161.326 Statistical Machine Learning
Part 6: Model Selection and Assessment

Mark Bebbington

Summary

This part is the equivalent of 4 lecture/lab hours, and deals with the question of how to find the `best' model, and how to estimate how `good' it is. There is very little on this in the text. I have given references for a couple of papers from the Journal of the Royal Statistical Society (with discussions) which go into this in some detail. You may find these rather hard going, but they can be very illuminating.

As usual, NumPy programs will be provided. These will be demonstrated on the linear regression model(s) for the prostate cancer data we met in Part 5.

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)).

Once you have finished the lecture slides, see if you can adapt the programs to comparing other models from Part 5. You should then be able to do Question 2 of Assignment 2. The remainder of this document is a commentary on the lecture slides, which should be read in conjunction with them.

Slide

3. This is a general philosophy. In general the rule is to have a procedure (or algorithm) in mind before examining the actual data (you should know the form of the data). The more you adapt your approach to the actual training data, the greater the bias in your model.
4. This is known as `over-fitting'.
5. We want to find the best model. But once we have the best model, we need to be able to quantify the inherent prediction error, so we can give bounds on any predictions made.
6. So, fit all candidate models to the training data to estimate the parameters. Use these parameters in testing the models on the validation data. We also need separate test data in order to get an unbiased estimate of the chosen model's variability – remember that it is the one with the least variability on the validation data, so that would be a biased estimate of variability.
7. An example fitting two models from Part 5 to the prostate cancer *training* data,
8. and testing them on the *test* data. We see that while the full model has a smaller RSS on the training data, its RSS on the test data is actually *larger*, i.e., it fits *worse*, in spite of the additional parameters!
9. See 'ex6_1.py. You can try more models than these. The 4.44 is the effective degrees of freedom (Part 5, Slide 60). In actual fact, this, lambda itself, and the parameters in the reduced and forward subset regression models should be reselected on the new training data.
10. Alas, there are never enough data to go 'round. So we will look at what we can do otherwise, and compare with the data splitting approach.
11. So, given input data X, targets (or responses) Y, and a model $\hat{f}(X)$ which estimates Y, we can calculate a `loss function'. In the linear regression context this will be the mean-squared error (RSS/N). In classification (see next section) it might be the error rate.

12. Note the distinction between test error, which is what we want to minimize, and training error, which is what we do minimize.
13. This is what `overfitting' means, and why it is bad. Remember that there is usually an error attached to any measurement. The structure in the RH plot is crucially dependent on these errors being zero!!
14. Summary.
15. 'Optimism' is defined as the difference between what we want to estimate (the in-sample error), and what we can estimate (the training error).
16. Optimism allows for direct estimation of prediction error. Otherwise we have to use resampling methods.
17. The program is in `prostate_split.py'. We will use the example of the full (9 predictors including the intercept) and reduced (only predictors 1,2,4 and 5) regression models from Part 5. The full model has lower training error (naturally), but greater optimism and in-sample error.
18. All the programs in this part include the data splitting method first for comparison. We see that the data splitting model and the calculation of optimism give the same result.
19. Some models we have met (e.g., ridge regression) have fewer 'effective parameters' than actual parameters. The C_p statistic is a general measure of prediction error for linear regression type models.
20. Example from Part 5 of a model (Ridge Regression) with fewer 'effective parameters' than actual parameters.
21. In linear regression, AIC is equivalent to C_p. More generally, AIC is defined as twice the number of parameters minus twice the log-likelihood, but we do not go into likelihood methods in this course.
22. Programme is in 'prostate_aic.py'. Note that σ^2 is calculated from a low-bias (i.e., highly complex, or many parameter) model.
23. Again, we duplicate the data-splitting result.
24. Another debugging exercise – the program runs, but produces different values.
25. See 'ex6_2.py'. We see that AIC chooses the model with all predictors except x_7, whereas the test data prefers only the intercept and x_1.
26. This is another likelihood-based method. Unlike AIC, it is asymptotically correct (as the amount of data becomes large, it selects the correct number of parameters). However, for small data sets it penalizes too heavily, selecting too few parameters, and hence AIC is preferred.
27. MDL (http://www.mdl-research.org/) is a different way of calculating things, which may appeal more to the computer science students.
28. See 'ex6_3.py'. The model favoured by BIC is the usual forward subset regression model, with 5 predictors including the intercept.
29. This allows us to quantify how much more preferable a model is, rather than just which is best. Note the dependence on the set M. In other words you need to have a good, varied, set of models to have much confidence in this approach.
30. We reserve a part of the data for validating models fitted to the remainder. If we reserve each part of the data in turn, and sum up the results from each part, we have cross-validation.
31. A nice picture from the text showing the split in train-validate-test cross-validation.
32. For 'tuning parameter', read β in the linear regression context.
33. Programme is in 'prostate_crossval.py'. We need to keep track of both the y's and X's in the kth set, because we are going to fit the estimated models to them to evaluate the loss function.

