Machine Learning - Michaelmas Term 2016
Lectures 4, 5 : Basis Expansion, Regularization, Validation

Lecturer: Varun Kanade

1 Basis Function Expansion

So far we've studied the linear model that can only capture a linear relationship between the output y and the inputs $\mathbf{x}$. Basis function expansion is an easy way to capture non-linear relationships.

1.1 Polynomial Basis Expansion

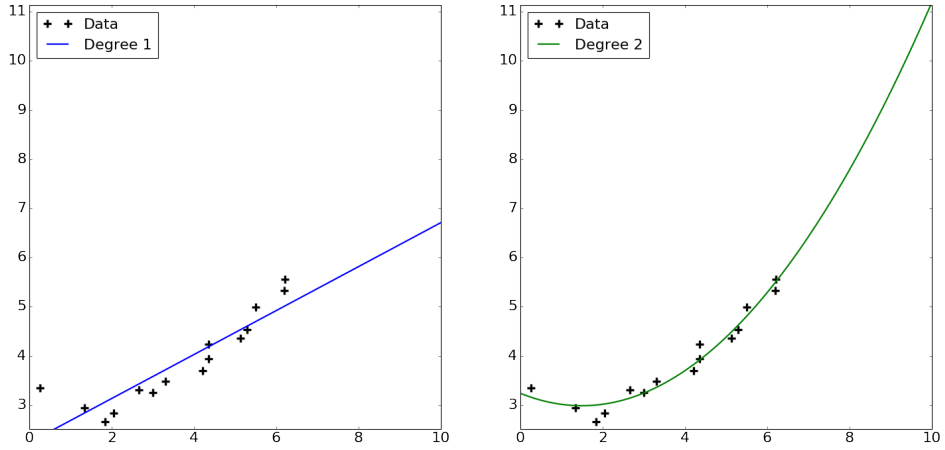
Let us first look at a simple example when the input is one dimensional. Suppose, we want the output to be a quadratic function of the input, then we can use the features 1, x and x^2, as inputs and then simply fit a linear model. Essentially, we've mapped the input from a 1-d space to a 3-d space, but the model in the 3-d space is just a linear model!

We'll denote such a feature expansion of input x by $\phi(x)$. In this case, $\phi(x) \in \mathbb{R}^3$, $\phi(x) = [1, x, x^2]^\mathsf{T}$. For $\mathbf{w} \in \mathbb{R}^3$ a linear model of the form $\mathbf{w} \cdot \phi(x)$ captures a quadratic relationship between the output y and the input x. Figure 1(a) and 1(b) show a linear model and a quadratic model fit to the same dataset; it can be seen that the qudratic model is a better fit. In general, it is possible to fit more complex models using a feature expansion that includes higher degree terms. The degree d feature expansion is given by $\phi(x) = [1, x, \ldots, x^d]^\mathsf{T}$. When using higher degree polynomials there is a danger that we may overfit the data, *i.e.,* use a model that works very well for the training data but will most likely peform badly on unseen data. We will discuss overfitting in much greater detail below. Figure 1(c) shows degree 7 and 14 polynomials fit to the same data, both of which are unnaturally "wiggly" and are probably overfitting. In fact, if we have N data points and use degree $d = N - 1$, then there exists a polynomial of degree d that fits all the data perfectly! Having large quantities of data can alleviate the problem of overfitting significantly as shown in Figure 1(d). In this case, all of the degree 2, 7 and 14 polynomials fit the data well; however, the linear model still performs poorly because it is incapable of capturing the true relationship between input and output. This last problem is referred to as *underfitting*.

Polynomial basis function can be carried out in higher dimensions. For example, in order to fit a quadratic function in 2 dimensions, for vector $\mathbf{x} = [x_1, x_2]^\mathsf{T} \in \mathbb{R}^2$, we use the feature expansion $\phi(\mathbf{x}) = [1, x_1, x_2, x_1^2, x_2^2, x_1x_2]^\mathsf{T}$. Now if we fit a linear model using $\phi(\mathbf{x})$ as the input, the output y can capture be an arbitrary quadratic function of the input. Figure 2 shows a linear model and a quadratic model (linear model with degree 2 basis expansion) in two dimensions. We can calculate the dimension in which the feature vector $\phi(\mathbf{x})$ lies if we want to fit degree d polynomials and the original input $\mathbf{x}$ is in D dimensions; this number is very large, roughly D^d. The large number of parameters required to fit such models will present both computational (requiring more computational resources) and statistical (requiring more data) challenges. We'll discuss to what extent these challenges may be addressed.

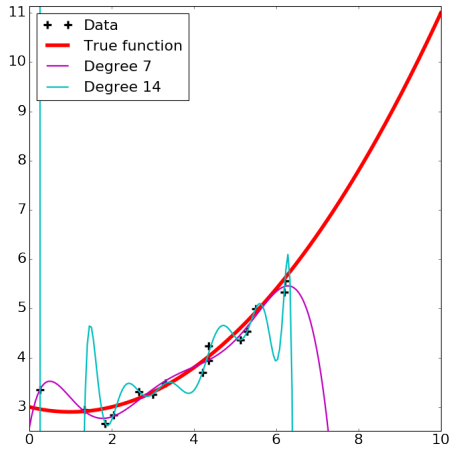
1.2 Basis Expansion Using Kernels

Let us now consider another way to capture non-linear relationships. We will perform basis expansion using kernels. For the time being, let us think of a kernel as a function that takes as input two points $\mathbf{x}$ and $\mathbf{x}'$ in the input space (*i.e.,* $\mathbb{R}^D$) and outputs a non-negative real

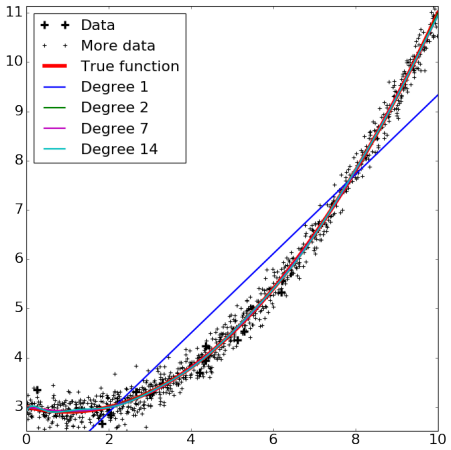


(a)

(b)



(c)



(d)

Figure 1: (a) Linear model fit to data. (b) Quadratic model fit to data. (c) The true model used to generate the data was quadratic. Higher degree models can be fit which may overfit the data. (d) When large quantities of data is available, overfitting can be avoided.

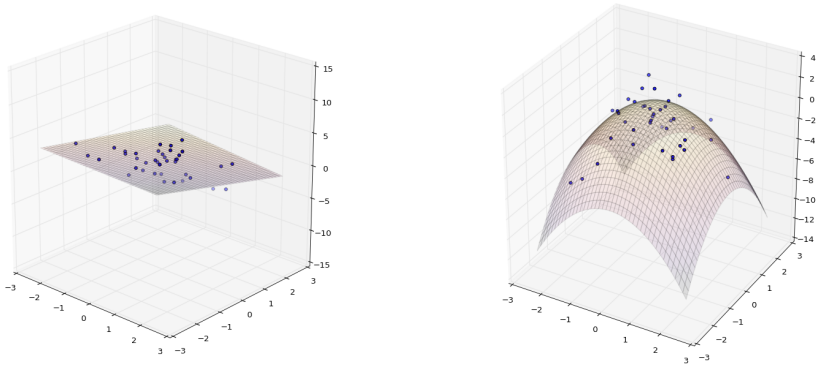
number.[1] In principle, any kernel may be used, however, let's focus on a particularly common one called the radial basis function (RBF) kernel.

Definition 1 (RBF Kernel)**.** *A* radial basis function *(RBF) kernel with width parameter* γ *is defined as* $\kappa(\mathbf{x}', \mathbf{x}) = \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2\right)$.

The name radial basis function reflects the fact that the value of $\kappa(\mathbf{x}', \mathbf{x})$ depends only on the distance between the two points $\mathbf{x}'$ and $\mathbf{x}$; for now, let us suppose that we are using Euclidean distances. We pick some centres, $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_M$ in the input space; we'll address the issue of how to pick centres shortly. Then for every point $\mathbf{x}$ we use the following kernel basis expansion

$$\phi(\mathbf{x}) = [1, \kappa(\boldsymbol{\mu}_1, \mathbf{x}), \kappa(\boldsymbol{\mu}_2, \mathbf{x}), \ldots, \kappa(\boldsymbol{\mu}_M, \mathbf{x})]^\mathsf{T}$$

[1] For a more formal definition refer to Murphy (2012, Chap 14).



