Learning from Data
Generalisation

Course lecturer: Amos Storkey
a.storkey@ed.ac.uk
Course page : http://www.anc.ed.ac.uk/∼amos/lfd/

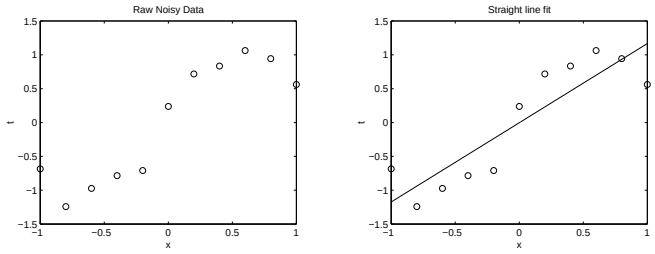

Figure 1: Left: Training Data for a regression problem. We wish to fit a function $f(x|\boldsymbol{\theta})$ to this data. Right: A straight line fit might look reasonable.

1 Introduction

One major goal in supervised learning is, on the basis of labelled training data, to encapsulate the underlying mechanism which generated the data, thereby learning a model with predictive power. That is, given a novel unlabelled instance, to make an accurate prediction.

1.1 Supervised Learning

Formally, supervised learning consists of the following scenario: A given set of training data, $D_{train} = \{(\mathbf{x}^\mu, t^\mu), \mu = 1, \ldots, P\}$ where each $\mathbf{x}^\mu$ is a vector of (in general real valued) attributes, and t^μ is the associated *target* for the μ^{th} pattern. (In the binary classification tasks we have been considering, $t^\mu \in \{0, 1\}$). If t^μ can take on one of only a finite set of values, we call this a *classification* problem. If t^μ can be any value, we call this a *regression* problem.

Our basic paradigm is the following : There is some underlying process which generates "clean" data. The data is then possibly corrupted by noise to give the actual data that we observe. The aim in supervised learning is to try to recover the underlying clean data generating mechanism.

Prediction without assumptions is meaningless

Consider the following problem. What is the next number in the sequence 1,2,3,4,5, ?[1] Well, there is no "correct" answer. I could predict anything, and who's to say that I'm incorrect? It may well be perfectly reasonable to a Martian to say the next number is 78. In fact, the "answer" that I was looking for was 63. This is the number of the bus that follows buses 1,2,3,4 and 5 in my home town. "Not fair!", you might say, "we didn't know that you were talking about busses in your home town". Well, that's what learning from data is about – you have to try to collate as much information about the data domain as you can, and hope that your assumptions are reasonable. Whatever you do, your predictions are only as good as your assumptions.

Consider the training data in fig(1). This is an example of a regression problem in which the targets t are real values. In this case, the inputs x are also real values. Our aim is to fit a function $f(x|\boldsymbol{\theta})$ to this data. What kinds of functions might we fit? For example, a straight line fit $f(x|\boldsymbol{\theta}) = \theta_0 + \theta_1 x$ may look reasonable.

[1] This is a supervised problem since a sequence has a temporal ordering and can be written as (t, x) pairs : $(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, ?)$.

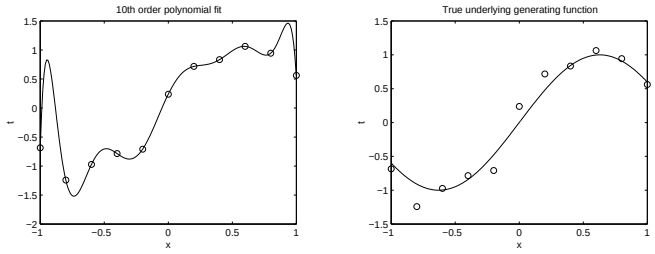

Figure 2: Left: What about a 10^th order polynomial. This has zero training error. Right: The "correct" clean underlying function which generated the data.

