COMP0078 Coursework 1

Srinath Kailasa, MSc Scientific Computing

14 November 2019

Part 1

1.1 Linear Regression

Considering a training set,

$$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_m, y_m)\} \tag{1}$$

where $\boldsymbol{x} = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$ and $y \in \mathbb{R}$. The linear regression algorithm finds a vector of weights $\boldsymbol{w} \in \mathbb{R}^n$ such that the sum of squared errors,

$$SSE = \sum_{t=1}^{m} (y_t - \boldsymbol{w} \cdot \boldsymbol{x}_t)^2 \tag{2}$$

is minimised. We can express this in matrix form, by forming a $(n \times m)$ 'design matrix' from our training data,

$$X = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m) \tag{3}$$

defining $\boldsymbol{y}$ as the column vector $\boldsymbol{y} = (y_1, y_2, ..., y_m)$, $\boldsymbol{w}$ then minimises,

$$(X\boldsymbol{w} - \boldsymbol{y})^T (X\boldsymbol{w} - \boldsymbol{y}) \tag{4}$$

We can also form a design matrix using a linear combination of basis functions $(\phi_1, \phi_2, ..., \phi_k)$ where $\phi_i : \mathbb{R}^n \to \mathbb{R}$ which define a feature map $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^k$. The main logic in machine learning for using such maps is to present the learning algorithm with data that it is better able to regress or classify. Our training set under this mapping goes to,

$$\{((\phi_1(\boldsymbol{x}_1), .., \phi_k(\boldsymbol{x}_1)), y_1), ..., (\phi_1(\boldsymbol{x}_m), .., \phi_k(\boldsymbol{x}_m)), y_m)\} \tag{5}$$

With the corresponding mapped design matrix,

$$\Phi = (\phi(\boldsymbol{x}_1), \phi(\boldsymbol{x}_2), ..., \phi(\boldsymbol{x}_m)) \tag{6}$$

we now seek a vector of weights $\boldsymbol{w} \in \mathbb{R}^k$ that minimises,

$$(\Phi \boldsymbol{w} - \boldsymbol{y})^T (\Phi \boldsymbol{w} - \boldsymbol{y}) \tag{7}$$

This is solved by the famous normal equations,

$$\begin{gathered}
\nabla_{\boldsymbol{w}} (\Phi \boldsymbol{w} - \boldsymbol{y})^T (\Phi \boldsymbol{w} - \boldsymbol{y}) = 0 \\
\nabla_{\boldsymbol{w}} (\boldsymbol{w}^T \Phi^T - \boldsymbol{y}^T)(\Phi \boldsymbol{w} - \boldsymbol{y}) = 0 \\
\nabla_{\boldsymbol{w}} (\boldsymbol{w}^T \Phi^T \Phi \boldsymbol{w} - 2\boldsymbol{w}^T \Phi^T \boldsymbol{y} + \boldsymbol{y}^T \boldsymbol{y}) = 0 \\
\boldsymbol{w} = (\Phi^T \Phi)^{-1} (\Phi^T \boldsymbol{y})
\end{gathered} \tag{8}$$

In this question, we use the polynomial basis to define our feature map, where $\{\phi_1(x) = 1, \phi_2(x) = x, ..., \phi_k(x) = x^{k-1}\}$. We plot the results of running a linear regression with four polynomial feature maps where $k = 1, 2, 3, 4$ in figure (1) on some sample data.

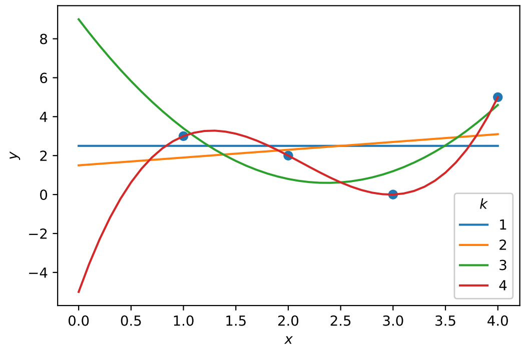


Figure 1: Fitted to data set $\{(1, 3), (2, 2), (3, 0), (4, 5)\}$

We note that for $k = 4$ as the number of parameters in our model matches the number of data points, we can fit exactly. The corresponding polynomial equations for these models are,

