Russel Stuart Daries
UCABRSD

STUDENT

FIRST LETTER OF SURNAME
D

PERSONAL TUTOR

# Coursework Coversheet

This document is the assessed coursework coversheet for all Computer Science modules conducted at UCL for this academic year. Please do the following when submitting coursework:

- Staple a completed and signed copy of this form to every piece of assessed coursework you submit for modules in the Department of Computer Science.
- avoid the use of document containers such as cardboard or plastic covers, document wallets, ring binders or folders (unless otherwise stated by the lecturer concerned).

We do not wish to discourage students from discussing their work with fellow students and collaborating in solving problems. However you should avoid allowing the collaborative phase to approach too close to a final solution which might make it impossible for you to make your own distinctive intellectual contribution. The key point is that you must not present the results of another person's work "as though they were your own". http://www.ucl.ac.uk/current-students/study/plagiarism/

UCL has now signed up to use a sophisticated detection system (JISC Turn-It-In) to scan work for evidence of plagiarism, and the Department intends to use this for assessed coursework. This system gives access to billions of sources worldwide, including websites and journals, as well as work previously submitted to the Department, UCL and other universities

The collection point for "returned marked" coursework will be in a public area. If you wish to have your coursework returned privately. You MUST collect your coursework as soon as you are informed.

## Submission Details

Please ensure that the details you give are accurate and completed to the best of your knowledge.

COURSEWORK TITLE: Coursework 1

LECTURER: Dr. Mark Herbster

MODULE CODE: COMPGI01

MODULE NAME: Supervised Learning

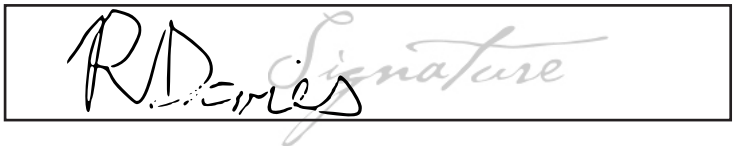
DEADLINE: 16 / 01 / 2017

## Declaration

I have read and understood the UCL and Departmental statements and guidelines concerning plagiarism.
I declare that:

1. This submission is entirely my own unaided work.
2. Wherever published, unpublished, printed, electronic or other information sources have been used as a contribution or component of this work, these are explicitly, clearly and individually acknowledged by appropriate use of quotation marks, citations, references and statements in the text.
3. In preparing this coursework, I discussed the general approach to take with the person(s) named below, however the content of the submission is my own work alone:

Person(s) with whom I have discussed this coursework: Nitish Mutha




## For Departmental Office Use

DATE AND TIME RECEIVED

INITIALS OR STAMP

# COMPGI01: Supervised Learning Coursework 1

Due on Monday, January 16, 2017

*Dr. Mark Herbster*

**Russel Daries: UCABRSD**
**Nitish Mutha: UCABMUT**

January 16, 2017

# Contents

**Exercise 1** **7**
(a) . . . 7
(b) . . . 7
(c) . . . 7
(d) . . . 8

**Exercise 2** **9**
(a) . . . 9
(b) . . . 9
(c) . . . 10

**Exercise 3** **11**
(1) . . . 11
(2) . . . 12

**Exercise 4** **13**
(a) . . . 13
(b) . . . 13
(c) . . . 14

**Exercise 5** **15**
(a) . . . 15
(b) . . . 15
(c) . . . 16
(d) . . . 16

**Exercise 6** **17**
(a) . . . 17
(b) . . . 17

**Exercise 7** **18**

**Exercise 8** **20**

**Exercise 9** **21**
(a) . . . 21
(b) . . . 21
(c) . . . 23

**Exercise 10** **24**
(a) . . . 24
(b) . . . 24
(c) . . . 24
(1) . . . 25
(2) . . . 25
(d) . . . 26

**Question 1** **27**

**Question 2** **28**
(a) . . . 28
(b) . . . 29

**Question 3** **33**
(a) . . . 33
(b) . . . 33

**Question 4** **34**
(a) . . . 34
(b) . . . 35
(c) . . . 35

**Appendices** **37**
Exercise 1: Code . . . 38
ex1.m . . . 38
Exercise 2: Code . . . 41
ex2.m . . . 41
Exercise 4: Code . . . 44
ex4.m . . . 44
Exercise 5: Code . . . 48
ex5.m . . . 48
Exercise 6: Code . . . 55
ex6.m . . . 55
Exercise 7: Code . . . 59
ex7.m . . . 59
ex7_4.m . . . 62
ex7_5.m . . . 63
ex7_6.m . . . 64
Exercise 9: Code . . . 65
Ex9_a.m . . . 65
Ex9_b.m . . . 66
Ex9_c.m . . . 68
Exercise 10: Code . . . 70
Ex10_RR.m . . . 70
dualcost.m . . . 72
kridgereg.m . . . 72
Question 4 (Part II): Code . . . 73
question4_1.m (least squares) . . . 73
question4_2.m (perceptron) . . . 74
question4_3.m (winnow) . . . 75
question4_4.m (1NN) . . . 77
Library Functions . . . 79
calculateLSR.m . . . 79
calculateLSRex2.m . . . 79
calculateLSRex4.m . . . 80
calculateLSRex5.m . . . 80
calculateLSRex6.m . . . 81
calculateLSRTest.m . . . 81
meanSquareError.m . . . 82

standard_deviation.m . . . . . 82
calK.m . . . . . 82
learnModel.m . . . . . 83

# List of Figures


1 MSE on training data. . . . . 8
2 MSE on test data. . . . . 8
3 Weight vector $\boldsymbol{w}$ for 200 trials . . . . . 8
4 MSE for train data ($\boldsymbol{x}\epsilon\Re^{10}$.). . . . . 10
5 MSE for test data ($\boldsymbol{x}\epsilon\Re^{10}$.). . . . . 10
6 Ridge Regression with 100 training samples. . . . . 13
7 Ridge Regression with 10 training samples. . . . . 14
8 Ridge Regression with average errors for 100 training samples. . . . . 14
9 Ridge Regression with average errors for 10 training samples. . . . . 14
10 MSEs for training(80), validation(20) and test data. . . . . 15
11 MSEs for training(8), validation(2) and test data. . . . . 15
12 MSEs for optimal gamma selection for 200 trials. . . . . 16
13 Ridge Regression with average errors for 100 training samples. . . . . 16
14 Ridge Regression with average errors for 10 training samples. . . . . 16
15 Cross validation with 100 training samples. . . . . 17
16 Cross validation with 10 training samples. . . . . 17
17 Optimal $\gamma$ for 100 training samples with tuning methods. . . . . 18
18 Optimal $\gamma$ for 10 training samples with tuning methods. . . . . 18
19 MSE on test data. . . . . 19
20 MSE on test data. . . . . 19
21 Average MSE for 13 attributes. . . . . 21
22 Box plot for MSE on the training set. . . . . 21
23 Box plot for MSE on the test set. . . . . 21
24 Box plot for MSE on the training set with all attributes. . . . . 23
25 Box plot for MSE on the test set with all attributes. . . . . 23
26 Cross validation error with Kernel Ridge Regression. . . . . 25
27 A separating hyperplane $(\boldsymbol{w}, b)$ for a two dimensional set [1]. . . . . 29
28 Estimated sample complexity of *Perceptron* algorithm. . . . . 34
29 Estimated sample complexity of *Winnow* algorithm. . . . . 34
30 Estimated sample complexity of *Least Squares* algorithm. . . . . 34
31 Estimated sample complexity of *1-Nearest Neighbour* algorithm. . . . . 34

# List of Tables


1 LSR errors for $\boldsymbol{w}^*$ with 100 training samples. . . . . 7
2 LSR errors for $\boldsymbol{w}^*$ with 10 training samples. . . . . 7
3 Average Mean Squared errors for varying training sample size repeated 200 times. . . . . 8
4 Average $\boldsymbol{w}^*$ for 200 trials. . . . . 8
5 LSR errors for $\boldsymbol{w}^*$ with 100 training samples. . . . . 9
6 LSR errors for $\boldsymbol{w}^*$ with 10 training samples. . . . . 9
7 Average Mean Squared errors for varying training sample size repeated 200 times. . . . . 9
8 Comparison of $\gamma$ tuning methods. . . . . 18
9 Comparison of average $\gamma$. . . . . 19
10 Comparison of MSE performances with naive regression. . . . . 21
11 Comparison of attribute MSE performances. . . . . 22
12 Comparison of attribute MSE performances with all attributes. . . . . 23
13 MSE for optimal parameters $\gamma^*$ and $\sigma^*$. . . . . 25
14 Comparison of regression methods. . . . . 26
15 Computational time for algorithms. . . . . 35

# Exercise 1

## (a)

The Least Squares Regression was applied to Exercise 1 in order to minimize the mean of the squared errors on all training samples. This can be expressed as finding the optimal weight vector $\boldsymbol{w}^*$ as shown in the Equation 1 below.

$$\boldsymbol{w}^* = argmin_{\boldsymbol{w}} \frac{1}{l} \sum_{i=1}^{l} (\boldsymbol{x'}_i \boldsymbol{w} - y_i)^2 \tag{1}$$

The relevant $\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{n}$ and $\boldsymbol{w}$ vectors were created as required for Exercise 1 as shown in below.

```
%% Exercise 1(a)
w = randn(1,1);
x = randn(600,1);
n = randn(600,1);
y = x*w + n;
```

## (b)

After all the required data matrices and vectors were created, using the Normal Equation as shown in Equation 2 below, the optimal $\boldsymbol{w}$ could be calculated.

$$\boldsymbol{w}^* = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{y} \tag{2}$$

The optimal $\boldsymbol{w}^*$ and mean squared errors on the train and test sets can be seen in Table 1 below.

| $\boldsymbol{w}^*$ | **Train Error** | **Test Error** |
|---|---|---|
| 0.5413 | 0.9601 | 1.0475 |

Table 1: LSR errors for $\boldsymbol{w}^*$ with 100 training samples.

It can be seen from Table 1 above that the test error was larger than the training error, with a 100 training examples and 500 test examples. The weight vector $\boldsymbol{w}$ did not generalise well.

## (c)

The same experiment as discussed in Excercise 1(b) was repeated with only 10 training samples and 500 test samples taken from standard normal distribution. The resultant errors and $\boldsymbol{w}^*$ can be seen in Table 2 below.

| $\boldsymbol{w}^*$ | **Train Error** | **Test Error** |
|---|---|---|
| 0.7475 | 0.7584 | 1.0700 |

Table 2: LSR errors for $\boldsymbol{w}^*$ with 10 training samples.

It can be seen that the training error in Table 2 was lower, but this can be attributed to only having sampled 10 times.The test error was higher by 0.0225 and the $\boldsymbol{w}^*$ was larger by 0.2061.

Exercise 1 continued on next page...

## (d)

The process followed in parts (a),(b) and (c) were repeated 200 times in order to evaluate the effect of varying the training size. Table 3 shown below demonstrated the average mean square errors for the training and test sets. Even though the training set performed well the test set performed poorly.

| No. Training Samples | Train Error | Test Error |
|---|---|---|
| 10 | 0.9247 | 1.1218 |
| 100 | 0.9921 | 1.0061 |

Table 3: Average Mean Squared errors for varying training sample size repeated 200 times.

Figure 1 and 2 shown below demonstrated the resultant MSE for each of the 200 trials on the training and the test set for the two training set sizes.

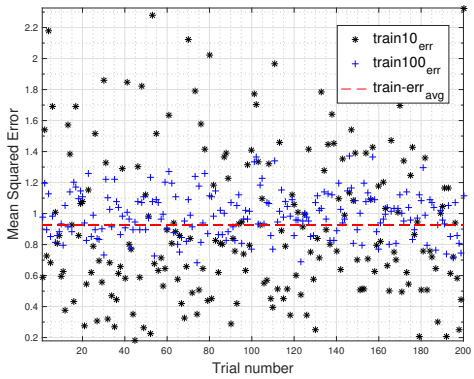



Figure 1: MSE on training data.

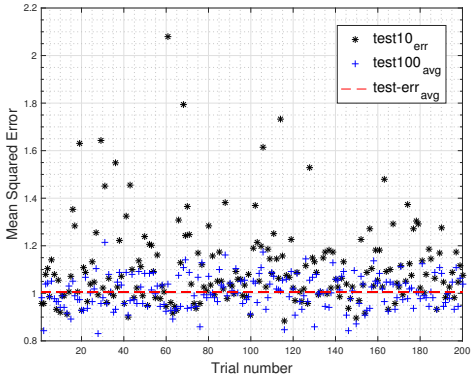



Figure 2: MSE on test data.

The average $\bar{\boldsymbol{w}}^*$ over the 200 experiments can be seen in Table 4 below.

| 10 Training Samples | 100 Training Samples |
|---|---|
| 0.0057 | -0.0222 |

Table 4: Average $\boldsymbol{w}^*$ for 200 trials.

Figure 3 below demonstrated the calculated $\boldsymbol{w}^*$ for each trial.

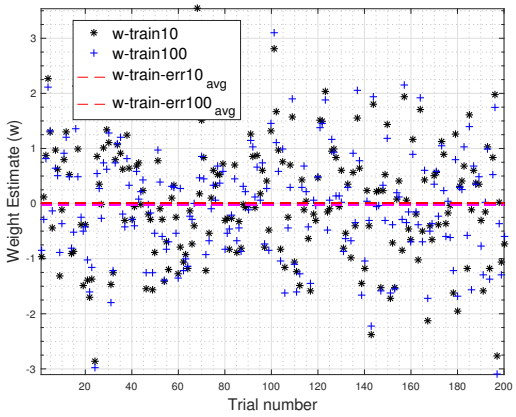



Figure 3: Weight vector $\boldsymbol{w}$ for 200 trials

# Exercise 2

## (a)

It could be seen in Exercise 1, the effect of the training and test corpus sizes was investigated in order to see how they affected the performance of the predictor. We will now extend the dimension of the data to $\boldsymbol{x} \epsilon \Re^{10}$.