(a) (b)

Figure 2: (a) Linear model in 2 dimensions. (b) Quadratic model in 2 dimensions obtained by first performing degree 2 polynomial basis expansion and then fitting a linear model.

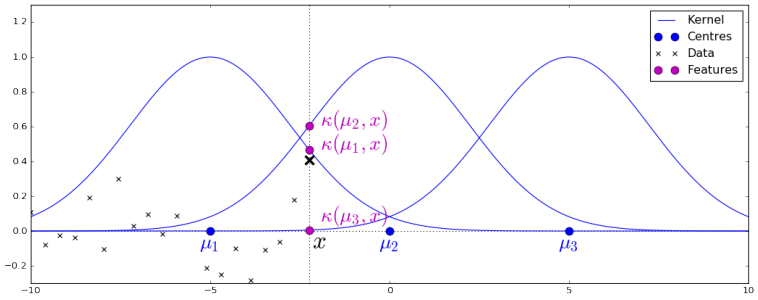


Figure 3: Basis expansion using kernels.

If we fit a linear model using the inputs $\phi(\mathbf{x})$, we get an output of the form:

$$\widehat{y}(\mathbf{x}) = w_0 + \sum_{i=1}^{M} \kappa(\boldsymbol{\mu}_i, \mathbf{x})$$

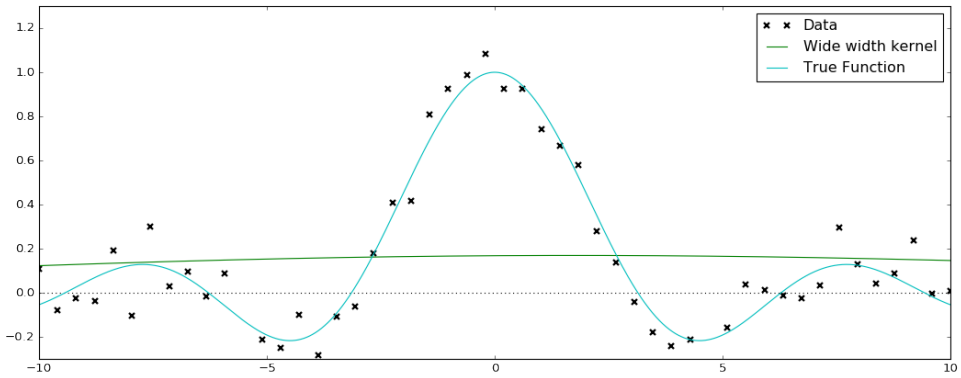
Thus, the model output is a linear combination of the kernel functions with M different centres.

Choice of Centres

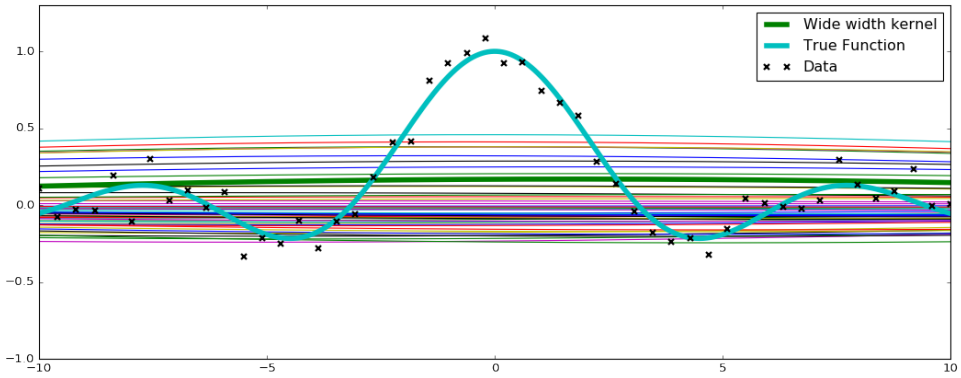
Figure 3 shows basis expansion using RBF kernels in one dimension using 3 centres. At the marked point x, the features will be given by $[1, \kappa(\mu_1, x), \kappa(\mu_2, x), \kappa(\mu_3, x)]^\mathsf{T}$. We see in the picture that $\kappa(\mu_3, x)$ is very close to 0 as the centre μ_3 is far from the point x. In fact in this picture most datapoints are very far from μ_3. The data mainly lies between $[-10, 2]$ and $\mu_3 = 5$. Thus, $\kappa(\mu_3, x) \approx 0$ for almost all the points in the dataset making this a relatively redundant feature. One way to avoid choosing such centres is to choose the datapoints themselves to be centres. This is the most common approach employed in practice. There is also good theoretical justification for this choice which is unfortunately beyond the scope of this course. The interested student is refererred to books on kernel methods, *e.g.,* (Schölkopf and Smola, 2002; Shawe-Taylor and Cristianini, 2004).

Choice of Width Paramater γ

As with the choice of degree in polynomial basis expansion, the width parameter γ plays an important role in kernel basis expansion. When γ is very small, the resulting kernel is "wide";



(a)



(b)

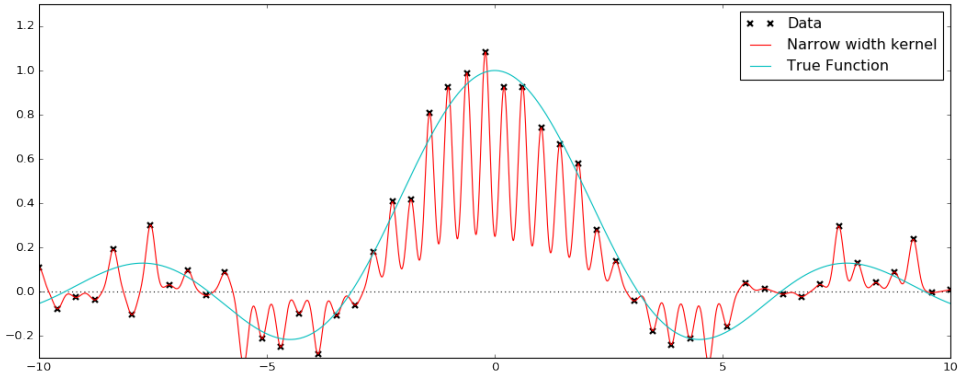
Figure 4: (a) True data generating function and linear model with kernel basis expansion for a wide kernel. (b) Composition of the fit model as a linear combination of kernels at datapoints as centres

suppose $\boldsymbol{\mu}$ is the centre, then even points $\mathbf{x}$ that are far from $\boldsymbol{\mu}$ will have $\kappa(\boldsymbol{\mu}, \mathbf{x}) \approx 1$. This often results in underfitting, as most points in the dataset will have the same feature value (see Fig. 4). On the other hand, if γ is very large, the kernel is "narrow"; except for points very close to the centre $\boldsymbol{\mu}$, the value $\kappa(\boldsymbol{\mu}, \mathbf{x}) \approx 0$. The resulting function fits the datapoints very well, but is unlikely to generalise to unseen data (see Fig. 5). A suitable choice of the width parameter will avoid both overfitting and underfitting (Fig. 6).

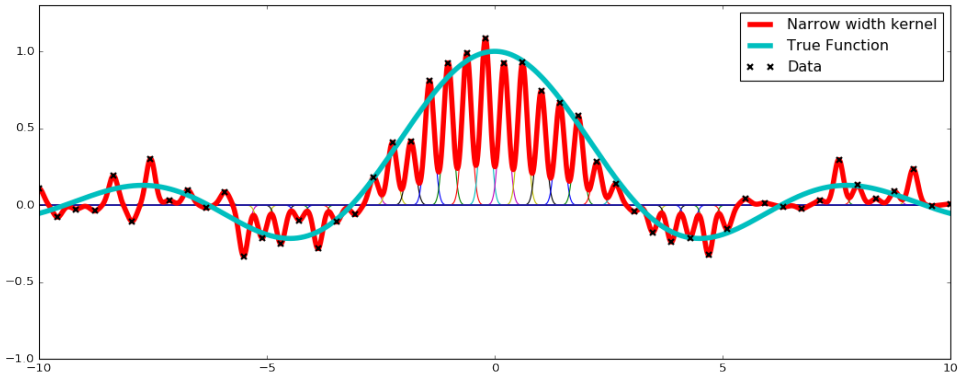
In high dimensions, we might suffer from the curse of dimensionality (see Problem 1 on Sheet 1). If the width of the kernel is chosen to be too large (very small γ), then it may result in underfitting. However, in high dimensions it is often not easy to find the "right" middle between a kernel that is 'too wide' and one that is 'too narrow'

