Statistical Learning CW1

Agnieszka Dobrowolska, Marco Carobene

November 14, 2019

1 PART I

1.1 Linear Regression

1.1.1 Polynomial Regression

In univariate polynomial regression, we fit a linear regression model to a polynomial basis of order k: $\{\phi_1(x) = 1, \phi_2 = x, \phi_3 = x^2, \ldots, \phi_k = x^{k-1}\}$.
In this section, we will investigate the effect of fitting polynomial regression models of increasing order.
In particular, we fit models using polynomial bases of order $k = 1, 2, 3, 4$ to the following set of points: $\{(1, 3), (2, 2), (3, 0), (4, 5)\}$.

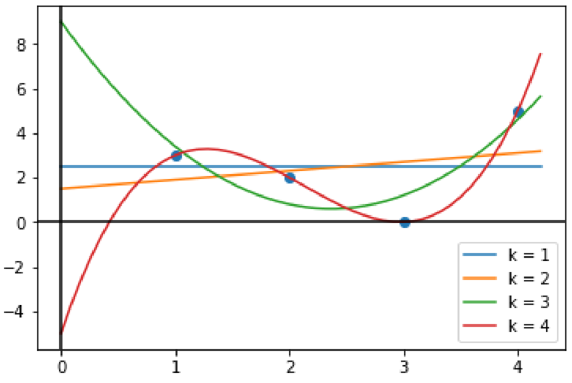


Figure 1: Plot of polynomial curves fit by each k-order polynomial basis.

$k =$	Curve Equation	MSE
1	2.5	3.25
2	$1.5 + 0.4x$	3.05
3	$9 - 7.1x + 1.5x^2$	0.80
4	$-5 + 15.17x - 8.5x^2 + 1.33x^3$	1.59×10^{-23}

As we can see from these MSE values, by increasing the degree of the polynomial basis, we can reduce the error made by the model on our training data. This may seem like a good thing at first, but as Figure 1 shows, we seem to be potentially *overfitting* our training data quite drastically by the time we get to $k = 4$, and this model is therefore unlikely to make accurate predictions on unseen data. We will investigate this phenomena in the next section.

1.1.2 Overfitting

We define the function

$$g_\sigma(x) := sin^2(2\pi x) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

And create the following dataset by taking 30 samples of $x_i \sim U[0, 1]$, then applying $g_{0.07}$, to achieve:

$$S_{0.07,30} = \{(x_1, g_{0.07}(x_1)), \dots, (x_{30}, g_{0.07}(x_{30}))$$

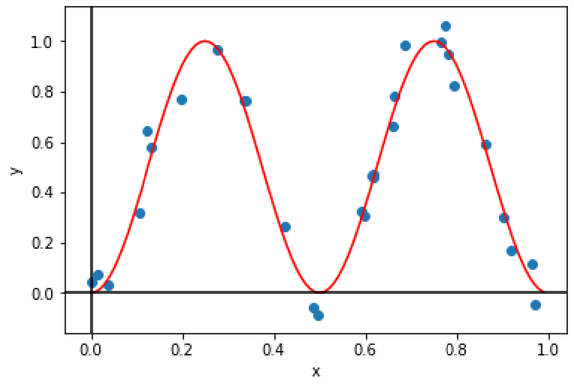


Figure 2: Plot of function $y = \sin^2(2\pi x)$ superimposed on our sample $S_{0.07,30}$

We examine the effect of fitting a polynomial regression curve to these data, of order $k = 2, 5, 10, 14, 18$:

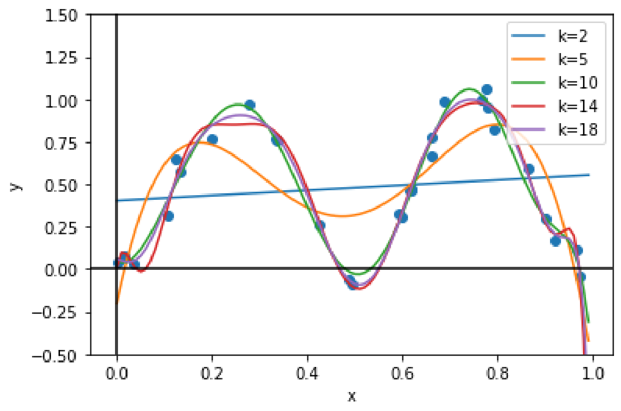


Figure 3: Plot of curves fit using k-order polynomial bases on our sample $S_{0.07,30}$

Define Training Error $te_k(S)$ = MSE of the fitting of S using a polynomial basis of order k. We plot this in Figure 4 to examine the effect of increasing the order of our polynomial basis.