Or is it a 10^{th} order polynomial, $f(x|\boldsymbol{\theta}) = \sum_{i=0}^{10} x^i \theta_i$, see fig(2)? In fact, the data was generated using the rule $t = \sin(2.5x) + \eta$, where η is zero mean Gaussian noise of variance 0.2^2.

To find a "good" curve, we need appropriate beliefs/assumptions about the smoothness of the "clean" underlying function *and* the level of noise. If our assumptions are not correct, our predictions will be poor.

Classes of Predictors

Our general approach to supervised learning is to make some function $f(\mathbf{x}|\boldsymbol{\theta})$ so that, given a novel point $\mathbf{x}$ our prediction $f(\mathbf{x}|\boldsymbol{\theta})$ will be accurate. What do we mean by accurate? If we had some extra data, D_{test}, different from the training data and generated in the same manner, then we would like that the error made by our predictions is roughly the same as the error that would be made even if we knew exactly what the clean underlying data generating process were. Of course, this is in some sense, an impossible task. However, we can devise procedures that can help give us some confidence in our predictions.

1.2 Training Error

The typical way that we train/learn the adjustable parameters $\boldsymbol{\theta}$ of our model is to optimise some objective function. For example, if our current model outputs $f(\mathbf{x}^\mu|\boldsymbol{\theta})$ on an input $\mathbf{x}^\mu$ and the training data output for that $\mathbf{x}^\mu$ is t^μ, we would like to adjust the parameters $\boldsymbol{\theta}$ such that $f(\mathbf{x}^\mu|\boldsymbol{\theta})$ and t^μ are close. We can measure how close these values are by using a function $d(x, y)$ which measures the discrepancy between two outputs x and y. To find the best parameter settings for the whole training set, we use

$$E_{train}(\boldsymbol{\theta}) = \sum_{(\mathbf{x}^\mu, t^\mu) \in D_{train}} d(f(\mathbf{x}^\mu|\boldsymbol{\theta}), t^\mu) \tag{1.1}$$

If we adjust the parameters $\boldsymbol{\theta}$ to minimise the training error, what does this tell us about the prediction performance on a novel point $\mathbf{x}$? In principle, nothing! However, in practice, since the mechanisms which generate data are in some sense smooth, we hope that our predictions will be accurate. We saw that in the case of using a perceptron, we can always find a hyperplane that separates data, provided that the dimension of the data space is larger that the number of training examples. In this case, the training error is zero. We saw, however, that the error on the 600 test examples was non-zero. Indeed, if the training data is believed to be a corrupted version of

some clean underlying process, we may not wish to have a zero training error solution since we would be "fitting the noise". What kind of error would we expect that our trained model would have on a novel set of test data?

1.3 Test Error

Imagine that we have gone through a procedure to minimise training error. How can we assess if this will have a good predictive performance – i.e., will *generalise* well? If we have an independent set of data D_{test}, the *test error*

$$E_{test}(\boldsymbol{\theta}) = \sum_{(\mathbf{x}^\mu, t^\mu) \in D_{test}} d(f(\mathbf{x}^\mu|\boldsymbol{\theta}), t^\mu) \tag{1.2}$$

is an unbiased estimate of the prediction error of our model $f(\mathbf{x}|\boldsymbol{\theta})$.

1.4 Validation Data

Consider two competing prediction model classes, $f_1(\mathbf{x}|\boldsymbol{\theta}_1)$ and $f_2(\mathbf{x}|\boldsymbol{\theta}_2)$. We train each of these by minimising the training error to end up with training error "optimal" parameter settings $\boldsymbol{\theta}_1^*$ and $\boldsymbol{\theta}_2^*$. Which is the better model? Is it the one with the lower training error? No. We can say that model with setting $\boldsymbol{\theta}_1^*$ is better than a model with setting $\boldsymbol{\theta}_2^*$ by comparing the *test* errors, $E_{test}(\boldsymbol{\theta}_1) < E_{test}(\boldsymbol{\theta}_2)$. Using test data in this way enables us to validate which is the better model.

