FIT2086 Lecture 8 Summary
Model Selection and Penalized Regression

Dr. Daniel F. Schmidt

October 29, 2019

1 Part I: Model Complexity

Underfitting and Overfitting. Recall the supervised learning problem: we have $p+1$ measured variables on n individuals. We nominate one of the variables as our target, say y, and the remaining p variables as our predictors, say $x_1, \ldots, x_p$. We then want to predict the value of y for an individual using these p predictors. In practice we often have a large number of predictors, and we assume that at least some of them will be useful in predicting our target. The question then is: should we just use them all, and if not, why not?

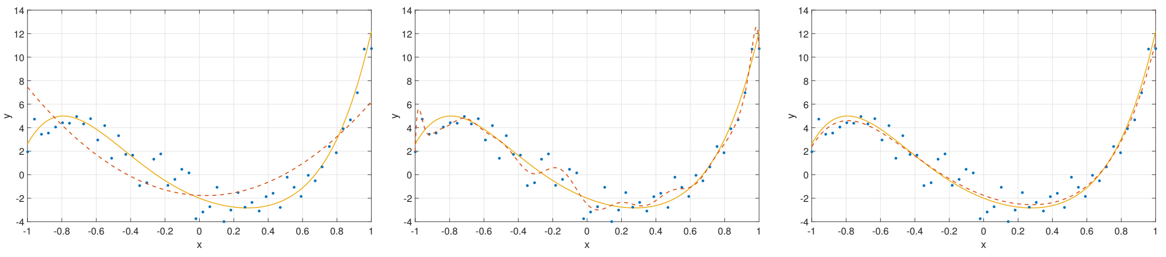
We can answer this question by thinking about the potential negative consequences of omitting or including predictors. The first negative consequence arises if we omit a predictor from our model and it turns out be important/associated with our target. This is called "underfitting". The consequence of underfitting is a systematic error ("bias") in predicting the target. This is intuitive: if a variable is associated with the target, and we don't include its effects, then we will be unable to make as good predictions about y as we potentially could. As an example, imagine we were trying to predict the time it would take for a plane to travel a certain distance and we did not include key information such as the maximum speed of the plane and the load the plane is carrying. The predictions would undoubtedly be poor.

Given the negative outcomes of omitting important variables, the second question is: why not just include all the predictors we have? The answer is that if we include spurious, or unassociated predictors, then we will learn noise or random patterns in our data, and the resulting model will have a poorer ability to predict onto new, unseen data from the same population. This phenomenon is called "overfitting", as we have moved beyond learning real patterns in the data and have begun to memorise/learn spurious patterns that are not reproducible in future samples.

Example: Overfitting and Underfitting. To demonstrate the ideas underlying under and overfitting more visually we examine a very typical example of these phenomena. Imagine we had observed n measurements of a variable x and a target y, and we want to build a model to predict the values of y given values of x, i.e., learn the relationship between x and y. As we learned in Lecture 6, a simple and flexible approach to doing this even if the relationship is nonlinear is to use polynomial regression. That is, we take x and transform it into p polynomial terms:

$$x \Rightarrow (x, x^2, x^3, \ldots, x^p)$$

This non-linear transformation creates a total of p predictor variables (the original x, i.e., the linear term plus $p-1$ polynomial terms) and if the variable y is linearly related to any of these polynomial



(a) Quadratic polynomial - underfitting (b) 20th order polynomial – overfitting (c) 5th order polynomial – "just right"

Figure 1: Modelling the relationship between two variables x (the predictor) and y (the target) using three different polynomial models. This figure clearly demonstrates how using too simple a model leads to systematic error due to inability to learn the underlying relationship, and how using too complex a model leads to error due to learning spurious patterns ("noise") in our data.

terms, we can capture the relationship between x and y using a linear regression model. The question is how many terms to include? The higher p, the more predictors we create, and the better the fit our model will have to the data. This is guaranteed as including more predictors in a linear model *always* reduces the residual sum-of-squares and hence improves the fit to the training data. But as hinted at above, improving the fit to the training data does not necessarily translate to better performance on new testing data drawn from the same population because we can end up learning *noise*. On the other hand, we want to include enough polynomial terms to adequately model the unknown relationship.

Figures 1(a) through to 1(c) show an example simulated dataset that demonstrates these concepts. The "true", population relationship between x and y is shown by the yellow curve. It is clearly non-linear, increasing with increasing x, then decreasing, then finally increasing again. The blue dots show our sample of $n = 50$ noisy observations y, and the dashed orange lines show the fit of three different models. In Figure 1(a) we see the fit when using a quadratic polynomial, i.e.,

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon$$

Comparing this to the unknown population relationship, it is clear that this model is too simple – it is able to capture only part of the true relationship and using this to predict onto future data would lead to large systematic errors, particularly around $x = -1$, $x = -0.6$, $x = 0.4$ and $x = 1$. A quadratic polynomial simply cannot model a relationship that increases, decreases and increases again, and so we are underfitting. Figure 1(b) shows the fit that we obtain if we use $p = 20$ polynomial terms; this is a very complex model that can model very complicated relationships between x and y. This more complex model clearly fits our *data* (the blue dots) much better than the previous simple model. However, the complexity of the model is now a problem as even though the model captures the overall increase-decrease-increase behaviour of the population curve it has also learned a large number of extra deviations that are really just artefacts of the noise; for example, the fitted curve fluctuates wildly near $x = -1$ while the population curve we are trying to learn does not. We are clearly overfitting and learning aspects of the relationship that are not really there, but are rather spurious patterns due to noise.