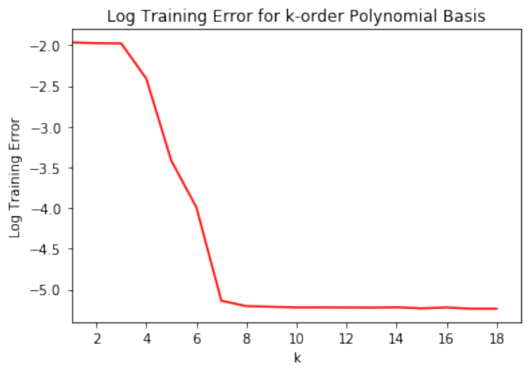


Figure 4: Plot of k against $\log(te_k(S))$

We can recognise immediately that the training error, $te_k(S)$, is a decreasing function of k. This suggests that as we increase the order of the polynomial basis, our model is able to more closely fit the training data.

We now generate a test set, T, using the same method as how we generated S:

$$T_{0.07,1000} = \{(x_1, g_{0.07}(x_1)), \ldots, (x_{1000}, g_{0.07}(x_{1000}))$$

Define Test Error $tse_k(S,T)$ = MSE of the test set T using a polynomial basis of order k, trained on set S. We examine the following phenomenon:

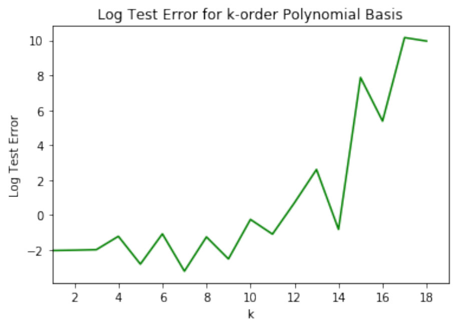


Figure 5: Plot of k against $\log(tse_k(S,T))$

Unlike the training error, the test error $tse_k(S,T)$ is seen to start as an increasing function of k, but suddenly starts increasing for $k > 5$. This suggests that when the order of the polynomial basis gets too high, we end up overfitting the training data. This means that the model cannot generalise to the unseen test data, and as a result makes large errors on its predictions.
We exemplify this difference by averaging both errors over 100 runs:

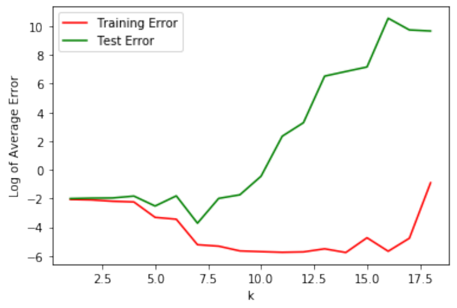


Figure 6: Effect of increasing the order of the polynomial basis on $\log(avg(te_k(S)))$ **(red)**, and $\log(avg(tse_k(S,T)))$ **(green)**

1.1.3 Sine Basis

An alternative basis to use when is the following Sine Basis of order k:

$$\{\phi_1(x) = \sin(1\pi x), \phi_2 = \sin(2\pi x), \phi_3 = \sin(3\pi x), \ldots, \phi_k = \sin(k\pi x)\}$$

As we happen to know the nature of where our data has come from, we may expect that a basis featuring the sine function may fit a better model to our data. We will investigate this claim by applying this basis with orders $k = 1, 2, ..., 18$. We first investigate the training and test errors of these bases on $S_{0.07,30}$ and $T_{0.07,1000}$:

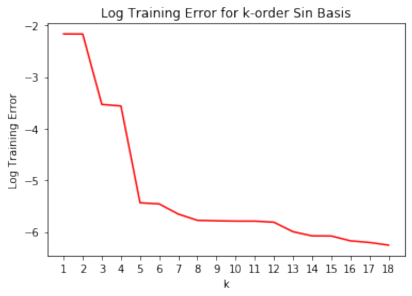


Figure 7: k against $\log(te_k(S))$

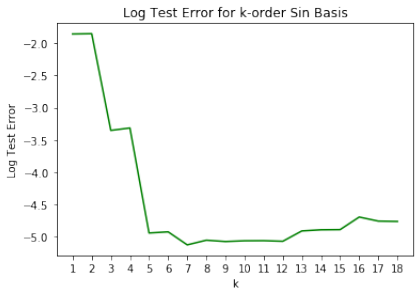


Figure 8: k against $\log(tse_k(S, T))$

Again, it helps to exemplify the evolutionary difference between the Training and Test Errors for each k-order basis, averaged over 100 trials:

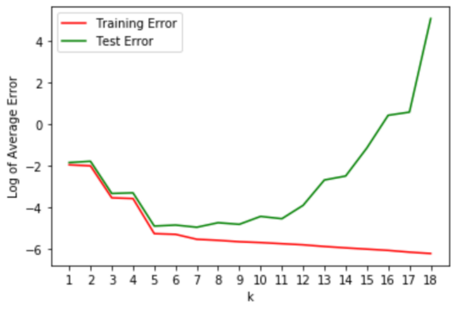


Figure 9: Effect of increasing the order of the sine basis on the $\log(avg(te_k(S)))$ **(red)**, and $\log(avg(tse_k(S, T)))$ **(green)**.