The relevant $\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{n}$ and $\boldsymbol{w}$ vectors were created as required for Exercise 2(a) as shown in below.

```
%% Exercise 1A
w = randn(10,1);
x = randn(600,10);
n = randn(600,1);
y = x*w + n;
```

The training and test sets were created and tested using the *calculateLSRex2* function which can be found in Appendix **??**.

## (b)

A similar process as discussed in Exercise 1 (b)-(d) was followed now with a higher dimensional $\boldsymbol{x}$. The results for the experiments can be seen in Tables 5 - 7 shown below.

| **Train Error** | **Test Error** |
|---|---|
| 0.9367 | 1.1269 |

Table 5: LSR errors for $\boldsymbol{w}^*$ with 100 training samples.

| **Train Error** | **Test Error** |
|---|---|
| 0.00 | 129.0357 |

Table 6: LSR errors for $\boldsymbol{w}^*$ with 10 training samples.

| **Training Samples** | **Train Error** | **Test Error** |
|---|---|---|
| 10 | 0.0 | 2282.20 |
| 100 | 0.9170 | 1.1053 |

Table 7: Average Mean Squared errors for varying training sample size repeated 200 times.

The MSEs for each experiment can be seen in Figure 4 and 5 on the following page.

Exercise 2 [(b)] continued on next page...

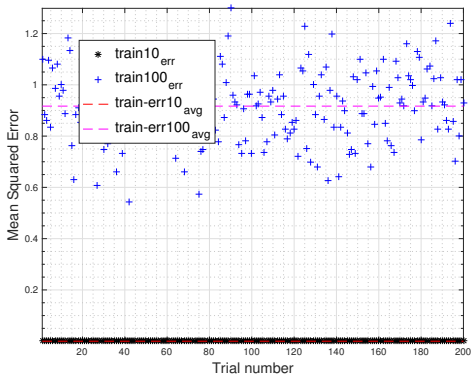



Figure 4: MSE for train data ($\boldsymbol{x}\epsilon\Re^{10}$.).

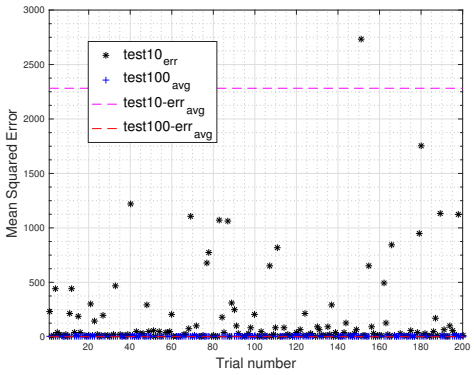



Figure 5: MSE for test data ($\boldsymbol{x}\epsilon\Re^{10}$.).

The code utilized to perform these calculations can be seen in the Appendix.

## (c)

It can be seen when comparing Figures 4 and 5 that the problem of overfitting is present. In the case of 10 training examples, the MSE was small 0.00 in comparison to MSE with 100 training examples with MSE 0.9170 . The resultant weight vector $\boldsymbol{w}_{10}$ was overfitted, so when it was tested on the 500 test examples the resultant average MSE was 2282.20 as compared to the $\boldsymbol{w}_{100}$ which generalised far better as the MSE of 1.1053 was much lower.

Thus is can seen, the effect of the training corpus size of 10 examples is suseptible to fitting noise contained in the signal of interest resulting in low training errors but large test errors. did not result in a large test error. This is clearly evident when comparing the test MSE's shown in Table 7. Whereas, with a substantial amount of training examples, which in this case was 100, $\boldsymbol{w}^*$ generalised far better when compared to 10 training examples even at higher dimensions of $\boldsymbol{x}$. By increasing the training examples to 100, however did not lead to a good indication of the performance of the regression function on the test set.

This is as a result of the regression function fitting the noise present in the signal of interest (overfitting).

# Exercise 3

## (1)

The optimization problem for Regularized Regression was solved in the following manner. Using Equation 1, Equation 3 could be proved.

$$\boldsymbol{w}^* = (\boldsymbol{X'X} + \gamma l \boldsymbol{I})^{-1} \boldsymbol{X'y} \tag{3}$$

The cost function $R(\boldsymbol{w})$ can be defined using Equation 1.

$$R(\boldsymbol{w}) = \frac{1}{l} \sum_{i=1}^{l} (\boldsymbol{x'}_i \boldsymbol{w} - y_i)^2 + \gamma \boldsymbol{w'w} \tag{4}$$

Equation 4 is minimised when $\nabla_{\boldsymbol{w}} R(\boldsymbol{w}) = 0$. Therefore, the following derivation was performed.

$$\begin{aligned}
\nabla_{\boldsymbol{w}} R(\boldsymbol{w}) &= \nabla_{\boldsymbol{w}} \Big( \frac{1}{l} \sum_{i=1}^{l} (\boldsymbol{x'}_i \boldsymbol{w} - y_i)^2 + \gamma \boldsymbol{w'w} \Big) \\
&= \frac{1}{l} \sum_{i=1}^{l} \nabla_{\boldsymbol{w}} \Big( (\boldsymbol{x'}_i \boldsymbol{w} - y_i)^2 \Big) + \nabla_{\boldsymbol{w}} \Big( \gamma \boldsymbol{w'w} \Big) \\
&= \frac{1}{l} \nabla_{\boldsymbol{w}} \Big( \boldsymbol{w'X'Xw} - 2\boldsymbol{y'Xw} + \boldsymbol{y'y} \Big) + \nabla_{\boldsymbol{w}} \Big( \gamma \boldsymbol{w'w} \Big) \\
\nabla_{\boldsymbol{w}} R(\boldsymbol{w}) &= \frac{1}{l} \Big( 2\boldsymbol{X'Xw} - 2\boldsymbol{X'y} \Big) + \Big( 2\gamma \boldsymbol{w} \Big)
\end{aligned}$$

Then we set the derivative of the cost function to zero as shown below and solve the equation to make $\boldsymbol{w}$ the subject of the formula.

$$\begin{aligned}
\nabla_{\boldsymbol{w}} R(\boldsymbol{w}^*) &= 0 \\
\frac{1}{l} \Big( 2\boldsymbol{X'Xw}^* - 2\boldsymbol{X'y} \Big) + \Big( 2\gamma \boldsymbol{w}^* \Big) &= 0 \\
\Big( \boldsymbol{X'Xw}^* - \boldsymbol{X'y} \Big) + \Big( \gamma l \boldsymbol{w}^* \Big) &= 0 \\
\boldsymbol{X'Xw}^* + \gamma l \boldsymbol{w}^* &= \boldsymbol{X'y} \\
\Big( \boldsymbol{X'X} + \gamma l \boldsymbol{I} \Big) \boldsymbol{w}^* &= \boldsymbol{X'y} \\
\boldsymbol{w}^* &= \Big( \boldsymbol{X'X} + \gamma l \boldsymbol{I} \Big)^{-1} \boldsymbol{X'y}
\end{aligned}$$

Therefore, as shown in the working above the optimization problem yields the following equation.

$$\boldsymbol{w}^* = \Big( \boldsymbol{X'X} + \gamma l \boldsymbol{I} \Big)^{-1} \boldsymbol{X'y} \tag{5}$$

Exercise 3 continued on next page...

## (2)

The proof demonstrating the fact that the $\boldsymbol{X}^{'}\boldsymbol{X} + \gamma l\boldsymbol{I}$ is positive definite matrix can be seen below.

The expression $\boldsymbol{X}^{'}\boldsymbol{X} + \gamma l\boldsymbol{I}$ can be broken into two seperate terms. For simplicity we will deal with the second term now.

$$\gamma l\boldsymbol{I}$$

Let matrix $\boldsymbol{S}$ be defined as positive semi-definite if $\boldsymbol{x}^T\boldsymbol{S}\boldsymbol{x} \geq 0$ for all vectors $\boldsymbol{x} \neq 0$. Therefore, using this definition and the fact that $\boldsymbol{S}$ is an identity matrix $\boldsymbol{S} = \boldsymbol{I}$, it is easy to prove that $\boldsymbol{I}$ is positive definite for a $3 \times 3$ matrix that can be extended to a $n \times n$ matrix.

$$\boldsymbol{x}^T\boldsymbol{S}\boldsymbol{x} = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = a^2 + b^2 + c^2 > 0 \tag{6}$$

Assuming the following inequalities hold true namely $\gamma > 0$ and $l > 0$. It is possible to state that the second term $\gamma l\boldsymbol{I}$ is positive definite.

The first term $\boldsymbol{X}^{'}\boldsymbol{X}$ can be proved to be positive semi-definite as follows.

$$\boldsymbol{z}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{z} > 0$$

$$\boldsymbol{X}\boldsymbol{z}^T\boldsymbol{X}\boldsymbol{z} > 0$$

$$||\boldsymbol{X}\boldsymbol{z}||_2{}^2 > 0$$

Therefore, combing the two results obtained above the following statement can be made:

$$\boldsymbol{X}^{'}\boldsymbol{X} + \gamma l\boldsymbol{I} > 0$$

# Exercise 4

## (a)

The weakness of normal regression was highlighted through the results presented in Exercises 1 and 2. However, through regularisation the freedom of the classifier is restricted such that the regression function will be less susceptible to noise and able to generalise better.

This is achieved by including a complexity term in the cost function. This is an adaptation of Equation 1 mentioned earlier. The new equation describing the optimal weight vector $\boldsymbol{w}^*$ can be seen in Equation 7 shown below.

$$\boldsymbol{w}^* = argmin_{\boldsymbol{w}} \frac{1}{l} \sum_{i=1}^{l} (\boldsymbol{x'}_i \boldsymbol{w} - y_i)^2 + \gamma \boldsymbol{w'} \boldsymbol{w} \tag{7}$$

The regularisation parameter $\gamma$ was varied over a range of values in order to investigate its effect on the performance of the regression function. Figure 6 shown below demonstrate training and test set errors as a function of $\gamma$.

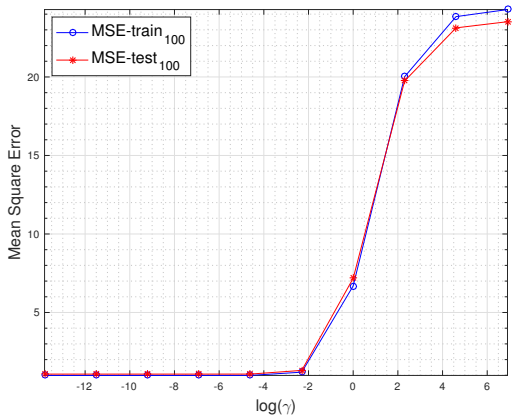



Figure 6: Ridge Regression with 100 training samples.

It can be seen from Figure 6 that the MSE of the test set is slightly higher than that of the training set but the regression function still generalises well with only 100 training samples.

## (b)

The same process was followed but with only 10 training samples and the resultant MSE's for the training and test sets can be seen in Figure 7 shown on the following page.

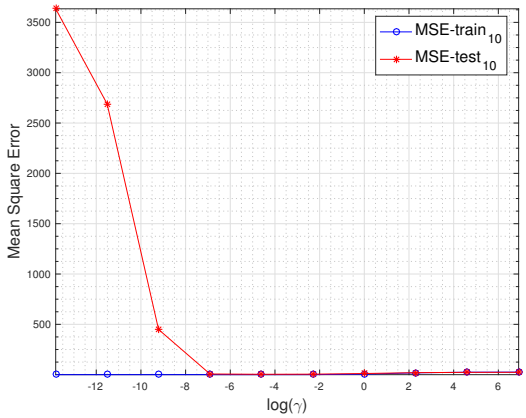



Figure 7: Ridge Regression with 10 training samples.

It can be seen that the phenomena of overfitting is still prevelant as the MSE for the training set is much lower when compared to the test set MSE.

## (c)

Following this result, the process discussed in Exercise 4(a) and 4(b) was repeated for 200 independent trials. The average of the MSEs for each $\gamma$ value for 200 trials was calculated in order to evaluate the performance. Figures 8 and 9 shown below demonstrates the results found.

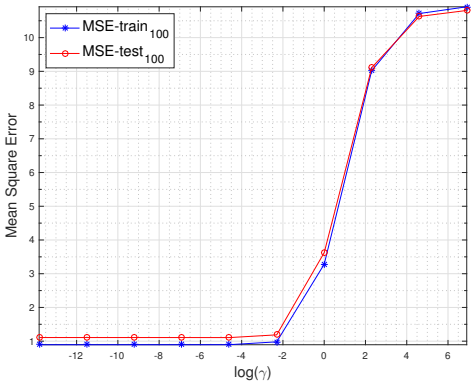



Figure 8: Ridge Regression with average errors for 100 training samples.

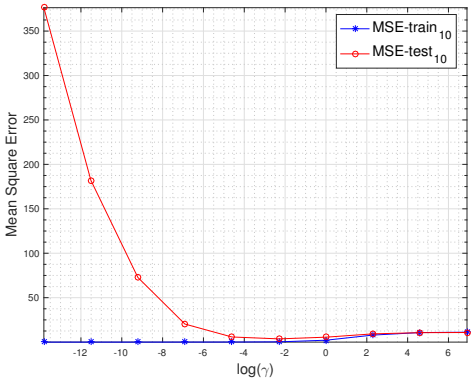



Figure 9: Ridge Regression with average errors for 10 training samples.

It can be seen from Figure 8 and 9 the general behaviour of the ridge regression over 200 trials remains matches the pattern identified in Exercise 4(a) and 4(b). However, just identifying the training set error and selecting the $\gamma$ that minimizes the MSE is not sufficient in predicting the performance of the regression function on the test set. It is a poor indicator of the expected performance.

# Exercise 5

## (a)

As discussed in the previous exercise, only using the training set to select the best $\gamma$ was insufficient. Therefore, as a result we will now use the validation set to select the best $\gamma$ value and perform training on the complete training set (100 samples) which include the validation set (20 samples) for 200 trials and average the results.