Remark 2. *High-dimensional geometry can sometimes appear counter-intuitive and takes some time and effort to get used to. For example, think about this simple question. What is the ratio of the area of a circle with diameter* 1 *to that or a unit square? In two dimensions the answer is simple, it's* $\pi/4$*. However, in higher dimensions the volume of the ball with unit diameter is exponentially (in the dimension) smaller than the volume of the unit box.*

Imagine the implications of this: if we sample points uniformly from the unit box in say 100 *dimensions, almost none of them lie in the sphere with unit diameter with centre at the centre of the box!*

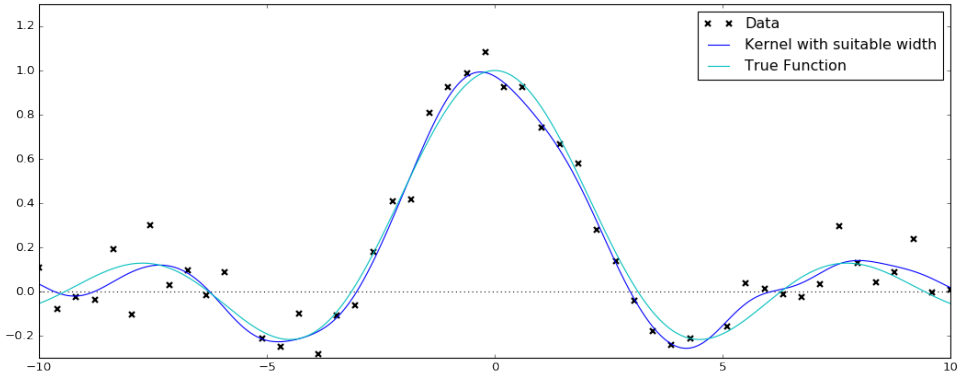


(a)

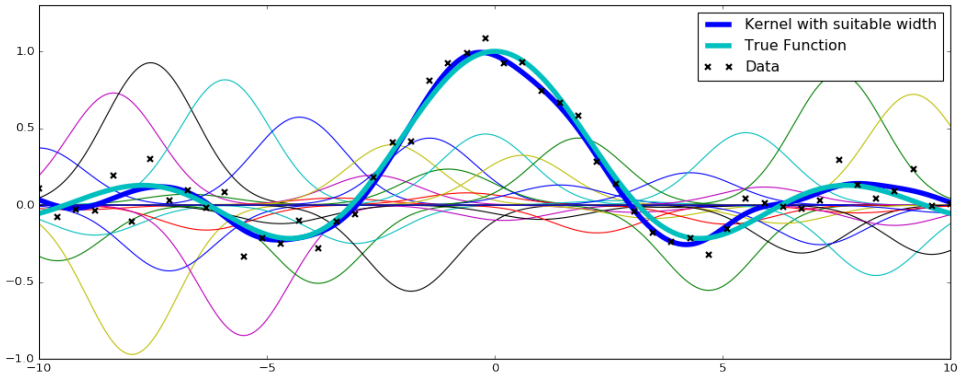


(b)

Figure 5: (a) True data generating function and linear model with kernel basis expansion for a narrow kernel. (b) Composition of the fit model as a linear combination of kernels at datapoints as centres

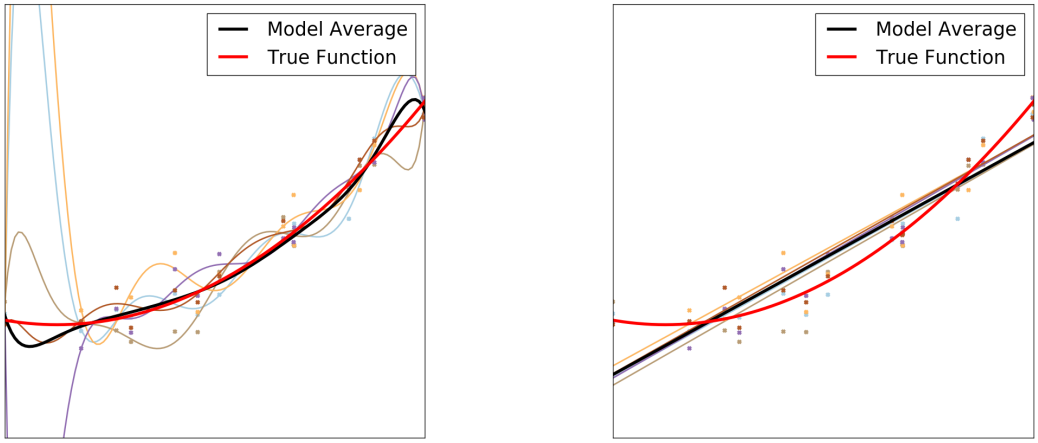


(a)



(b)

Figure 6: (a) True data generating function and linear model with kernel basis expansion for an intermediate width kernel. (b) Composition of the fit model as a linear combination of kernels at datapoints as centres



(a) (b)

Figure 7: (a) Different high-degree polynomials fit on data generated from the same true quadratic function with Gaussian noise. There is high-variance in the models, but the average (thick black line) is close to the true function (thick red line). (b) The same setting as Fig. (a) but this time a linear model is fit. There is very little variance, but high bias.

Discussion on Basis Expansion

Both polynomials and radial basis functions are universal; what this means is, as we increase the degree in the case of polynomials or the number of centres in the case of RBF kernels, the resulting model can approximate any *continuous* function. While this means that these models are very powerful, it also suggests that they may be too powerful! Ultimately, we want to fit models that can be trained using reasonable quantities of data and computational resources.

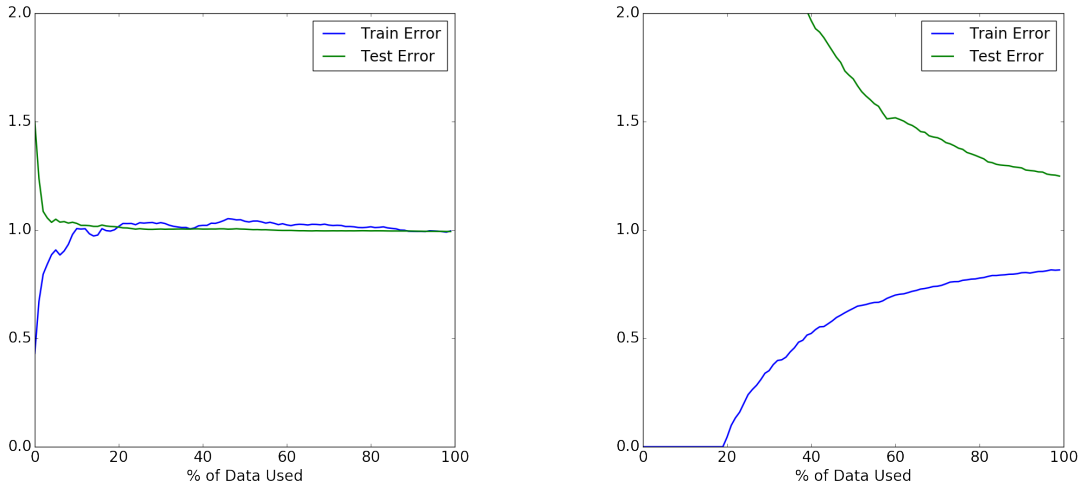
2 Overfitting and the Bias Variance Tradeoff

We've already alluded to the possibility of overfitting the data when we increase the complexity of the model by increasing the number of parameters as a result of basis expansion. On the other hand, when the model we attempt to fit is inadequate to explain the data, it results in *underfitting*. In statistical learning theory, these terms are referrred to as having "high variance" and "high bias" respectively. Choosing the correct model complexity to avoid both overfitting and underfitting is referred to as the *bias-variance* tradeoff.

2.1 Bias-Variance Tradeoff

We'll not go into the precise definitions of bias and variance in the context of statistical learning here. (On Problem Sheet 2 you are asked to work out the bias and variance of the least squares estimator.) High variance refers to the fact that when we fit the model to different realisations of the dataset we get very different models. As an example, if we fit a model using 5 different sets of 10 patients each, we might get completely different models (in terms of predictions on unseen patients) in each case. High bias refers to the fact that the model makes predictions that will inherently be different (either higher or lower) from the observations on unseen points.