$$\begin{gathered}
k = 1 : y = 2.5 \\
k = 2 : y = 1.5 + 0.4x \\
k = 3 : y = 9 - 7.1x + 1.5x^2 \\
k = 4 : y = -5 + 15.2x - 8.5x^2 + 1.33x^3
\end{gathered} \tag{9}$$

where recurring decimals have been reported to 3 significant figures. Defining mean square error as,

$$MSE = \frac{SSE}{m} \tag{10}$$

We can calculate the MSE for each of the above models as shown in figure (2).

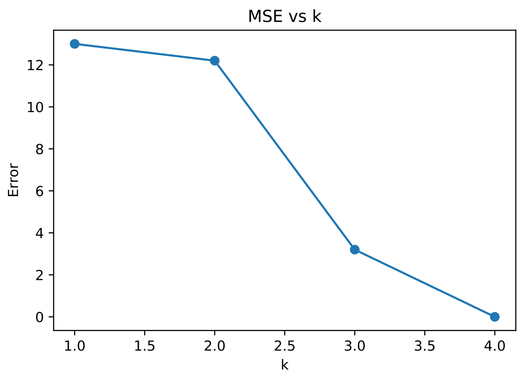


Figure 2: Mean Square Error vs k

Overfitting is the phenomenon of building a model that corresponds too closely to the training data, and therefore fails to generalise when presented with new data points. We saw a hint of this with the polynomial model $k = 4$, which will allow us to fit any 4 observed data points exactly. In this section we perform further analysis to illustrate this phenomenon.

Let's define,

$$g_\sigma(x) := \sin^2(2\pi x) + \epsilon \tag{11}$$

where ϵ is a random variable distributed normally with 0 mean and σ^2 variance. Sampling x_i uniformly at random in the interval $[0, 1]$ 30 times $(x_1, ..., x_{30})$, we can create a dataset,

$$S_{0.07,30} = \{(x_1, g_{0.07}(x_1), ..., (x_{30}, g_{0.07}(x_{30}))\} \tag{12}$$

This is plotted in figure (4), where the random number generators are seeded to ensure reproducibility. Let's fit this data using our polynomial feature map of dimension $k = 2, 5, 10, 14, 18$, these results are plotted in figure (3). Let the training error $te_k(S)$ denote the MSE of the fitting of data set S with a polynomial basis of dimension k, we can then plot the natural logarithm of the training error vs k, shown in figure (5A). This function in theory should be strictly decreasing, however the product of our design matrix Φ with it's transpose $\Phi^T\Phi$ is extremely ill conditioned for large k (see fig. 5B) resulting in increasing test error as our linear regression algorithm relies on the calculation of it's inverse (8). To understand this we need to consider the condition number, which can be seen as a way of providing a bound on the error in using a numerical method to solve the following linear system [2],

$$A\boldsymbol{x} = \boldsymbol{b} \tag{13}$$

Consider a numerical method to invert A in (13), we will in general obtain some perturbed solution of the form,

$$A(\boldsymbol{x} + \delta\boldsymbol{x}) = \boldsymbol{b} + \delta\boldsymbol{b}, \tag{14}$$

Where $\delta\boldsymbol{x}$ is the accuracy of the numerical method. If we define the condition number of a matrix as K_M,

$$K_M(A) = \left\|A^{-1}\right\|_M \|A\|_M \, . \tag{15}$$

Where we take $\|\cdot\|_M$ to be a matrix norm induced by some vector norm $\|\cdot\|_V$. Dropping the subscripts for clarity, we have by linearity,

$$\|\delta\boldsymbol{x}\| = \left\|A^{-1}\delta\boldsymbol{b}\right\| \leq \left\|A^{-1}\right\| \|\delta\boldsymbol{b}\| \tag{16}$$

Furthermore we also have,

$$\|\boldsymbol{b}\| = \|A\boldsymbol{x}\| \leq \|A\| \|\boldsymbol{x}\| \tag{17}$$

Giving us an upper bound on numerical error in terms of condition number for inverting the matrix A,