34. Note that the results will differ slightly in each run, as the data is split randomly in each run. This is generally safer, in case there is any systematic structure in the data. Consider, for example, what would happen if the data were ordered.
35. Again, we reproduce the data-splitting result.
36. Another debugging exercise. This time it doesn't run. Remember – try to get it to run without looking at the version which works. You won't have the luxury of doing so when it's your program.
37. See 'ex6_4.py'. You should keep the groups random (basic statistical principal – always randomize to neutralize other possible sources of variation.
38. How many parts do we divide the data into? Accuracy is not linear in the data – typically there is some sort of exponential convergence.
39. See 'ex6_5.py'. We see that things appear to plateau out after K = 5 or so.
40. The bootstrap is an outgrowth of cross-validation. The key point is that we are sampling with replacement, rather than without replacement.
41. We can estimate any quantity of a statistic of the data,
42. for example the slope in univariate regression, or RSS, or …
43. An example of estimating prediction error. We average over the number of bootstrap samples, so the estimate should become more accurate as B increases.
44. We are going to compare the full and reduced regression models. The bootstrap sample involves taking a random y (ytemp) *along with the corresponding* X (Xtemp, and hence Xredtemp). In other words, we select both the inputs and corresponding output together, preserving the input-output link which, after all, is the object of our investigation. Note sampling with replacement.
45. However, we are still using the same (sort of) data for training and validation – how do we fix this?
46. First introduce the cross-validation principle to our bootstrap – don't estimate the prediction error on cases where targets appear in the bootstrap sample. This requires some pretty fancy programming (see next slide).
47. As promised, a lot of program. Need to keep careful track of whether a data point has been used to estimate the parameters on not. We proceed over the bootstrap samples, evaluating contributions to the loss function as we go.
48. However, as we have only N possibilities to make up our bootstrap sample of size N, there will be duplicates. Hence the bootstrap sample is more variable than we want (cf. Var(X+Y) = Var(X) + Var (Y), but Var (2X) = 4Var(X))
49. The result is a weighted average of our bootstrap CV error, and the usual training error (Slide 12). Full program for Slides 44, 47 and 49) is in `prostate_boot.py'.
50. Again, the same conclusion as the data-splitting result. However, have a glance at by `how much' each method confirmed the correct result.
51. Given a number of possible models, the predictions can be averaged to give a better predictor. Note that this is already built into the linear regression method, so no advantage is gained. The result would be something like penalized regression.
52. Model averaging is a way of making predictions less sensitive to individual model assumptions. The unweighted average is sometimes called the `Committee Method'.
53. Here the weights are the posterior estimates of the model, estimated (for example) by the BIC using the nice little formula from Slide 31.
54. Bootstrapping can also be seen as model averaging, and is by far the simplest of the methods.
55. See 'ex6_6.py'. You can also calculate the standard deviation for the estimates and compare with those from the regular linear regression.