Figure 7(a) shows five different high-degree polynomials fit to points generated from the same quadratic function with different noise added in each case. The model has high variance, but no bias. The average of the 5 models shown by the thick black line is actually pretty close



(a) (b)

Figure 8: (a) Plot of training error and test error as a fraction of training data used. This shows that the model is either underfitting or fitting correctly. (b) Same as Fig. (a). In this case, the model is clearly overfitting.

to true model generating the data (thick red line). Figure 7(b) shows linear models fit to the same five datasets. In this case, there is not much variance but there is a high bias. In most of the input space the models will make predictions that are either higher or lower than the true function.

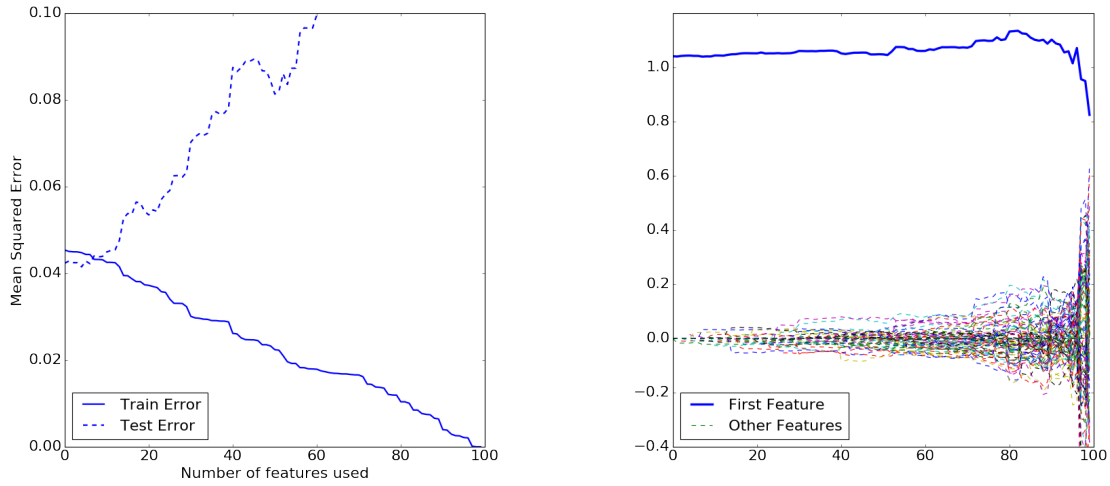
Learning Curves

An important task in practice is understanding whether the trained model has high bias (underfitting), high variance (overfitting) or neither. When the data is not one-dimensional and synthetically generated as we've seen in the pictures, how can we tell whether we are underfitting or overfitting?

One way to do this is to plot learning curves (see Fig. 8). If sufficient data is available, we first keep aside part of the dataset called the "test set" which will not use to fit the model at all. The remaining data is called the "training set". We can train the model on increasing sizes of the training data. For each model we've fit we can compute the training error (error on the data used for training) as well as the test error (error on the test set).

If the model has high bias (or suitable bias), the training and test error curves approach each other and then stay level. On the other hand if the model has high-variance, the training error will start increasing as we increase the amount of training data, and the test error will start decreasing. Eventually in the limit of infinite data, we do expect them to approach each other. Looking at the learning curves is a good way to understand whether our model is overfitting or underfitting, and develop possible solutions if necessary.

If the model is underfitting, then of course, we can try to design more complex models, *e.g.,* by basis function expansion. If the model is overfitting, then we can either simplify the model by directly reducing the number of parameters, or apply methods that control overfitting that we shall discuss next. Another, but often more expensive solution, is to obtain more data; if we get more data, the curves for the training and test error get closer to each other.



(a) (b)

Figure 9: (a) Plot of training and test error vs number of features used in the linear model to fit the toy problem described in Section 2.2 (b) Plot showing the weights to each feature in the models as a function of the number of features used.

2.2 How does overfitting occur?

Let us try to understand why models overfit the data. Of course, the obvious answer is because the number of parameters in the model can be very large, especially after performing basis expansion. Here's a quote from Enrico Fermi as reported by Freeman Dyson, [2]

> I remember my friend Johnny von Neumann used to say, with four parameters I can fit an elephant, and with five I can make him wiggle his trunk.

Compare this to the number of parameters in our model, if our original inputs are in $\mathbb{R}^{100}$ and we use degree 10 basis expansion. The number of parameters in the model is $\approx 10^{20}$! It is unlikely that we'll ever get datasets of this size and even if we did, we wouldn't have the computational resources to fit them.

Overfitting occurs because we have a finite-size dataset as well as noise in the observations. Let's perform the following thought experiment. Take two unbiased coins and flip them each independently. Repeat the experiment one hundred times and count the number of times the two coins tosses produced the same outcome. Now, we expect this number to be roughly 50. However, we know that actually outcome is more likely to some number in the range 40 to 60 other than 50! Let's say this number was 43, then we can conclude that by knowing the result of the first coin, we can predict the output of the second coin with about 57% accuracy, which is higher than 50%! Of course this is patently absurd as the second coin is independent of the first one. This is essentially what is going on when we fit a complex model to a finite-size dataset.

Consider a toy dataset generated as follows: $N = 100$ inputs are generated in $D = 100$ dimensions, each entry of the $N \times D$ matrix is drawn from the normal distribution with mean 0 and variance 1. The output $y = x_1 + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ for $\sigma = 0.2$. We know that the input features $x_2, \ldots, x_{100}$ have nothing to do with the output y. We fit a series of linear models $\mathbf{w}_1, \ldots, \mathbf{w}_{100}$, where the model $\mathbf{w}_i$ only uses the first i features.

Figure 9(a) shows the training error and test error as a function of the number of features used. We can see that as we add more features the training error goes down but the test error actually increases! Figure 9(b) shows the actual weights on the i^{th} feature. Obviously, for model

[2] `http://www.nature.com/nature/journal/v427/n6972/full/427297a.html`

$\mathbf{w}_k$ the weight on all features $j \geq k$ is 0 because they are unused. Initially almost all the weight is on the first feature, but as we allow the model to use more features, we see that other features also start getting larger weights. Essentially, by the time we reach $\mathbf{w}_{100}$ the model has fit the noise in the output variables perfectly using the irrelevant features!

3 Regularization

Let us now consider a few approaches to reducing overfitting. Of course, one approach is to reduce the number of features used in the model, which in turn reduces the number of model parameters and hence overfitting. However, this is somewhat less satisfying as this might leave our models unable to capture interesting relationships in the data. For example, it is not possible to know *a priori* which higher-degree monomials may be important when using polynomial basis expansion. Thus, we add all terms up to a certain degree.

However, as discussed in Section 2.2, having a large number of (possibly irrelevant) features makes the learning algorithms attempt to fit noise. What we would like is a penalty to be imposed for putting weights on features that contribute little to predicting the signal. However, we don't know which features are irrelevant, and so we add a weight penalty for every feature. The optimization objective is now a combination of the loss and penalty term; the optimization procedure has to balance the tradeoff between minimizing the loss and the penalty.

3.1 Ridge Regression

In ridge regression, we use the least squares objective and add a penalty on the sum of the squares of the weight parameters. For $\lambda > 0$,

$$\mathcal{L}_{\text{ridge}}(\mathbf{w}) = (\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \sum_{i=1}^{D} w_i^2 \tag{1}$$

Before proceeding find to the $\mathbf{w}$ that minimises $\mathcal{L}_{\text{ridge}}(\mathbf{w})$, a few words are in order. First, notice that we've left out the w_0 term (for the constant 1 feature) out of the penalty. We think of the magnitudes of the weights w_i as a measure of the complexity of the model. However, a translation of the output does not correspond to any additional model complexity. As a more concrete example, if we think of predicting the temperature using measurements of pressure, moisture, *etc.*, we may choose to output the answer in $^\circ C$ (celsius) or K (kelvin). The fact that we need to add 273 to every output to get the value in K does not make the model any more complex!

Standardizing Inputs

Likewise, let's consider the inputs $\mathbf{x}$. Let's consider a very simple model, $\widehat{y} = w_0 + w_1 x$, where x is the temperature measured in $^\circ C$ (celsius). Now, if instead we use x' which is the temperature in $^\circ F$ (fahrenheit), the model becomes $\widehat{y} = \left(w_0 - \frac{160}{9} w_1\right) + \frac{5}{9} w_1 x'$. Thus, in one case, we would get the term w_1^2, in the other $25w_1^2/81$, which less than one third of w_1^2.

