Part 2: Univariate Linear Regression

An introduction to NumPy

Statistical Machine Learning

Generalisation

Produce sensible outputs for inputs that were not seen in learning

Also known as Pattern Recognition

Supervised Learning

- ‘Learn’ from data:
  - *Outcome* measurement, either quantitative (stock price) or categorical (heart attack)
  - Predict from *features* (clinical measurements…)
  - Using a set of *training data* in which we know both the feature measurements and outcomes
  - Result is a prediction model, or *learner*, which can predict outcome for new feature data.
    - ✓A good learner is one that accurately predicts.

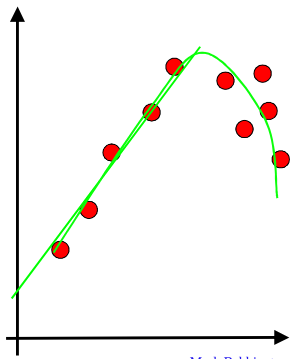
Notation

- Inputs (vectors): $x_i = (x_{ji}, j = 1,\dots,p), i=1,\dots,n$
- Outputs : y_i, $i = 1,\dots,n$
- Targets : t_i, $i = 1,\dots,n$

- Variable Types
  - Quantitative – continuous
    - Regression
  - Qualitative – categorical (discrete)
    - Classification
  - Ordinal (ordered categorical)

Regression

- Curve Fitting (with *noise*)
- Function Approximation



Univariate Linear Regression

Predict t via the output

$$\hat{y} = \beta_0 + \beta_1 x$$

Pick β_0, β_1 to minimize the residual sum of squares

$$\begin{aligned} RSS(\beta_0, \beta_1) &= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2 \end{aligned}$$

Differentiating wrt β_0, β_1 we get the equations

$$\begin{aligned} 0 &= -2\sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_i\right) \\ &= n\beta_0 - n\bar{y} - n\beta_1\bar{x}, \quad \Rightarrow \hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x} \end{aligned}$$

Univariate Linear Regression

and

$$\begin{aligned} 0 &= -2\sum_{i=1}^{n} x_i \left(y_i - \beta_0 - \beta_1 x_i\right) \\ &= n\beta_0\bar{x} - \sum_{i=1}^{n} x_i y_i + \beta_1 \sum_{i=1}^{n} x_i^2 \\ &= n\bar{x}\bar{y} - \sum_{i=1}^{n} x_i y_i - n\beta_1\bar{x}^2 + \beta_1 \sum_{i=1}^{n} x_i^2 \\ \Rightarrow \hat{\beta}_1 &= \frac{n\bar{x}\bar{y} - \sum_{i=1}^{n} x_i y_i}{n\bar{x}^2 - \sum_{i=1}^{n} x_i^2} \end{aligned}$$

NumPy: The working directory

Your NumPy programs will start as follows:

```
import os
os.chdir(“path”)
```

Where `path` is the directory path to the directory you want to work in. It is easiest to keep everything for a program in the one directory.

Initializing NumPy

Next we have

```
from pylab import *
from numpy import *
```

These bring in, respectively, the plotting package (pylab) and NumPy itself. Best done in this order, as otherwise the random number commands in NumPy get overwritten.

All the commands (as indicated by the wildcard `*’) in both packages are now available.

Reading data into NumPy

```
data = loadtxt(‘data_file.txt')
# this reads in the file data_file.txt as a list
# of numbers (1-D array)

# if you have P columns in your data, then
data = data.reshape(-1,P)
# will reshape it into an array with P columns
# the `-1’ indicates that you want as many rows
# as it takes to form P columns
```

Note use of the `#’ symbol – everything on a line after this in a programme is a comment, i.e., is not executed.

Univariate Linear Regression in NumPy

The file ‘height_age.txt’ contains the heights (in cm) and ages (in months) of 83 children between 18 and 29 months of age. We will fit a linear relationship to this data, in order to predict the age of a child from its height.

```
import os
os.chdir(“path”)
from pylab import *
from numpy import *
data = loadtxt(‘height_age.txt')
data = data.reshape(-1,2) # 2 columns – age and height
```

Looking at the data

```
print data
➢ [[ 23.  86.]
➢  [ 25.  79.]
➢  ......
➢  [ 20.  81.]]
# data is an array with 2 columns indexed `0’ and`1’
# NB indexing in NumPy starts at 0, not 1 !!!

# separate x and y using the `slice’ operator
y = data[:,0]
x = data[:,1]
# square brackets select sub-arrays,
# the `:’ indicates all rows, ‘0’ the first column,
# etc.
# In order to select the 2nd through 5th (say)
# columns of an array Z, we use Z[:,1:5]
# this starts with index 1 (ie, the second column) and
# finishes BEFORE index 5 (ie, the sixth column)
```