Using a sine basis with order $k = 5$ gives us the lowest average test error of all seen bases. Hence this is the optimal choice of basis for this dataset.

1.2 Boston Housing and Kernels

In this section, we will be using the Boston Housing dataset (the original 14 column version). As this dataset features a range of explanatory variables, we are able to investigate the differences between baseline, univariate and multivariate linear regression models.
We will also extent this investigation to a Kernelised Ridge Regression model.

For each of the models, we will perform 20 train-test splits of the data (each being split 2/3-1/3) and averaging our results.

1.2.1 Baseline Model

We start by fitting a Naive Regression model to the data. This involves using a vector of ones in place for our X matrix of input values.
In this case, we have that our fitted weights are intended to minimise
$SSE = \sum_{i=1}^{N}(y_i - \boldsymbol{w}^\intercal \boldsymbol{x}_i)^2 = \sum_{i=1}^{N}(y_i - w)^2$.

$$\frac{\partial}{\partial w} SSE = 0 = 2\sum_{i=1}^{N} y_i - 2N\hat{w}$$
$$\hat{w} = \frac{1}{N}\sum_{i=1}^{N} y_i = \bar{y}$$

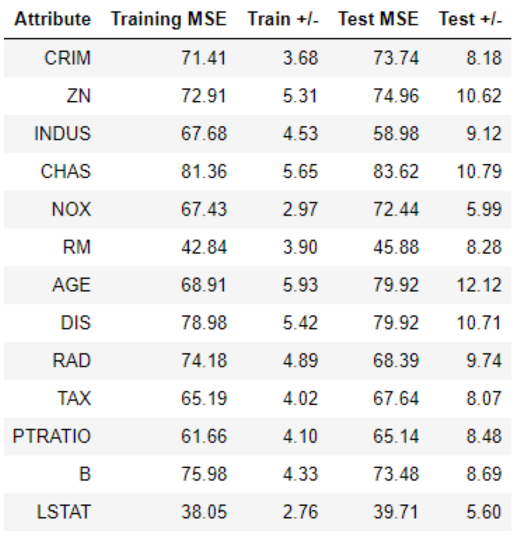
Thus it is evident that fitting with naive regression is equivalent to predicting with the mean y-value of the training set.

Fitting a Naive Regression model to the dataset gives us the following average MSE values,

<u>Average Training MSE:</u> 85.68
<u>Average Test MSE:</u> 82.14

1.2.2 Simple Linear Regression

We now fit a series of Univariate Linear Regression models to the training data. For each of the 13 attributes in the dataset, we will fit a linear regression model using the single attribute and a bias term, i.e. $\boldsymbol{x} = (1, x_i)$.
Each of the Univariate Linear Regression Models give us the following average MSE values:



Attribute	Training MSE	Train +/-	Test MSE	Test +/-
CRIM	71.41	3.68	73.74	8.18
ZN	72.91	5.31	74.96	10.62
INDUS	67.68	4.53	58.98	9.12
CHAS	81.36	5.65	83.62	10.79
NOX	67.43	2.97	72.44	5.99
RM	42.84	3.90	45.88	8.28
AGE	68.91	5.93	79.92	12.12
DIS	78.98	5.42	79.92	10.71
RAD	74.18	4.89	68.39	9.74
TAX	65.19	4.02	67.64	8.07
PTRATIO	61.66	4.10	65.14	8.48
B	75.98	4.33	73.48	8.69
LSTAT	38.05	2.76	39.71	5.60

1.2.3 Full Linear Regression

We now examine the performance of our full model, a Multivariate Linear Regression model which learns from all 13 of our attributes and a bias term, i.e. $\boldsymbol{x} = (1, x_1, \ldots, x_{13})$. This model achieves the following average MSE values,

Average Training MSE: 21.88
Average Test MSE: 23.46

Which, as expected, outperforms all of the univariate models.

1.3 Kernelised Ridge Regression

The use of a kernel function can often be beneficial to the performance of a regression model for both the training and the test data.
Here, we will perform Kernel Ridge Regression using the Gaussian Kernel,

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

Let ℓ be the number of training examples. Once we have a fitted model, we can predict on a new input $\boldsymbol{x}'$ via the following:

$$y_{\text{pred}} = \sum_{i=1}^{\ell} \alpha_i^* K(\boldsymbol{x}_i, \boldsymbol{x}')$$

where we have calculated our optimal $\boldsymbol{\alpha}^*$ using:

$$\boldsymbol{\alpha}^* = (\boldsymbol{K} + \gamma\ell\mathbb{I}_\ell)^{-1}\boldsymbol{y}$$

