161.326 Statistical Machine Learning
Part 5: Multivariate Linear Regression

Mark Bebbington

Summary

This part is the equivalent of 6 lecture/lab hours. Some of the statistical concepts may be new to all students, particularly the matrix notation involved. On the other hand, the programming is no more difficult than in the previous part.

The book has a little material on linear regression (Section 2.4). Thus copies of some pages from Hastie et al can be found in the `reading' folder in Part 5. The lecture slides follow the material in the same order, with the addition of a lot of worked programming examples in NumPy.

As a general approach, I recommend following through the lecture slides, trying all the programs as you go along (remember to change the "os.chdir" argument to wherever you have put the data file(s)), and reading the relevant pages from Hastie et al once you finish a sub-topic. On the other hand, if another approach works better for you, go for it.

Halfway through the lecture slides you should be able to do the last question of Assignment 1. The remainder of this document is a commentary on the lecture slides, which should be read in conjunction with them.

Slide

4. "Nearest neighbour" here means clustering techniques which we will meet later. They are non-parametric type methods, where inputs that are "close" are assumed to have similar outputs.
5. The linear model, on the other hand, assumes a linear response, i.e., if you double an input, you get twice the response.
6. Pros and cons ….
7. Least squares works on minimizing the differences between y and ŷ. Thus we can use it for any univariate output, regardless of whether the input is univariate or multivariate.
8. All lower case letters are assumed to be column vectors (n rows by 1 column), and the superscript "T" indicates the transpose, i.e., a row vector. Upper case letters generally represent matrices. We will usually have n data points (or replications) and p predictors, thus y will be an n-vector, and X an n-by-(p+1) matrix. Subscripts identify particular elements of the vector/matrix, although it is frequently abused as in this slide, where x_j is the jth row (if you start at 0) of X.
9. Note that the solution for β has exactly the same form as for univariate regression (previous section).
10. In 2-dimensions we fit a line, 3-dimensions a plane, and in higher dimensions a hyperplane. In all cases the residuals are the distance from the observation to the hyperplane along the observation (i.e., y) axis.
11. Define a new function for a p-dimensional input. Now X is a matrix, otherwise there is no difference from "linreg1". This is a function definition, from the "def". The colon should automatically indent subsequent lines, which is important, as Python does not use "end" commands – the end of a function or loop is wherever the indentation ends. <u>Spaces are important.</u> shape(X)[0] is the first argument returned from querying the shape of X, i.e., the

number of rows. Thus we take a vector (a no. rows of X by 1 matrix) of ones and attach it (concatenate) to the matrix X. axis=1 indicates that it is attached as another column, not another row. Note the seeming excess number of parentheses – they are all required. Transpose should be obvious, and linalg.inv() is the inverse of a matrix. The "return" statement sends back three results, beta is a (p+1)-by-1 vector of coefficients, yhat an n-by-1 vector of fitted values, and RSS a scalar.