Finally, Figure 1(c) demonstrates the fit we would obtain on this data if we used $p = 5$, i.e., a $5th$-order polynomial. In this case we see the fitted curve closely matches the unknown population relationship, and appears to be about "just right". Of course the curve will never exactly match the true population curve due to the noise/randomness in our measurements, but this model appears to be a pretty good fit to the population curve. The obvious question is: how do we find this "best fitting" model using nothing but the data?

Mean-Squared Prediction Error. One way of thinking about these problems more formally is to consider the average error our fitted model would incur when making predictions about new data sampled from our population. This is a generalization of the idea of "mean-squared error" we examined in Lecture 3 when characterising the behaviour of estimators of parameters. To formalise this we first must introduce the idea of a true, underlying population model; this is the model that "generated" the data we sampled. Our aim is obviously to build a model from our data that is close to this true, population model. For regression problems we can write a description of the population model as:

$$\mathbb{E}\left[Y \mid x_1, \ldots, x_p\right] = f(x_1, \ldots, x_p)$$

where $f(\cdot)$ is some arbitrary function relating the predictors to the expected value of our target Y. In practice we usually don't know anything about the population model $f(\cdot)$ and must make our model building decisions based primarily on the data sample we have gathered. We introduce the term training data to refer to the sample of data $\mathbf{y} = (y_1, \ldots, y_n)$ we have collected. We use this data to build a model $\hat{\mathcal{M}}(\mathbf{y}) \equiv \hat{\mathcal{M}}$ that we will use to estimate the unknown relationship $f(\cdot)$. In the case of a linear regression, for example, our model $\mathcal{M}$ is a particular linear combination of:

- predictors $x_1, \ldots, x_p$, and
- specific non-linear transformations of these predictors.

In general we would like for the model that we end up with to be able to predict well onto new, unseen data that we might collect from the same population we gathered our training data from. If it predicts well then we can use the predictions of the model to make decisions or inferences about the population we are trying to model. We call this the model's ability to generalize to unseen data that it was not trained on.

To formally measure how well a model can generalize we can think about the situation in which we somehow obtain a new set of n' observations from the same population, say $\mathbf{y}' = (y'_1, \ldots, y'_{n'})$, along with their measured predictors, and use this data to test how well our model generalizes. We call this new data the testing data. To test the model $\hat{\mathcal{M}}(\mathbf{y})$ we have learned from our training data on our testing data we first make predictions using our model for each of the individuals in our testing data:

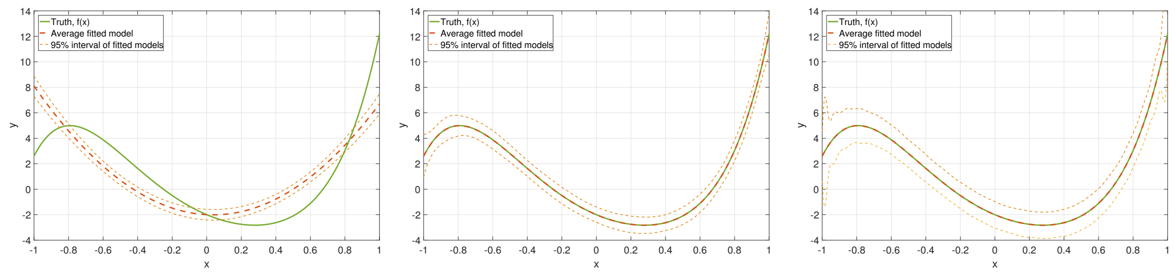
$$\hat{y}_i(\hat{\mathcal{M}}(\mathbf{y})) = \hat{y}\left(x'_{i,1}, \ldots, x'_{i,p}, \hat{\mathcal{M}}(\mathbf{y})\right)$$

using the predictors $x'_{i,1}, \ldots, x'_{i,p}$ associated with each of the individuals in the testing data. We can then measure how close our *predictions* are to the actual values of y associated with these new testing individuals by computing the mean-squared prediction error:

$$\text{MSPE}(\hat{\mathcal{M}}(\mathbf{y})) = \frac{1}{n'} \sum_{i=1}^{n'} (y'_i - \hat{y}_i(\hat{\mathcal{M}}(\mathbf{y})))^2 \tag{1}$$

This captures how much our predictions about the new individuals from the testing data, made use the model we learned on the training data, deviate on average from the actual values of the targets of these individuals. The smaller the MSPE, the more accurate our predictions are. Therefore we would ideally like to find a model with a small MSPE.