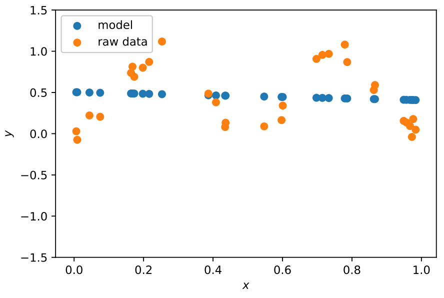
$$\frac{\|\delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \|A\| \left\|A^{-1}\right\| \frac{\|\delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|} = K_M(A)\frac{\|\delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|} \tag{18}$$

We can perform some similar analysis to find a lower bound also in terms of condition number, we simply quote the result here,

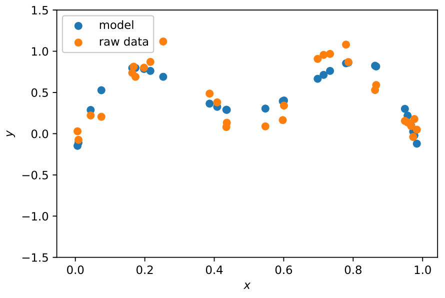
$$\frac{1}{K_M(A)}\frac{\|\delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|} \leq \frac{\|\delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq K_M(A)\frac{\|\delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|} \tag{19}$$

If $K_M(A) \approx 1$ we say that A is well conditioned, and we can see that the relative error in the solution is bounded by the relative residual $\|\delta\boldsymbol{b}\| / \|\boldsymbol{b}\|$, however if $K_M(A)$ is very large the inversion is susceptible to large errors with small perturbations of the residual - giving us some background on why train error $te_k(S)$ increases for large k, at least in our implementation of the linear regression algorithm.

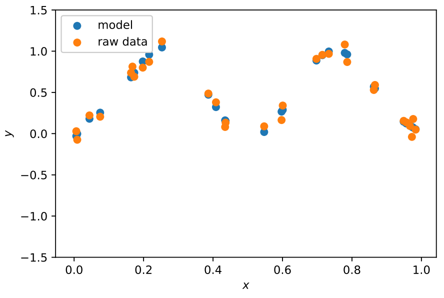
Regardless of this, we can still demonstrate the phenomenon of overfitting using these models. Let's generate a new test set on which to measure model performance.



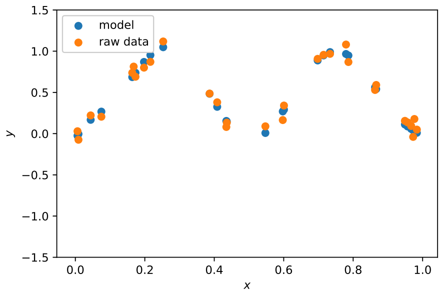
(a) k=2



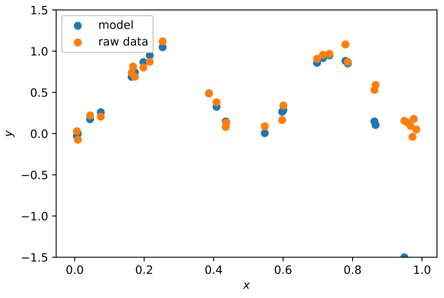
(b) k=5



(c) k= 10



(d) k=14



(e) k=18

Figure 3: $S_{0.07,30}$ fitted with polynomial bases of dimension k

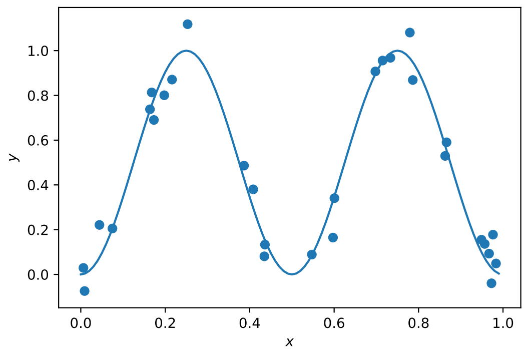
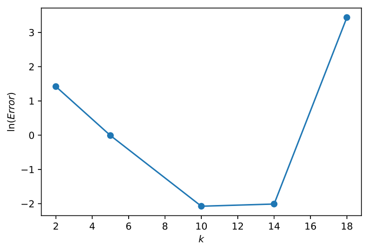
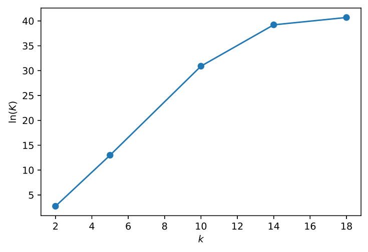


Figure 4: $S_{0.07,30}$, with the function $\sin^2(2\pi x)$ plotted over the same range.