To avoid issues of scaling and translation, it is good practice to standardise all the input variables, make them have mean 0 and variance 1 before fitting a model to the data. Don't forget to apply the same transformation to the test data!

If in addition we also centre the output variables y_is, then in the case of ridge regression we will always get $w_0 = 0$ (Problem Sheet 2). Thus, we can succinctly re-write the objective $\mathcal{L}_{\text{ridge}}$ in vector form as shown below.

Remark 3. *Don't forget to standardize the inputs and centering the outputs before applying the estimates derived in this section!*

$$\mathcal{L}_{\text{ridge}}(\mathbf{w}) = (\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^{\mathsf{T}}\mathbf{w} \tag{2}$$

Suppose the data is $\langle (\mathbf{x}_i, y_i) \rangle_{i=1}^{N}$ with inputs standardised and output centered. Let us now take the gradient of the above expression to find the optimal $\mathbf{w}$.

$$\mathcal{L}_{\text{ridge}}(\mathbf{w}) = \mathbf{w}^{\mathsf{T}}\mathbf{X}^{\mathsf{T}}\mathbf{X}\mathbf{w} - 2\mathbf{y}^{\mathsf{T}}\mathbf{X}\mathbf{w} + \mathbf{y}^{\mathsf{T}}\mathbf{y} + \lambda \mathbf{w}^{\mathsf{T}}\mathbf{w}$$

Taking the gradient with respect to $\mathbf{w}$

$$\begin{aligned}
\nabla_{\mathbf{w}} \mathcal{L}_{\text{ridge}} &= 2(\mathbf{X}^{\mathsf{T}}\mathbf{X})\mathbf{w} - 2\mathbf{X}^{\mathsf{T}}\mathbf{y} + 2\lambda \mathbf{w} \\
&= 2\left(\left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda \mathbf{I}_D \right) \mathbf{w} - \mathbf{X}^{\mathsf{T}}\mathbf{y} \right)
\end{aligned}$$

Above $\mathbf{I}_D$ is the $D \times D$ identity matrix. We set the gradient to $\mathbf{0}$ and solve for $\mathbf{w}$

$$\left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda \mathbf{I}_D \right) \mathbf{w} = \mathbf{X}^{\mathsf{T}}\mathbf{y}$$

to obtain the solution

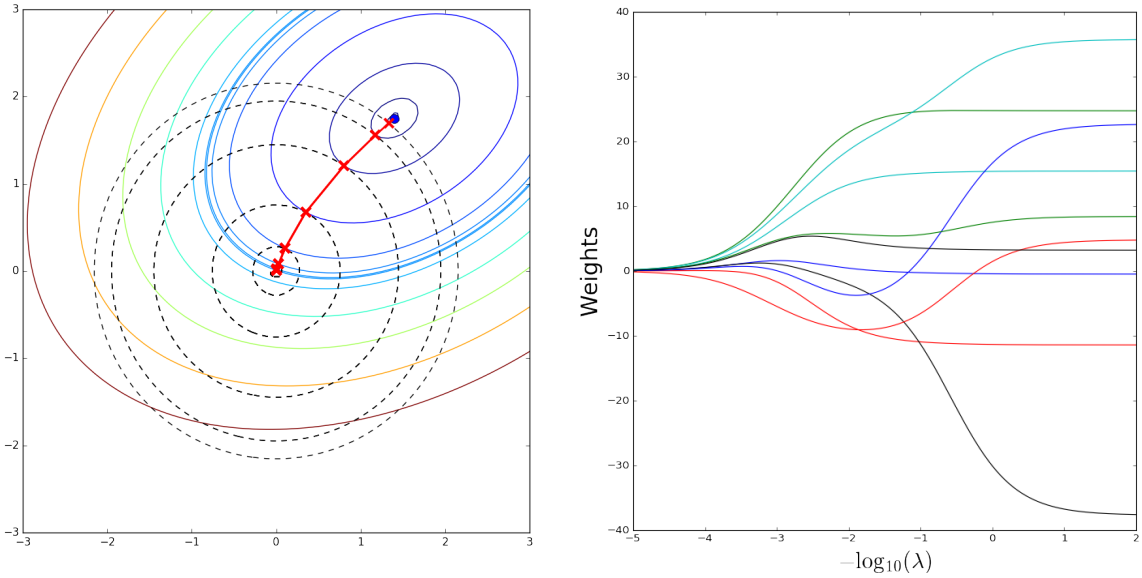
$$\mathbf{w}_{\text{ridge}} = \left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda \mathbf{I}_D \right)^{-1} \mathbf{X}^{\mathsf{T}}\mathbf{y} \tag{3}$$

Unlike in the case of the least-squares estimate, we do not need to be concerned about whether or not the matrix $(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda \mathbf{I}_D)$ is invertible. For $\lambda > 0$, this is always invertible (Exercise: Show that this is the case.). The quantity λ controls the tradeoff between minimising the prediction error and reducing the model complexity. As $\lambda \to 0$, we recover the least-squares estimate, where we are only concerned with minimising the sum of the squares of the residuals. On the other hand as $\lambda \to \infty$, we will get $\mathbf{w} = \mathbf{0}$ as the solution. Clearly, this is not desirable and the goal is to pick a λ that balances the two parts of the objective more evenly.

We'll return to the question of choosing λ shortly, but let's look at an alternative formulation of ridge regression.

$$\begin{aligned}
&\text{minimise} && (\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w} - \mathbf{y}) \\
&\text{subject to:} && \mathbf{w}^{\mathsf{T}}\mathbf{w} \leq R
\end{aligned}$$

It is not that hard to show that these two formulations are equivalent (the form in Eq. (2) is called the Lagrangean form), however, we'll leave that as an excercise for the interested student. When $R \to \infty$, there is essentially no constraint and the solution is that given by the least squares estimate, let's call it $\mathbf{w}_{\text{LS}}$ (in fact this is the case for any $R \geq \mathbf{w}_{\text{LS}}^{\mathsf{T}}\mathbf{w}_{\text{LS}}$). For smaller R, we'll get a solution $\mathbf{w}$ such that $\mathbf{w}^{\mathsf{T}}\mathbf{w} = R$, and the contour curves of $\mathbf{w}^{\mathsf{T}}\mathbf{w}$ and $(\mathbf{X}\mathbf{w}-\mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w}-\mathbf{y})$ are tangent at the solution. Figure 10(a) shows the solution to the objective function as a function of R (equivalently λ). Figure 10(b) shows how the weights on features vary as a function of λ for ridge regression performed on the diabetes dataset (available in scikit-learn).



(a) (b)

Figure 10: (a) Solution to ridge regression as a function of R (or λ) (b) Plot showing the weights of each feature obtained in ridge regression as a function $-\log(\lambda)$.

3.2 The Lasso

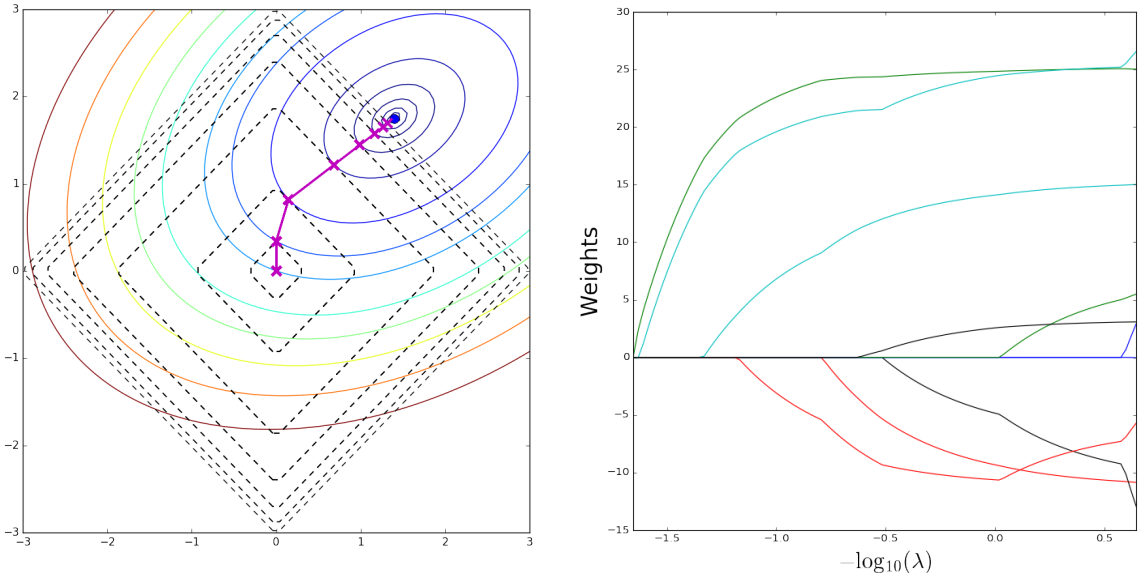
The Lasso (which stands for least absolute shrinkage and selection operator) is an alternative way to penalise model complexity. In the Lasso objective, instead of adding the sum of the squares of the weights as a penalty term, we add the sum of the absolute values of the weights, *i.e.*, $\sum_{i=1}^{D} |w_i|$, as a penalty term to the objective function. As in the case of Ridge regression, it is a good idea to standardize the input variables. For $\lambda > 0$, the lasso objective is expressed as:

$$\mathcal{L}_{\text{lasso}}(\mathbf{w}) = (\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \sum_{i=1}^{D} |w_i| \tag{4}$$