Bias and Variance. One way of understanding the effects of underfitting and overfitting is through their effect on the MSPE described above. An important thing to remember about the MSPE calculated by equation (1) is that it is the mean-squared prediction error of the particular model $\hat{\mathcal{M}}(\mathbf{y})$ fitted



(a) 2nd order polynomial:
EMSPE $= 3.33$,
bias$^2 = 3.27$,
variance $= 0.058$

(b) 7th order polynomial:
EMSPE $= 0.147$,
bias$^2 = 0.0$,
variance $= 0.147$

(c) 20th order polynomial:
EMSPE $= 0.486$,
bias$^2 = 0.0$,
variance $= 0.486$

Figure 2: Average and 95% interval of predictions produced by polynomial regression on 10,000 training data sets sampled from the true population (a 5th order polynomial) along with the expected mean-squared prediction error (EMSPE), squared-bias and variance. Note how the intervals are wider for the more complex models due to the extra variance introduced by estimating more parameters.

on the particular training sample $\mathbf{y}$. If we sampled a different training sample from our population and learned a different model, we would get different predictions onto our testing data, and therefore a different MSPE.

If we remember our population-sample-model framework of statistical learning, the training sample we have observed is just one of an infinite number of different training samples of size n we might have observed, but did not. To remove the dependency of (1) on the particular training sample we could instead average the MSPE over all the different possible models we would learn from all the different possible training samples of size n we might draw from our population. We call this the expected MSPE, and write this as

$$\text{EMSPE} = \mathbb{E}\left[\text{MSPE}(\hat{\mathcal{M}}(\mathbf{y}))\right] = \text{bias}^2 + \text{variance}$$

where the expectation is over all the possible samples of size n we could draw from our population, and

- "bias" measures the systematic error of our model at predicting future data from our population;
- "variance" measures the error in predicting future data from our population due to random variation (i.e., the fact our training data is a noisy random sample)

Therefore, to minimise MSPE we would like to simultaneously keep both the bias and the variance down.

Example: Bias and Variance. We revisit our previous example of learning the relationship between a variable x and a target y using polynomial regression. Figures 1(a) through to 1(c) demonstrated underfitting and overfitting on a particular sample of data (the blue dots). Let us now look at this example in terms of the expected MSPE behaviour.

To do this I undertook the following experiment: I first generated 10,000 different training samples of size $n = 50$ from the true population model. This approximates "all the possible training data sets we might draw from our population". Then, for each sample I fitted a 2nd-order polynomial model to each of the training samples, giving us 10,000 different models. Finally I produced predictions using each of these models for $5,000$ equally spaced x' values from -1 to 1, and then calculated: (i) the average prediction of the $10,000$ models at each of these points, (ii) the variance of the predictions of the $10,000$ models at each of these points, and (iii) the squared-bias and variance for the overall predictions for the $10,000$ models. I then repeated this for 7th order and 20th order polynomial models.

For visualisation purposes I plotted the average predictions along with the 2.75th and 97.5th percentiles of the predictions made by the $10,000$ models for each of the three different complexity models. These are shown in Figures 2(a) through 2(c). Note that the true population relationship (the green line) is actually a specific 5th order polynomial.

In Figure 2(a), the 2nd order polynomial is too simple to properly model the unknown true population relationship between x and y, which happens to be a 5th order polynomial. We see that the average prediction made by a 2nd order polynomial fitted to a random sample of size $n = 50$ from our population is given by the thick dashed line, and demonstrates clearly the systematic error such a simple model has in this case. The average fit is very poor and has a high bias due to underfitting. It doesn't seem to really matter what particular sample we draw from our population, the 2nd order polynomial is just too simple to capture the population curve well. However, on the plus side the model is simple and the variability in the fit (the light dashed lines) is not great, so the variance is low. However, the overall EMSPE (sum of bias-squared and variance) is high due to the fact the model is too simple.

Figure 2(b) shows the results of using a 7th order polynomial. We know the truth happens to be a 5th order polynomial, so our model is unnecessarily complex. We are including unassociated predictors (x^6 and x^7). These extra predictors introduce a relationship between predictor and target that is not really there, and the additional patterns we learn are just noise – they do not replicate on future data. The results show the *average predictions* of a 7th order polynomial fitted to a random training sample is perfect in this case, so there is no bias. The problem is that the variability is inflated due to the inclusion of the additional unnecessary x^6 and x^7 terms. The bias is zero but the variance is larger than for the quadratic model; however, overall, the EMSPE is much smaller because we have reduced the bias due to systematic error to zero, which is a substantial reduction over the simple quadratic model. Finally Figure 2(c) shows the results for the 20th order polynomial. This quite substantially overfits the truth, which is a 5th order polynomial, by including an additional 15 terms (x^6 through to x^{20}). Once again, the bias is zero because the model is more complex than the truth, but the variance is now much greater (as seen by the wider 95% intervals on the predictions) due to the estimation of 15 unnecessary parameters. This model has an overall worse EMSPE than the 7th order polynomial due to increased variance.

Summary. In general we can say that:

- Simpler models have (potentially) higher bias and (always) lower variance as they have less flexibility and less parameters to learn;
- More complex models have lower bias and higher variance, as they are more flexible and can learn more complicated relationships. The price they pay for this flexibility is an increased propensity for learning "noisy" patterns in the data;
- Variance will always decrease with increasing training sample size as we get more information about our population, but bias will not; if our model is not complex enough to learn the true population relationship well then it does not matter how much data we obtain, we will always have a systematic error;
- Variance increases with the number of predictors we include, as there are more parameters to try and estimate from the random training data.

