**Machine Learning**

# Assignment 6

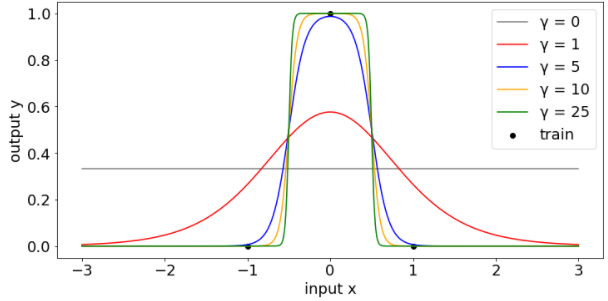
**Q(i)** For the first part of this assignment, I created a csv data called dummy.csv that has the values: (-1, 0), (0, 1), (1, 0) and imported this to my program.

1. *(Full code in Figure 1)*

For this dataset, I used the KNeighborsRegressor function from sklearn’s library to create kNN predictions.

****

Using a k of 3 and a range of values 0, 1, 5, 10 and 25 as parameter γ of the Gaussian kernel, the plotted predictions were as follows:

****

1. As γ is varied, the predictions also change as γ determines what is close and far and the distance the training set reaches. When γ is 25, the predictions are close to 1 for input values between -0.5 and 0.5 because as the value of the γ parameter increases, it suggests that it is close and have a higher chance of distant data being similar.
2. The KernelRidge function from sklearn’s library was used to train a kernalised ridge regression model on this data. *(Full Code in Figure 2)*

To get the kernel ridge parameters, the dual\_coef\_ function was used.

****

The predictions were plotted as follows for C values 0.1, 1, 10 and 1000.

|  |  |
| --- | --- |
|  |  |
|  |  |

The kernel ridge parameters of the following plots were as follows:

|  |  |
| --- | --- |
|  |  |
|  |  |

1. As C and γ is increased, the predictions plotted in part (c) get closer to the training data (0, 1). From the plot, when C has a large value such as 1000, the predictions contain all the three training sets in the data: (-1, 0), (0, 1), (1, 0).

Parameters theta change with the input so they affect the predictions when the KernelRidge() function is applied.

In the kNN predictions, the predictions seem closer to each other as γ has values 5, 10 and 25 and it looks as if the predictions have already reached the training set (0, 1) when γ is 10.

As for the kernalised ridge regression predictions, the predictions look evenly spaced out when γ is 5, 10 and 25.

When γ is 1 in the kernalised ridge regression, the predictions of output y go below 0 in comparison to the kNN predictions where the predictions remain above 0.

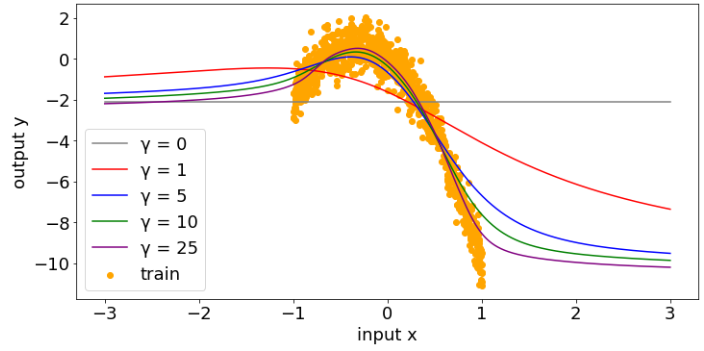
Another comparison is that the plotted predictions look more curved than the predictions using kNN.

**Q(ii)**

1. In this program, I loaded the *week6.csv* data that was provided in the assignment sheet and predictions from a grid of feature values from -3 to 3 were generated. There are 998 points in this dataset, so I used this figure as the value for k in the KNeighborsRegressor function. *(Full code in Figure 3)*



Again, the range of values I used for γ are 0, 1, 5, 10 and 25. Using these values, the predictions were plotted as follows:

****

From this graph, as γ increases the predictions change as the γ parameter controls how quickly the gaussian kernel decreases as the distance between the input and training points grows.

From the plot above, the training set show a negative curve. The predictions from γ parameter 1-25 also show a descending curve but not as intense downward as the training data. In fact, it looks like the predictions stopped decreasing from input x of 1 and looks as if the plots will show stable predictions beyond input 3 of as more x inputs are introduced.