(a) $\ln(te_k(S))$ vs k



(b) $\ln(K_M(\Phi^T\Phi))$ vs k

Figure 5: Training error, $te_k(S)$, and condition number, $K_M(\Phi^T\Phi)$, as functions of polynomial dimension k.

$$T_{0.07,1000} = \{(x_1, g_{0.07}(x_1), ..., (x_{1000}, g_{0.07}(x_{1000}))\} \tag{20}$$

Running our models over $T_{0.07,1000}$, we can plot the test error $tse_k(S,T)$ which is the MSE of the test set (fig. 6).

As these experiments will vary with the generation of different training and test sets, we can repeat our experiments and average our results to get a better picture. Let's do this for 100 runs. The results of this experiment are plotted in figure (7).

It is difficult to analyse how much numerical error is forcing test error to increase as a function of k in the experiment of figure (7), or compare it's impact to that of generalisation error from overfitting. However, by designing a better conditioned design matrix, we should be able to observe this. Consider the following sinusoidal feature map, $\{\phi_1(x) = \sin(1\pi x), \phi_2(x) = \sin(2\pi x), ..., \phi_k(x) = \sin(k\pi x)\}$, it's intuitive that a model constructed with a periodic basis should

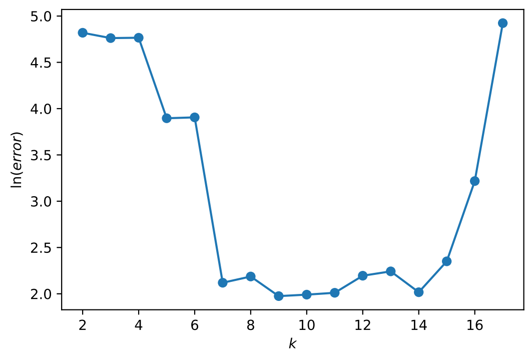


Figure 6: Test error $tse_k(S,T)$ as a function of k

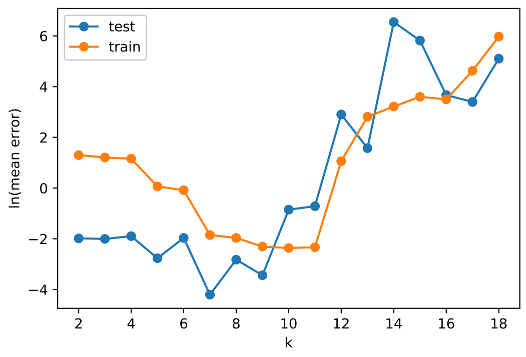


Figure 7: Test and Train error plotted as a function of k, with the results averaged over 100 runs using a polynomial basis for our feature map.

be better able to regress on periodic-looking data. The results of repeating the above experiment with this feature map are shown in figure (8). Here we see the behaviour we expected, arbitrarily decreasing training error by increasing the complexity of our model via the dimension of the basis k, resulting in an eventual increase of the test error. This can loosely can be interpreted as fitting noise within the training data.

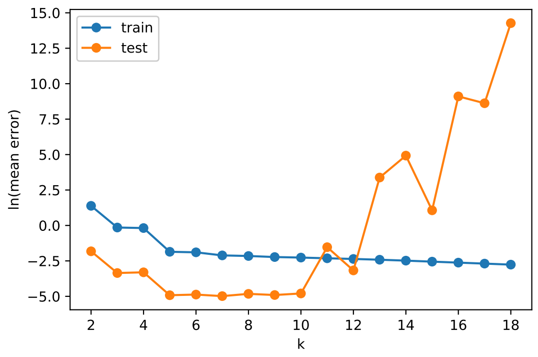


Figure 8: Test and Train error plotted as a function of k, with the results averaged over 100 runs using a sinusoidal basis for our feature map.