Unlike Ridge Regression, there is no closed form solution for $\mathbf{w}$ that minimises the Lasso objective; we have to resort to general optimisation methods. Clearly, as in the case of Ridge Regression as when $\lambda = 0$, we recover the least squares solution, whereas when $\lambda \to \infty$, we get the solution $\mathbf{w} = \mathbf{0}$. The equivalent constrained optimisation form for the Lasso objective is the following:

$$\begin{aligned} &\text{minimise} && (\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}}(\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &\text{subject to:} && \sum_{i=1}^{D} |w_i| \leq R \end{aligned}$$

As $R \to \infty$ (which is equivalent to $\lambda \to 0$) we recover the least squares solution; when $R = 0$ (which is equivalent to $\lambda \to \infty$) we get $\mathbf{w} = 0$ as the solution. For intermediate values of R we get a solution such that $\sum_{i=1}^{D} |w_i| = R$ and the contour curves of $\sum_{i=1}^{D} |w_i|$ and $(\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathsf{T}} (\mathbf{X}\mathbf{w} - \mathbf{y})$ are tangent at the solution. However the contour curves of $\sum_{i=1}^{D} |w_i|$ have corners, where the tangent is not well defined. This happens when some of the w_i are exactly zero! As a result, the model fit by Lasso may have several zero weights, and hence it can be seen as a form of *variable selection* (hence the name Lasso). Figure 11(a) shows the solution to the Lasso objective as a function of R (equivalently λ). Figure 11(b) shows how the weights



(a) (b)

Figure 11: (a) Solution to the Lasso as a function of R (or λ) (b) Plot showing the weights of each feature obtained using Lasso as a function of $-\log(\lambda)$.

on features vary as a function of λ for Lasso performed on the diabetes dataset (available in scikit-learn). We can see that for λ up to a certain level, several of the weights are exactly 0; compare this to Ridge Regression in Fig. 10(b).

3.3 Discussion

Let us return to the toy problem introduced in Section 2.2. Figure 12 shows the training and test error for least squares, Ridge Regression and Lasso on the problem. We see that both least squares and Ridge Regression do quite badly as we allow the model to use more and more irrelevant features. Recall that the first feature is the only relevant feature; but as we introduce more irrelevant features these models start fitting noise. Lasso actually does very well even when all 100 features (99 of which are irrelevant) are used. The reason why Ridge Regression performs poorly, while Lasso does very well, in this case is probably due to the fact that it is important to have most features other than the first one have zero weight. As discussed above this is more likely to be the case with Lasso than with Ridge Regression because of the corners in the Lasso penalty term. This can be seen in Figures 12(b), (c), (d) where the actual weights on the features are shown as a function of the number of features used in the model. As more and more irrelevant features are allowed, both the least squares and Ridge Regression model put weight on irrelevant features; this happens to a much lesser extent in the case of Lasso.

4 Bayesian Approach to Machine Learning

In this section, we briefly describe the Bayesian approach to machine learning and it's connections to Ridge Regression and Lasso. For the most part in this course, we'll not adopt the Bayesian approach. However, it is well worth understanding at least the basic aspects of this approach. The description of the Bayesian approach and its relation to the "frequentist" approach provided here is far from adequate and those interested should refer to Murphy (2012, Chap. 5, 6) and beyond.

In the "frequentist" approach, the assumption is that there are "true" parameters, unknown

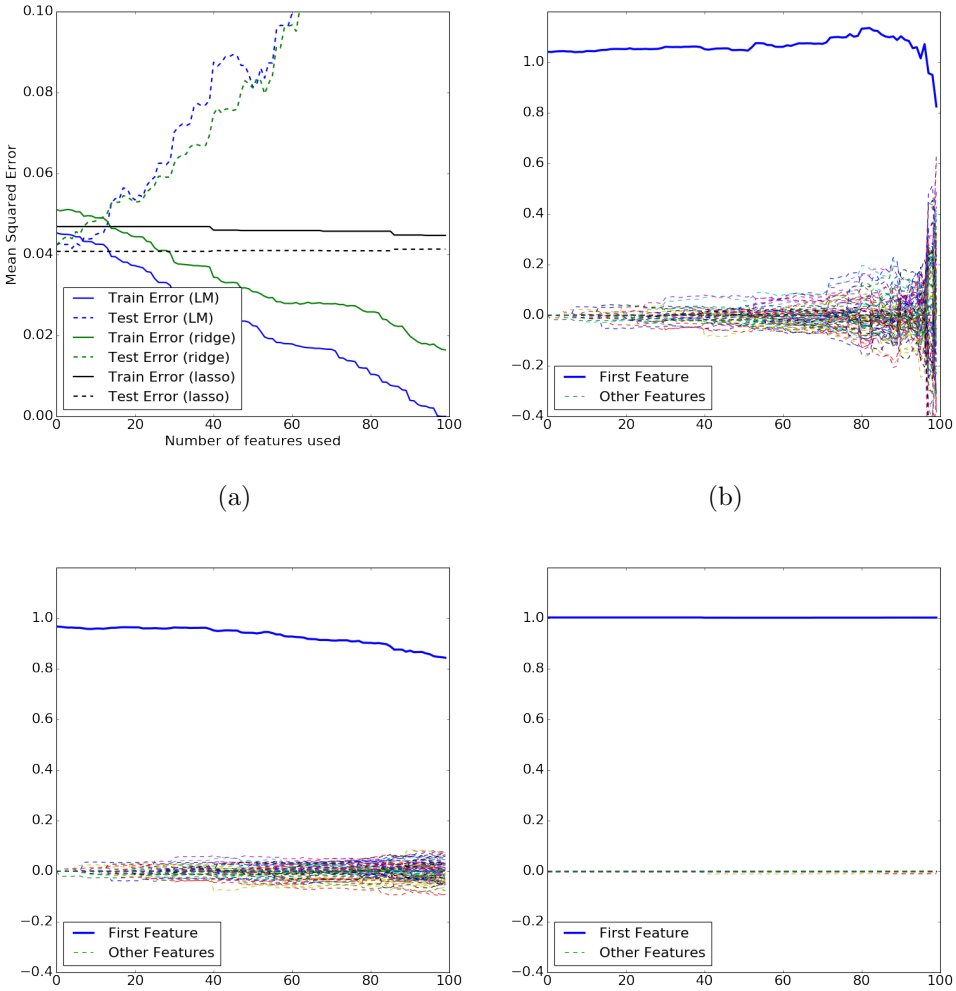


Figure 12: All plots concern the toy problem instroduced in Section 2.2 (a) Training and test error for least squares, Ridge Regression and Lasso as a function of the number of features used. (b) Weights as a function of the number of features used in the model for least squares (c) Weights as the number of features used for Ridge Regression (d) Weights as a function of the number of features used for Lasso

though they may be to us. The goal is to make use of data, which depends on the true parameters and which we observe through some random process, to infer the true "unknown" parameters. In the Bayesian approach, in the absence of any data, a belief about what the parameters may be is represented by a *prior* distribution on the parameters; let us denote this prior on the parameters by $p(\mathbf{w})$. As in the frequentist setting, the data will depend on parameters and will be observed through some random process. When the data, denoted by $\mathcal{D}$ is observed, the belief about the parameters is updated and represented using what is called the *posterior* distribution. This distribution is obtained using Bayes' Rule using the prior distribution $p(\mathbf{w})$ and the (probabilistic) data model, denoted by $p(\mathcal{D} \mid \mathbf{w})$. Then, the posterior on the model parameters given the data is given by,

$$p(\mathbf{w} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \mathbf{w}) \cdot p(\mathbf{w})}{p(\mathcal{D})} \tag{5}$$

Thus, the posterior reflects the updated belief about the parameters after observing some data.

