CSU44061 Machine Learning - Week 6
Samuel Petit - 17333946
Dataset # id:8–80–16

Question i

Code for all questions provided in the appendix.

Part a

Let's start with going through what we need to train these models, first we need to generate our dataset. Using the linspace method over the range -1 to 1 with 3 number of data points, we generate our X points. I then hardcoded the Y values. In order to work with the single feature, I reshape it thanks to panda's methods and we are ready to use this data with sklearn models.

```
m = 3
Xtrain = np.linspace(-1.0, 1.0, num=m)
ytrain = np.sign(np.array([0, 1, 0]))
Xtrain = Xtrain.reshape(-1, 1)
```

The gaussian kernel used is the same as previously used in lectures so I won't spend too much time explaining it:

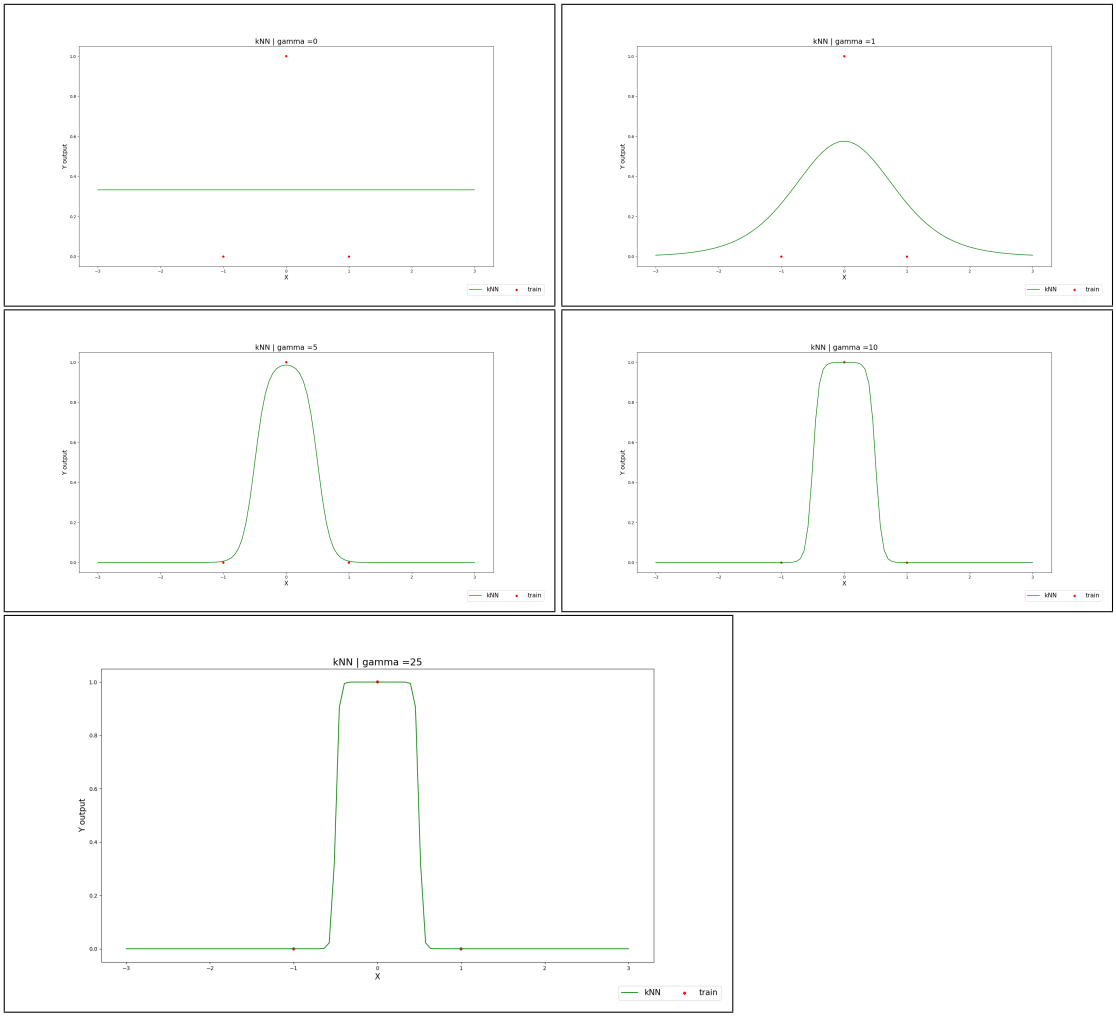
```
def gaussian_kernel(distances):
    weights = np.exp(-V * (distances ** 2))
    return weights/np.sum(weights)
```

