy (variable mpg):

pg = x0 + x1pounds.
     

What’s the value of the model for an input of 2000 pounds?

 14.9  19.4  21.1  25.0  28.8  33.9  35.2

2.

Use the data to fit the following model of fuel economy (variable mpg):

pg = y0 + y1pounds+ y2horsepower.
     

(a)

What’s the value of the model for an input of 2000 pounds and 150 horsepower?

 14.9  19.4  21.1  25.0  28.8  33.9  35.2

(b)

What’s the value of the model for an input of 2000 pounds and 50 horsepower?

 14.9  19.4  21.1  25.0  28.8  33.9  35.2

3.

Construct a linear function that uses pounds, horsepower and cylinders to model mpg. We don’t have a good way to plot out functions of three input variables, but you can still write down the formula. What is it?

Exercise 3: The Intercept. Go back to the problem where you fit polynomials to the ccf versus temp data. Do it again, but this time tell the software to remove the intercept from the set of vectors. (You do this with the notation -1 in the project( ) operator.)

Plot out the polynomials you find over a temperature range from -10 to 50 degrees, and plot the raw data over them. There’s something very strange about the models you will get. What is it?

|  |  |
| --- | --- |
| A | The computer refuses to carry out this instruction. |
| B | All the models show a constant output of ccf. |
| C | All the models have a ccf of zero when temp is zero. |
| D | All the models are exactly the same! |
|  |  |