12. You are assumed to know the meaning of "uncorrelated", "variance" and "covariance matrix". If you don't, ASK!
13. "Gaussian" and "Normal" are synonymous in this course. The symbol "~" means "is distributed as". $N(\mu,\sigma^2)$ means a normal distribution with mean μ and variance σ^2. These become a vector of means and a covariance matrix in more than one dimension.
14. Test statistic, as in the univariate case.
15. Testing significance in NumPy
16. Note the use of the "reshape" and slice ":" commands. More details about the dataset are available in Hastie et al.
17. Note the use of nested for-loops, and the "range" command. The plots disclose that there are some indicator (0 or 1) variables, and a lot of correlation. This is obviously a poor graphic, as it is hard to read. NB
18. Obtaining the correlation matrix of the inputs from the covariance function. Note that cov() assumes that the replications are the columns, not the rows, hence the transpose.
19. Debugging exercise. See later for solution. The basic debugging strategy is to use a lot of print commands to see what is what at each line of your code.
20. After debugging, using 'prostate_linreg1.py'
21. Solution to debugging exercise.
22. How well does the fitted model generalize to data that is not part of the fitting process?
23. A "loss function" is sometimes called mean square error, etc. …
24. The purpose of the machinery here is to find the best trade-off between bias and error, or *how much* improvement you get from each additional parameter. You should look in a dictionary to find the difference between "precision" (lack of variation) and "accuracy" (lack of bias). Both are desirable, but cannot be simultaneously achieved.
25. Remember that the effect of predictors may be different if other predictors are added or removed. As an example, try adding in y as a predictor and see what happens to the x's.
26. Standardization is important to avoid numerical errors in the program. Try the program in 'prostate_linreg2.py'
27. "Nested models" are where each one is contained in the next largest one. In this case, each level adds another coefficient. Now, it is obvious that adding more parameters in a series of nested models cannot make the fit any worse, so why not do it? The basic answer here is that you then end up fitting the *training data* perfectly. On the other hand, the data you want to use the model on (the *test data*) will be different, so over-fitting leads to bias and error.
28. Note the form of the null-hypothesis.
29. Since one of the predictors we might discard could be the intercept (β_0) we need to define linear regression without an intercept term Note that there are applications where it is obvious the intercept must be zero, so it may be of use for its own sake.
30. Here we have the *full model* X1 (all 9 predictors, including the intercept) and the *reduced model* X0 (only 4 predictors). We check the z-scores for the reduced model, finding that they are still all significant (they cannot become less significant).
31. And now the F-statistic shows that the extra parameters do not do enough to make them worthwhile. Try the program in 'prostate_linreg3.py'

You can now read Hastie et al, pp. 41-47.

32. v_j is defined on slide 14, σ-hat on slide 12. $z_{1-\alpha}$ is the percentile of the standard normal distribution (1.96 etc.). The generalization of a confidence interval in more than one dimension is a confidence set. Note that you cannot take the product of individual confidence intervals: If you use 95% confidence intervals you would get a 95^P% confidence set, and even this only works if the predictors are independent.
33. See ‘ex5_2.py’. The easiest (mathematically) way to plot the confidence set is by simulating points and accepting or rejecting them. Note the 2-D projections for the solution.
34. Bias-variance trade off again. Do the last few parameters give useful information, or is it just noise?
35. One way to examine this is to fit the model (i.e., estimate β) using one set of data, and compare its performance on a different set of data. See ‘ex5_3.py’. Of course, we get a `better’ fit (smaller RSS) by optimizing the parameters to the data. We see that parameter 4 is not significant according to the test data. Note that both sets of data must be standardized to the same variance.
36. The next few slides outline possible ways of arriving at a “best” model including only some of the predictors. Of course, for a sufficiently small set of predictors, we could just fit all the possible subsets.

You can now attempt Assignment 1, Q4.

37. Forward stepwise selection adds one predictor (the best improvement) at a time, until none of the remaining predictors are a worthwhile improvement according to the F-ratio (slide 28).
38. Sorry about the font size, but I wanted all of the function on the same slide. The idea is that one has an existing model “oldX1”, “oldbeta”, “oldRSS”, and the list of included predictors “inclpred”, where the ‘-1’ predictor is the intercept. Then all the possible single parameter additions to the model are examined, with the best (“bestRSS”, etc.) being tested to see if it is good enough. If it is, it is added to the new (oldX1, oldbeta, etc.). “continu” is just a flag that stops the program when it becomes 0 (“false”). (bestpred, bestbeta, bestRSS) keeps track of the best additional predictor at the current level. Note the additional stopping conditions – the statisticians might be unfamiliar with some of these pitfalls.
39. The program to run the fstepreg function on the prostate data. Run the program ‘prostate_fwdsubreg.py’. Put a few print statements in so you can see what is going on.
40. The best model turns out to be the same as the ‘reduced model’, except that the intercept is included as well (naturally enough, as it is the starting point).
41. See ‘ex5_4.py’. We see that the first predictor in is x_2, followed by the intercept, then x_4 and x_3. This has an RSS of 32.9, while starting with the intercept, even though it is the second term in the other model, gives a different model of the intercept, x_1, x_2, x_5 and x_4. Interesting, eh?
42. The key point is that stepwise models are not guaranteed to arrive at the best model, regardless of the criteria (see Assignment 1, Q2).
43. Instead of starting with the intercept and add predictors, we can start with the full model and drop them out. Or we can do both.
44. See ‘ex5_5.py’ Note that `worstpred’ is the index in ‘inclpred’, not the predictor itself. We see that x_7 is removed first, followed by the intercept. The rest are retained. Here we need evidence to <u>remove</u> a predictor (significant until proven otherwise) while in the forward case, we need evidence to <u>add</u> a predictor (insignificant until proven otherwise).