Model selection for prediction is about balancing the bias and variance. We want a trade-off of a model that is complex enough to fit the truth well, but not too complex so that we are not learning spurious, noisy patterns.

In linear and logistic models the complexity of a model is increased by having more predictors, or more transformations of predictors. In general, we have the following rule: omitting a predictor will always decrease variance, but may lead to an increase in bias if the predictor is important. However, model selection even for linear/logistic models is quite subtle. There are a few interesting aspects that make the problem quite delicate:

- In linear models, simply including all the predictors that are associated with the target is not necessarily enough to keep the bias low. If some of the predictors are associated with the target in a non-linear fashion, we will also need to include an appropriate number of transformations.
- It is possible to overfit and underfit at the same time. Imagine we fit a model that includes an unimportant predictor and omits an important one. Now we have excess variance due to the unimportant predictor and a systematic error due to the omission of the relevant predictor.
- Sometimes, we can actually improve our prediction error by not including a relevant (associated) predictor. This can happen if the association is weak and the noise is large. In this case it may be hard to actually get a good estimate of the coefficient, and we might do better by not trying.

In general, we use a combination of real world and domain knowledge coupled with statistical tools when trying to build models and control how complex they should be.

2 Selecting Predictors

Hypothesis testing and Multiple Testing Problems. One technique for determining whether to include a predictor in a linear/logistic model is hypothesis testing. Here we exploit the fact that a coefficient of $\beta_j = 0$ implies that predictor j is not associated with the target $\mathbf{y}$. We can then test the hypothesis

$$H_0 : \beta_j = 0 \text{ vs } H_A : \beta_j \neq 0$$

with smaller p-values being stronger evidence against the null hypothesis that $\beta_j = 0$, i.e., that there is no association. If the p-value is sufficiently small we may reject the null and conclude that there is an association between predictor j and the target.

In practice there is a potential problem with this approach. Imagine we decided to use p-values to decide whether to include a predictor in a model or not. We might choose to include the predictor if the p-value is less than some nominal value, say $\alpha = 0.05$. If we were only testing a single predictor and rejected the null (of no association) if the p-value was less than 0.05, then 5% of the time we would erroneously include the predictor in our model even if it was not associated with the target at the population level, i.e., we would overfit 5% of the time. But if we have p predictors, then we are doing p tests, not a single test. So now, even if all p predictors were unassociated with the target, we would expect $0.05\,p$ of the tests to have p-values less than 0.05 *just by chance.* If p is big enough, this means we will end up including a lot of unimportant variables in our model. For example, if $p = 1,000$, then 50 predictors will on average have p-values less than 0.05 *even if none of them are associated with the target y.* This can easily happen if we take our original predictors and include many transformations of them, i.e., logs, squares, interactions etc. This problem is sometimes referred to as data dredging or "p-hacking", because if we include enough variables some of them will eventually have p-values smaller than 0.05 just by chance.

More formally this issue is called the multiple testing problem. There are a lot of ways we could try and resolve this problem, but the one we will examine is called the Bonferroni procedure. Let α

be our significance level; for example, it is common to use $\alpha = 0.05$. Then, if we do p different tests then the Bonferroni procedure says we should only reject the null hypothesis for each of the tests if

$$p\text{-value} < \frac{\alpha}{p}.$$

That is we divide the significance level we would be happy with if we were doing one test by the number of tests we are doing to get our new adjusted level. For example, if $\alpha = 0.05$, and we tested $p = 1,000$ different predictors for inclusion in a linear model, then we would need to see p-values smaller than $0.05/1000 = 5 \times 10^{-5}$ to reject the null hypothesis and believe that the predictor is associated with the target.

The Bonferroni procedure guarantees that the probability of including at least one unassociated/unimportant predictor is α. This is very conservative but at least you can be sure that if your predictors have p-values that pass the Bonferroni procedure they are very likely to be relevant.

Likelihood and Information Criteria. We learned in Lecture 3 that the negative log-likelihood $L(\mathbf{y} \,|\, \hat{\beta}_0, \hat{\beta})$ is a measure of how well a probability model fits a data sample. The smaller the negative log-likelihood, the more probability the model assigns to the data sample, and the more compatible the sample is with the model. For a model with probability distribution $p(y_i \,|\, \hat{\theta})$, where $\hat{\theta}$ are estimates of model parameters, the negative log-likelihood of the data under the model is

$$L(\mathbf{y} \,|\, \hat{\theta}) = -\sum_{i=1}^{n} \log p(y_i \,|\, \hat{\theta})$$

We can use the negative log-likelihood as a guide for selecting models. The problem with using it directly is that the more complex a model is, the better it fits the training sample and the smaller the negative log-likelihood will be. For example, in the case of linear models, the negative log-likelihood always decreases as we add more predictors. We saw in Lecture 6 the way to counter this is to form an information criterion score that penalizes the negative log-likelihood by the number of predictors we include in the model. The idea is the model selection criterion assigns a score to a model which takes into account how well the model fits the data (likelihood) as well as how complex the model is. We then choose the model that minimises this score. More generally, let

- $\mathcal{M}$ denote a model (set of predictors to use, for example, in the case of linear/logistic models);
- $\hat{\mathcal{M}}$ denote the model fitted to data $\mathbf{y}$;
- $L(\mathbf{y} \,|\, \hat{\mathcal{M}})$ denote the minimised negative log-likelihood for the model $\mathcal{M}$;
- $k_{\mathcal{M}}$ denote the number of predictors in model $\mathcal{M}$.