1.2 Filtered Boston housing and kernels

The Boston Housing data set is a classic data set in machine learning, using 13 predictive values we can predict the price of a house in a given neighbourhood. We'll use this data set in this section to extend our discussion of linear regression using kernel methods. We begin by comparing how well baseline linear regression - i.e. predicting with a constant function, is changed by adding more features. Performing a 2:1 train-test split of our data, the MSE of predicting with a constant function, a given single feature of the data set as well as a bias term, and all the features of the data set combined as well as a bias term, using linear regression is given in table (1). We observe that linear regression with any single feature and a bias term improves MSE in comparison to predicting with a constant function, which is again further improved by using all of the features in conjunction with each other and a bias term. We note that predicting with a constant function is akin to finding a mean of the training set labels. We note that Feature 13 corresponds to the percentage of 'lower class' people in a given neighbourhood - and it may be ethically suspect to include it in any predictive analysis as it encodes people's bias. However, we also see that if it alone is used to train our model, the MSE is actually the smallest of any regression model constructed with a single feature. This is probably due to the distorting effects of class and race on the American housing market, and people's resultant perception of value.

Table 1: Regression Methods, MSE reported to 3 significant figures.

Method	MSE train	MSE test
Naive Regression	88.1	77.4
Linear Regression (Feature 1)	75.7	64.2
Linear Regression (Feature 2)	78.3	64.2
Linear Regression (Feature 3)	69.8	55.2
Linear Regression (Feature 4)	83.4	79.8
Linear Regression (Feature 5)	72.7	62.3
Linear Regression (Feature 6)	45.8	39.1
Linear Regression (Feature 7)	76.7	64.5
Linear Regression (Feature 8)	82.8	72.4
Linear Regression (Feature 9)	75.3	66.2
Linear Regression (Feature 10)	70.3	57.5
Linear Regression (Feature 11)	67.0	54.1
Linear Regression (Feature 12)	78.5	68.4
Linear Regression (Feature 13)	38.7	38.4
Linear Regression (All Features)	23.0	20.7
Kernel Ridge Regression	12.4	11.2

1.3 Kernelised ridge regression

Intuitively linear regression will fail to achieve great results with non-linear data sets, here we repeat our experiments in the previous section making use of Kernel ridge regression with the following Gaussian kernel,

$$K(\boldsymbol{x}_i, \boldsymbol{k}_j) = \exp(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}) \tag{21}$$

As the feature space of the Gaussian kernel will be infinite dimensional we work in the dual, solving the following dual optimisation problem,

$$\boldsymbol{\alpha}^* = \text{argmin}_{\boldsymbol{\alpha} \in \mathbb{R}^l} \frac{1}{l} \sum_{i=1}^{l} (\sum_{i=j}^{l} \alpha_j K_{i,j} - y_i)^2 + \gamma \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} \tag{22}$$

The solution of which we simply quote below,

$$\boldsymbol{\alpha}^* = (\boldsymbol{K} + \gamma l \boldsymbol{I}_l)^{-1} \boldsymbol{y} \tag{23}$$

For which a test point is evaluated as,

$$y_{\text{test}} = \sum_{i=1}^{l} \alpha_i^* K(\boldsymbol{x}_i, \boldsymbol{x}_{test}) \tag{24}$$

There are optimum parameters choices for σ - the standard deviation of the Gaussian kernel, and γ - the regularisation parameter. We can find these using K-Fold cross validation to exhaustively test all possible combinations of these parameters for the combination that produces the least test error. We have taken all combinations of $\gamma \in [2^{-40}, 2^{-39}, ..., 2^{-26}]$ and $\sigma \in [2^{7}, 2^{7.5}, ..., 2^{13}]$, and performed 5-fold cross-validation. We can plot the cross-validation error, which we've taken as a mean of the MSE over each fold, against γ and σ. This is shown in figure (9). The MSE calculated with the optimal parameters of γ_{opt} and σ_{opt} is shown in table (1).