Note the use of V in that method. That is a global variable set within the code such as to be able to change the V value of that method easily.

Finally, the last piece we need is the testing set of X values which, as stated in the question is over the range [-3, 3]. I also use linspace to generate it:

```
Xtest = np.linspace(-3.0, 3.0, num=100).reshape(-1, 1)
```

Now, using all these and a range of γ values, we can train a kNN model which yeild the following predictions:



Part b

A kNN model makes predictions based on its closest neighbors, in a kNN regression it is the average of all k neighbors. Here we use $k = 3$ which is the entire dataset. However we also use a Gaussian function here which essentially means our feature is mapped through the gaussian function which gives a form of distance estimation.

Given that 2 out of our 3 points are 0, we understand that value of $\gamma = 0$ makes the gaussian function not have an effect and thus our predictions take the form of a constant line that is closer to 0 than 1 on the y axis (the average of all $k = 3$ neighbors).

We notice that the more we increase γ, the more our predictions tend to 1 for x values in the range [-0.5, 0.5] (roughly). We can clearly see the impact of

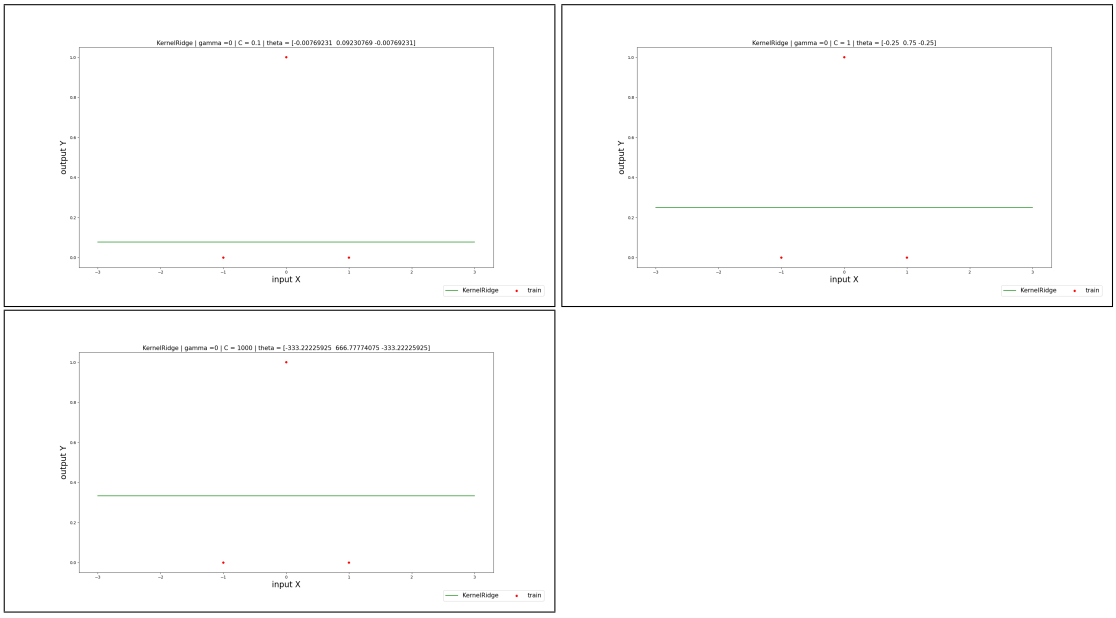
the gaussian function there, going from some form of a smooth quadratic shape to a more binary prediction model for high values of γ, where it is predicting 0 most of the time, and 1 when x gets close to 0.