The following information criteria are commonly used in the literature:

- The Akaike information criterion (AIC):

  $$\text{AIC}(\hat{\mathcal{M}}) = L(\mathbf{y} \,|\, \hat{\mathcal{M}}) + k_{\mathcal{M}}$$

  and the Kullback–Leibler information criterion (KIC):

  $$\text{KIC}(\hat{\mathcal{M}}) = L(\mathbf{y} \,|\, \hat{\mathcal{M}}) + 3/2\, k_{\mathcal{M}}$$

  Both of these criteria tend to lead to some overfitting, even for large n. However, this is not necessarily a problem as moderate overfitting is less damaging for large n as there is more information in our training sample with which to estimate the parameters more accurately, and both rarely underfit which reduces problems with bias. In practice, KIC often performs better than AIC by suffering from less overfitting.

- The Bayesian information criterion (BIC):

$$\mathrm{BIC}(\hat{\mathcal{M}}) = L(\mathbf{y} \mid \hat{\mathcal{M}}) + k_{\mathcal{M}}/2 \log n$$

  where n is the number of samples used to fit $\hat{\mathcal{M}}$. The BIC has the interesting property that as n increases the probability of overfitting goes to zero. However, this leads to the BIC being more conservative, and for small n it can sometimes suffer from too much underfitting.

- The Risk inflation information criterion (RIC):

$$\mathrm{RIC}(\hat{\mathcal{M}}) = L(\mathbf{y} \mid \hat{\mathcal{M}}) + k_{\mathcal{M}} \log p$$

  where p is total number of possible predictors. The idea behind the RIC is similar to the multiple testing ideas: the more predictors we try, the more conservative we need to be to avoid overfitting. The RIC can be quite conservative but is usually recommended over AIC/KIC if p is very large.

Cross-Validation. The method Cross-validation (CV) is a very general strategy for controlling model complexity that is used frequently in practice. The motivation behind cross-validation is very intuitive which is another reason it is very popular. Ideally, from our previous discussion, we know that we would like to select the model that minimises the prediction error onto future data $y'_1, \ldots, y'_m$

$$\mathrm{MSPE}(\hat{\mathcal{M}}(\mathbf{y})) = \frac{1}{m}\sum_{i=1}^{m}(y'_i - \hat{y}_i(\hat{\mathcal{M}}(\mathbf{y})))^2$$

The problem with this "strategy" is that it requires knowing our testing data ahead of time, which is not possible. However, even though we don't have access to future testing data, we do know that the data $\mathbf{y}$ we are using to fit our model is generated by the same population that we would like to predict on in the future. So we can use the data sample we have collected to try and estimate the MSPE. To do this we take our sample, and divide it into two parts: a training sample, which we use to build our model, and a testing sample which we use to assess how well our model predicts onto "new data". In this way we simulate the procedure of drawing new testing data from our population. More generally, the basic cross validation algorithm is:

1. We randomly partition the data sample $\mathbf{y}$ we have into two sets:
   - A training set $\mathbf{y}_{\text{train}}$,
   - and a testing set $\mathbf{y}_{\text{test}}$
2. Fit a model $\mathcal{M}$ to the training data $\mathbf{y}_{\text{train}}$
3. Compute the MSPE of this model on the withheld testing data $\mathbf{y}_{\text{test}}$
4. Repeat this procedure $m > 1$ times and average the MSPE

This gives us an estimate of how well our model would predict future data from the same population. We can then do this for all the different models we are considering and then choose the one that has the smallest CV error, the hope being that it will also have the smallest prediction error on actual new data from the same population. CV has one obvious problem: the CV scores will vary from run to run of the CV algorithm because of the random way in which the data is partitioned. To counter this we can make the number of repetitions m as large as possible to keep the variability of our estimate down. There are two specific variants of CV that are often used in statistical packages:

1. The first is called K-fold CV, and this works by split into K equal sized non-overlapping random partitions. We then train our model on $K - 1$ of the partitions and test the model on the remaining partition; we do this for all K combinations and repeat this m times, and compute the average prediction error achieved by the model. Usually $K = 10$ performs well, and the larger the number of repetitions m, the more stable the estimate of the prediction error will be.

2. The second variance is called Leave-one-out CV (LOO CV). This works by training on $n - 1$ of the data points and using the remaining data point for testing. We can then repeat this process for all n possible partitions. Thus the number of repititions is fixed at n and the CV estimate will be the same every time we run LOO. This is one of the advantages of this variant.

Overall, cross-validation is popular because it is simple to implement and very flexible. There is no need for special derivations or extra mathematics; we just run CV, partition our data using whatever strategy we have chosen and estimate the MSPE. A drawback is that it is potentially slow if model fitting is computationally expensive, as we need to fit many models as part of the CV process.