The regularization value $\gamma$ was varied over the validation set as shown in Figure 10 shown below.

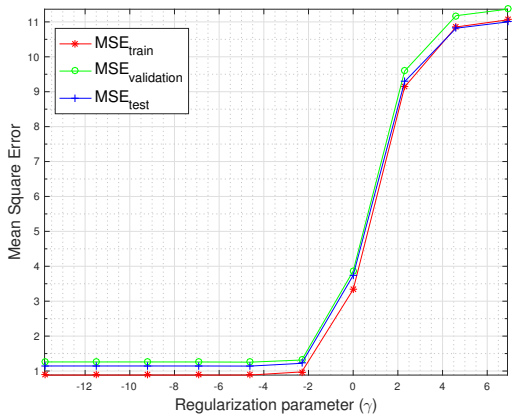



Figure 10: MSEs for training(80), validation(20) and test data.

The optimal regularizaton parameter $\bar{\gamma}^{(100)}$ was found to be 0.0445.

## (b)

A similiar process for selection of $\gamma$ was utilized as mentioned in Exercise 5(a) but with 8 training samples and 2 validation samples for 200 trials. The resultant plot can be seen in Figure 11 below.

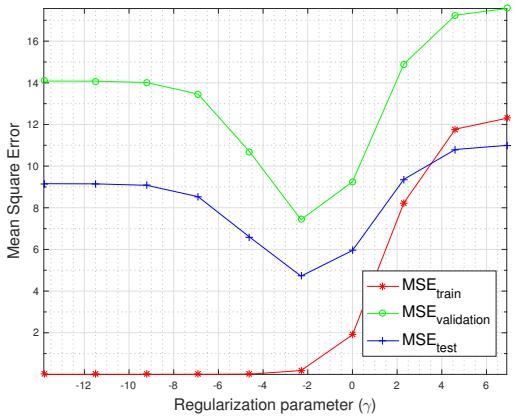



Figure 11: MSEs for training(8), validation(2) and test data.

The optimal regularizaton parameter $\bar{\gamma}^{(10)}$ was found to be 77.0543.

## (c)

It can be seen from Figure 12 below that the MSE on the test set for 10 training samples even with the optimal gamma value is larger than that of 100 training samples.

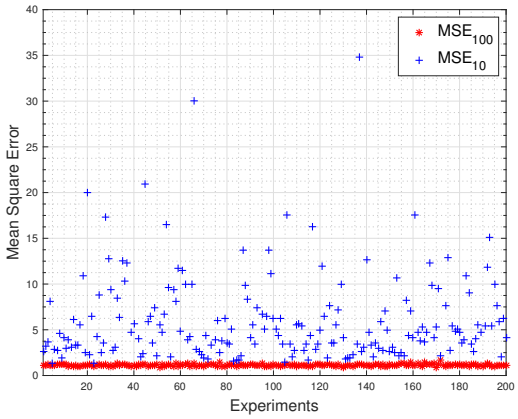



Figure 12: MSEs for optimal gamma selection for 200 trials.

It can also be seen that $\bar{\gamma}^{(10)} > \bar{\gamma}^{(100)}$ is larger because the complexity has larger weighting (Equation 7) in order to penalise the possibility of overfitting is greater for the 10 training samples.

## (d)

A similar trend as that observed in Exercise 4(a) and 4(b) was observed when the dimensions of $\boldsymbol{x}$ was reduced to one dimension. The results demonstrating this trend can be seen below in Figure 13 and 14.

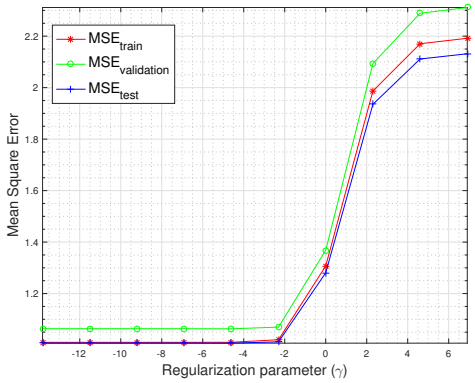



Figure 13: Ridge Regression with average errors for 100 training samples.

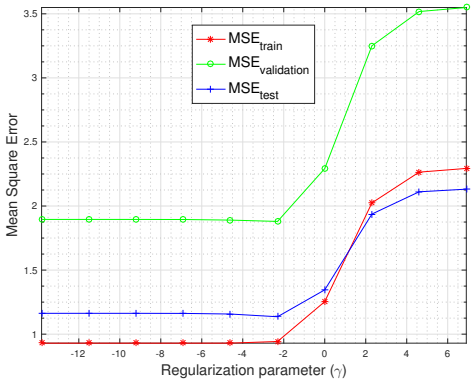



Figure 14: Ridge Regression with average errors for 10 training samples.

The validation MSE was considerably higher as can be seen in Figure 14, where as the test error was considerably less. The curves shown in Figure 13 matched relatively closely.

# Exercise 6

## (a)

The technique of using 5-fold cross-validation was used in order to select the optimal regularization parameter $\gamma$. As shown below the resultant cross-validation scores can be seen in Figure 15 with a training set size of 100.

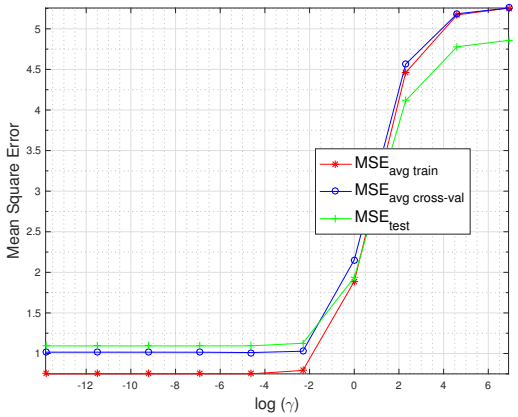



Figure 15: Cross validation with 100 training samples.

It can be seen from Figure 15 above the average cross-validation error score is slightly higher than the actual test set, suggesting the cross-validation method given enough samples, is able to generalise well. This is a similiar behaviour to other selection methods presented in Exercise 5 and 6.

## (b)

A similar process was followed for a training set size of 10. The result can be seen below in Figure 16.

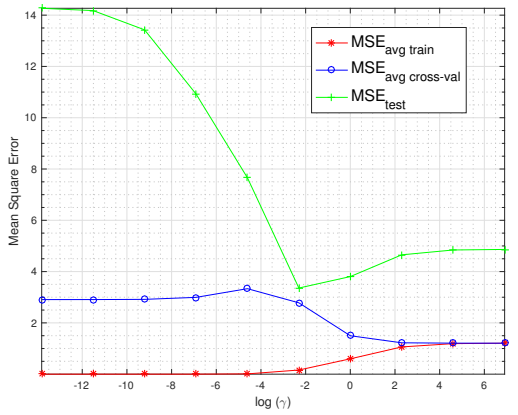



Figure 16: Cross validation with 10 training samples.

Analysing Figure 16 above, we can see that the test set error is larger than both the training and cross-validation scores. The trained regression function is a poor predictor. This can be attributed to the regression approach having insufficient data (10 samples) in order to perform well.

# Exercise 7

Table 8 shown below demonstrates the comparison of $\gamma$ tuning methods as covered in Exercises 4-6.

| Tuning Method | No. of Training Samples | Average Test Error | Standard Deviation |
|---|---|---|---|
| Min Train Err. | 10 | 2061.70 | 15938.00 |
| Min Valid Err. | 10 | 243.3223 | 2382.10 |
| Cross Validate | 10 | 6.1998 | 9.4548 |
| Min Train Err. | 100 | 1.1070 | 0.0864 |
| Min Valid Err. | 100 | 1.1200 | 0.0918 |
| Cross Validate | 100 | 1.1057 | 0.0854 |

Table 8: Comparison of $\gamma$ tuning methods.

It is clearly evident from Table 8 that regardless of the optimal choice of $\gamma$ the MSE on the test set given 10 training samples remains high with a large standard deviation.

The MSE on the test set with 100 training samples in Table 8 is far less with a very low standard deviation approximately 0.085 across all 3 tuning methods. The cross-validation method did achieve the best result on the test set followed by minimizing the training error method. The cross-validation method did achieve the best results due to the data segmentation allowing it to generalise better. The minimization of the training error makes the weights susceptible to over-fitting which is undesirable.

The mean error values for the optimal gamma selected for each of the tuning methods can be seen in Figure 17 and 18 can be seen below. It can be seen from these figures that the cross-validation error usually falls between the bounds of the training and validation errors.

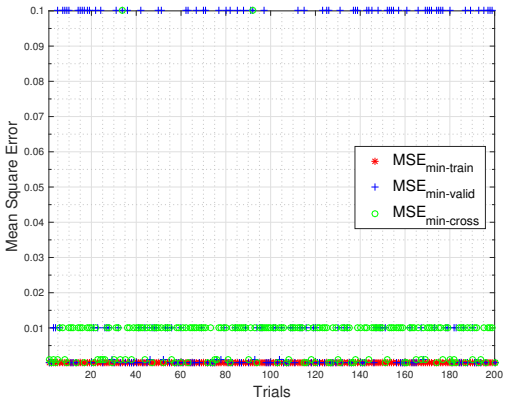



Figure 17: Optimal $\gamma$ for 100 training samples with tuning methods.

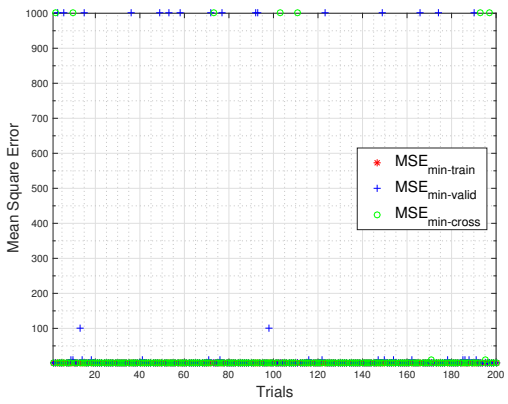



Figure 18: Optimal $\gamma$ for 10 training samples with tuning methods.

Exercise 7 continued on next page...

Table 9 below shows the average $\gamma$ over 200 trials.

| **Tuning Method** | **No. of Training Samples** | **Average $\gamma$** |
|---|---|---|
| Min Train Err. | 10 | $1 \times 10^{-6}$ |
| Min Valid Err. | 10 | 82.11 |
| Cross Validate | 10 | 35.28 |
| Min Train Err. | 100 | $1 \times 10^{-6}$ |
| Min Valid Err. | 100 | 0.0365 |
| Cross Validate | 100 | 0.0076 |

Table 9: Comparison of average $\gamma$.

The relevant scatter plots for each of the three tuning methods with training samples can be seen in Figure 19 (100 samples) and 20 (10 samples) below.

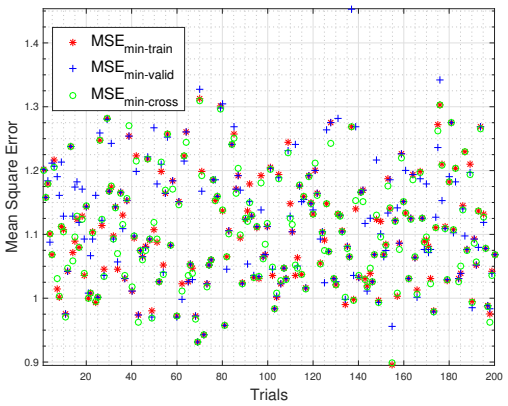



Figure 19: MSE on test data.

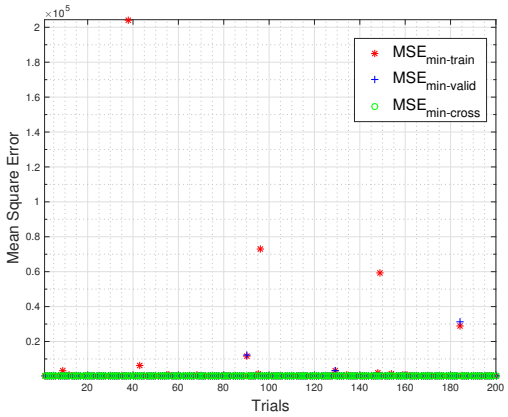



Figure 20: MSE on test data.

It can be seen by comparing the plots of Figure 19 and 20 that the range of values of the MSE vary by a substantial amount in Figure 20 which confirms the results obtained in Table 8 where all of the tuning methods with 10 training samples had large standard deviations indication large variations in the MSE values as shown in Figure 20.

# Exercise 8

The MATLAB data file *boston.mat* was loaded into the workspace for processing as required as can be seen in code *ex9.m* which will be submitted with the assignment.

# Exercise 9

## (a)

Naive Regression was performed using vectors of ones for the training ($\boldsymbol{x}_{train}$) and test ($\boldsymbol{x}_{test}$) sets such that we would be learning the simple equation $\hat{y} = b$. The resultant MSE for the training and test sets can be seen in Table 10 shown below.

| **MSE Training** | $\sigma_{train}$ | **MSE Test** | $\sigma_{test}$ |
|---|---|---|---|
| 85.9461 | 4.8111 | 81.6110 | 9.4262 |

Table 10: Comparison of MSE performances with naive regression.

## (b)

Based on the data loaded from the Boston housing data set. Linear regression was then performed on each one the of the attributes while incorportating a bias term such that the weight vector $\boldsymbol{w}\epsilon\Re^2$ is learnt. The results obtained can be seen in Figures 21 - 23 below.

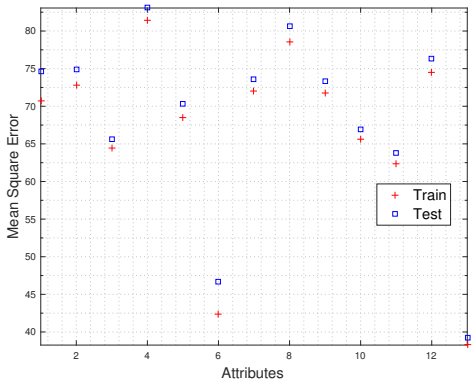



Figure 21: Average MSE for 13 attributes.