You can now read Hastie et al, pp. 55-56.

45. Shrinkage methods “weight” the various predictors. We shall look at ridge regression mainly, as the lasso needs optimization methods we will not get to in this course.
46. Basically, we make large coefficients smaller. Hence standardization is essential, or we will immediately penalize the coefficients for predictors given in small numbers. For example, suppose one factor is distance. If we measure it in km, rather than m, the β will be 1000 times larger, and hence the penalty term will be 1,000,000 times larger!
47. We see that, once the standardizing has been done, the formulae are very similar to the basic case. “Full rank” and “non-singular” mean that it is invertible, i.e., we can find an inverse. Orthogonal inputs are ones with zero dot-products, i.e., they are at right-angles to one another.
48. The ‘answer’ should not depend on the units or the addition of a constant (for example, imagine if temperature was a factor – the scale used should make no difference). See Exercise 5.6
49. Basically this is a computational convenience, as the intercept is not penalized. See slide 48.
50. “eye(8)” is the 8-by-8 identity matrix. Note that “lambda” is a reserved word in Python – don’t use it as a variable. We need to “glue” the two bits of beta together. Note how the RSS is calculated. Centering is easily accomplished.
51. Try the program (prostate_ridgereg.py) . Compare the estimated beta with that of the full model (slide 20, although you might want to go back and center the inputs in that program).
52. See ‘ex5_6.py’. Scaling the inputs changes the coefficients (and not proportionally to the scaling). With no penalty on the intercept, shifting y amounts to a shift in the intercept. The other coefficients are unchanged. This is not the case if there is a penalty on the intercept. Note how increasing lambda results in a worse RSS.
53. The idea of singular value decomposition is to find a transformation (U,V) of the input matrix (X) such that the result is a series of orthogonal vectors of (ordered) length D.
54. Nothing could be easier in NumPy (this is as good as it gets).
55. You may have heard of principal components. First, this is how the SVD and eigen decompositions relate.
56. This shows how one extracts the principal components, which are basically the directions in the predictor space which explain the most.
57. A picture worth a thousand words?
58. The prostate data. Tough to get anything from this …
59. This explains how ridge regression works in terms of singular value decomposition.
60. Degrees of freedom is a vital concept. It is the amount of `wiggle room’ you give the model to match variation in the data. Too much leads to overfitting. Back to the bias-variance tradeoff. Try the program (prostate_svd.py). The bottom version is “elegant”, the loop is easier to read. Your choice – I want correct, I’m not worried about elegance.
61. See ‘ex5_7.py’. Note how we have to get around the problem of ridge regression being y-origin dependent.
62. This is how the Lasso works. Due to the nonlinearity, much more complex fitting machinery is required. The key difference to ridge regression is that it can effectively eliminate some parameters. Not examinable.
63. As we have the principal components, which are a transformation of the inputs, why not regress directly on them? Or, more interestingly, on only the more important of them.
64. This is another method of discarding the less important inputs. However, these are the PC inputs, so the included PC inputs may include parts of all, or only some, of the originals.
65. Try the program (prostate_pcr.py). Compare betahat_pcr with the fitted beta from other methods (slides 20 and 51). Vary M to see what happens.
66. This may make sense in a few weeks, for now just nod thoughfully and move on.

You can now read Hastie et al, pp. 59-66.
You can now attempt Assignment 2, Q1