3 Penalized Regression

Statistical Instability. For linear and logistic regressions we have learned several different algorithms that can be used in conjunction with information criteria to try and select a good set of predictors: (i) all-subsets selection which tries all combination of predictors to find the model with the smallest model selection criterion score, (ii) the forward selection algorithm, in which we start with an empty model containing no predictors, and iteratively enlarge the model by including the most important predictor among those that are currently not in the model; and (iii) backwards selection in which we start with the full model including all predictors and iteratively remove the least important predictor from the model. See Lecture summary 6 for more details on these algorithms.

However, there are a number of potential problems with these traditional methods. The all-subsets selection procedure is infeasible for moderate numbers of predictors due to the exponential growth in the number of predictor combinations we need to try, i.e., if we have p predictors there are 2^p different combinations of these predictors we could form. The large number of different models we are testing/trying also results in multiple hypothesis testing problems. Stepwise methods are more computationally tractable than all subsets approach but are adversely affected by correlation in predictors as it becomes difficult to determine which of many similar predictors we should include. They are also slow when there are a large number of predictors to try. However, perhaps most important, is statistically instability. A procedure is said to be statistically unstable if small changes in the training data result in big changes to the model that the procedure produces. All of the above methods suffer from high instability.

Statistical Instability: Example. To see what effect this instability has, let us consider the following experiment. I took a well known example dataset of $p = 10$ predictors measured on $n = 354$ individuals. The target `Y` is a measure of diabetes progression of an individual, and the predictors are: `AGE`, `SEX`, `BMI`, `BP` (blood pressure) and `S1`,...,`S6` (six blood serum measurements).

I wrote some R code to do the following procedure: I removed one of the 354 samples at random from the dataset, and then found "best" combination of predictors using all subsets. To do this I checked all possible (1024) combinations of predictors, and selected the combination that had the smallest cross-validation error. I then repeated this experiment four times, and recorded which predictors appeared in the best model for all these four slightly modified datasets. The results are shown in Table 1.

As is immediately obvious, each of the four experiments resulted in all-subsets selection choosing a different model! All we did in each experiment was remove a *single* data point from the $n = 354$ data points, but the "all-or-nothing" nature of this type of model selection (either completely exclude or

Test	AGE	SEX	BMI	BP	S1	S2	S3	S4	S5	S6
1		✓	✓	✓	✓	✓	✓		✓	
2		✓	✓	✓	✓	✓	✓			
3	✓	✓	✓	✓	✓	✓	✓	✓		
4	✓	✓	✓	✓	✓	✓	✓		✓	

Table 1: Predictors chosen by selecting model that minimises cross-validation score using all-subsets selection method. For each test, a single one of the $n = 354$ samples was removed at random.

completely include a predictor) means that the "best" model can change substantially with only a small change to the data. One can only imagine how unstable such a method is when the sample size is small!

Penalized Regression. An alternative to using information criteria to decide whether to include predictors is to directly apply a "penalty" to the coefficients when we are estimating them from the training data. For linear regression, we have the penalized least-squares approach which says to find the estimates of the coefficients that minimise a penalized residual sum-of-squares:

$$\left(\hat{\beta}_0, \hat{\boldsymbol{\beta}}_\lambda\right) = \underset{\beta_0, \boldsymbol{\beta}}{\arg\min} \left\{ \text{RSS}(\beta_0, \boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} g(\beta_j) \right\} \tag{2}$$

In comparison, the regular least squares we saw in Lecture 6 only considers the minimisation of the goodness-of-fit (RSS) and does not take into account the values of the coefficients. An obvious first question is: how do we choose the penalty function $g(\cdot)$? Well, in general, we choose it to increase with increasing magnitude of β_j, so that the larger $|\beta_j|$ is the more penalty is applied. This approach generalises easily to logistic regression, i.e., penalized maximum likelihood.

An obvious second question is: why do we choose a penalty of this form? The following provides the intuition behind penalised least squares. We note that large values of the coefficients imply strong effects, i.e., if a coefficient is large then the change in expected value of the target $\mathbb{E}[Y]$ is large per unit change in the predictor. We also note that a coefficient of zero implies *no association* between the predictor and the target. Therefore our penalty should encourage smaller values of the coefficients to try and reduce overfitting. Note that we *never* penalize the intercept β_0 as the value of this is determined by the choice of unit of measurement for the target y and therefore $\beta_0 = 0$ is not a 'special value'.

For this interpretation of the size of $|\beta_j|$ to be sensible we need to standardise our predictors before fitting, i.e., ensure that

$$\sum_{i=1}^{n} x_{i,j} = 0 \text{ and } \sum_{i=1}^{n} x_{i,j}^2 = n$$

This procedure is easily done by subtracting mean and dividing by standard deviation, and is usually handled by packages that implement these techniques. The idea behind such a standardisation is that each predictor is then unitless and on the same (arbitrary) scale, which further implies that a larger β_j means a stronger association.

The penalty term in (2) includes a user-chosen hyperparameter λ. A hyperparameter is a parameter of the learning/estimation method that is not actually a parameter of the final model. In this case, the value of λ controls how *strong* the penalty is. Generally, for penalized regression:

- As λ goes to zero, the estimates of $\boldsymbol{\beta}$ are closer to least-squares because the penalty has less effect;
- As λ gets larger, the estimates of $\boldsymbol{\beta}$ become smaller as the penalty as more effect.