We start by using 5-fold cross-validation to select the optimal values of γ and σ. We optimise our choices of $\gamma \in \{2^{-40}, 2^{-39}, \dots, 2^{-26}\}$, and $\sigma \in \{2^7, 2^{7.5}, \dots, 2^{12.5}, 2^{13}\}$. As some values of the error are relatively very large, we visualise the natural logarithm of the "cross-validation error", the validation error averaged over folds,

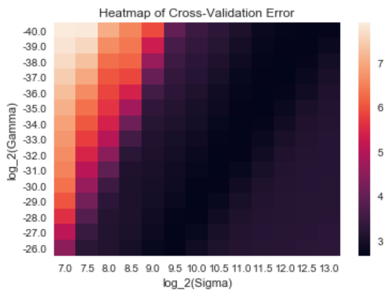


Figure 10: Heatmap displaying the natural log of the Cross-Validation Error for each choice of γ and σ

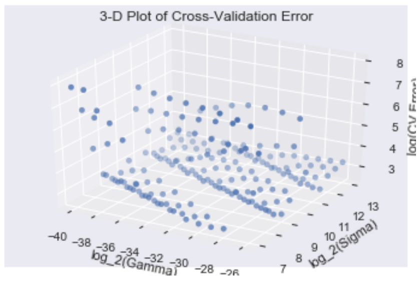
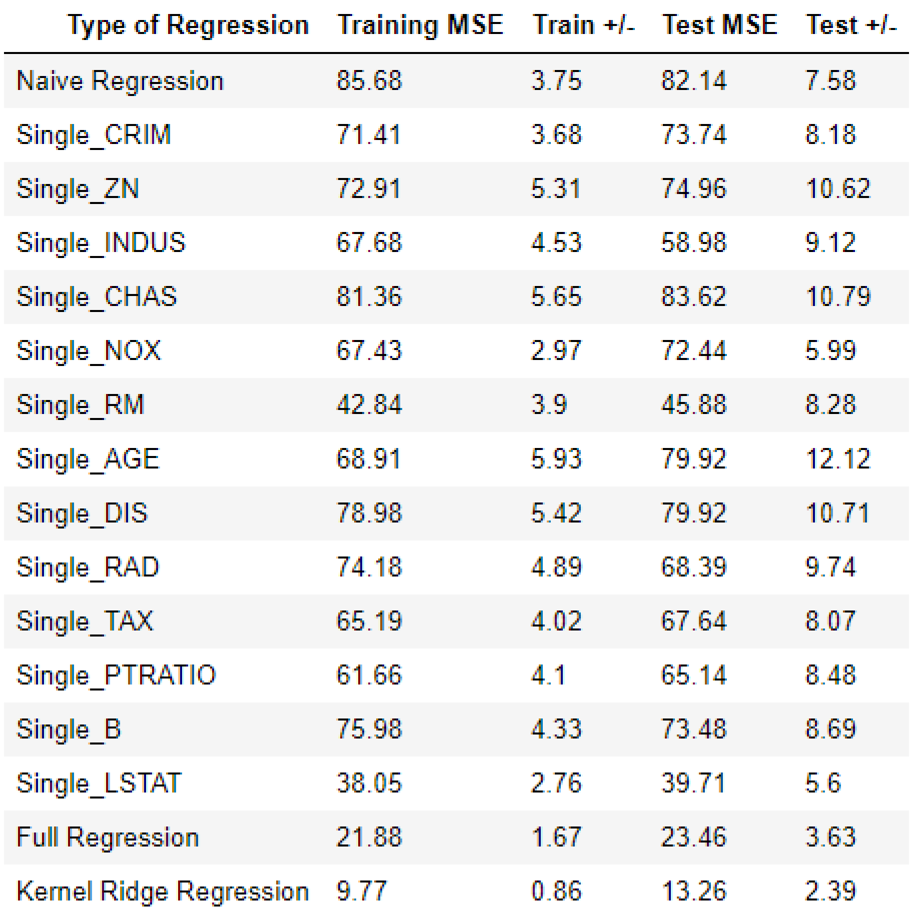


Figure 11: 3d plot displaying the natural log of the Cross-Validation Error for each choice of γ and σ

Via cross-validation, we determine $\gamma = 2^{-31}$, $\sigma = 2^{10}$ to be our best choice of our parameters. This selection of the parameters gives the following average MSE values,

Average Training MSE: 9.77
Average Test MSE: 13.26

1.3.1 Conclusion



Type of Regression	Training MSE	Train +/-	Test MSE	Test +/-
Naive Regression	85.68	3.75	82.14	7.58
Single_CRIM	71.41	3.68	73.74	8.18
Single_ZN	72.91	5.31	74.96	10.62
Single_INDUS	67.68	4.53	58.98	9.12
Single_CHAS	81.36	5.65	83.62	10.79
Single_NOX	67.43	2.97	72.44	5.99
Single_RM	42.84	3.9	45.88	8.28
Single_AGE	68.91	5.93	79.92	12.12
Single_DIS	78.98	5.42	79.92	10.71
Single_RAD	74.18	4.89	68.39	9.74
Single_TAX	65.19	4.02	67.64	8.07
Single_PTRATIO	61.66	4.1	65.14	8.48
Single_B	75.98	4.33	73.48	8.69
Single_LSTAT	38.05	2.76	39.71	5.6
Full Regression	21.88	1.67	23.46	3.63
Kernel Ridge Regression	9.77	0.86	13.26	2.39