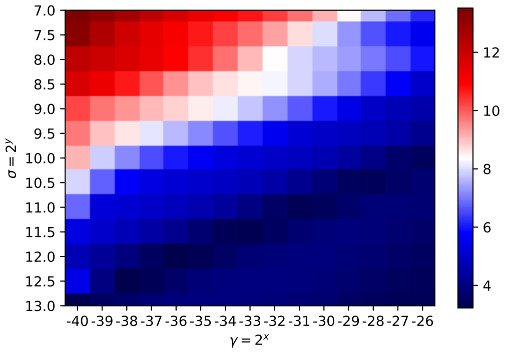


Figure 9: Cross Validation Error as a function of σ and γ. Here we plot $\ln(error))$ to show the contrast

In order to get some empirical bounds for the performance of all of our regression techniques, we can repeat all of the experiments above 20 times with random 2:1 train-test splits. Note that Kernel ridge regression is repeated with the optimum parameters only. The results of this are shown in table (2).

Part 2

A Bayes estimator, $\hat{y}^*$, is an estimator that minimizes the posterior expected value of a loss function $L(y, \hat{y})$,

$$\hat{y}^* := \text{argmin}_{\hat{y}} \varepsilon(\hat{y}) \tag{25}$$

where $\varepsilon(\hat{y})$ is defined as,

$$\varepsilon(\hat{y}) := \mathbb{E}(L(y, \hat{y})) \tag{26}$$

In this question, we overload $[\cdot]$ as $[n] := \{1, 2, ..., n\}$ if n is a positive integer, and $[pred] = 1$ if a logical predicate is true, or $[pred] = 0$ if it is false. We consider a probability mass function $p(x, y)$ over X and Y, where they are both finite, and therefore $\sum_{x \in X} \sum_{y \in Y} p(x, y) = 1$.

Table 2: Regression Methods each repeated 20 times, standard deviation reported to 1 decimal place.

Method	MSE train	MSE test
Naive Regression	83.5 ± 5.2	86.6 ± 10.7
Linear Regression (Feature 1)	70.7 ± 5.4	74.5 ± 10.9
Linear Regression (Feature 2)	72.1 ± 5.0	76.7 ± 10.3
Linear Regression (Feature 3)	64.0 ± 5.2	66.3 ± 10.7
Linear Regression (Feature 4)	80.4 ± 5.0	85.5 ± 10.5
Linear Regression (Feature 5)	68.1 ± 5.2	71.2 ± 10.6
Linear Regression (Feature 6)	43.7 ± 2.5	43.9 ± 5.2
Linear Regression (Feature 7)	71.4 ± 5.6	75.0 ± 11.3
Linear Regression (Feature 8)	78.1 ± 5.5	81.8 ± 11.2
Linear Regression (Feature 9)	71.5 ± 5.6	73.6 ± 11.4
Linear Regression (Feature 10)	65.3 ± 5.5	67.4 ± 11.3
Linear Regression (Feature 11)	62.1 ± 4.2	64.1 ± 8.8
Linear Regression (Feature 12)	74.2 ± 4.9	77.0 ± 10.1
Linear Regression (Feature 13)	37.5 ± 2.5	40.9 ± 5.3
Linear Regression (All Features)	21.1 ± 1.3	24.9 ± 3.0
Kernel Ridge Regression	10.8 ± 0.7	15.1 ± 2.2

For $Y = [k] = \{1, ..., k\}$ and $\boldsymbol{c} = [0, \infty)^k$ a vector of k costs, a loss $L_{\boldsymbol{c}}$ is defined as,

$$L_{\boldsymbol{c}} := [y \neq \hat{y}]c_y \tag{27}$$

The Bayes estimator of this is found as follows,

$$\begin{gathered}\varepsilon(\hat{y}^*) = \text{argmin}_{\hat{y}}\mathbb{E}[L_{\boldsymbol{c}}] \\ \varepsilon(\hat{y}^*) = \text{argmin}_{\hat{y}}\mathbb{E}[c_y[y \neq \hat{y}]] \\ \varepsilon(\hat{y}^*) = \text{argmin}_{\hat{y}} \sum_{x\in X}\sum_{y\in Y} c_y[y \neq \hat{y}]p(x, y) \\ \varepsilon(\hat{y}^*) = \text{argmin}_{\hat{y}} \sum_{x\in X}[\sum_{y\in Y} c_y[y \neq \hat{y}]p(y|x)]p(x)\end{gathered} \tag{28}$$