We've seen in class that as γ grows, our value for $K(x^i, x)$ decreases much faster as the distance between x^i and x grows. That is clearly noticeable as the plot for $\gamma = 25$ decreases much faster than say $\gamma = 1$ where we can see increases and decreases over most of the range of x values.

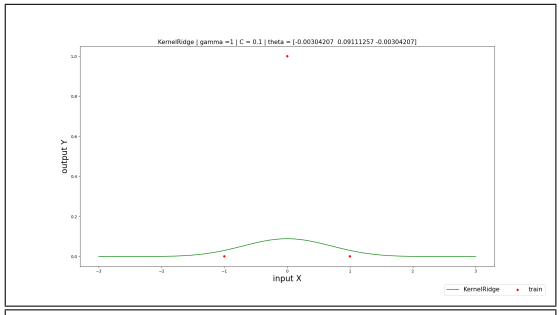
Part c

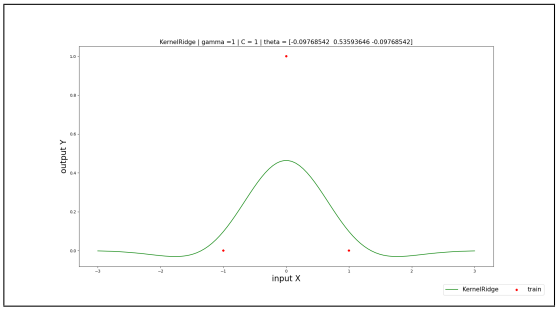
I won't go over explaining how the data was formatted and prepared again as that was covered in part a. Iterating over the provided range of $C = [0.1, 1, 1000]$ and $\gamma = [0, 1, 5, 10, 25]$, we obtain the following graphs (grouped by their γ value). Note that the dual coef parameter is included within the title of each graph.