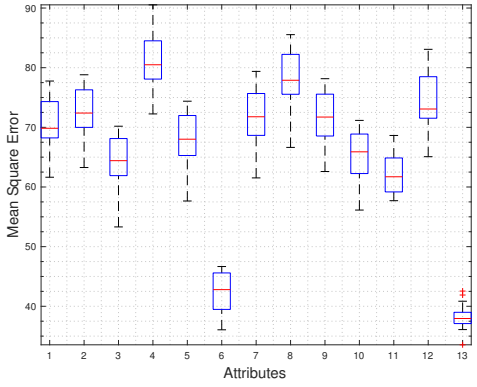



Figure 22: Box plot for MSE on the training set.

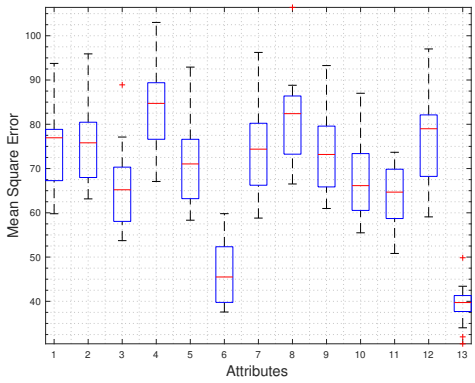



Figure 23: Box plot for MSE on the test set.

It can be seen from Figure 21 that the test errors were slightly larger than the training error. It can be seen that attributes 6 and 13 result in the lowest MSE for the training and test set (Figure 22.

Exercise 9 [(b)] continued on next page...

23). Therefore, it can be concluded that they aid the most in predicting the output value $\hat{y} \approx y$.

The resultant mean and standard deviation for each attribute can be seen in Table 11 shown below.

| **Attribute No.** | **MSE Training** | $\sigma_{train}$ | **MSE Test** | $\sigma_{test}$ |
|---|---|---|---|---|
| 1 | 70.7447 | 4.2548 | 74.6865 | 9.1147 |
| 2 | 72.8396 | 3.9868 | 74.9556 | 8.2195 |
| 3 | 64.3904 | 4.4036 | 65.5785 | 8.9016 |
| 4 | 81.4900 | 4.7935 | 83.0593 | 9.4631 |
| 5 | 68.4933 | 4.2524 | 70.3485 | 8.5533 |
| 6 | 42.3054 | 3.5653 | 46.7082 | 7.2898 |
| 7 | 72.0200 | 4.5887 | 73.5604 | 9.3796 |
| 8 | 78.5866 | 4.6642 | 80.5866 | 9.5948 |
| 9 | 71.7204 | 4.2897 | 73.3460 | 8.6945 |
| 10 | 65.5571 | 4.0711 | 66.9019 | 8.2134 |
| 11 | 62.3123 | 3.6134 | 63.7735 | 7.3950 |
| 12 | 74.4986 | 4.5331 | 76.2912 | 9.2620 |
| 13 | 38.2555 | 2.0485 | 39.1809 | 4.2550 |

Table 11: Comparison of attribute MSE performances.

Exercise 9 continued on next page...

## (c)

Linear regression was then performed using all the attributes available through the Boston housing data. The resultant training and test results can be seen in Table 12 below.

| **MSE Training** | $\sigma_{train}$ | **MSE Test** | $\sigma_{test}$ |
|---|---|---|---|
| 21.6457 | 1.2055 | 24.0536 | 2.9680 |

Table 12: Comparison of attribute MSE performances with all attributes.

It can be seen from Table 12 above that the resultant weight vector $\boldsymbol{w}$ performs far better than any of the individual regressors presented in Table 11 of Exercise 9(b). The figures demonstrating these results in a more graphical form can be seen in Figures 24 and 25 shown below.

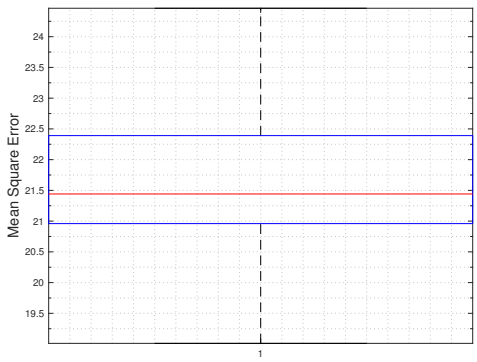



Figure 24: Box plot for MSE on the training set with all attributes.

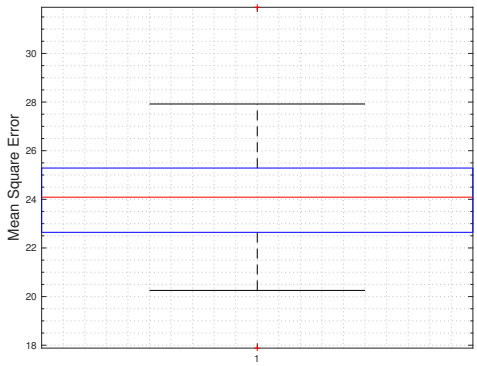



Figure 25: Box plot for MSE on the test set with all attributes.

# Exercise 10

## (a)

In this exercise Kernel Ridge Regression was computed on the Boston housing data in order to evaluate its performance in Exercise 10 against the performance of Linear Regression utilized in Exercise 9.

The function *kridgereg.m* was created with input parameters $\mathbf{K}$, $\boldsymbol{y}$ and $\gamma$ and output vector $\boldsymbol{\alpha}$ of dual weights. This function implements the following equation shown below.

$$\boldsymbol{\alpha} = (\boldsymbol{K} + \gamma l \boldsymbol{I}_l)^{-1} \boldsymbol{y} \tag{8}$$

The matlab code of the function *kridgereg.m* can be seen below.

```
function a = kridgereg(K, y, gamma)

a = (K + (gamma*size(y,1)*eye(size(y,1))) ) \ y;
```

In order to evaluate that the implementation is correct using the result from the assignment question page 6 the following equation was formed.

$$\boldsymbol{\alpha} = (\boldsymbol{X}\boldsymbol{X}' + \gamma l \boldsymbol{I}_l)^{-1} \boldsymbol{y} \tag{9}$$

This implementation correctly computes $\boldsymbol{\alpha}$ because the MATLAB left divide operation is used, instead of the explicit matrix inversion of the term $(\boldsymbol{K} + \gamma l \boldsymbol{I}_l)^{-1}$. The function correctly implements Equation 8 as required. Explicit matrix inversion is computationally expensive when compared to the left divide operation in MATLAB, thus is is computed slower. The residual error due to left divide operation is several orders of magnitude smaller than the explicit inverse function $inv((\boldsymbol{K} + \gamma l \boldsymbol{I}_l)^{-1})$ that could be used in MATLAB [2].

## (b)

The function *dualcost.m* was created as required with the following input parameters $\mathbf{K}$, $\boldsymbol{y}$ and the dual weight vector $\boldsymbol{\alpha}$ and returns the Mean Square Error (MSE) as shown below.

```
function mse = dualcost(K,y,a)

mse = (K*a - y)'*(K*a - y) / size(y,1);
```

The function shown above implements Equation 10 shown below.

$$MSE = \frac{1}{l}(\boldsymbol{K}_{test}\boldsymbol{\alpha} - \boldsymbol{y})'(\boldsymbol{K}_{test}\boldsymbol{\alpha} - \boldsymbol{y}) \tag{10}$$

## (c)

The vector of $\boldsymbol{\gamma}$ and $\boldsymbol{\sigma}$ were created as specified. Using the Gausian Kernel as shown in Equation 11 shown below, Kernel Ridge Regression was performed.

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}) \tag{11}$$

Exercise 10 [(c)] continued on next page...

## (1)

Figure 26 shown below demonstrates the cross-validation error as a function of $\gamma$ and $\sigma$.

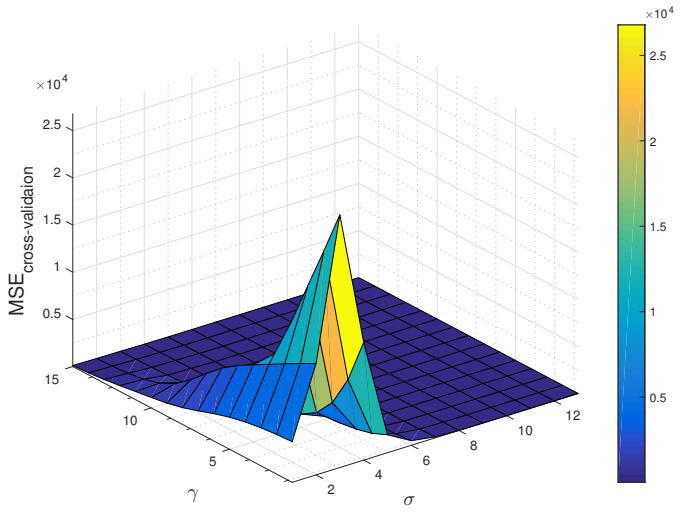



Figure 26: Cross validation error with Kernel Ridge Regression.

## (2)

The optimal hyper-parameters $\gamma^*$ and $\sigma^*$ that minimized the training and test sets were found to be $1.8190 \times 10^{-12}$ and 4096 respectively. The resultant MSE for the training and test sets can be seen in Table 13 below.

| **MSE train** | **MSE test** |
|---|---|
| 7.6349 | 14.4546 |

Table 13: MSE for optimal parameters $\gamma^*$ and $\sigma^*$.

Exercise 10 continued on next page...

## (d)

Exercise 10(a)-(c) was repeated over 20 random splits of the data and the resultant training and test errors were stored. Their relevant means and standard deviations were calculated and tabulated with the results obtained in Exercise 9 as can be seen in Table 14 shown below.

| **Method** | **MSE train** | **MSE test** |
|---|---|---|
| Naive Regression | 85.94±4.81 | 81.61±9.42 |
| Linear Regression (attribute 1) | 70.74 ± 4.25 | 74.68±9.11 |
| Linear Regression (attribute 2) | 72.83±3.98 | 74.95±8.21 |
| Linear Regression (attribute 3) | 64.39±4.40 | 65.57±8.90 |
| Linear Regression (attribute 4) | 81.49±4.79 | 83.05±9.46 |
| Linear Regression (attribute 5) | 68.49±4.25 | 70.34±8.55 |
| Linear Regression (attribute 6) | 42.30±3.56 | 46.70±7.28 |
| Linear Regression (attribute 7) | 72.02±4.58 | 73.56±9.37 |
| Linear Regression (attribute 8) | 78.58±4.66 | 80.58±9.59 |
| Linear Regression (attribute 9) | 71.72±4.28 | 73.34±8.69 |
| Linear Regression (attribute 10) | 65.55±4.07 | 66.90±8.21 |
| Linear Regression (attribute 11) | 62.31±3.61 | 63.77±7.39 |
| Linear Regression (attribute 12) | 74.49±4.53 | 76.29±9.26 |
| Linear Regression (attribute 13) | 38.25±2.04 | 39.18±4.25 |
| Kernel Ridge Regression | 7.20±1.53 | 19.89±25.50 |

Table 14: Comparison of regression methods.

It can be seen from Table 14 that Kernel Ridge Regression performed the best when compared to other linear regressors. Thus, it is clearly evident from these results that performing linear regression on the non-linear dataset does not yield good predictive performance.

Therefore by using the Gaussian Kernel function corresponding to a non-linear feature map, linear regression using Ridge Regression can be utilised in the feature space, thus allowing the creation of a nonlinear regression function on the nonlinear dataset in the input space as stated in the paragraph above. This is not computational expensive as the Kernel trick makes use of inner products of the feature vector $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)'\phi(\boldsymbol{x}_j)$ and rather than the actual vectors $\boldsymbol{x}_i$.

# Question 1

In order to understand the relationship between linear regression with a Gaussion Kernel $K_\beta(\boldsymbol{x}, \boldsymbol{t}) = exp(-\beta||\boldsymbol{x} - \boldsymbol{t}||^2)$ and 1-Nearest Neighbour classifier is to first understand the latter.

The 1-Nearest Neighbour algorithm can be broken down into three steps [3]:

1. Given training data $D = \{\boldsymbol{t}_i, y_i\}$ and an input $\boldsymbol{x}$ data point.
2. Find index $j = \text{argmin}_i\, d(\boldsymbol{x}, \boldsymbol{t}_i)$ that minimizes the euclidean distance ($||\boldsymbol{x} - \boldsymbol{t}_i||_2$).
3. $\hat{y} = sign(y_j)$ where $\hat{y} \in \pm 1$. Therefore, the input $\boldsymbol{x}$ has been classified as $\hat{y}$.

The downside of simple 1-Nearest Neighbour classification is that the nearest neighbour around a data point $\boldsymbol{x}$ of interest will receive equal weight as can be seen in Step 3 of the algorithm shown above. The 1-NN algorithm can be adjusted to incorporate distance by means of the euclidean distance as can be seen in Equation 12 below.

$$\hat{y}(\boldsymbol{x}) = sign(d(\boldsymbol{x}, \boldsymbol{t}_j) y_j) \tag{12}$$

$$\hat{y}(\boldsymbol{x}) = sign(||\boldsymbol{x} - \boldsymbol{t}_j||_2 y_j) \tag{13}$$

However, when using the Gaussian Kernel in classification, the index number of the nearest neighbours (K=1) for an input $\mathbf{x}$ does not need to found, in order to classify. Instead, all neighbours surrounding $\boldsymbol{x}$ can be summed up. Thus, data points (neighbours) closer to $\boldsymbol{x}$ will receive a larger weight and contribute more to the summation as shown below.

$$\hat{y}(\boldsymbol{x}) = sign(\sum_{j=1}^{n} K_\beta(\boldsymbol{x}, \boldsymbol{t}_j) y_j) \tag{14}$$

$$\hat{y}(\boldsymbol{x}) = sign(\sum_{j=1}^{n} exp\Big( - \beta||\boldsymbol{x} - \boldsymbol{t}_j||^2\Big) y_j) \tag{15}$$

$$\hat{y}(\boldsymbol{x}) = sign(\sum_{j=1}^{n} exp\Big( - \frac{||\boldsymbol{x} - \boldsymbol{t}_j||^2}{2\sigma^2}\Big) y_j) \tag{16}$$

It can be seen from Equation 15 and 16 take a similar form to that of Equation 13 shown earlier for 1-Nearest Neighbour. It is important to note now, that the parameter $\beta$ or correspondingly so, $\sigma$ defines the kernel width. In saying this, the parameter $\beta$ affects how suddenly the weighting of surrounding data points $\boldsymbol{t}_i$ reduces as their distance ($||\boldsymbol{x} - \boldsymbol{t}_j||^2$) increases.