Plot the data

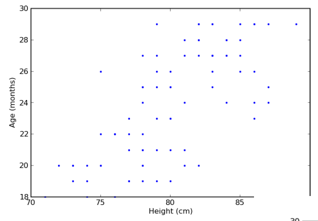
```
figure(1)
plot(x,y,'.')
xlabel('Height (cm)')
Ylabel('Age (months)')
show()   # forces pylab to print current figure
```

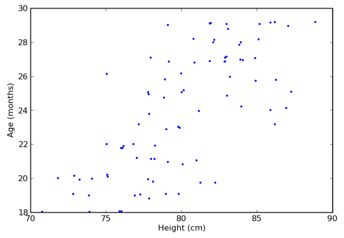
161.326 2.13 Mark Bebbington

Jitter the data

```
figure(2)
xtemp = x + uniform(-0.25, 0.25, x.shape)
ytemp = y + uniform(-0.25, 0.25, y.shape)
# this adds a uniform r.v. on (-0.25,0.25) to
# each x and y value to separate them so we can
# see all the points. The last argument produces
# a random vector the same size as x, so you can
# add them together.
plot(xtemp,ytemp,'.')
xlabel('Height (cm)')
Ylabel('Age (months)')
show()
```

161.326 2.14 Mark Bebbington





161.326 2.15 Mark Bebbington

Sums and means

```
N = len(x)  # length of the vector x
➢   n = 83
xbar = sum(x)/N
ybar = sum(y)/N
print '(xbar,ybar) = (',xbar,',',ybar,')'
➢   (xbar,ybar) = ( 80.0361445783 , 23.9036144578 )
```

Note that you can't control the number of decimal digits

In order to calculate $\Sigma_i x_i y_i$ (and $\Sigma_i x_i^2$) we can use a simple loop

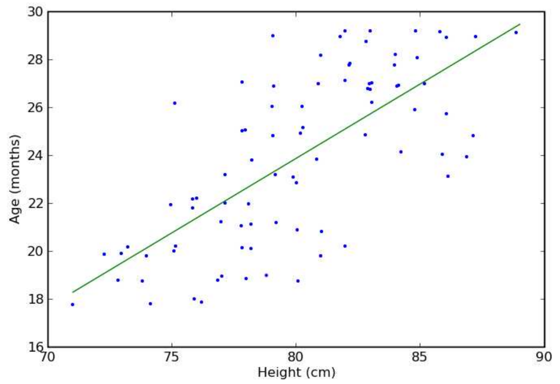
```
sumxy = 0  # initialize
for i in range(N):
# runs over 0,…,N-1. The ':' tells python a loop is needed
# – nested loops are controlled by indenting
    sumxy = sumxy + x[i]*y[i]
print 'sumxy =',sumxy   # loop ended by end of indentation
➢   sumxy = 159637.0
```

161.326 2.16 Mark Bebbington

Regression equations

```
beta1hat = (N*xbar*ybar - sumxy)/(N*(xbar**2) - sumx2)
beta0hat = ybar - beta1hat*xbar
print 'beta =',beta0hat,beta1hat
➢ beta = -25.8091523983 0.62112895515

# calculate endpoints of regression line and plot
minx = min(x)
maxx = max(x)
miny = beta0hat + beta1hat*minx
maxy = beta0hat + beta1hat*maxx
xline = array([minx,maxx])  # note how to
yline = array([miny,maxy])  # input a vector
figure(2)  # adds to previous plot
plot(xline,yline,'-')
show()
```



➢ program in ‘height_age1.py’ on WebCT

Univariate Regression in Vector-Matrix notation

Can rewrite $\hat{y} = \beta_0 + \beta_1 x$ as

$$\hat{y} = \begin{pmatrix} 1 & x \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} = {}_1x\beta$$

where β is a 2x1 matrix (2 rows and 1 column), ${}_1x$ is Nx2, where the first column consists of ones, and $\hat{y}$ is an Nx1 matrix. Then

$$\mathrm{RSS}(\beta_0,\beta_1) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = (y-\hat{y})^T (y-\hat{y})$$
$$= (y - {}_1x\beta)^T (y - {}_1x\beta)$$

The superscript `T’ denotes the transpose, which is the reflection about the diagonal elements.