2 PART II

2.1 Bayes Estimator

We note that the Bayes Estimator for a loss function is defined to be f^* s.t.

$$f^* := \arg\min_f \{\mathcal{E}(f)\}$$

where $\mathcal{E}(f) = \mathbb{E}_{X,Y}[L(y, f(x))]$.

a) Consider k-class classification, s.t. $Y = [k]$ and $\boldsymbol{c} \in [0, \infty)^k$ the vector of costs.

We define

$$L_{\boldsymbol{c}}(y, \hat{y}) := \mathbb{I}_{[y \neq \hat{y}]} c_y$$

Thus,

$$\mathbb{E}[L(y, f(x))] = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} \mathbb{I}_{[y \neq f(x)]} c_y P(x, y)$$

We consider the task of finding f^* for a specific $x = x'$.
i.e. We minimise the function

$$\begin{aligned}\mathbb{E}[L|X = x'] &= \sum_{y=1}^{k} \mathbb{I}_{[y \neq f(x')]} c_y P(x', y) \\ &\propto \sum_{y=1}^{k} \mathbb{I}_{[y \neq f(x')]} c_y P(y|x')\end{aligned}$$

Set $z = f(x')$ **Note:** $\sum_{y=1}^{k} c_y P(y|x') = \sum_{y=1}^{k} (\mathbb{I}_{[y=z]} + \mathbb{I}_{[y \neq z]}) c_y P(y|x')$.
Thus, $\sum_{y=1}^{k} \mathbb{I}_{[y \neq z]} c_y P(y|x') = \sum_{y=1}^{k} (1 - \mathbb{I}_{[y=z]}) c_y P(y|x')$.

$$\begin{aligned}\mathbb{E}[L|X = x'] &= \sum_{y=1}^{k} c_y P(y|x') - \sum_{y=1}^{k} \mathbb{I}_{[y=z]} c_y P(y|x') \\ &= \mathbb{E}_{Y|X=x'}[c_y] - c_z \mathbb{P}(Y = z|X = x')\end{aligned}$$

This function is minimised for

$$\boxed{f^*(x') = \arg\max_{z \in [k]} \{c_z \mathbb{P}(Y = z|X = x')\}}$$

b) We set $Y \subset \mathbb{R}$, and define the loss function as follows,

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

Thus,

$$\mathbb{E}[L(y, f(x))] = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} |y - f(x)| P(x, y)$$

Again, consider finding f^* at a specific point $x = x'$, and set $z = f(x')$.

$$\begin{aligned}\mathbb{E}[L(y, z)|X = x'] &\propto \sum_{y \in \mathcal{Y}} |y - z| P(y|x') \\ &= \sum_{y<z} (z - y) P(y|x') + \sum_{y>z} (y - z) P(y|x')\end{aligned}$$

We look to minimise this w.r.t. z, thus we take the derivative and set it equal to 0:

$$\frac{\partial}{\partial z} \mathbb{E}[L(y, z)|X = x'] = 0 = \sum_{y<z} P(y|x') - \sum_{y>z} P(y|x')$$

i.e. We select $f^*(x')$ to be the choice of z for which $\sum_{y<z} P(y|x') = \sum_{y>z} P(y|x')$. Thus:

$$\boxed{f^*(x') = Median[P(y|x')]}$$

2.2 Kernel Modification

a) Given K_c is defined as:

$$K_c(\mathbf{x}, \mathbf{z}) := c + \sum_{i=1}^{n} x_i z_i, \text{ where } \mathbf{x}, \mathbf{z} \in \mathbb{R}^n$$

The kernel function K_c is a positive semi-definite (PSD) kernel function if its respective kernel matrix, $\mathbf{K}$ (s.t. $\mathbf{K}_{ij} = K_c(\mathbf{x}_i, \mathbf{x}_j)$ where $\mathbf{x}_i, \mathbf{x}_j$ are input data), is such that:

$$\boldsymbol{y}^T \mathbf{K} \boldsymbol{y} \geq 0, \text{ for any } \boldsymbol{y} \in \mathbb{R}^n$$