The Gaussian Kernel in linear classification performs a similar weighting role as $d(\cdot, \cdot)$ mentioned earlier for 1-NN algorithm.

The variation of $\beta$ is an important role to understand in Equation 15, thus we will consider two cases namely:

1. $\beta \to 0$: As $\beta$ becomes smaller, the Gaussian Kernel will approach the value of 1 ($K_\beta(\boldsymbol{x}, \boldsymbol{t}_j) \to 1$), thus the weighting of each neighbour will become equally weighted regardless of distance. In affect, the width of the Gaussian Kernel is increasing, allowing more neighbours to weigh in.
2. $\beta \to \infty$: As $\beta$ becomes larger, the width of the Gaussian Kernel becomes smaller ($K_\beta(\boldsymbol{x}, \boldsymbol{t}_j) \to 0$). Thus in effect only taking into account neighbours in the immediate vicinity surrounding $\boldsymbol{x}$. As $\beta$ approaches zero, the classifier tends to the same behaviour as that of a 1-NN algorithm.

# Question 2

## (a)

In order to understand how we would change the perceptron to learn linear classifier with bias $b\Re$ as shown below in Equation 18.

$$f_{\boldsymbol{w},b}(\boldsymbol{x}) = sign(\boldsymbol{w}^T\boldsymbol{x} + b) \tag{17}$$

We need to evaluate the perceptron to learn linear classifier with no bias $b = 0$ as shown in Equation 18 below.

$$f_{\boldsymbol{w},b}(\boldsymbol{x}) = sign(\boldsymbol{w}^T\boldsymbol{x}) \tag{18}$$

Using the notation defined in [1], let $X$ denote the input space and $Y$ denote the output domain. Where, $X \subseteq \Re^n$, $Y = \{-1, 1\}$. More formally, this can be labelled as the set training set $S$:

$$S = \big((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ....., (\boldsymbol{x}_m, y_m)\big) \subseteq (X \times Y)^m$$

The Perceptron algorithm pseudocode with no bias can be seen below in Algorithm 1 [1],[4]:

**Algorithm 1** Perceptron Algorithm without bias

Given training set $S$ and learning rate $\eta\epsilon\Re^+$
$\boldsymbol{w}_1 \leftarrow \boldsymbol{0}, k \leftarrow 0$
**for** $i = 1$ to $m$
    if $y_i(sign(< \boldsymbol{w}_k \cdot \boldsymbol{x}_i >)) \leq 0$
        $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k + \eta y_i \boldsymbol{x}_i$
        $k \leftarrow k + 1$
    end if
end for
return $\boldsymbol{w}_k$

The Perceptron algorithm shown above was then updated to include the bias term $b$ as can be seen in Algorithm 2 shown below.

**Algorithm 2** Perceptron Algorithm with bias

Given training set $S$ and learning rate $\eta\epsilon\Re^+$
$\boldsymbol{w}_1 \leftarrow \boldsymbol{0}; k \leftarrow 0; b_0 \leftarrow 0$
$R \leftarrow \max_{1 \leq i \leq m} ||\boldsymbol{x}_i||$
**for** $i = 1$ to $m$
    if $y_i(sign(< \boldsymbol{w}_k \cdot \boldsymbol{x}_i > + b_k)) \leq 0$
        $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k + \eta y_i \boldsymbol{x}_i$
        $b_{k+1} \leftarrow b_k + \eta y_i R^2$
        $k \leftarrow k + 1$
    end if
end for
return $\boldsymbol{w}_k, b_k$

It can be seen when comparing Algorithm 1 and 2 there only subtle differences the update procedure with $b_k$ now included when a mistake is made, but the predominate procedure of the Perceptron remains unchanged.

Question 2 [(a)] continued on next page...

Intuitively, the expression contained within the $sign(\cdot)$ function namely, $< \boldsymbol{w} \cdot \boldsymbol{x} > +b$ splits the data $S$ by means of a hyperplane that satisfies the equation:

$$< \boldsymbol{w} \cdot \boldsymbol{x} > +b = 0$$

Thus, the separating of the data into two half spaces corresponding to the positive class $+1$ and the negative class $-1$. The weight vector $\boldsymbol{w}$ defines the direction perpendicular to the separating hyperplane. The term $b$ is the bias term, which moves the hyperplane parallel to itself as can be seen in Figure 27 below.

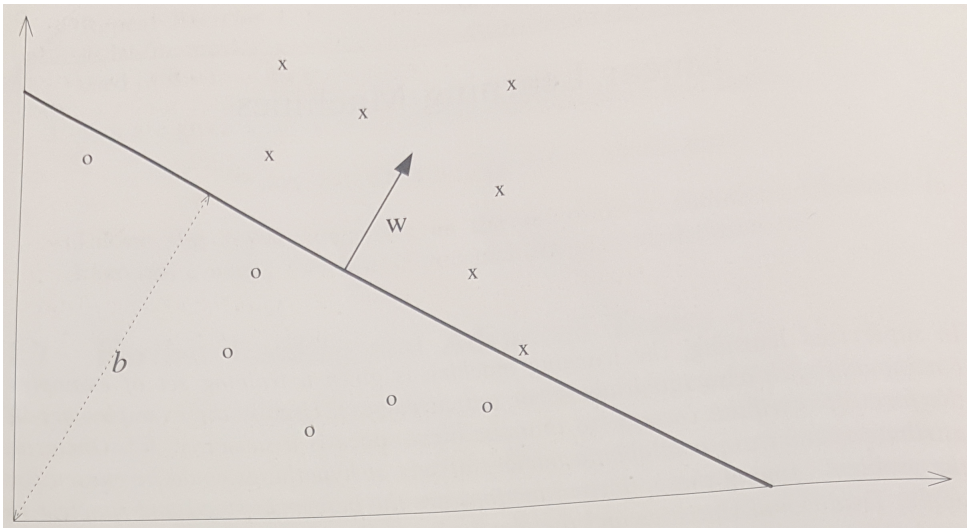



Figure 27: A separating hyperplane $(\boldsymbol{w}, b)$ for a two dimensional set [1].

It is clear to see from the algorithms derived for the Perceptron and the geometric interpretation that by including a bias term $b$ changes the threshold of the algorithm seen in line 5 of Algorithm 2 and update procedure. Thus, by varying $b$, it moves the hyperplane parallel to itself.

## (b)

Firstly, we will consider the bound on the number of mistakes incurred by the Perceptron algorithm without bias. This result was derived through Novikoff Thereom [1].

$$k \leq \left(\frac{R_2}{\gamma}\right)^2 \tag{19}$$

Where:

- $R_2$: $||\boldsymbol{x}_i|| \leq R_2$ for all $i$.

We will now derive the upper bounds on the number of mistakes the perceptron makes when the bias term is incorporated.

Given a sequence of data:

$$S = [(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ...., (\boldsymbol{x}_m, y_m)]$$

Therefore, there must exist a weight vector $\boldsymbol{w}^*$ such that is of unit length, $||\boldsymbol{w}^*|| = 1$ and the following relationship hold true for all points in S.

Question 2 [(b)] continued on next page... 

$$y_i\big( < \boldsymbol{w}^* \cdot \boldsymbol{x}_i > \big) \geq \gamma \tag{20}$$

We will derive the lower bound and the upper bounds on $||\boldsymbol{w}^*||$ to determine bound on perceptron mistakes. We will make a few changes to the $\boldsymbol{x_i}$ and $\boldsymbol{w}$ vectors to incorporate the bias term.

$$\hat{\boldsymbol{x}} = (\boldsymbol{x}^T, R_1)^T$$

$$\hat{\boldsymbol{w}} = (\boldsymbol{w}^T, \frac{b}{R_1})^T$$

Where:

- $R_1$: $||\hat{\boldsymbol{x}_i}|| \leq R_1$ for all $i$.

Therefore, let $\boldsymbol{w}_{k-1}$ be the weight used, when the k-th mistake was made. Starting the algorithm with $\boldsymbol{w}_0 = \boldsymbol{0}$. Assuming on the k-th mistake using data point $(\boldsymbol{x}_i, y_i)$ such that:

$$sign(\hat{\boldsymbol{w}_{k-1}}^T \hat{\boldsymbol{x}_i}) = sign(\boldsymbol{w}_{k-1}{}^T \boldsymbol{x}_i + b_{k-1}) \neq y_i$$

The incorrect classification implies:

$$y_i < \hat{\boldsymbol{w}_{k-1}} \cdot \hat{\boldsymbol{x}_i} >= y_i\big( < \boldsymbol{w}_{k-1} \cdot \boldsymbol{x}_i > + b_{k-1}\big) \leq 0 \tag{21}$$

Therefore, the Perceptron Algorithm would update in the following manner:

$$\hat{\boldsymbol{w}_k} = \hat{\boldsymbol{w}_{k-1}} + \eta y_i \hat{\boldsymbol{x}_i} = (\boldsymbol{w}_{k-1}{}^T, \frac{b_{k-1}}{R_1})^T + \eta y_i (\boldsymbol{x}_i{}^T, R_1)^T$$

Because the following is true [1]:

$$\frac{b_k}{R_1} = \frac{b_{k-1}}{R_1} + \eta y_i R_1$$

$$< \hat{\boldsymbol{w}_k} \cdot \hat{\boldsymbol{w}^*} >=< \hat{\boldsymbol{w}_{k-1}} \cdot \hat{\boldsymbol{w}^*} > + \eta y_i < \hat{\boldsymbol{x}_i} \cdot \hat{\boldsymbol{w}^*} > \tag{22}$$

This equation above can be the lower bound using Equation 21. Thus, the following expression becomes evident:

$$\eta\gamma + < \hat{\boldsymbol{w}_{k-1}} \cdot \hat{\boldsymbol{w}^*} > \leq \eta\gamma < \hat{\boldsymbol{x}_i} \cdot \hat{\boldsymbol{w}^*} > + < \hat{\boldsymbol{w}_{k-1}} \cdot \hat{\boldsymbol{w}}^* >$$

$$\eta\gamma + < \hat{\boldsymbol{w}_{k-1}} \cdot \hat{\boldsymbol{w}^*} > \leq < \hat{\boldsymbol{w}_k} \cdot \hat{\boldsymbol{w}^*} >$$

Therefore, by induction the following relationship is formed [5]:

$$k\eta\gamma \leq < \hat{\boldsymbol{w}_k} \cdot \hat{\boldsymbol{w}^*} > \tag{23}$$

In a similar manner the upper bound can be formed:

$$||\hat{\boldsymbol{w}_k}||^2 = ||\hat{\boldsymbol{w}_{k-1}} + \eta y_i \hat{\boldsymbol{x}_i}||^2$$

$$= ||\hat{\boldsymbol{w}_{k-1}}||^2 + 2\eta y_i < \hat{\boldsymbol{w}_{k-1}} \cdot \hat{\boldsymbol{x}_i} > + \eta^2 ||\hat{\boldsymbol{x}_i}||^2$$

Using the fact that ${y_i}^2 = 1$ and Equation 20.

$$||\hat{\boldsymbol{w}}_k||^2 \leq ||\hat{\boldsymbol{w}_{k-1}}||^2 + \eta^2||\hat{\boldsymbol{x}}_i||^2$$

$$||\hat{\boldsymbol{w}}_k||^2 \leq ||\hat{\boldsymbol{w}_{k-1}}||^2 + \eta^2(||\boldsymbol{x}_i||^2 + {R_1}^2)$$

Using the fact that $||\boldsymbol{x}_i|| \leq R_1$.

$$||\boldsymbol{x}_i||^2 \leq {R_1}^2$$

$$||\hat{\boldsymbol{w}}_k||^2 \leq ||\hat{\boldsymbol{w}_{k-1}}||^2 + \eta^2({R_1}^2 + {R_1}^2)$$

$$||\hat{\boldsymbol{w}}_k||^2 \leq ||\hat{\boldsymbol{w}_{k-1}}||^2 + 2\eta^2{R_1}^2$$

Therefore, through induction the following inequality can be shown to exist:

$$||\hat{\boldsymbol{w}}_k||^2 \leq 2\eta^2 k{R_1}^2$$
$$||\hat{\boldsymbol{w}}_k|| \leq \sqrt{2k}\eta R_1$$

$$k\eta\gamma \leq < \hat{\boldsymbol{w}}_k \cdot \hat{\boldsymbol{w}}^* > \leq ||\hat{\boldsymbol{w}}_k||||\hat{\boldsymbol{w}}^*|| \leq \sqrt{2k}\eta R_1||\hat{\boldsymbol{w}}^*||$$

$$k\eta\gamma \leq ||\hat{\boldsymbol{w}}_k||||\hat{\boldsymbol{w}}^*|| \leq \sqrt{2k}\eta R_1||\hat{\boldsymbol{w}}^*||$$

$$k\eta\gamma \leq \sqrt{2k}\eta R_1||\hat{\boldsymbol{w}}^*||$$

$$k\eta \leq \frac{\sqrt{2k}\eta R_1||\hat{\boldsymbol{w}}^*||}{\gamma}$$
$$k \leq \frac{2{R_1}^2||\hat{\boldsymbol{w}}^*||^{2\,2}}{\gamma}$$

Therefore, due to the fact that $b^* \leq R_1$ must hold true for a non-trivial seperation of the data [1], and the fact that $||\boldsymbol{w}^*|| = 1$ [6].

$$||\hat{\boldsymbol{w}}^*||^2 \leq ||\boldsymbol{w}^*||^2 + 1 = 1 + 1 = 2$$

$$k \leq \frac{4{R_1}^2}{\gamma^2}$$
$$k \leq \left(\frac{2R_1}{\gamma}\right)^2 \tag{24}$$

It is important to note the relationship $R_1$ and $R_2$ have in determining whether the bounds increases, assuming $\gamma$ remains unchanged. It can be shown at $R_1 > R_2$ in the following manner:

$$R_1 > R_2$$
$$||\hat{\boldsymbol{x}}_i|| > ||\boldsymbol{x}_i||$$

$$\sqrt{1^2 + \sum_{i=1}^{m} {x_i}^2} > \sqrt{\sum_{i=1}^{m} {x_i}^2}$$

Therefore when comparing Equations 19 and 24, it is clear that the bound does increase by a factor of 4. Thus, the bound on the number of mistakes increases by adding a bias term in the Perceptron Algorithm.