Regression Equations in Vector-Matrix form

Differentiating w.r.t. β, we get the *normal equations*

$$0 = {}_1x^T (y - {}_1x\hat{\beta}) = {}_1x^T y - {}_1x^T {}_1x\hat{\beta}$$
$$\Rightarrow \hat{\beta} = \left({}_1x^T {}_1x\right)^{-1} {}_1x^T y$$

In NumPy

```
x1 = concatenate((ones((shape(x)[0],1)),x),axis=1)
```

‘shape(x)[0]’ is the number of rows of x, ‘ones(N,M)’ is an NxM matrix of ones, ‘concatenate((A,B),axis = 1)’ joins A and B together along axis 1, ie, side by side.

However, when we try it , we get

```
➢ValueError: arrays must have same number of
dimensions
```

Huh? Think about it - `data’ is an Nx2 matrix, `x’ is the second column, so they are both Nx1 vectors?

```
print 'x.shape =',x.shape
➢x.shape = (83,)
```

The problem is that x has been stored as a 1-D array. This is a problem peculiar to NumPy. We need to make it into a matrix.

Try again

```
x = transpose(matrix(x))
           #'matrix(z)' makes array z into a matrix.
           #Note the transpose. NumPy makes
           #all 1-D arrays into ROW vectors
y = transpose(matrix(y)) #(needed later)
print 'x.shape =',x.shape
➢ x.shape = (83, 1)
x1 = concatenate((ones((shape(x)[0],1)),x),axis=1)
print x1
➢ [[  1.  86.]
➢ [  1.  79.]
➢ ......
➢ [  1.  81.]]
```

Matrices and Arrays

Note that if you use

`x = data[:,1:2]` in place of `x = data[:,1]` etc., then x is Nx1 ☺

BUT

This is an Nx1 ARRAY, not an Nx1 MATRIX, so you will need to use `dot(A,B)` instead of `A*B` for matrix multiplication, etc. ☹

Regression line

```
betahat = linalg.inv(transpose(x1)*x1)*transpose(x1)*y
# we have invoked the ‘inverse’ function from a linear
# algebra package. The inverse of z is inv(z) such
# that z*inv(z) is the identity matrix (ones on
# diagonal, zeros off-diagonal).
print 'betahat =',betahat
betahat = [[-25.8091524 ]
           [  0.62112896]]
# Note that betahat is now a 2x1 matrix
```

Which is exactly the same answer as before.

➢program in ‘height_age2.py’ on WebCT

Defining a Function

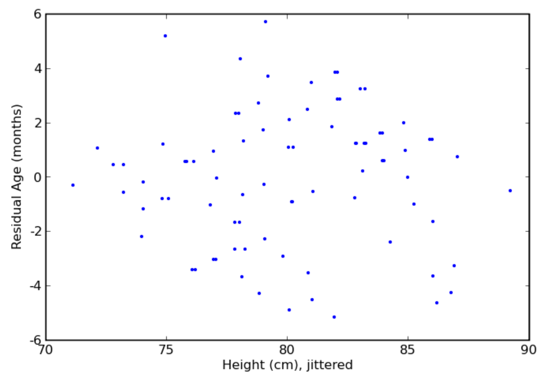
➢We might want to use linear regression more than once. Also, we might want to know the predicted ŷ, or the residual sum of squares (RSS). Hence (after the ‘import’ command(s):

```
def linreg1(x,y):
    # takes Nx1 matrices x and y, and performs
    # linear regression y = beta x. NOTE indenting
    x1 = concatenate((ones((shape(x)[0],1)),x),axis=1)
    betahat = linalg.inv(transpose(x1)*x1)*transpose(x1)*y
    yhat = x1*betahat
    RSS = transpose(y - yhat)*(y - yhat)
    return betahat,yhat,RSS
```

So ‘linreg1(x,y)’ returns 3 arguments – a 2x1 vector betahat, an Nx1 vector yhat, and a scalar RSS

Residuals

```
temp = linreg1(x,y) # this avoids calling the
  function 3 times
betahat = temp[0]
yhat = temp[1]
residual = y - yhat # residuals
RSS = temp[2] # residual sum of squares
print 'RSS =',RSS
➢ RSS = [[ 512.19534501]]
figure(2)
plot(array(xtemp),array(residual),'.')
xlabel('Height (cm), jittered')
ylabel('Residual Age (months)')
show()
```



We see no systematic pattern of residuals

Exercise 2.1

Plot a histogram of the residuals in order to assess their normality.