We have that

$$\begin{aligned}
\boldsymbol{y}^T \mathbf{K} \boldsymbol{y} &= \sum_{i=1}^{n} \sum_{j=1}^{n} y_i K_c(\mathbf{x}_i, \mathbf{x}_j) y_j \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} y_i [c + \sum_{k=1}^{n} x_{ik} x_{jk}] y_j \\
&= c \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j + \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} y_i y_j x_{ik} x_{jk} \\
&= \underbrace{c \Big(\sum_{i=1}^{n} y_i \Big)^2}_{\geq 0} + \underbrace{\sum_{k=1}^{n} \Big(\sum_{i=1}^{n} y_i x_{ik} \Big)^2}_{\geq 0}
\end{aligned}$$

The only way to ascertain that $\boldsymbol{y}^T \mathbf{K} \boldsymbol{y} \geq 0$ for any $\boldsymbol{y} \in \mathbb{R}^n$, for any set of input data $\{\mathbf{x}_i\}_{i=1}^{N}$, is if we set $c \geq 0$.
Therefore,

$$\boxed{K_c \text{ is only certain to be a PSD kernel for } c \geq 0}$$

b) The use of a kernel function in Linear Regression (with least squares) arises when we train the model using a basis function, $\boldsymbol{\phi}(.)$.

In the *primal form*, we are attempting to find the following optimal setting of $\mathbf{w}$:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \Big\{ \sum_{i=1}^{n} (y_i - \mathbf{w}^\intercal \boldsymbol{\phi}(\mathbf{x}_i))^2 \Big\} \tag{1}$$

As derived in lectures, the *dual form* of this optimisation problem results in fitting a new weights parameter $\boldsymbol{\alpha}$ via:

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \Big\{ \sum_{i=1}^{n} \sum_{j=1}^{n} (y_i - \alpha_j K_{i,j})^2 \Big\} \tag{2}$$

where $K_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j) = \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle$

This means that for any kernelised linear regression, we can relate the problem in (2) to that in (1) via the derivation of a basis function, $\boldsymbol{\phi}(.)$, which generates the kernel K.

As the question asks, we now consider using the kernel K_c for linear regression (with least squares). We note that:

$$\begin{aligned} K_c(\mathbf{x}, \mathbf{z}) &= c + \sum_{i=1}^{n} x_i z_i \\ &= c + \langle \mathbf{x}, \mathbf{z} \rangle \\ &= \langle \boldsymbol{\psi}(\mathbf{x}), \boldsymbol{\psi}(\mathbf{z}) \rangle \end{aligned}$$

by setting $\boldsymbol{\psi}(\mathbf{x}) := \begin{pmatrix} \sqrt{c} \\ \mathbf{x} \end{pmatrix}$.

Thus, when using the kernel K_c to do linear regression, we are solving the optimisation problem

$$\begin{aligned} \mathbf{w}^* &= \arg\min_{\mathbf{w}} \Big\{ \sum_{i=1}^{n} (y_i - \mathbf{w}^\intercal \boldsymbol{\psi}(\mathbf{x}_i))^2 \Big\} \\ &= \arg\min_{w_0, \mathbf{w}_{1:d}} \Big\{ \sum_{i=1}^{n} (y_i - (w_0\sqrt{c} + \mathbf{w}_{1:d}^\intercal \mathbf{x}_i))^2 \Big\} \end{aligned}$$

where $\mathbf{w} = \begin{pmatrix} w_0 \\ \mathbf{w}_{1:d} \end{pmatrix}$.

This has the exact same form as the classic multivariate linear regression problem, with the addition of a bias term, $w_0\sqrt{c}$, added the calculation of $\hat{y}_i$ for each $\mathbf{x}_i$. The choice of c is not too important, as the bias entry of the weight-vector, w_0, will rescale this bias term to reflect the true bias in the data.

There is also evidence to show that c introduces a bias term when we look at the solution to the problem in the dual form. In particular, the solution results in us setting

$$\boldsymbol{\alpha}^* = K^{-1}\boldsymbol{y}$$

It is evident from this expression that if we were to introduce a large value of c, this would dominate the values in the K matrix, and resultantly shrink all values of $\boldsymbol{\alpha}^*$ by the same amount. This seems like it may have some influence on the solution beyond a bias term until we examine how we predict for a new value of $\mathbf{x}'$:

$$\begin{aligned}\hat{y} &= \sum_{i=1}^{n} \alpha_i^* K(\mathbf{x}_i, \mathbf{x}') \\ &= \sum_{i=1}^{n} \alpha_i^* \langle \mathbf{x}_i, \mathbf{x}' \rangle + c \sum_{i=1}^{n} \alpha_i^*\end{aligned}$$

In particular, we again see the presence of a bias term (bias as this term is independent of x, and is therefore added to any arbitraty prediction) in our predictions of y. A large value of c will downscale the α_i^*s, but this downscaling will be counteracted by the bias term when we make a prediction.

2.3 Gaussian Kernels and NN

We fit a linear regression model to data s.t. $y \in \{-1, 1\}$, using a kernel $K_\beta(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\beta \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)$. How can we select a function $\hat{\beta}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m, \boldsymbol{t})$, given a new test point $\boldsymbol{t}$, such that our classifier simulates a *1-Nearest Neighbour Classifier* if we set $\beta = \hat{\beta}$?