In the limit of infinite data, the posterior distribution will become a point mass at the maximum likelihood estimate (as long as the prior has non-zero mass everywhere).

Let us now discuss how a prediction is made using this approach. To make things a bit more concrete, let's suppose that the new input point is $\mathbf{x}_{\text{new}}$ and we wish to predict the output y_{new} (or in general a distribution over the output). There are two approaches to this, the first is to use a point-estimate (or a plugin estimate) which uses a single set of parameters that are chosen to represent the posterior distribution. For example, this may be the posterior mean, median or mode. The second approach is to use the entire posterior distribution to make the prediction, sometimes referred to as the full Bayesian approach, by integrating out the parameters $\mathbf{w}$. Thus, we may express:

$$p(y \mid \mathbf{x}_{\text{new}}, \mathcal{D}) = \int_{\mathbf{w}} p(y \mid \mathbf{w}, \mathbf{x}_{\text{new}}) \cdot p(\mathbf{w} \mid \mathcal{D}) \, \mathrm{d}\mathbf{w}.$$

While the full Bayesian approach is certainly desirable as it accounts for all our prior beliefs as well as the observed data, for all but the simplest of models this can be computationally expensive (or even intractable!). There is a lot of research on developing approximate methods in the case of computational intractability, which we will not cover in the course.

Let us now return to the first approach, which is to obtain a point estimate. The mode, however unrepresentative of the distribution as whole it may be, stands out for one reason. In order to compute the median or mean of the posterior distribution it is necessary to compute the denominator of (5). This denominator represents just the probability of observing the data, obtained by integrating out the parameters $\mathbf{w}$, *i.e.,*

$$p(\mathcal{D}) = \int_{\mathbf{w}} p(\mathcal{D} \mid \mathbf{w}) \cdot p(\mathbf{w}) \, \mathrm{d}\mathbf{w}.$$

However, except in relatively simple cases, even this integral may be computationally expensive to evaluate. In order to obtain the mode though, the denominator is unnecessary, it can be obtained by simply looking for $\mathbf{w}$ where the numerator of (5) achieves the maximum value. Thus, it is often common to express the posterior as,

$$p(\mathbf{w} \mid \mathcal{D}) \propto p(\mathcal{D} \mid \mathbf{w}) \cdot p(\mathbf{w}) \tag{6}$$

The mode of the posterior is a point estimate known as the *maximum a posteriori* or MAP estimate. This can be obtained by finding $\mathbf{w}$ that maximises the RHS of (6). For most of this section, we'll focus on computing the MAP estimate.

4.1 Bayesian Linear Regression

Let us now look specifically at linear regression through the Bayesian approach. We'll still consider the linear model given by,

$$p(y \mid \mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{w} \cdot \mathbf{x}, \sigma^2)$$

Throughout this section, we'll think of σ^2 as fixed and known. Thus, we're only representing $\mathbf{w}$ as the parameters. We need to define a prior distribution over $\mathbf{w}$; let us assume that this is a spherical Gaussian distribution with mean $\mathbf{0}$ and variance τ^2 in each direction,

$$\begin{aligned} p(\mathbf{w}) &= \mathcal{N}(\mathbf{0}, \mathbf{\Lambda}), && \text{where } \ \mathbf{\Lambda} = \tau^2 \mathbf{I}_D \\ &= \frac{1}{(2\pi\tau^2)^{D/2}} \cdot \exp\left(-\frac{\mathbf{w}^\mathsf{T}\mathbf{w}}{2\tau^2}\right) \end{aligned}$$

As we've been doing so far, given data $\mathcal{D} = \langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$, we can represent $\mathbf{y}$ given model parameters $\mathbf{w}$ and inputs $\mathbf{X}$ as,

$$p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = \frac{1}{(2\pi\sigma^2)^{N/2}} \cdot \exp\left(-\frac{(\mathbf{y} - \mathbf{X}\mathbf{w})^\mathsf{T}(\mathbf{y} - \mathbf{X}\mathbf{w})}{2\sigma^2}\right)$$

Thus, we can express the posterior as,

$$p(\mathbf{w} \mid \mathbf{X}, \mathbf{y}) \propto \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{(\mathbf{y} - \mathbf{X}\mathbf{w})^\mathsf{T}(\mathbf{y} - \mathbf{X}\mathbf{w})}{2\sigma^2}\right) \cdot \frac{1}{(2\pi\tau^2)^{D/2}} \exp\left(-\frac{\mathbf{w}^\mathsf{T}\mathbf{w}}{2\tau^2}\right)$$

The *maximum a posteriori* or MAP estimate is obtained by finding the value of $\mathbf{w}$ that maximises the RHS of the above expression. Since σ and τ are fixed, we can express this as:

$$\mathbf{w}_{\text{map}} = \underset{\mathbf{w}}{\text{argmax}} \exp\left(-\frac{(\mathbf{y} - \mathbf{X}\mathbf{w})^\mathsf{T}(\mathbf{y} - \mathbf{X}\mathbf{w})}{2\sigma^2} - \frac{\mathbf{w}^\mathsf{T}\mathbf{w}}{2\tau^2}\right)$$

Using the fact that log is monotone and coverting argmax to argmin by flipping signs, we get

$$\mathbf{w}_{\text{map}} = \underset{\mathbf{w}}{\text{argmin}} \left(\frac{(\mathbf{y} - \mathbf{X}\mathbf{w})^\mathsf{T}(\mathbf{y} - \mathbf{X}\mathbf{w})}{2\sigma^2} + \frac{\mathbf{w}^\mathsf{T}\mathbf{w}}{2\tau^2}\right)$$

$$\mathbf{w}_{\text{map}} = \underset{\mathbf{w}}{\text{argmin}} \left((\mathbf{y} - \mathbf{X}\mathbf{w})^\mathsf{T}(\mathbf{y} - \mathbf{X}\mathbf{w}) + \frac{\sigma^2}{\tau^2} \cdot \mathbf{w}^\mathsf{T}\mathbf{w}\right) \tag{7}$$

Comparing the form of (1) and (7), we see that the MAP estimate is exactly that given by minimising the Ridge Regression objective with $\lambda = \sigma^2/\tau^2$.

Exercise: Choose a suitable prior on $\mathbf{w}$ so that the MAP estimate can be viewed as minimising the Lasso objective.

4.1.1 Full Bayesian Approach for Linear Regression

We mentioned earlier that the full Bayesian approach can be computationally expensive. In the case of Bayesian linear regression discussed in this section, everything can be expressed in closed form. The calculations are a bit tedious and the interested student is referred to Murphy (2012, Sec 7.6). However, let us see the advantage of this approach. For the setting described here, we can express the distribution over the output y of a new data point $\mathbf{x}_{\text{new}}$ as follows:

$$p(y \mid \mathbf{w}_{\text{new}}, \mathcal{D}) = \mathcal{N}(\mathbf{w}_{\text{map}} \cdot \mathbf{x}_{\text{new}}, \sigma^2 + \mathbf{x}_{\text{new}}^\mathsf{T}\mathbf{V}_N\mathbf{x}_{\text{new}}) \tag{8}$$

where

$$\mathbf{V}_N = \sigma^2\left(\mathbf{X}^\mathsf{T}\mathbf{X} + \frac{\sigma^2}{\tau^2} \cdot \mathbf{I}_D\right)^{-1} \tag{9}$$

It can be shown (using singular value decomposition) that the variance in (8) is relatively small for $\mathbf{x}_\mathbf{w}$ that "look like" previously observed data and large for those that don't. Thus, the predictions include higher degree of uncertainty in parts of the input space where there is scarce data and less uncertainty where data is plenty. Figure 13 shows this for polynomial regression in one dimension. As shown in the figure, one way to think of the full Bayesian approach is to make prediction using $\mathbf{w}$ sampled from the posterior distribution; the figure shows models represented by several samples of $\mathbf{w}$ drawn from the posterior distribution as well as error bars representing the uncertainty. It can be seen that in the region where there is a lot of data almost all models drawn from the posterior make almost the same predictions, but in regions where data is scarce the predictions can be quite different.