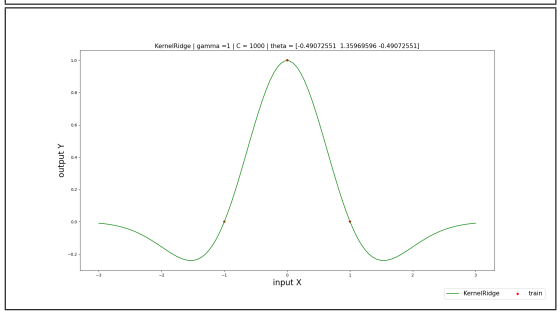
$\gamma = 0$



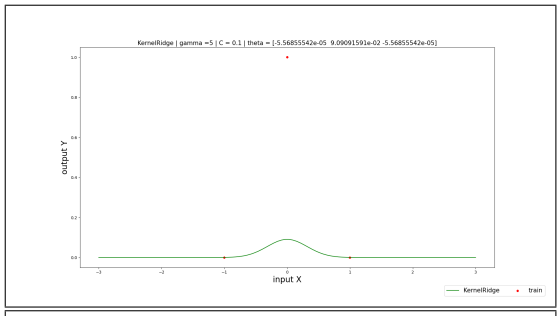
$\gamma = 1$

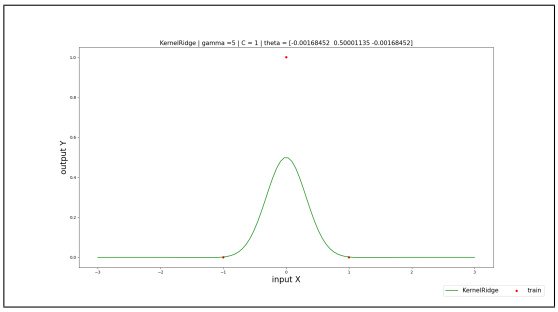


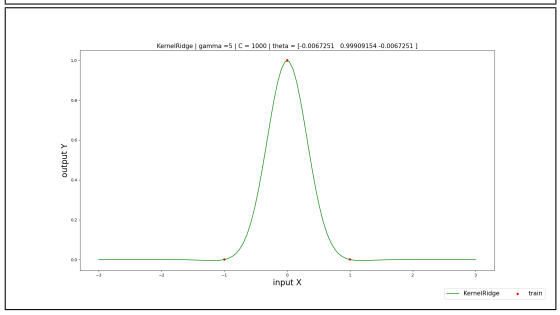




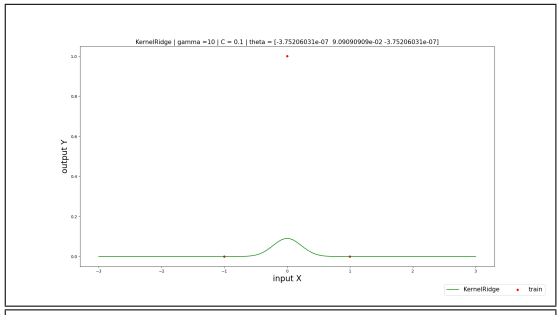
$\gamma = 5$

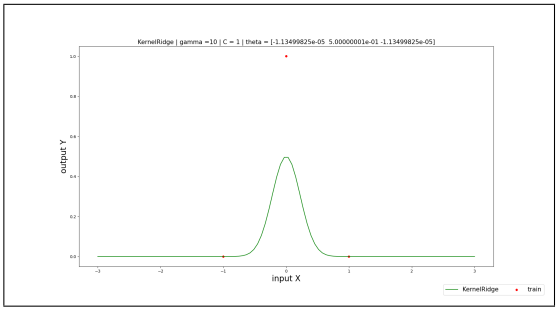


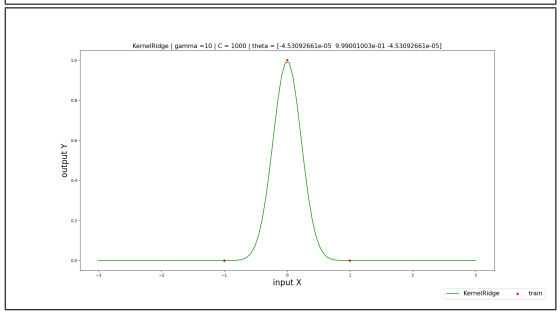




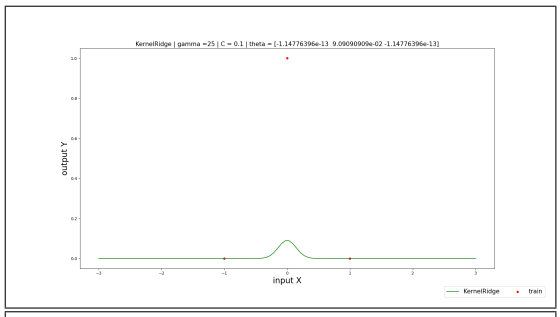
$\gamma = 10$

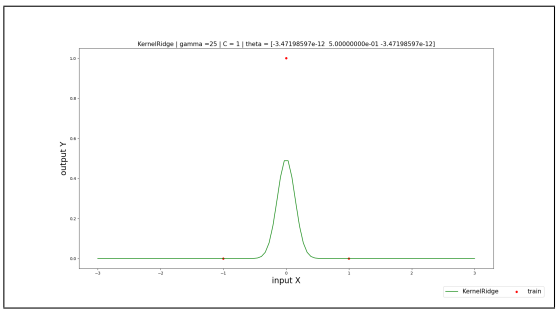

KernelRidge | gamma =10 | C = 0.1 | theta = [-3.75206031e-07 9.09090909e-02 -3.75206031e-07]
output Y
input X
KernelRidge
train

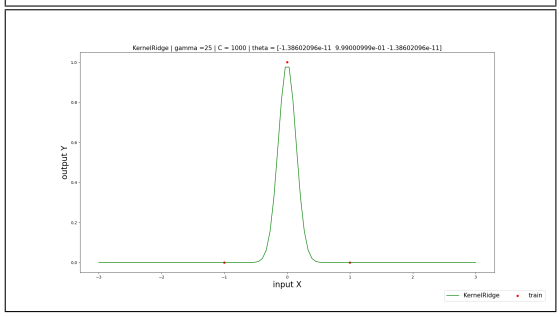

KernelRidge | gamma =10 | C = 1 | theta = [-1.13499825e-05 5.00000001e-01 -1.13499825e-05]
output Y
input X
KernelRidge
train