The standard procedure is to split any training data into three sets. The first is the training data, D_{train}, used to train any model. The second $D_{validate}$ is used to assess which model has a better test performance. Once we have chosen our optimal model on the basis of using validation data, we can get an unbiased estimate of the expected performance of this model by using a third set of independent data D_{test}. This data should not have been used in any way during the training procedure if we wish to obtain an unbiased estimate of the expected test performance of the model.

1.5 Dodgy Joe and Lucky Jim

Perhaps the following parody will make the above arguments clearer:

Let me introduce two characters, "Lucky Jim" and "Dodgy Joe". Lucky Jim invents some new procedure, and initially, finds that it works quite well. With further experimentation, he finds that it doesn't always work, and that perhaps it requires some rather fine tuning to each problem. Undeterred, this charismatic scientist attracts both funds and attention enough to stimulate a world wide examination of his method. Working independently of each other, surely enough research groups from around the world begin to report that they manage to achieve zero test error on each problem encountered. Eventually, some research group reports that they have found a procedure, based on Lucky Jim's method that is able to give *zero* test error on *every* problem that has ever been known to exist. After so many years of hard work, Lucky Jim happily announces his universal predictor (perhaps a billion hidden unit neural network with fixed parameters), with the (indeed true) claim that it gives zero test error on every known problem that ever existed. He markets this product and hopes to claim a fortune.

Contrast the dogged determination of Lucky Jim now with the downright unscrupulous behaviour of Dodgy Joe. Quite frankly, he doesn't have the patience of Lucky Jim, and he simply assembles all the known problems that ever existed, and their corresponding test sets. He then constructs his method such that, when asked to perform the prediction on problem A with corresponding test set B, he simply makes the output of his method the output for the test set B (which he of course knows). That is, his algorithm is nothing more than a lookup table - if the user says, "this is the test set B" then Dodgy Joe's algorithm simply reads off the predictions for test set B which, by definition, will give zero error. He then also markets his universal predictor package as giving zero test performance on every known problem (which is indeed true) and also hopes to make a fortune.

If we look at this objectively, both Lucky Jim and Dodgy Joe's programs are doing the same thing, even though they arrived at the actual code for each method in a different way. They are both nothing more than lookup tables. The point is that we have *no* confidence whatsoever that either Lucky Jim's or Dodgy Joe's package will help us in our predictions for a *novel* problem. We can only have confidence that a method is suitable for our novel problem if we believe that a particular method was successful on a *similar* problem to ours in the past, or the assumptions that resulted in successful prediction on a previous problem might well be expected to hold for a novel problem – smoothness of the problems for example.

The above also highlights the issue that it is not enough to assess a method only on the reported results of a subset of *independent* research groups. It may be that, with the same method (eg neural nets with a fixed architecture but undetermined parameters) one of a hundred groups which decide to tackle a particular problem is able to find that particular set of parameter values (essentially by chance) that gives good test performance, whilst the other 99 groups naturally do not report their poor results. In principal, real comparison of a method on a problem requires the collation of *all* results from *all* sources (attempts).

WowCo.com

WowCo.com is a new startup prediction company. After years of failures, they eventually find a neural network with a trillion hidden units that achieves zero test error on every learning problem posted on the internet up till January 2002. Each learning problem included a training and test set. Proud of their achievement, they market their product aggressively with the claim that it 'predicts perfectly on all known problems'. Would you buy this product?

Model Comparison : An example

Let us reconsider our favourite digit classification problem. There are 1200 examples of the digit 1 and 7. Let us split this to form a new training set of 400 examples, and a validation set of 200 examples. We will retain a further 600 examples to measure the test error. I used PCA to reduce the dimensionality of the inputs, and then nearest neighbours to perform the classification on the 200 validation examples. Based on the validation results, I selected 19 as the number of PCA components retained, see fig(3). The independent test error on 600 independent examples using 19 dimensions is 14. Once we have used the validation data to select the best model, can we use both training and validation data to retrain the optimal model? In this case, we would have decided that 19 is the optimal dimension to use, based on 200 training and 100 validation points. Can we now, having