If $\lambda = 0$ there is no penalty and (2) reduces to the usual least-squares method. Therefore, we can interpret the choice of λ as setting the "complexity" of the resulting regression model. By varying λ from zero to a large value we can generate a "path" of regression models, one for each solution of (2) for each value of λ that we are trying. The models become more complex as we move along the path as the penalty becomes less and less strong and the models fit the data better and better.

The penalized regression approach has several advantages over stepwise selection methods. The first is a dramatic increase in statistical stability. Small changes in the training data (removing or adding a data point) will result in small changes to the model that we end up with, in contrast to the all subsets example presented previously. Another advantage is that they can be directly applied even when the sample size is smaller than the number of predictors (the $n < p$ setting), which is difficult for some other methods. Finally, they can more gracefully handle correlation between predictors. When at least two of the predictors are highly correlated it can be problematic because it becomes is difficult to distinguish between their relationship with the target, as the two predictors convey similar information. In traditional least-squares this leads to increased variance; in stepwise methods, this dramatically increases instability as the choice of which predictor to include becomes more difficult. We now examine the two most popular penalized regression methods.

Ridge Regression. Ridge regression was the first penalized method introduced and was first proposed back in the 1970s. The ridge regression estimator is found by solving

$$\left(\hat{\beta}_0, \hat{\boldsymbol{\beta}}_\lambda\right) = \underset{\beta_0, \boldsymbol{\beta}}{\arg\min} \left\{ \mathrm{RSS}(\beta_0, \boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} \tag{3}$$

so that the penalty is $g(\beta_j) = \beta_j^2$, i.e., the squared value of the coefficient. This satisfies our criteria of being an increasing function of the magnitude of the coefficient, and is smallest when $\beta_j = 0$. Ridge regression has several strengths:

- It is very quick to run, even for very large n and p;
- It produces *very* stable, low variance estimates even when many predictors are highly correleated.

However, it also suffers from one very distinct weakness: just like least-squares, it cannot estimate coefficients to be exactly zero, i.e., it cannot do variable selection. This means that no matter what choice of $\lambda < \infty$ we take the solution to (3) will never have any of the coefficients equal to exactly zero. The only way to set coefficients to zero is to take $\lambda = \infty$, in which case *all* the coefficients are set to zero.

Lasso regression. A more modern penalized regression estimator, introduced in the 1990s is lasso regression. The lasso regression estimator is found by solving

$$\left(\hat{\beta}_0, \hat{\boldsymbol{\beta}}_\lambda\right) = \underset{\beta_0, \boldsymbol{\beta}}{\arg\min} \left\{ \mathrm{RSS}(\beta_0, \boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} |\beta_j| \right\} \tag{4}$$

so that the penalty function is $g(\beta_j) = |\beta_j|$, i.e., the absolute value of the coefficient. Remarkably, the use of the absolute value rather than the square in the penalty leads to a method with very different behaviour to ridge regression. The lasso has several strengths:

- The lasso is very stable, and small changes to the training sample will result in similar lasso estimates;
- As with ridge regression, efficient algorithms exist that can solve (4) even for large p or n.

- In contrast to ridge regression, the lasso can estimate coefficients to be exactly zero, i.e., it can perform variable selection

The last advantage stems from the fact that for particular values of λ, the lasso estimates that solve (4) will result in some of the values of the coefficients being *exactly equal to zero*. Thus the lasso simulteneously performs variable selection and estimation of the parameters, which is very useful. However, the lasso also has several weaknesses:

- The nature of the penalty means that it produces biased estimates for large coefficients (smaller on average than they should be);
- Correlated predictors can be problematic and are not handled as well as by ridge regression
- The lasso is known to potentially overfit (include unassociated predictors) in real data situations.

Example of Paths. To demonstrate how the ridge and lasso regression work we ran the R `glmnet` package on the diabetes progression dataset we examined above for both ridge regression and lasso regression. Figure 3(a) and 3(b) show the "paths" produced by the package for both methods as λ is varied. For each λ value the values of the coefficients associated with the $p = 10$ predictors are plotted. We see the key difference between ridge and lasso is that for the ridge path none of the coefficients are equal to zero without all of them being equal to zero, so that for every value of $\lambda < 80$ all predictors are included in the model. In contrast, we see that for the lasso some of the coefficients disappear to zero while others are still non-zero. For example, for the lasso, when $\lambda = 30$ all the coefficients are non-zero but when $\lambda = 60$ only three of the coefficients remain non-zero (and subsequently, the model contains only three predictors).