considering just a single point $x' \in X$,

$$\begin{gathered}e = \varepsilon(\hat{y}^*(x')) = \text{argmin}_{\hat{y}} \sum_{y\in Y} c_y[y \neq \hat{y}(x')]p(y|x')p(x') \\ e \propto \sum_{y\in Y} c_y[y \neq \hat{y}(x')]p(y|x')\end{gathered} \tag{29}$$

We can then see that the sum e is minimised when $p(y|x')$ is maximised, so $\hat{y}$ is taken to be the 'most likely' value - or the mode of the data set.

Now consider the following loss function,

$$L(y,\hat{y}) := |y-\hat{y}| \tag{30}$$

We can derive the Bayes estimator using a similar strategy which results in an expression that mirrors (29),

$$e \propto \sum_{y\in Y} |y-\hat{y}(x')|p(y|x') \tag{31}$$

Again we can find when this sum is minimised, to do this we optimise with respect to $\hat{y}(x')$

$$\begin{aligned} e &\propto \sum_{y\in Y} |y-\hat{y}(x'))|p(y|x') \\ &= \sum_{1}^{\hat{y}(x')} (y-\hat{y}(x'))p(y|x') + \sum_{\hat{y}(x')}^{k} (y-\hat{y}(x'))p(y|x') \end{aligned} \tag{32}$$

differentiating and setting to 0,

$$\begin{aligned} \sum_{1}^{\hat{y}(x')} p(y|x') &= \sum_{\hat{y}(x')}^{k} p(y|x') \\ \Rightarrow 2\sum_{1}^{\hat{y}(x')} p(y|x') &= \sum_{y\in Y} p(y|x') = 1 \\ \sum_{1}^{\hat{y}(x')} p(y|x') &= \frac{1}{2} \end{aligned} \tag{33}$$

We therefore find that the Bayes estimator for this loss function can be interpreted as the median of the data set.

Now let's consider the kernel,

$$K_c(\boldsymbol{x},\boldsymbol{z}) := c + \sum_{i=1}^{n} x_i z_i \tag{34}$$

where $\boldsymbol{x},\boldsymbol{z}\in\mathbb{R}^n$ and $c\in\mathbb{R}$. For what values of c does K_c fulfill the properties of a positive semi-definite kernel? This is equivalent to requiring that any finite matrix constructed by pairwise evaluation, $K_c^{i,j} = K_c(x_i,x_j)$, has either entirely non-negative eigenvalues. Equivalently, using the following theorem, a Kernel function is PSD iff,

$$K_c(\boldsymbol{x},\boldsymbol{z}) = \langle\Phi(\boldsymbol{x}),\Phi(\boldsymbol{z})\rangle\,, \tag{35}$$

Where $\Phi(\cdot)$ defines a feature map $\Phi:\mathbb{R}^n\to W$ where W is a Hilbert space. By observation we note that (34) can be written as,

$$K_c(\boldsymbol{x}, \boldsymbol{z}) = \begin{pmatrix} x_1 & x_2 & ... & x_n & \sqrt{c} \end{pmatrix}^T \begin{pmatrix} z_1 \\ z_2 \\ ... \\ z_n \\ \sqrt{c} \end{pmatrix} = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{z}) \rangle \tag{36}$$

Where $\Phi(\boldsymbol{x}) = (x_1, x_2, ..., x_n, \sqrt{c})$. Therefore the constraint is simply that $c \geq 0$ in order for this kernel function to be PSD [1].