# Question 3

## (a)

$K_c$ is positive semidefinite kernel for $c \geq 0$.

$$K_c(x, z) := c + \sum_{i=1}^{n} x_i z_i \tag{25}$$

Since $\sum_{i=1}^{n} x_i z_i$ is a linear kernel it is positive semidefinite.

By the property we know that sum of two positive semidefinites is a positive seimidefinite. Hence if we assume c is 1x1 matrix and if it satisfies $yCy' \geq 0 \; \forall y \in \mathbb{R}$ condition, then we can say c is positive semidefinite.

We also know that every 1x1 dimension matrix is positive semidefinite if $c_{11} \geq 0$. Therefore, for $K_c$ to be a positive semidefinite, we need $c \geq 0$

## (b)

The c parameter does not directly influence kernel but controls the trade-off between the margin and the size of the slack variables. [1]

# Question 4

## (a)

The four algorithms were considered in order to evaluate the sample complexity of each, namely *The Perceptron*, *Winnow*, *Least Squares* and *1-nearest neighbour* [4].

In this question, we are concerned with looking at the relationship between the ($n$) number of features against the number of training examples ($m$) required which would lead to a reduction in the generalisation error. More formely, the "working definition" as stated in the question page, "..sample complexity is the minimum number of examples ($m$) to incure no more than 10% generalisation error (on average)" [7].

Thus, the algorithms mentioned earlier were, implemented and their performance was evaluated. The sample complexity plots for each algorithm can be seen in Figures 28 - 31 shown below.

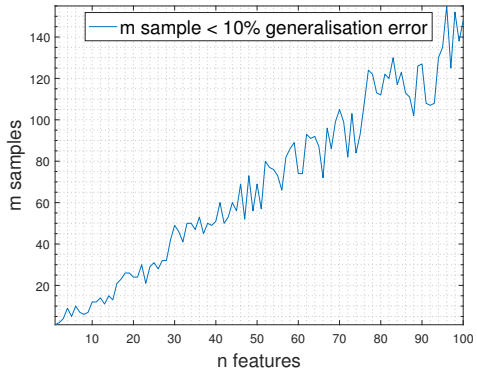



Figure 28: Estimated sample complexity of *Perceptron* algorithm.

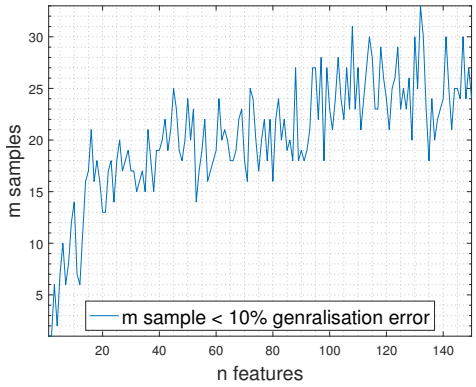



Figure 29: Estimated sample complexity of *Winnow* algorithm.

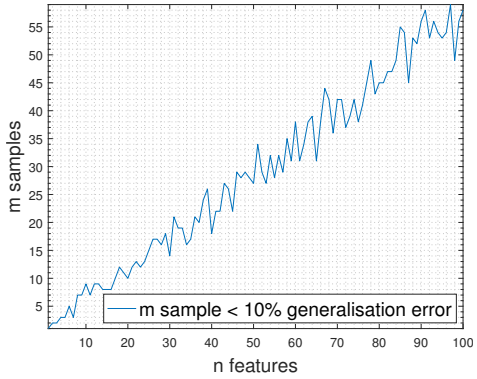



Figure 30: Estimated sample complexity of *Least Squares* algorithm.

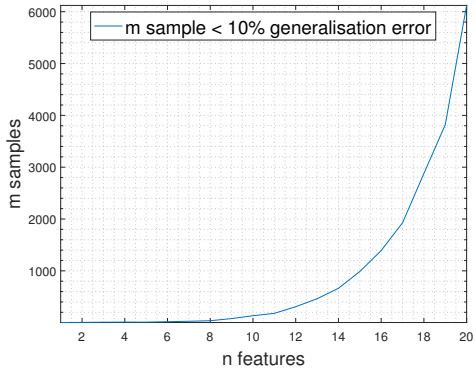



Figure 31: Estimated sample complexity of *1-Nearest Neighbour* algorithm.

It can be seen from Figure 28 - 31 shown above that for each algorithm as the number of features increase, the corresponding number of training samples increase. 'Therefore, the sample complexity increases as a function of features.

All the MATLAB code required to generate these results can be seen in Appendix .

## (b)

It became evident through empirically testing in Q4(a), that computing the "sample complexity" exactly through simulation became extremely expensive.

In order to test the sample complexity exactly, we iteratively increased $n$ dimensions of the features of the input data. And for each such vector, we iteratively trained and tested on those trained parameters by varying the lengths of samples $m$. For each combination of $n$ feature and $m$ samples, we computed the percent error of mis-classification. For each value of $n$ we evaluated the percent error and continued to test, until the error for a particular value of $m$ is not at least 10%. When error drops below or equal to 10%, we record the value of $m$ and do the same process for the next value of $n$.

The computational time for each algorithm to converge to a solution as the number of samples ($m$) and number of features ($n$) were increased can be seen in the following table.

| **Algorithm** | **Mean Comp. Time** | $\sigma_{comptime}$ | **n** | **m** | **Test sample size** |
|---|---|---|---|---|---|
| Perceptron | 337s | $\pm 10s$ | 100 | 500 | 500 |
| Winnow | 198s | $\pm 6s$ | 100 | 500 | 500 |
| Least Squares | 140s | $\pm 3s$ | 100 | 500 | 500 |
| 1-NN | 21682s | $\pm 200s$ | 20 | 10000 | 500 |

Table 15: Computational time for algorithms.

It can be seen from Table 15 that *1 Nearest Neighbour* algorithm took the longest, whereas *Least Square* algorithm was the fastest amongst all.
Thus, in order to evaluate the sample complexity for each algorithm certain trade-off were made namely:

1. Number of independent trial were reduced: In order to find the generalized error, we need to take test on multiple trials and then average out the errors. We had to reduce them to 100 and trade off on the accuracy for speed of execution.

2. Dimension ($n$) was limited for each algorithm: In case of *1-NN* we had to limit the dimension to just 20 features which took around 6 hours to compute. Where as are we limited the dimension $n$ for other algorithms at 100.

3. In order to faster execute the sample complexity over the range, we further enhanced the method to iterate over odd values of the dimensions ($n$) and samples ($m$), at the cost of accuracy.

## (c)

In order to fully understand the relationship of $m$ vs. $n$, the Figures 28 - 31 generated in Q4(a) were analysed closer.

For Perceptron algorithm [8] it can be seen from Figure 28 that as $n \to \infty$, the sample complexity increased as a linear function of dimension ($m = \Theta(n)$). It is found that Perceptron make mistakes in each trial and is $\Omega(kN)$ mistakes.

If we then consider the complexity of the Winnow algorithm, it can be deduced from Figure 29 that the sample complexity is a logarithmic function of dimension ($m = \Theta(log(n))$). As we increase the dimensions ($n$) the number of samples ($m$) increase at a certain extent and then slows down and nearly stays constant at $\infty$. The mistake bound is $\Omega(kln2^n)$

In both Perceptron algorithm and Winnow, the new hypothesis depends on the old hypothesis at a learning rate $\eta > 0$.

The sampling complexity of *Least Squares* algorithm was found to be linear in dimensions ($n$) with respect to sampling ($m$). This algorithm is the fastest to run out of other three. The complexity was found to be $m = \Theta(n)$ for $n \to \infty$ from Figure 30.

Finally, the 1-Nearest Neighbour algorithm was inspected and the found that the sample complexity is exponential in nature with respect to dimensions ($n$) $\to \infty$ from 31. 1-NN performance was seen to be worst out of all four algorithms. It took more than 6 hours to run for just up to 20 dimensions. It made most errors and hence as a result it requires more number of samples to converge for generalization error below 10%

# Appendices

## Exercise 1: Code

In this and all of the following sections of the code we have used external functions at multiple places. Please refer Library Functions section at the end of the appendix for implementation.

**ex1.m**

```
%-------------------------------------------------------------------------%
% Module: GI01 - Supervised Learning
% Assignment : Coursework 1
% Author : Russel Daries, Nitish Mutha
% Student ID: 16079408, 15113106
% Question: 1
% Section: Part 1
% Description: Least Squares Regression
% ------------------------------------------------------------------------%

% clearing memory
clear all
close all
clc

addpath('../library')

%% Ex1(a)

w = randn(1,1);
x = randn(600,1);
n = randn(600,1);

%% Ex1(b)
part_b_result = calculateLSR(x,w,n,100,500)


%% Ex1(c)
part_c_result = calculateLSR(x,w,n,10,500)

%% Ex1(d)

result_100 = [];
result_10 = [];
for j = 1:200
    w_j = randn(1,1);
    x_j = randn(600,1);
    n_j = randn(600,1);
    result_100 = [result_100 ; calculateLSR(x_j,w_j,n_j,100,500)];
    result_10 = [result_10 ; calculateLSR(x_j,w_j,n_j,10,500)];
end

average_result_100 = mean(result_100)
```

```
average_result_10 = mean(result_10)

% Code to plot resultant figures

figure;
plot(result_10(:,2),'k*')
hold on
plot(result_100(:,2),'b+')
hold on
plot(average_result_10(2)*ones(length(result_10),1),'r--')
hold on
plot(average_result_100(2)*ones(length(result_100),1),'m--')
xlabel('Trial number','FontSize',15)
ylabel('Mean Squared Error','FontSize',15)
grid on;
set(gcf, 'Color', 'w');
leg=legend('train10_{err}','train100_{err}','train-err10_{avg}','train-err100_
    {avg}','Location','Best')
set(leg,'FontSize',15)
set(gca,'YMinorTick','on')
grid minor
axis tight;
print('ex1d_err_train','-depsc')
close all;


figure
plot(result_10(:,3),'k*')
grid on;
hold on
plot(result_100(:,3),'b+')
hold on
plot(average_result_10(2)*ones(length(result_10),1),'c--')
hold on
plot(average_result_100(3)*ones(length(result_100),1),'r--')
hold on
xlabel('Trial number','FontSize',15)
ylabel('Mean Squared Error','FontSize',15)
set(gcf, 'Color', 'w');
leg=legend('test10_{err}','test100_{avg}','test10-err_{avg}','test100-err_{avg
    }','Location','Best');
set(leg,'FontSize',15)
set(gca,'YMinorTick','on')
grid minor
axis tight;
ylim([0.8 2.2])
print('ex1d_err_test','-depsc')
close all;

```

```
figure;
plot(result_10(:,1),'k*')
hold on
plot(result_100(:,1),'b+')
hold on
plot(average_result_10(1)*ones(length(result_10),1),'r--')
hold on
plot(average_result_100(1)*ones(length(result_100),1),'m--')
xlabel('Trial number','FontSize',15)
ylabel('Weight Estimate (w)','FontSize',15)
grid on;
set(gcf, 'Color', 'w');
leg=legend('w-train10','w-train100','w-train-err10_{avg}','w-train-err100_{avg}','Location','Best')
set(leg,'FontSize',15)
set(gca,'YMinorTick','on')
grid minor
axis tight;
print('ex1d_err_wtrain','-depsc')
close all;
```