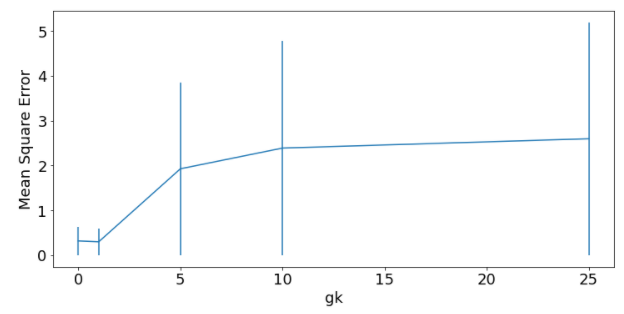
1. Similarly, from part (i)(c), the sklearn KernelRidge function was used to train a kernalised ridge regression model on the *week6.csv* data that was given in the assignment sheet.*(Full code in Figure 4)*

When C is 0.1, 1, 10, and 1000, the predictions were as follows:

|  |  |
| --- | --- |
|  |  |
|  |  |

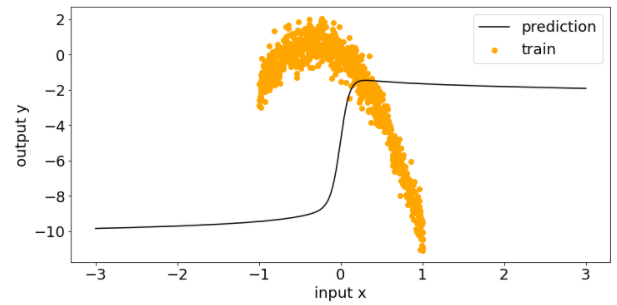
From the predictions in the plots above, we can see that as γ is varied, the predictions seem to widen, and it starts to show more negative curves from input x of -1 onwards which is evident when C is 1000 and when γ has a range from 5-25. There is not a lot of change in predictions when input x is greater than 1 onwards from the 4 plots above but this is not the case when input x has negative values.

1. Using cross validation to choose a reasonable hyperparameter value for a kNN model, the error bar plot of this model is as follows:

****

From the plot above, I have chosen 5 as the value for hyperparameter γ for the kNN model. I chose this because the mean square error (MSE) from the error bar plot is not too high as the larger MSE is the larger the error.

Using 5 as the hyperparameter value γ, the predictions for this kNN model were generated and plotted below:



As for the kernalised ridge regression model, I also generated error bar plots to choose a reasonable value for hypeparameter γ and alpha.

Using cross-validation to choose γ hyperparameter for the kernalised ridge regression model, the error bar plots of each were generated as follows:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

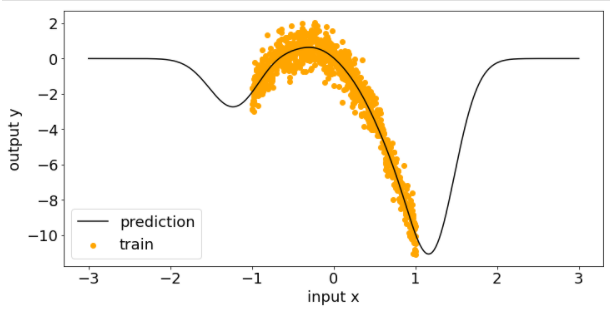
To choose alpha, the error bar plots were generated:

|  |  |
| --- | --- |
|  |  |
|  |  |

However, from the error bar plots above to choose alpha, the plots show similar error bars for 0.1, 1, 10 and 1000 as values of C.

Therefore, the resulting hyperparameter I chose was 5 for γ as it has the least mse and 10 for alpha.

When C = 10 and γ = 1, the predictions for a kernalised ridge regression model were:



Comparing the predictions from the kNN model and the kernalised ridge regression model using cross-validation in both, the predictions look more accurate in the kernalised ridge regression model than the predictions from the kNN model. The prediction line from the kNN model only shows a positive curve while the prediction line from the kernalised ridge regression model shows both negative and positive predictions as feature values are extended to -3 to 3. It looks like the predictions are based off and continues from the actual training data which is more reliable. Also, the MSE in the kNN model is much higher than the kernalised ridge regression model (5 vs 0.06). The kernalised ridge regression MSE is much lesser and shows smaller error than kNN. Therefore, the model that generated the better predictions for me was the kernalised ridge regression model.

## APPENDIX

Figure – Q(i)(a): as6q1a.py



Figure – Q(i)(c): as6q1c.py







Figure – Q(ii)(a)



Figure – Q(ii)(b)





Figure – Q(ii)(c)(i)



Figure – Q(ii)(c)(ii) – as6q2c1.py





as6q2c2.py





Figure – Q(ii)(c)(ii) - as6q2c3.py