Choosing λ. An obvious question remains: how to choose the hyperparameter λ? It is clear that for both ridge regression and lasso regression the choice crucially affects what model we end up with. A standard procedure is to use cross-validation to choose the value of λ that results in the best (estimated) prediction error onto new data. Specifically, we do the following:

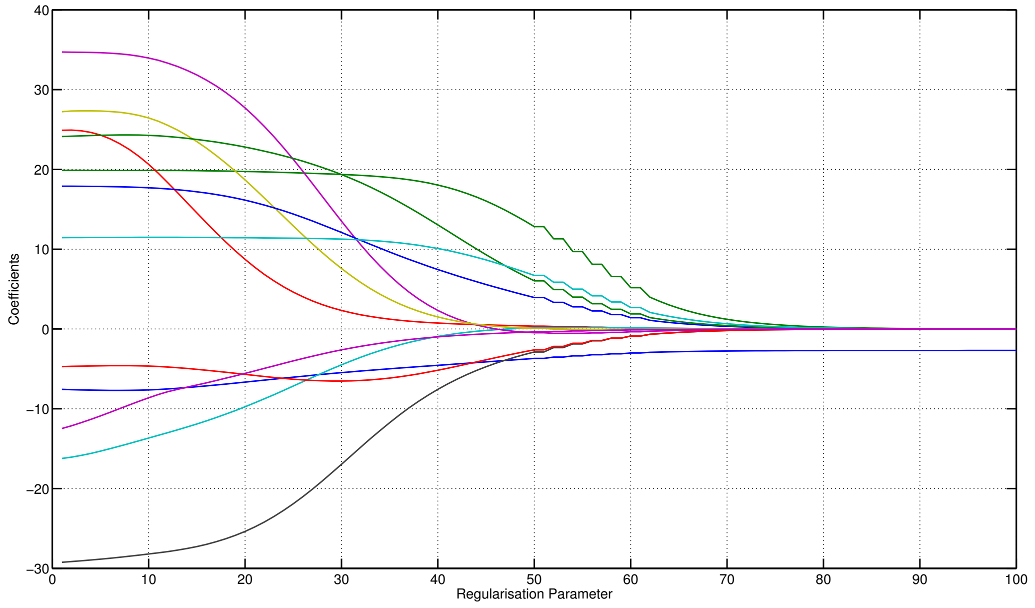
1. Vary λ over some pre-specified grid of values from $\lambda = 0$ (no penalty) to λ large (big penalty);
2. For each λ we trying, use cross-validation to estimate prediction error;
3. Select the λ with smallest cross-validation error;
4. Use that λ to estimate our final model from all the data.

There are a number of efficient implementations in standard packages: (i) in MATLAB we can use the `lasso()` and `lassoglm()` functions; (ii) in R, we use the `glmnet` package which works efficiently even for very large number of predictors p.

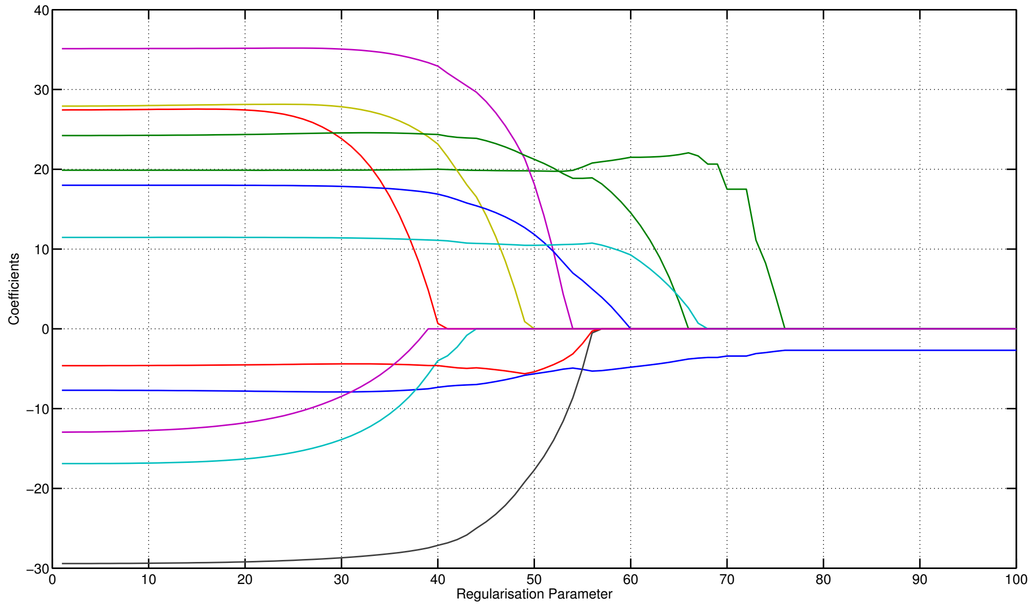
Links to Bias/Variance Trade-Off. A different way of thinking about penalized regression is through a bias/variance motivation. Recall that the expected mean-squared prediction error EMSPE of a model can be written as

$$\text{EMSPE} = \text{bias}^2 + \text{variance}$$

It turns out that if all relevant predictors are included then the usual least-squares is unbiased in the sense that they will neither over- nor underestimate the true population values of the coefficients, on average. However, the variance of the least-squares estimates can be high because we do not constrain them at all. Penalized regression introduces some bias through the penalty but as a result reduces variance of the estimates by shrinking the coefficients towards zero. In this way choosing a good λ means we can reduce the overall EMSPE by (potentially) increasing bias by a small amount but reducing variance by a corresponding large amount.



(a) Ridge regression



(b) Lasso regression

Figure 3: Coefficient paths produced by ridge regression and lasso regression for the diabetes data set with $p = 10$ predictors plus an intercept as the hyperparameter λ is varied. Note how the estimated coefficients get smaller as the penalty λ increases, and that the intercept (blue line) never goes to zero as we do not penalize this.