We first examine the behavior of the 1-NN Classifier:
Our training data is composed of the following input data $\boldsymbol{x}_i$ and their classifications, y_i:

$$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$$

For a new point $\boldsymbol{x}$, for which we hope to predict a y category, we re-order the training set into $\{(\boldsymbol{x}_{(1)}, y_{(1)}), (\boldsymbol{x}_{(2)}, y_{(2)}), \ldots, (\boldsymbol{x}_{(m)}, y_{(m)})\}$ such that

$$\|\boldsymbol{x}_{(1)} - \boldsymbol{x}\| \leq \|\boldsymbol{x}_{(2)} - \boldsymbol{x}\| \leq \ldots \leq \|\boldsymbol{x}_{(m)} - \boldsymbol{x}\|$$

We then choose $\boldsymbol{x}$ to be categorised as $y_{(1)}$.

We now note that the Gaussian kernel also incorporates a distance metric into its formula, and explore the behaviour of the Gaussian kernel classifier.
Using the formulae as derived in lecture, we define the Kernel matrix to be s.t. $(K)_{ij} = K_\beta(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and we optimise our kernelised regression (without ridge) by setting

$$\boldsymbol{\alpha}^* = K^{-1}\boldsymbol{y}$$

where $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)^\intercal$.
This then allows us to make predictions via $\hat{y} = \sum_{i=1}^m \alpha_i K_\beta(\boldsymbol{x}_i, \boldsymbol{x})$.

If we consider the case of sending $\beta \to +\infty$ then when it comes to the kernel function acting on our training data, we have that

$$K_\beta(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\beta \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2) \to \begin{cases} 1, & \text{if } \boldsymbol{x}_i = \boldsymbol{x}_j. \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

So that the matrix $K \to \mathbb{I}_m$, and therefore $\boldsymbol{\alpha}^* \to \boldsymbol{y}$.
For a new data-point $\boldsymbol{t}$, we classify this point as

$$\hat{y} = \text{sign}[f(\boldsymbol{t})] = \text{sign}[\sum_{i=1}^m y_i \exp(-\frac{1}{\beta}\|\boldsymbol{x}_i - \boldsymbol{t}\|^2)]$$

Similarly to $1-NN$, we can reorder our training dataset into $\{(\boldsymbol{x}_{(1)}, y_{(1)}), (\boldsymbol{x}_{(2)}, y_{(2)}), \dots, (\boldsymbol{x}_{(m)}, y_{(m)})\}$ such that

$$\|\boldsymbol{x}_{(1)} - \boldsymbol{t}\| \leq \|\boldsymbol{x}_{(2)} - \boldsymbol{t}\| \leq \dots \leq \|\boldsymbol{x}_{(m)} - \boldsymbol{t}\|$$

So that we may rewrite $f(\boldsymbol{t})$ into the form

$$f(\boldsymbol{t}) = y_{(1)} e^{-\frac{1}{\beta}\|\boldsymbol{x}_{(1)} - \boldsymbol{t}\|^2} + y_{(2)} e^{-\frac{1}{\beta}\|\boldsymbol{x}_{(2)} - \boldsymbol{t}\|^2} + \dots + y_{(m)} e^{-\frac{1}{\beta}\|\boldsymbol{x}_{(m)} - \boldsymbol{t}\|^2}$$

If we iteratively increase β, effectively sending $\beta \to +\infty$, the terms in this summation will become approximately zero working backwards from the $(\boldsymbol{x}_{(m)}, y_{(m)})$ term, until the only non-zero term remaining is the first term. This term has sign equivalent to that of $y_{(1)}$, thus $\boldsymbol{t}$ is categorised to be the same as $y_{(1)}$ - meaning that this classifier behaves like a $1 - NNClassifier$.
Obviously, one cannot set $\beta = +\infty$ when using the kernel, and this would come with massive problems if it were possible. Namely that **all** terms in $f(\boldsymbol{t})$ would be $= 0$ unless $\boldsymbol{t}$ were exactly equal to some $\boldsymbol{x}_i$ in our training set. However, there will be some choice of $\hat{\beta}$ such that the argument above would hold.

Our choice for $\hat{\beta}$ is simply to set it equal to l, where l represents a very large number compared to all $\|\boldsymbol{x}_{(i)} - \boldsymbol{t}\|^2$.

There must be a trade-off between ensuring that all terms after $y_{(1)} e^{-\frac{1}{\beta}\|\boldsymbol{x}_{(1)} - \boldsymbol{t}\|^2}$ in our equation for $f(\boldsymbol{t})$ are ≈ 0, but that this first term is still somewhat maintained.