KernelRidge | gamma =10 | C = 1000 | theta = [-4.53092661e-05 9.99001003e-01 -4.53092661e-05]
output Y
input X
KernelRidge
train

$\gamma = 25$


KernelRidge | gamma =25 | C = 0.1 | theta = [-1.14776396e-13 9.09090909e-02 -1.14776396e-13]
output Y
input X
KernelRidge
train


KernelRidge | gamma =25 | C = 1 | theta = [-3.47198597e-12 5.00000000e-01 -3.47198597e-12]
output Y
input X
KernelRidge
train


KernelRidge | gamma =25 | C = 1000 | theta = [-1.38602096e-11 9.99000999e-01 -1.38602096e-11]
output Y
input X
KernelRidge
train

Part d

A kernel ridge regression uses a ridge regression model (that is a model where the loss function is least squares). As mentioned in the previous question it also uses an L2 penalty, that is its regularisation term is of $\alpha = \frac{1}{2C}$. Now, the particularity of a Kernel ridge regression is that it using the "kernel trick", that is using a linear model to solve a non-linear problem, it essentially maps our features into new, linearly seperable data points. In this situation we are using a gaussian kernel, that is the KernelRidge's kernel parameter in sklearn is set to 'rbf'.

Let's now look into how predictions change with respect to γ and C changing. Starting with varying values for γ, we notice that the same observation as we've pointed out for the kNN model applies: as γ grows, our value for $K(x^i, x)$ decreases much faster as the distance between x^i and x grows, thus, once again we can see that small values of γ see small fluctuations over a large part of the range of x values, while large values such as $\gamma = 25$ go from a low predictiong to a high predictions very fast, over a very small distance (or range) of x.

In terms of varying our C value, we can take any value of γ as a reference given we notice the same trend, a very low value for C such as 0.1 will enable for very small fluctuations of predictions, and that fluctuation will increase as C does too. This makes sense as instead of optimising for simply the loss function, we try to minimize the loss + the penalty term. Thus when C is very small, α is very large, thus making the penalty term very big we obtain very little variation. When C is very large, α is very small thus that penalty term is not as impactful and our model will have more variations based on the data it is trained with.

Looking at θ from the title of these graphs, we notice a few trends. Throughout, we see that as C becomes bigger, so do the coefficients, taking $\gamma = 5$ as an example we have: $\theta = -5.5 * 10^{-5}$, $9 * 10^{-2}$ and $-5.56 * 10^{-5}$ when $C = 0.1$, and when $C = 1000 : \theta = -0.0067$, 1 and -0.0067 (roughly). And the same is noticeable for the other values of γ.