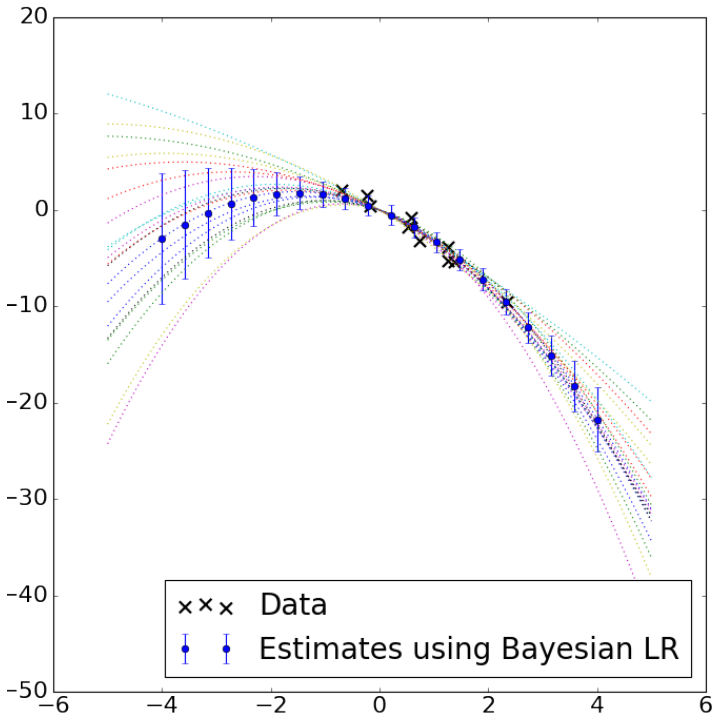


Figure 13: Full Bayesian Approach for polynomial regression in one dimension.

Choosing a Prior

How to choose a prior in the Bayesian approach? That is one of the central questions and often a point of criticism of this approach. While in principle one should choose a prior that reflects the true beliefs, often priors are chosen for mathematical convenience. In the absence of any definite beliefs about the prior, one should choose a prior that is as uninformative as possible. We'll not cover these aspects of the Bayesian approach in the course; the interested student may refer to the textbook by Murphy (2012).

5 Model Selection

Let us now return to the question alluded to several times so far about how to select various hyperparmeters such as λ (in Ridge and Lasso), the degree (in polynomial basis expansion), and the width parameter γ (in kernel regression). In general, as we start using more and more complex models to fit the data, we might have more hyperparameters that need to be selected.

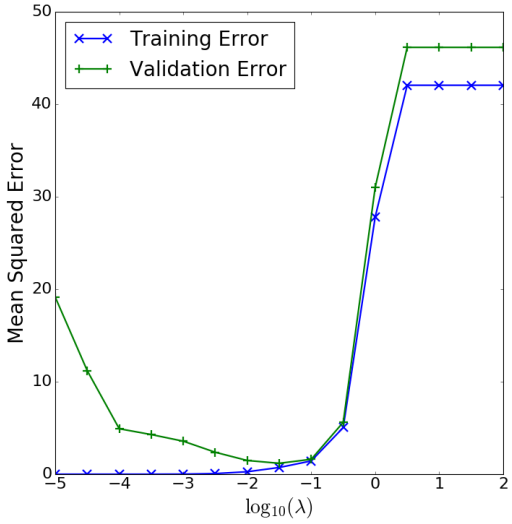
5.1 Validation

Let us start with the setting where the data is relatively plenty. In this case, we divide the data into a *training* set and a *validation* set. The training set will be used to actually train the model parameters (such as $\mathbf{w}$, not the hyperparameters!) and the validation set will be used to pick suitable values for the hyperparameters. Of course, in reality we care about testing our model on completely unseen data. When applying machine learning in the real-world, tests will present themselves! However, in academic settings, we can keep aside yet another part of the data called the *test* set, where we'll evaluate the performance of our models after performing training and validation. In academic settings, the test set should not be touched at all except for reporting the performance of the models!

Now, let's say we have only one hyperparameter to choose from, say λ. We start with a possible set of values that we may want to assign to λ, *e.g.,* $\lambda \in \{0.01, 0.1, 1, 10, 100\}$. In general,

λ	training error(%)	validation error(%)
0.01	0	89
0.1	0	43
1	2	12
10	10	8
100	25	27

(a)



(b)

Figure 14: (a) (Made-up) Errors on training and validation sets. (b) Curves showing error on training and validation sets as a function of λ for Lasso.

the range of hyperparameters should be chosen depending on the sensitivity of the trained model to the hyperparameters; thus they may be on a log scale, linear scale, etc.[3] Fig 14(a) summarises (made-up values of) the error on the training set and the validation set for some model. Since, we've not used the validation set as part of the training set, we'll trust the performance on the validation set as being more representative than that on the training set. On the training set, we may have overfit depending on the complexity of the model. Thus, in this case we'll pick $\lambda = 10$ as the value for the hyperparameter. Once the value of the hyperparameter is fixed, it is often a good idea to train the model using all available data (including the one previously used for validation), since the more data we use in the training the more accurate our model is likely to be. If we plot the curves for the training and validation error as a function of the hyperparameter, the validation error curve typically has a U-shape, where the validation error is high on one-side because of overfitting (where the training error is typically low) and on the other side because of underfitting (where the training error is also high). The optimal hyperparameter to be chosen is at the bottom of the U shape (see Fig. 14(b)). When there are multiple hyperparameters to be chosen, we can make a "grid" or all possible combinations of values for the hyperparameters and pick the combination that is most suitable; this is called *grid search*. Grid search may be very costly even when the number of hyperparameters is relatively modest because of the exponential size of the search space. Techniques such as random search, or Bayesian black-box optimisation can be applied; we will not cover them in this course.

When data is scarce, keeping aside a validation set is not a good idea. In this case, it is more common to divide the data into K parts (called folds) and then use $K-1$ parts as the training set and use the performance on the Kth part for validation (see Fig. 15). This is then repeated across all the folds and the average error on the fold used as the validation set (over the K different choices) is used as a proxy for the validation error. This has the effect of reducing the variance in the validation error and hence is usually more suitable than using a small validation set. It is common to use $K = 5$ or $K = 10$. One extreme case is to use $K = N$, where N is the number of datapoints. In this case, in each instance we are training on $N-1$ datapoints and testing the performance on the N^{th} one, averaging over all possible choices. This

[3]While looking at existing literature will provide clues as to what scales should be chosen for various hyperparmeters, there will be times when this will only be clear after one round of validation.

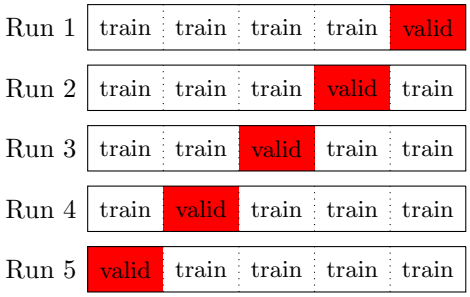


Figure 15: 5-fold cross validation.

method is called leave-one-out-cross-validation or LOOCV for short. However, this means that we are running the training algorithm N times which can be computationally expensive. Some methods (such as least squares) have the property that the influence of one datapoint from the trained model can be quickly removed without explicitly re-training, however in general this is unlikely to be the case.

Discussion

The question of model selection is an important one. We've only seen the very basic approaches employed in practice. We'll return to these questions a few times later in the course. In particular, for classification problems, it is often necessary to treat errors of different kinds differently. Predicting that a tumor is malignant when it is not vs predicting a tumor is benign when it is not are not equally problematic errors! Thus, when performing model selection, it is important to assign different costs to different types of errors. In unsupervised learning, where we have no access to the ground truth the problem of model selection requires using different criteria.

The machine learning pipeline we've see so far is: get data, choose a model, train a model and select hyperparameters, and test the model. What happens when the test turns out to be bad? We have no option but to start from scratch; however, in general we may not always get new data. We may simply be left with the option of choosing a different model or hyperparameters and use the existing data. However, notice that we've already "seen" the test set, although only implicitly, when testing our model. Thus, through this process we've *leaked* some information from the test data to the machine learning pipeline and hence can no longer assume that our model is completely blind to the test set. When using the test set sparingly, this does not usually pose a huge problem, however, if the test set is not utilised carefully this can lead to serious overfitting! For example, in Kaggle competitions it often happens that some top teams on the public leaderboard are not close to the top on the private leaderboard! This may happen because these teams are submitting too many entries and implicitly overfitting on the data used for the public leaderboard.[4]

References

Kevin P. Murphy. *Machine Learning : A Probabilistic Perspective*. MIT Press, 2012.

Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[4] An interesting blog post about topping the public leaderboard in Kaggle without reading the data is available here: `http://blog.mrtz.org/2015/03/09/competition.html`.