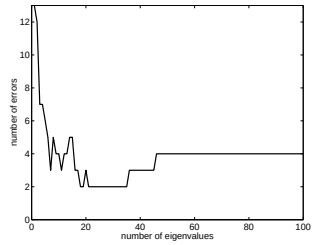

Figure 3: 400 training examples are used, and the validation error plotted on 200 further examples. Based on the validation error, we see that a dimension of 19 is reasonable.

decided to use 19 components, retrain the model on the 300 training and validation points? This is a fairly subtle point. In principle, the answer is no, since the new procedure, strictly speaking, corresponds to a different model. However, in practice, there is probably little to be lost by doing so, and possibly something to be gained since we are making use of more training data in setting many parameters. These issues highlight some of the philosophical complications that can arise based on a frequentist interpretation of probability. No such difficulties arise in the Bayesian framework, where all the training data can be used in a clear and consistent manner for model selection.

1.6 Regularisation

If the data generation process includes noise (additive), then the true, clean data generating process will typically be smoother than the observed data would directly suggest. To try to discover this smoother clean underlying process, we need to ensure that our model for the clean underlying process does not fit the noise in the observed data. That is, it is undesirable to have a zero training error, and we need to encourage our model to be smooth. One way to achieve this is through regularisation in which an extra "penalty" term is added on the the standard training error, to form the regularised training error:

$$E_{regtrain}(\boldsymbol{\theta}, \lambda) = E_{train}(\boldsymbol{\theta}) + \lambda E_{reg}(\boldsymbol{\theta}) \tag{1.3}$$

The larger λ is, the smoother will be solution which minimises the regularised training error. If we regularise too much, the solution will be inappropriate and too smooth. If we don't regularise at all, the solution will be over complex, and the solution will be fitting the noise in the data. (Regularisation only really makes sense in the case of models which are complex enough that overfitting is a potential problem. There is little point in taming a pussy-cat; taming a lion however, might be worthwhile!). How do we find the "optimal" value for λ? Training is then done in two stages:

- For a fixed value of λ, find $\boldsymbol{\theta}^*$ that optimises $E_{regtrain}$. Repeat this for each value of λ that you wish to consider. This gives rise to a set of models, $\left\{\boldsymbol{\theta}^*_{\lambda_i}, i = 1, \ldots, V\right\}$.
- For each of these models, on a separate validation set of data (different from the training data used in the first step), calculate the validation

error:

$$E_{val}(\boldsymbol{\theta}^*) = \sum_{(\mathbf{x}^\mu, t^\mu) \in D_{val}} d(f(\mathbf{x}^\mu | \boldsymbol{\theta}^*), t^\mu) \tag{1.4}$$

The "optimal" model is that which has the lowest validation error.

Regularisation : An example

In fig(4), we fit the function $t = a\sin(wx)$ to data, learning the parameters a and w. The unregularised solution badly overfits the data, and has a high validation error. To encourage a smoother solution, I used a regularisation term $E_{reg} = w^2$. I then computed the validation error based on several different values of the regularisation parameter λ, finding that $\lambda = 0.5$ gave a low validation error.

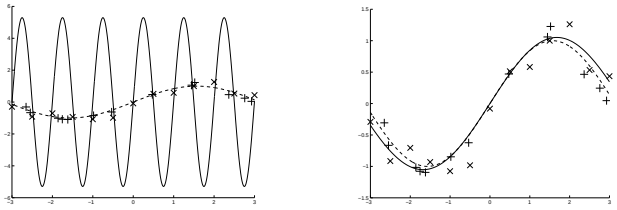

Figure 4: Left: The unregularised fit ($\lambda = 0$) to training given by $\times$. Whilst the training data is well fitted, the error on the validation examples, $+$ is high. Right: the regularised fit ($\lambda = 0.5$). Whilst the training error is high, the validation error (which is all important) is low. The true function which generated this noisy data is the dashed line, and the function learned from the data is the solid line.