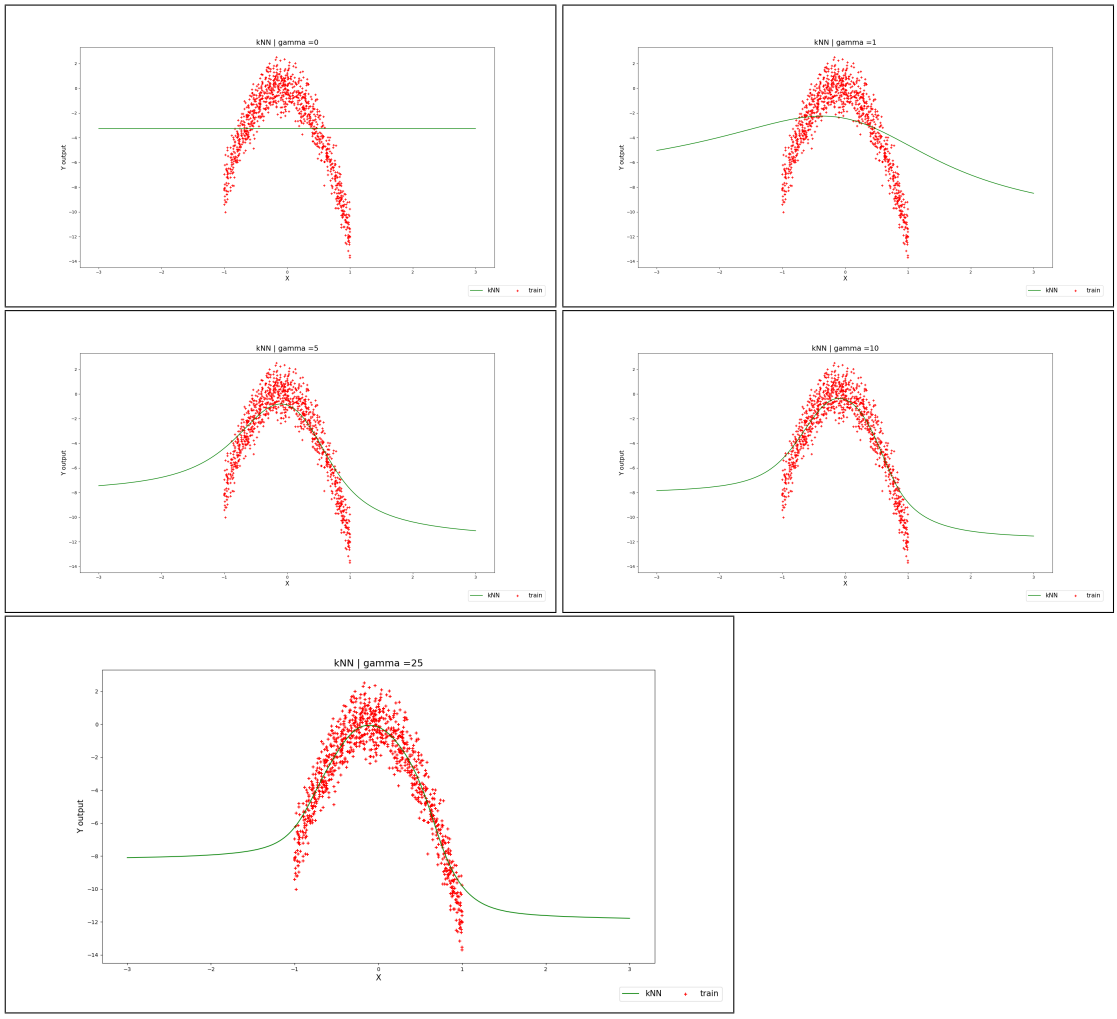
Finally, we also notice that as γ gets biggers, our coefficients become smaller, having for example: $\theta = -333, 666, -333$ when $\gamma = 0$ and $C = 1000$, and when $\gamma = 25$ and $C = 1000 : \theta = 1.38 * 10^{-11}, 0.99, -1.38 * 10^{11}$.

1 Question ii

Part a

Let's first clarify the range I have used for testing my models; I decided to follow the trend as suggested by the exercise and the went with the range $[-3 * abs(min), 3 * max]$. In practise though, the minimum was a negative value thus I only have to use $[3 * min, 3 * max]$.

The method is the same as explained in question i thus I will not go over the explaining here. Using the provided dataset, $k = \#pointsindataset$ and a range for $\gamma = [0, 1, 5, 10, 25]$, we obtain the following predictions after training a kNN model:



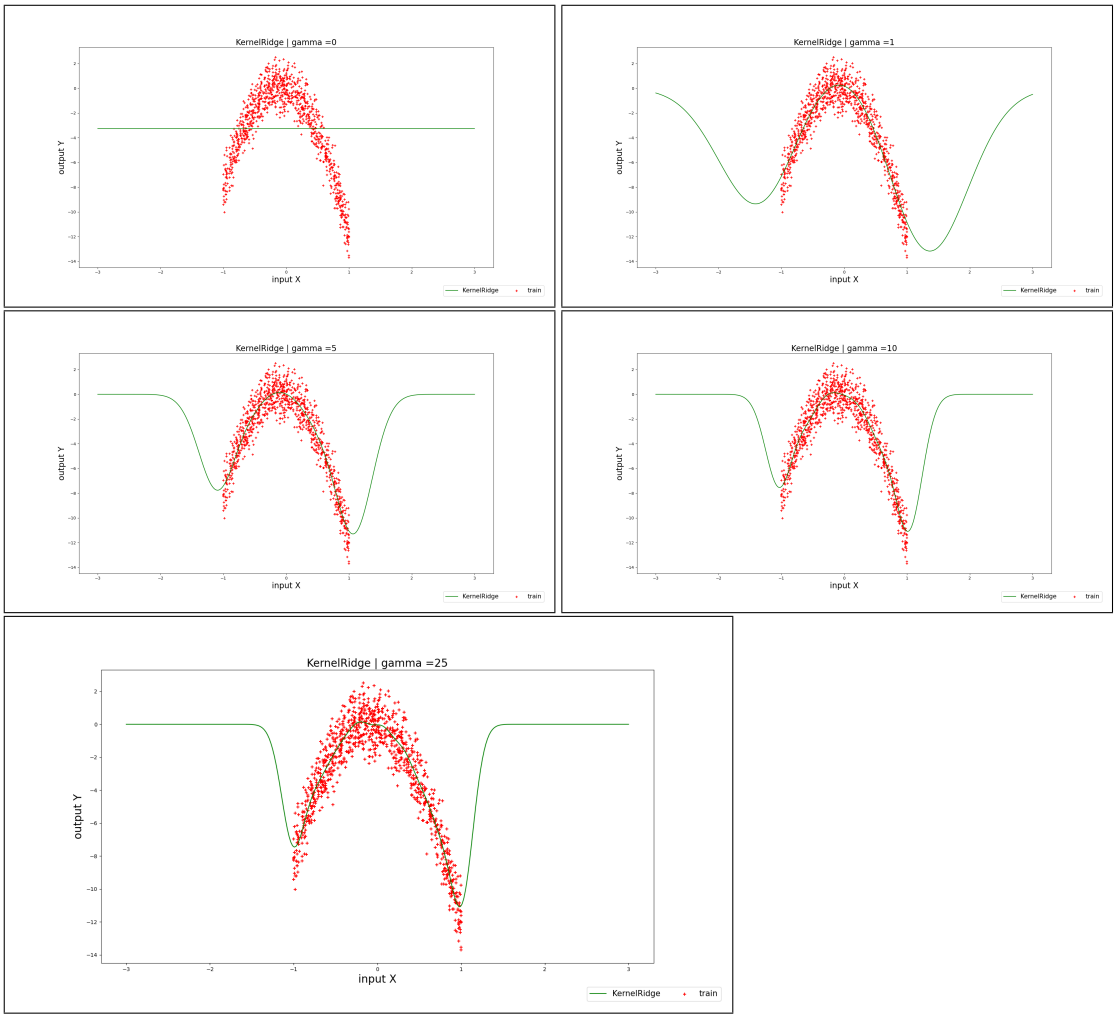
As mentioned twice before, we notice the same behavior as γ grows, the prediction plot adapts much faster to the data it is trained with, going from starting to follow a trend with $\gamma = 1$ to fitting quite well the training data when $\gamma = 25$.

In terms of how the models generalise outside of the trained data range, we notice that most models are quite flat looking, with the one for $\gamma = 25$ matching most of the taining data's curve but still flattening out a bit later, which, given the look of the training data is likely not what would happen.n fact it seems

slightly wrong to try expand the range for this training data as it clearly seems to be set within a set interval of x values.

Part b

Similarly to our previous question I have explained most of how this works already so I will not go into details again, using the provided dataset as the training data and making the model predict values over the range we have previously set of $[3 * min, 3 * max]$. We can train KernelRidge models with varying values of γ. This question does not mention the loss parameter C at all thus I will use sklearn's default value, which is 1. We obtain the following plots:



Part c

Appendix