## Exercise 2: Code

**ex2.m**

```
%-------------------------------------------------------------------------%
% Course : Supervised Learning
% Assignment : 1
% Excercise 2
% Programmers: Russel Daries, Nitish Mutha
% Student ID: 16079408, 15113106
% Question: 2
% Section: Part 1
% Description: Least Squares Regression
% ------------------------------------------------------------------------%

clear all
close all
clc

addpath('../library')

%% Ex2(a)
w = randn(10,1);
x = randn(600,10);
n = randn(600,1);

%% Ex2(b)
[part_b_result_w,part_b_result_mse_train,part_b_result_mse_test] =
    calculateLSRex2(x,w,n,100,500);
[part_c_result_w,part_c_result_mse_train,part_c_result_mse_test] =
    calculateLSRex2(x,w,n,10,500);

% Initilizae variables
result_100 = [];
result_10 = [];
w_100 = [];
w_10 = [];
mse_train_100 = [];
mse_train_10 = [];
mse_test_100 = [];
mse_test_10 =[];

for j = 1:200
    % Generate dataset
    w_j = randn(10,1);
    x_j = randn(600,10);
    n_j = randn(600,1);

    % Store resultant calculation
    % 100 training examples
```

```
45      [result_w_100_j, result_mse_train_100_j, result_mse_test_100_j] =
            calculateLSRex2(x_j,w_j,n_j,100,500);
46      w_100 = [w_100; result_w_100_j];
47      mse_train_100 = [mse_train_100 ; result_mse_train_100_j];
48      mse_test_100 = [mse_test_100; result_mse_test_100_j];
49
50      % 10 training example
51      [result_w_10_j, result_mse_train_10_j, result_mse_test_10_j] =
            calculateLSRex2(x_j,w_j,n_j,10,500);
52      w_10 = [w_10; result_w_10_j];
53      mse_train_10 = [mse_train_10 ; result_mse_train_10_j];
54      mse_test_10 = [mse_test_10; result_mse_test_10_j];
55
56  end
57
58  % Average MSE for 10,100 training examples
59  average_mse_train_100 = mean(mse_train_100)
60  average_mse_test_100 = mean(mse_test_100)
61
62  % Average MSE on the test sets
63  average_mse_train_10 = mean(mse_train_10)
64  average_mse_test_10 = mean(mse_test_10)
65
66  % Code to plot resultant figures
67
68  figure;
69  h1 = plot(mse_train_10,'k*')
70  hold on
71  h2 = plot(mse_train_100,'b+')
72  hold on
73  h3 =plot(average_mse_train_10*ones(length(mse_train_10),1),'r--')
74  hold on
75  h4 = plot(average_mse_train_100*ones(length(mse_test_100),1),'m--')
76  xlabel('Trial number','FontSize',15)
77  ylabel('Mean Squared Error','FontSize',15)
78  grid on;
79  set(gcf, 'Color', 'w');
80  leg=legend('train10_{err}','train100_{err}','train-err10_{avg}','train-err100_
        {avg}','Location','Best')
81  set(leg,'FontSize',15)
82  set(gca,'YMinorTick','on')
83  grid minor
84  axis tight;
85  print('ex2b_err_train','-depsc')
86  close all;
87
88
89  figure
90  plot(mse_test_10,'k*')
```

```
91  grid on;
92  hold on
93  plot(mse_test_100,'b+')
94  hold on
95  plot(average_mse_test_10*ones(length(mse_test_10),1),'m--')
96  hold on
97  plot(average_mse_test_100*ones(length(mse_test_100),1),'r--')
98  hold on
99  xlabel('Trial number','FontSize',15)
100 ylabel('Mean Squared Error','FontSize',15)
101 set(gcf, 'Color', 'w');
102 leg=legend('test10_{err}','test100_{avg}','test10-err_{avg}','test100-err_{avg
        }','Location','Best');
103 set(leg,'FontSize',15)
104 set(gca,'YMinorTick','on')
105 grid minor
106 axis tight;
107 ylim([0.0 3e3])
108 print('ex2b_err_test','-depsc')
109 close all;
110
111 figure;
112 plot(w_10,'k*')
113 hold on
114 plot(w_100,'b+')
115 hold on
116 plot(mean(w_10)*ones(length(w_10),1),'r--')
117 hold on
118 plot(mean(w_100)*ones(length(w_100),1),'m--')
119 xlabel('Trial number','FontSize',15)
120 ylabel('Weight Estimate (w)','FontSize',15)
121 grid on;
122 set(gcf, 'Color', 'w');
123 leg=legend('w-train10','w-train100','w-train10_{avg}','w-train100_{avg}','
        Location','Best')
124 set(leg,'FontSize',15)
125 set(gca,'YMinorTick','on')
126 grid minor
127 axis tight;
128 print('ex2b_wtrain','-depsc')
129 close all;
```

## Exercise 4: Code

**ex4.m**

```
%-------------------------------------------------------------------------%
% Module: GI01 - Supervised Learning
% Assignment : Coursework 1
% Author : Russel Stuart Daries, Nitish Mutha
% Student ID: 16079408 ,15113106
% Question: 4
% Section: Part 1
% Description: Effect of Regularisation Parameter
% -------------------------------------------------------------------------%

%%
clear all
close all
clc

addpath('../library')

%% Ex4(a)
w = randn(10,1);
x = randn(600,10);
n = randn(600,1);
gamma_power = -6:1:3;
% Range of gamma values
gamma = 10.^gamma_power;

%Initialise variables
result_mse_train_100 = [];
result_mse_test_100 = [];
for i = 1 : size(gamma,2)
    [w_estimator_100, mse_train_100, mse_test_100] = calculateLSRex4(x,w,n
        ,100,500,gamma(i));
    result_mse_train_100 = [result_mse_train_100 ; mse_train_100];
    result_mse_test_100 = [result_mse_test_100 ; mse_test_100];
end

figure
plot(log(gamma),result_mse_train_100,'b-o');
hold on;
plot(log(gamma),result_mse_test_100,'r-*');
grid on;
set(gcf, 'Color', 'w');
xlabel('{log(\gamma)}','FontSize',15);
ylabel('Mean Square Error','FontSize',15);
leg=legend('MSE-train_{100}','MSE-test_{100}','Location','Best');
set(leg,'FontSize',15);
set(gca,'YMinorTick','on');
```

```
grid minor
axis tight;
print('ex4a_mse_100','-depsc');
close all;


%% Ex4(b)
result_mse_train_10 = [];
result_mse_test_10 = [];
for i = 1 : size(gamma,2)
    [w_estimator_10, mse_train_10, mse_test_10] = calculateLSRex4(x,w,n
        ,10,500,gamma(i));
    result_mse_train_10 = [result_mse_train_10 ; mse_train_10];
    result_mse_test_10 = [result_mse_test_10 ; mse_test_10];
end

figure
plot(log(gamma),result_mse_train_10,'b-o');
hold on;
plot(log(gamma),result_mse_test_10,'r-*');
grid on;
set(gcf, 'Color', 'w');
xlabel('{log(\gamma)}','FontSize',15);
ylabel('Mean Square Error','FontSize',15);
leg=legend('MSE-train_{10}','MSE-test_{10}','Location','Best');
set(leg,'FontSize',15);
set(gca,'YMinorTick','on');
grid minor
axis tight;
print('ex4b_mse_10','-depsc');
close all;


%% Ex4(c)

result_mse_train_100 = [];
result_mse_train_10 = [];
result_mse_test_100 = [];
result_mse_test_10 = [];
result_w_100 = [];
result_w_10 = [];

% 200 independant trials
for j = 1:200

    w_j = randn(10,1);
    x_j = randn(600,10);
    n_j = randn(600,1);
    w_100 = [];
```

```
94      w_10 = [];
95
96      mse_train_100 = [];
97      mse_train_10 = [];
98      mse_test_100 = [];
99      mse_test_10 =[];
100
101     % Varying regularisation parameter
102     for i=1: size(gamma,2)
103         [result_w_100_j, result_mse_train_100_j, result_mse_test_100_j] =
                calculateLSRex4(x_j,w_j,n_j,100,500,gamma(i));
104         w_100 = [w_100, result_w_100_j];
105         mse_train_100 = [mse_train_100, result_mse_train_100_j];
106         mse_test_100 = [mse_test_100, result_mse_test_100_j];
107
108         [result_w_10_j, result_mse_train_10_j, result_mse_test_10_j] =
                calculateLSRex4(x_j,w_j,n_j,10,500,gamma(i));
109         w_10 = [w_10, result_w_10_j];
110         mse_train_10 = [mse_train_10, result_mse_train_10_j];
111         mse_test_10 = [mse_test_10, result_mse_test_10_j];
112
113     end
114     result_mse_train_100 = [result_mse_train_100 ; mse_train_100];
115     result_mse_train_10 = [result_mse_train_10 ; mse_train_10];
116
117     result_mse_test_100 = [result_mse_test_100 ; mse_test_100];
118     result_mse_test_10 = [result_mse_test_10 ; mse_test_10];
119
120     result_w_100 = [result_w_100 ; w_100];
121     result_w_10 = [result_w_10 ; w_10];
122
123 end
124
125 % Computing the average result
126 average_mse_train_100 = mean(result_mse_train_100);
127 average_mse_test_100 = mean(result_mse_test_100);
128 average_mse_train_10 = mean(result_mse_train_10);
129 average_mse_test_10 = mean(result_mse_test_10);
130
131
132 figure
133 plot(log(gamma),average_mse_train_100,'b-*');
134 hold on;
135 grid on;
136 plot(log(gamma),average_mse_test_100,'r-o');
137 set(gcf, 'Color', 'w');
138 xlabel('{log(\gamma)}','FontSize',15);
139 ylabel('Mean Square Error','FontSize',15);
140 leg=legend('MSE-train_{100}','MSE-test_{100}','Location','Best');
```

```
141 set(leg,'FontSize',15);
142 set(gca,'YMinorTick','on');
143 grid minor
144 axis tight;
145 print('ex4c_mse_200_100','-depsc');
146 close all;
147
148 figure;
149 plot(log(gamma),average_mse_train_10,'b-*');
150 hold on; grid on;
151 plot(log(gamma),average_mse_test_10,'r-o');
152 grid on;
153 xlabel('{log(\gamma)}','FontSize',15);
154 ylabel('Mean Square Error','FontSize',15);
155 leg=legend('MSE-train_{10}','MSE-test_{10}','Location','Best');
156 set(leg,'FontSize',15);
157 set(gca,'YMinorTick','on');
158 grid minor
159 axis tight;
160 print('ex4c_mse_200_10','-depsc');
161 close all;
```