Tests of Significance

Recall that $y = \beta_0 + \beta_1 x + \varepsilon$, where $\varepsilon \sim N(0,\sigma^2)$. Our estimate of σ^2 is s^2 = RSS/(N-2)

```
s2 = RSS/(N-2)
print 's2 =',s2
➢s2 = [[ 6.32339932]]
```

And then $\hat{\beta} \sim N\left(\beta, \left({}_1x^T{}_1x\right)^{-1} s^2\right)$

Thus $t_j = \dfrac{\hat{\beta}_j}{s\sqrt{v_j}}$, where v_j is the jth diagonal element of $({}_1x^T{}_1x)^{-1}$, has the student-t distribution with N-2 degrees of freedom

Is there a significant slope?

```
def linreg1(x,y):
    x1 = concatenate((ones((shape(x)[0],1)),x),axis=1)
    V = linalg.inv(transpose(x1)*x1)
    betahat = V*transpose(x1)*y
    yhat = x1*betahat
    RSS = transpose(y - yhat)*(y - yhat)
    return betahat,yhat,RSS,V

V = temp[3]
tstat = linalg.inv(diag(sqrt(diag(V))))*betahat/sqrt(s2)
#take diagonal elements of V, square-root, then make into
#diagonal matrix, invert (i.e., v -> 1/v), etc., to get
#student t on N-2 d.o.f.
print 'tstat =',tstat # approx. > 2 is significantly
                      # different from zero
➢ tstat = [[-4.72464234]
           [ 9.11209486]]
```

So both the intercept and slope are significantly different from zero.

➢ program in ‘height_age3.py’ on WebCT

Exercise 2.2

The file ‘leaf_decay.txt’ contains the time (in weeks) and the weight of organic material in a bag.

Fit a linear regression, examine the residuals, and then edit your program to include a suitable transformation of the data, not forgetting to back-transform the results.

More NumPy Stuff

- ➢NumPy aims at Matlab syntax.
  - ❖Sometimes misses by a little
  - ❖Sometimes by a lot
- ➢Actually seems to work a little more like `R’
  - ❖But there is the odd hiccup!

Python `Features’

- It is strongly typed (this means you need to be careful when dividing integers. If a = 7, say, then
  - a/2 = 3
  - a/2.0 = 3.5
  - If a is a `float’ (floating point, i.e., not integer), then no problem
- Works by reference
  - a=b makes a reference to b in a, not a copy
    - ✓ So if b is later changed, a changes also
  - Use a = b[:] to make a copy

Python Lists

- Note 0 indexing
- a[-1] is last element of a, etc.
- a.append(x) – adds x to the end of a
- a.count(x) – number of x’s in a
- a.index(x) – index of first x in a
- a.insert(i,x) – inserts x at index i
- a.pop(i) – removes the element at index i
- a.remove(x) – removes first x from a
- reverse(a) – reverses the order of a
- sort(a) – sorts a
- a == a – compares a and b element-wise, producing 1 in positions where the elements are equal, a 0 otherwise.

Control Flow

```
if statement:                for var in set:
   commands                     commands
elif:                        else:
   commands                     commands
else:
   commands                  for i in range(10):

while(condition):            range(start,stop,step)
```

Comments, Help, Documentation

- Comments are marked with a #
- Functions can have a doc string
  - “”” description of function “””
- info(functionname)

Making Arrays

- arange(5)
- arange(3,7,2) (note where this stops)
- ones(3)
- ones((3,4)) (note the double brackets)
- zeros((2,3))
- eye(2,3)
- linspace(3,7,3)

Multiplication on arrays

- Standard multiplication (A*B) is elementwise
- Matrix multiplication uses dot(A,B)
- Matrix exponentiation pow(A,2)
- Inner product inner(a,b)

Multiplication on Matrices

- Standard multiplication (A*B) is matrix multiplication
- Element multiplication uses multiply(A,B)
- Matrix exponentiation A**2
- Inner product inner(a,b)

More Arrays

- x=arange(6).reshape(3,2)
- x.transpose()
- x.min() min(x[:,1]) x.min(axis=0)
- Similar for max, sum
- mean(a), var(a)

Random Numbers

- from numpy.random import *
- random.rand(matsize)
- random.randn(matsize)
- random.normal(mean,stdev,matsize)
- random.uniform(low,high,matsize)
- random.randint(low,high,matsize)

Linear Algebra

- from numpy import linalg
- linalg.inv
- linalg.pinv
- linalg.det
- linalg.eig
- linalg.svd

And Lots More

http://www.scipy.org/Tentative_NumPy_Tutorial

http://www.scipy.org/Numpy_Example_List_With_Doc

http://www.scipy.org/NumPy_forMatlab_Users

http://dirac.cnrs-orleans.fr/plone/Members/hinsen/courses-and-lecture-notes/python/an-introduction-to-python-for-scientists/

http://mathplotlib.sourceforge.net/