2.4 The Whack-A-Mole Problem

This solution was inspired and aided by the 1998 paper, *'Turning Lights Out with Linear Algebra'*, M. Anderson, T. Feil.

Let the current state of the board be represented by a modulo-2 matrix $B \in \mathbb{Z}_2^{n\times n}$, where

$$B_{ij} = \begin{cases} 1, & \text{if the mole is present} \\ 0, & \text{if the mole is hiding} \end{cases}$$

Due to the properties of modulo-2, hitting any given hole an even number of times from some initial state returns the same initial state. This means that any given hole should only be hit at most one time.

The change induced by hitting a hole B_{ij} can be represented by the addition of some modification matrix, H_{ij} such that

$$H_{ij} = \begin{pmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & ⋰ \\ \dots & 0 & 0 & 0 & 0 & 0 & \dots \\ \dots & 0 & 0 & 1 & 0 & 0 & \dots \\ \dots & 0 & 1 & 1 & 1 & 0 & \dots \\ \dots & 0 & 0 & 1 & 0 & 0 & \dots \\ \dots & 0 & 0 & 0 & 0 & 0 & \dots \\ ⋰ & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

where the displayed elements are centered about the (i,j)th entry. For example, in a 4x4 game then we may want to hit the hole B_{32}. Then our new state is found via,

$$B' = B + H_{32} = B + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Again due to properties of the B matrix being modulo-2, we have that it does not matter in which order the holes get hit, only which holes are hit during the course of the game.Thus we can write the problem in the following form:

We are attempting to find the set of binary indicator variables, s^*_{ij}, where

$$s^*_{ij} = \begin{cases} 1, & \text{if the (ij)th hole should be hit to win the game} \\ 0, & \text{otherwise} \end{cases}$$

so that

$$B + \sum_{i,j} s^*_{ij} H_{ij} = O$$

where O represents an $n \times n$ matrix of zeros.
i.e. we intend to find the set of holes that must be hit to solve the game, starting from its initial state.

We can vectorise this problem as follows:
We reshape the B matrix to be a vector $\mathbf{b}$, where

$$\mathbf{b} = [B_{11}, B_{12}, \ldots, B_{n(n-1)}, B_{nn}]^\intercal$$

we set

$$\mathbf{s}^* = [s^*_{11}, s^*_{12}, \ldots, s^*_{n(n-1)}, s^*_{nn}]^\intercal$$

and define a matrix $\mathbf{H}$ to be such that we vectorise and stack the H_{ij} matrices

$$\mathbf{H} = \begin{bmatrix} — & H_{11} & — \\ — & H_{12} & — \\ & \vdots & \\ — & H_{n(n-1)} & — \\ — & H_{nn} & — \end{bmatrix}$$

and consider the result of multiplying $\mathbf{Hs}^*$. We note that $\mathbf{H}$ is an $n^2 \times n^2$ *block tridiagonal matrix*, with I_n matrices along its lower and upper diagonals, and the matrix A along the main diagonal, where A is an $n \times n$ *tridiagonal matrix* with ones along its lower, main and upper diagonals.

Because this $\mathbf{H}$ matrix is trivially symmetric, then we recognise that the ijth row of $\mathbf{H}$ is equivalent to its ijth column, and that we may therefore retrieve the ijth row of $\mathbf{H}$ by multiplying it by the basis vector

$$\mathbf{e}_{ij} = \begin{bmatrix} 0 & 0 & \ldots & 0 & \underbrace{1}_{ij\text{th element}} & 0 & \ldots & 0 \end{bmatrix}^\intercal$$

i.e. $H_{ij} = \mathbf{He}_{ij}$.
We further recognise that, due to the fact that $\mathbf{s}^*$ is a vector of zeros and ones, then it may be written in the form $\mathbf{s}^* = \sum_{i,j:s^*_{ij}=1} \mathbf{e}_{ij}$. Therefore we can write

$$\begin{aligned} \sum_{i,j} s^*_{ij} H_{ij} &= \sum_{i,j:s^*_{ij}=1} H_{ij} \\ &= \sum_{i,j:s^*_{ij}=1} \mathbf{He}_{ij} \\ &= \mathbf{H} \sum_{i,j:s^*_{ij}=1} \mathbf{e}_{ij} \\ &= \mathbf{Hs}^* \end{aligned}$$

All of this results in us attempting to find a $\mathbf{s}^*$ satisfying the equation

$$\begin{aligned} &\mathbf{b} + \mathbf{Hs}^* = \mathbf{0} \\ &\Rightarrow \mathbf{Hs}^* = -\mathbf{b} \equiv \mathbf{b} \pmod 2 \end{aligned}$$

We are then able to solve for $\mathbf{s}^*$ simply via solving the system of linear equations given by $\mathbf{Hs}^* = \mathbf{b}$ for $\mathbf{s}^*$. This is solved for $\mathbf{s}^* = \mathbf{H}^{-1}\mathbf{b}$.