Consider performing linear regression with the following Gaussian kernel,

$$K_\beta(\boldsymbol{x}, \boldsymbol{t}) = \exp(-\beta \|\boldsymbol{x} - \boldsymbol{t}\|^2) \tag{37}$$

To train a classifier on $\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_m, y_m)\} \in \mathbb{R}^n \times \{-1, 1\}$. In what scenario does the choice of β result in this algorithm emulate the action of the 1-Nearest Neighbours classifier on the same data set? In fact there are a family of choices of β that would fulfill this condition. Consider the following choice,

$$\beta = \frac{\|\boldsymbol{x} - \boldsymbol{t}\|^2_{\min}}{\|\boldsymbol{x} - \boldsymbol{t}\|^p_{\min}} = \frac{d^2_{\min}}{d^p_{\min}} \tag{38}$$

where $p \in \mathbb{R}$, and $d_{\min}$ is the minimum distance between two points $\boldsymbol{x}, \boldsymbol{t} \in \mathbb{R}^n$ in the data set. The value of the kernel evaluated between these two points then goes to,

$$K^{\min}_\beta = \exp(-d^{-p}_{\min}) \tag{39}$$

If we choose to work in a norm such that $d_{\min} = 1$ for convenience, then we arrive at,

$$K^{\min}_\beta = \exp(-1) = \frac{1}{e} \tag{40}$$

for any other $d_+ > d_{\min}$,

$$\lim_{p \to -\infty} K^+_\beta = \lim_{p \to -\infty} \exp(-d^{-p}_+) = 0 \tag{41}$$

Similarly for $d_- < d_{\min}$,

$$\lim_{p \to -\infty} K^-_\beta = \lim_{p \to -\infty} \exp(-d^{-p}_-) = 1 \tag{42}$$

Furthermore assume that the data we are regressing on is distributed in such a way that the minimum separating distances of one point from any other point d evaluated in a given norm $\|\cdot\|$ lies in the interval $d \in [d_{\min} - \xi, d_{\min} + \xi]$ where ξ is a 'small' number such that,

$$\begin{aligned} K^{\min}_\beta &= \exp(-(1 \pm \xi)^{-p}) \\ K^{\min}_\beta &= \frac{1}{e} \pm \frac{p\xi}{e} + O(\xi^2) \approx \frac{1}{e}(1 \pm p\xi) \end{aligned} \tag{43}$$

and we impose a condition on ξ such that $|\xi p| << 1$. Then we can infer that there exists a choice of norm, and a choice of p such that the kernel function evaluated between a test point and a given data point from the training set tends to 0 for all other points in the training set if the test point is 'closer' to the given evaluation point in terms of our chosen norm. Therefore making predictions with such a kernel will emulate the action of 1-Nearest Neighbours, where the heaviest weight for our prediction will come from the nearest point in the training set.

For the final question in this coursework we are consider the generalised problem of 'Whack-a-Mole' for an $n \times n$ game board, for which we want to find a polynomial time solution. We begin by noting that hitting a given mole an even number of times results in the board resetting to it's position before hitting commenced, therefore it's not the number of times a hit takes place that's important - but whether a hit took place (odd times) or did not take place at all (even times).

The configuration of the board at each step of the algorithm can be viewed in matrix form, $M \in \mathbb{R}^n \times \mathbb{R}^n$, where each entry is in $\{0, 1\}$ signifying the absence/presence of a mole respectively. The action of hittting a mole at position (i, j) can be interpreted as a matrix addition of the form,

$$M + A_{i,j} \tag{44}$$

Note that as matrix additions are commutative the order of hits does not matter. Let's signify the number of times a particular position has to be hit as $x_{i,j}$, in which case our board M is completely free of moles when,

$$M + \sum_{i,j} x_{i,j} A_{i,j} = 0 \tag{45}$$

alternatively we can interpret this as a Matrix-Vector product,

$$\begin{gathered} \sum_{i,j} x_{i,j} A_{j,i} = M \\ A\boldsymbol{x} = \boldsymbol{M} \end{gathered} \tag{46}$$

Where A is an $n^2 \times n^2$ matrix of all the possible actions. In order to find the hits needed to solve the board, $\boldsymbol{x}$, all we need to do is invert the matrix A,

$$\boldsymbol{x} = A^{-1}\boldsymbol{M} \tag{47}$$

As all of the actions are different A will be a full-rank matrix, therefore an inverse does exist for A. Well known algorithms can be used for the inversion, for example Gaussian elimination, which can do this calculation in $O(n^3)$ - polynomial time [2].

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] E. Süli and D.F. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003.