## Exercise 5: Code

**ex5.m**

```
%-------------------------------------------------------------------------%
% Module: GI01 - Supervised Learning
% Assignment : Coursework 1
% Author : Russel Stuart Daries, Nitish Mutha
% Student ID: 16079408 ,15113106
% Question: 5
% Section: Part 1
% Description: Tuning Regularization Parameter with Validation Set
% -------------------------------------------------------------------------%

%%
clear all
close all
clc

addpath('../library')

%% Ex5(a) and Ex5(b)

gamma_power = -6:1:3;
gamma = 10.^gamma_power;

result_mse_train_100 = [];
result_mse_train_10 = [];

result_mse_valid_100 = [];
result_mse_valid_10 = [];

result_mse_test_100 = [];
result_mse_test_10 = [];

result_w_100 = [];
result_w_10 = [];

best_gamma_100 = zeros(200,1);
best_gamma_10 = zeros(200,1);

g_average_mse_test_100 = [];
g_average_mse_test_10 = [];

for j = 1:200

    w_j = randn(10,1);
    x_j = randn(600,10);
    n_j = randn(600,1);
    w_100 = [];
```

```
47      w_10 = [];
48
49      mse_train_100 = [];
50      mse_valid_100 = [];
51      mse_test_100 = [];
52
53      mse_train_10 = [];
54      mse_valid_10 = [];
55      mse_test_10 =[];
56
57
58      for i=1: size(gamma,2)
59          % Calculate MSE for each data set with 100 training samples
60          [result_w_100_j, result_mse_train_100_j,result_mse_valid_100_j,
                result_mse_test_100_j] = calculateLSRex5(x_j,w_j,n_j,80,20,500,
                gamma(i));
61          w_100 = [w_100; result_w_100_j];
62          mse_train_100 = [mse_train_100, result_mse_train_100_j];
63          mse_valid_100 = [mse_valid_100, result_mse_valid_100_j];
64          mse_test_100 = [mse_test_100, result_mse_test_100_j];
65          % Calculate MSE for each data set with 10 training samples
66          [result_w_10_j, result_mse_train_10_j,result_mse_valid_10_j,
                result_mse_test_10_j] = calculateLSRex5(x_j,w_j,n_j,8,2,500,gamma(
                i));
67          w_10 = [w_10; result_w_10_j];
68          mse_train_10 = [mse_train_10, result_mse_train_10_j];
69          mse_valid_10 = [mse_valid_10, result_mse_valid_10_j];
70          mse_test_10 = [mse_test_10, result_mse_test_10_j];
71
72      end
73      % Find optimal gamma value from validation set MSE
74      [min_mse_100,idx_min_100] = min(mse_valid_100);
75      best_gamma_100(j,1) = gamma(idx_min_100);
76
77      [min_mse_10,idx_min_10] = min(mse_valid_10);
78      best_gamma_10(j,1) = gamma(idx_min_10);
79
80      result_w_100 = [result_w_100 ; w_100];
81      result_w_10 = [result_w_10 ; w_10];
82
83      result_mse_train_100 = [result_mse_train_100 ; mse_train_100];
84      result_mse_train_10 = [result_mse_train_10 ; mse_train_10];
85
86      result_mse_valid_100 = [result_mse_valid_100 ; mse_valid_100];
87      result_mse_valid_10 = [result_mse_valid_10 ; mse_valid_10];
88
89      result_mse_test_100 = [result_mse_test_100 ; mse_test_100];
90      result_mse_test_10 = [result_mse_test_10 ; mse_test_10];
91
```

```
92      [w_test_100, g_result_mse_test_100] = calculateLSRTest(x_j,w_j,n_j
            ,80,20,500,gamma(idx_min_100));
93      g_average_mse_test_100 = [g_average_mse_test_100, g_result_mse_test_100];
94
95      [w_test_10, g_result_mse_test_10] = calculateLSRTest(x_j,w_j,n_j,8,2,500,
            gamma(idx_min_10));
96      g_average_mse_test_10 = [g_average_mse_test_10, g_result_mse_test_10];
97
98
99  end
100
101 % Compute averages
102 average_mse_train_100 = mean(result_mse_train_100);
103 average_mse_valid_100 = mean(result_mse_valid_100);
104 average_mse_test_100 = mean(result_mse_test_100);
105
106 average_mse_train_10 = mean(result_mse_train_10);
107 average_mse_valid_10 = mean(result_mse_valid_10);
108 average_mse_test_10 = mean(result_mse_test_10);
109
110 % plots for mse vs gamma values for 100
111 figure;
112 plot(log(gamma),average_mse_train_100,'r-*');
113 hold on;
114 plot(log(gamma),average_mse_valid_100,'g-o');
115 plot(log(gamma),average_mse_test_100,'b-+');
116 grid on;
117 set(gcf, 'Color', 'w');
118 xlabel('Regularization parameter (\gamma)','FontSize',15);
119 ylabel('Mean Square Error','FontSize',15);
120 leg=legend('MSE_{train}','MSE_{validation}','MSE_{test}','Location','Best');
121 set(leg,'FontSize',15);
122 set(gca,'YMinorTick','on');
123 grid minor
124 axis tight;
125 print('ex5a_80_20','-depsc');
126 close all;
127
128 % plots for mse vs gamma values for 10
129 figure;
130 plot(log(gamma),average_mse_train_10,'r-*');
131 hold on;
132 plot(log(gamma),average_mse_valid_10,'g-o');
133 plot(log(gamma),average_mse_test_10,'b-+');
134 grid on;
135 set(gcf, 'Color', 'w');
136 xlabel('Regularization parameter (\gamma)','FontSize',15);
137 ylabel('Mean Square Error','FontSize',15);
138 leg=legend('MSE_{train}','MSE_{validation}','MSE_{test}','Location','Best');
```

```
139 set(leg,'FontSize',15);
140 set(gca,'YMinorTick','on');
141 grid minor
142 axis tight;
143 print('ex5b_8_2','-depsc');
144 close all;
145
146 % plots for test mse vs experiments for 100 and 10
147 figure;
148 plot(g_average_mse_test_100,'r*');
149 hold on;
150 plot(g_average_mse_test_10,'b+');
151 grid on;
152 set(gcf, 'Color', 'w');
153 xlabel('Experiments','FontSize',15);
154 ylabel('Mean Square Error','FontSize',15);
155 leg=legend('MSE_{100}','MSE_{10}','Location','Best');
156 set(leg,'FontSize',15);
157 set(gca,'YMinorTick','on');
158 grid minor
159 axis tight;
160 ylim([0.0 40])
161 print('ex5b_mse_optimal_gamma','-depsc');
162 close all;
163
164 %% Ex5(c)
165 mean_gamma_100 = mean(best_gamma_100)
166 mean_gamma_10 = mean(best_gamma_10)
167
168 %% Ex5(d)
169
170 % Repeating tuning of regularization parameter with validation set with
171 % data similair to Ex1(a)-(d)
172 result_mse_train_100 = [];
173 result_mse_train_10 = [];
174
175 result_mse_valid_100 = [];
176 result_mse_valid_10 = [];
177
178 result_mse_test_100 = [];
179 result_mse_test_10 = [];
180
181 result_w_100 = [];
182 result_w_10 = [];
183
184 best_gamma_100 = zeros(200,1);
185 best_gamma_10 = zeros(200,1);
186
187 g_average_mse_test_100 = [];
```

```
188 g_average_mse_test_10 = [];
189
190 for j = 1:200
191
192     w_j = randn(1,1);
193     x_j = randn(600,1);
194     n_j = randn(600,1);
195     w_100 = [];
196     w_10 = [];
197
198     mse_train_100 = [];
199     mse_valid_100 = [];
200     mse_test_100 = [];
201
202     mse_train_10 = [];
203     mse_valid_10 = [];
204     mse_test_10 =[];
205
206
207     for i=1: size(gamma,2)
208         % Calculate MSE for each data set with 100 training samples
209         [result_w_100_j, result_mse_train_100_j,result_mse_valid_100_j,
                result_mse_test_100_j] = calculateLSRex5(x_j,w_j,n_j,80,20,500,
                gamma(i));
210         w_100 = [w_100; result_w_100_j];
211         mse_train_100 = [mse_train_100, result_mse_train_100_j];
212         mse_valid_100 = [mse_valid_100, result_mse_valid_100_j];
213         mse_test_100 = [mse_test_100, result_mse_test_100_j];
214         % Calculate MSE for each data set with 10 training samples
215         [result_w_10_j, result_mse_train_10_j,result_mse_valid_10_j,
                result_mse_test_10_j] = calculateLSRex5(x_j,w_j,n_j,8,2,500,gamma(
                i));
216         w_10 = [w_10; result_w_10_j];
217         mse_train_10 = [mse_train_10, result_mse_train_10_j];
218         mse_valid_10 = [mse_valid_10, result_mse_valid_10_j];
219         mse_test_10 = [mse_test_10, result_mse_test_10_j];
220
221     end
222     % Find optimal gamma value from validation set MSE
223     [min_mse_100,idx_min_100] = min(mse_valid_100);
224     best_gamma_100(j,1) = gamma(idx_min_100);
225
226     [min_mse_10,idx_min_10] = min(mse_valid_10);
227     best_gamma_10(j,1) = gamma(idx_min_10);
228
229     result_w_100 = [result_w_100 ; w_100];
230     result_w_10 = [result_w_10 ; w_10];
231
232     result_mse_train_100 = [result_mse_train_100 ; mse_train_100];
```

```
233     result_mse_train_10 = [result_mse_train_10 ; mse_train_10];
234
235     result_mse_valid_100 = [result_mse_valid_100 ; mse_valid_100];
236     result_mse_valid_10 = [result_mse_valid_10 ; mse_valid_10];
237
238     result_mse_test_100 = [result_mse_test_100 ; mse_test_100];
239     result_mse_test_10 = [result_mse_test_10 ; mse_test_10];
240
241     [w_test_100, g_result_mse_test_100] = calculateLSRTest(x_j,w_j,n_j
            ,80,20,500,gamma(idx_min_100));
242     g_average_mse_test_100 = [g_average_mse_test_100, g_result_mse_test_100];
243
244     [w_test_10, g_result_mse_test_10] = calculateLSRTest(x_j,w_j,n_j,8,2,500,
            gamma(idx_min_10));
245     g_average_mse_test_10 = [g_average_mse_test_10, g_result_mse_test_10];
246
247
248 end
249
250 % Compute averages
251 average_mse_train_100 = mean(result_mse_train_100);
252 average_mse_valid_100 = mean(result_mse_valid_100);
253 average_mse_test_100 = mean(result_mse_test_100);
254
255 average_mse_train_10 = mean(result_mse_train_10);
256 average_mse_valid_10 = mean(result_mse_valid_10);
257 average_mse_test_10 = mean(result_mse_test_10);
258
259 % plots for mse vs gamma values for 100
260 figure;
261 plot(log(gamma),average_mse_train_100,'r-*');
262 hold on;
263 plot(log(gamma),average_mse_valid_100,'g-o');
264 plot(log(gamma),average_mse_test_100,'b-+');
265 grid on;
266 set(gcf, 'Color', 'w');
267 xlabel('Regularization parameter (\gamma)','FontSize',15);
268 ylabel('Mean Square Error','FontSize',15);
269 leg=legend('MSE_{train}','MSE_{validation}','MSE_{test}','Location','Best');
270 set(leg,'FontSize',15);
271 set(gca,'YMinorTick','on');
272 grid minor
273 axis tight;
274 print('ex5d_80_20','-depsc');
275 close all;
276
277 % plots for mse vs gamma values for 10
278 figure;
279 plot(log(gamma),average_mse_train_10,'r-*');
```

```
hold on;
plot(log(gamma),average_mse_valid_10,'g-o');
plot(log(gamma),average_mse_test_10,'b-+');
grid on;
set(gcf, 'Color', 'w');
xlabel('Regularization parameter (\gamma)','FontSize',15);
ylabel('Mean Square Error','FontSize',15);
leg=legend('MSE_{train}','MSE_{validation}','MSE_{test}','Location','Best');
set(leg,'FontSize',15);
set(gca,'YMinorTick','on');
grid minor
axis tight;
print('ex5d_8_2','-depsc');
close all;
```

## Exercise 6: Code

**ex6.m**

```
%-------------------------------------------------------------------------%
% Module: GI01 - Supervised Learning
% Assignment : Coursework 1
% Author : Russel Stuart Daries, Nitish Mutha
% Student ID: 16079408 ,15113106
% Question: 6
% Section: Part 1
% Description: Tuning Regularization Parameter with Cross-Validation
% -------------------------------------------------------------------------%

%%
clear all
close all
clc

addpath('../library')

%% Ex6(a) and Ex6(b)

gamma_power = -6:1:3;
gamma = 10.^gamma_power;

result_mse_train_100 = [];
result_mse_train_10 = [];

result_mse_valid_100 = [];
result_mse_valid_10 = [];

result_mse_test_100 = [];
result_mse_test_10 = [];

result_w_100 = [];
result_w_10 = [];


g_average_mse_test_100 = [];
g_average_mse_test_10 = [];

w_j = randn(10,1);
x_j = randn(600,10);
n_j = randn(600,1);

% Perform 5 fold cross validation
for j = 1:5


```

```
47      w_100 = [];
48      w_10 = [];
49
50      mse_train_100 = [];
51      mse_valid_100 = [];
52
53
54      mse_train_10 = [];
55      mse_valid_10 = [];
56
57
58      for i=1: size(gamma,2)
59          [result_w_100_j, result_mse_train_100_j,result_mse_valid_100_j] =
                calculateLSRex6(x_j,w_j,n_j,100,gamma(i),j);
60          w_100 = [w_100; result_w_100_j];
61          mse_train_100 = [mse_train_100, result_mse_train_100_j];
62          mse_valid_100 = [mse_valid_100, result_mse_valid_100_j];
63
64          [result_w_10_j, result_mse_train_10_j,result_mse_valid_10_j] =
                calculateLSRex6(x_j,w_j,n_j,10,gamma(i),j);
65          w_10 = [w_10; result_w_10_j];
66          mse_train_10 = [mse_train_10, result_mse_train_10_j];
67          mse_valid_10 = [mse_valid_10, result_mse_valid_10_j];
68
69      end
70      %Identify gamma that minimises MSE
71      [min_mse_100,idx_min_100] = min(mse_valid_100);
72      best_gamma_100(j,1) = gamma(idx_min_100);
73
74      [min_mse_10,idx_min_10] = min(mse_valid_10);
75      best_gamma_10(j,1) = gamma(idx_min_10);
76
77      result_w_100 = [result_w_100 ; w_100];
78      result_w_10 = [result_w_10 ; w_10];
79
80      result_mse_train_100 = [result_mse_train_100 ; mse_train_100];
81      result_mse_train_10 = [result_mse_train_10 ; mse_train_10];
82
83      result_mse_valid_100 = [result_mse_valid_100 ; mse_valid_100];
84      result_mse_valid_10 = [result_mse_valid_10 ; mse_valid_10];
85
86  end
87
88  mean_gamma_100 = mean(best_gamma_100)
89  mean_gamma_10 = mean(best_gamma_10)
90
91  % Average all dataset errors
92  average_mse_train_100 = mean(result_mse_train_100);
93  average_mse_valid_100 = mean(result_mse_valid_100);
```

```
94
95  average_mse_train_10 = mean(result_mse_train_10);
96  average_mse_valid_10 = mean(result_mse_valid_10);
97
98  mse_test_100 = [];
99  mse_test_10 =[];
100 % Calculate MSE for test set
101 for i=1: size(gamma,2)
102     [result_w_100_j, result_mse_train_100_j,result_mse_test_100_j] =
            calculateLSRex4(x_j,w_j,n_j,100,500,gamma(i));
103     mse_test_100 = [mse_test_100, result_mse_test_100_j];
104
105     [result_w_10_j, result_mse_train_10_j,result_mse_test_10_j] =
            calculateLSRex4(x_j,w_j,n_j,10,500,gamma(i));
106     mse_test_10 = [mse_test_10, result_mse_test_10_j];
107
108 end
109
110 % plots for mse vs gamma values for 100
111 figure;
112 plot(log(gamma),average_mse_train_100,'r-*');
113 hold on;
114 plot(log(gamma),average_mse_valid_100,'b-o');
115 plot(log(gamma),mse_test_100,'g-+');
116 grid on;
117 set(gcf, 'Color', 'w');
118 xlabel('log (\gamma)','FontSize',15);
119 ylabel('Mean Square Error','FontSize',15);
120 leg=legend('MSE_{avg train}','MSE_{avg cross-val}','MSE_{test}','Location','
        Best');
121 set(leg,'FontSize',15);
122 set(gca,'YMinorTick','on');
123 grid minor
124 axis tight;
125 print('ex6_cv_100','-depsc');
126 close all;
127
128 % plots for mse vs gamma values for 10
129 figure;
130 plot(log(gamma),average_mse_train_10,'r-*');
131 hold on;
132 plot(log(gamma),average_mse_valid_10,'b-o');
133 plot(log(gamma),mse_test_10,'g-+');
134 grid on;
135 set(gcf, 'Color', 'w');
136 xlabel('log (\gamma)','FontSize',12);
137 ylabel('Mean Square Error','FontSize',12);
138 leg=legend('MSE_{avg train}','MSE_{avg cross-val}','MSE_{test}','Location','
        Best');
```

```
139 set(leg,'FontSize',15);
140 set(gca,'YMinorTick','on');
141 grid minor
142 axis tight;
143 print('ex6_cv_10','-depsc');
144 close all;
```

## Exercise 7: Code

**ex7.m**

```
%-------------------------------------------------------------------------%
% Module: GI01 - Supervised Learning
% Assignment : Coursework 1
% Author : Russel Stuart Daries, Nitish Mutha
% Student ID: 16079408 ,15113106
% Question: 7
% Section: Part 1
% Description: Comparing tuning-methods for regulrisation parameters
% -------------------------------------------------------------------------%

%%
clear all
close all
clc

addpath('../library')

gamma_power = -6:1:3;
gamma = 10.^gamma_power;

%% Initialize variables of interest
test_4_100 = zeros(200,1);
test_4_10 = zeros(200,1);

test_5_100 = zeros(200,1);
test_5_10 = zeros(200,1);

test_6_100 = zeros(200,1);
test_6_10 = zeros(200,1);

all_gamma_4_100 = zeros(200,1);
all_gamma_4_10 = zeros(200,1);

all_gamma_5_100 = zeros(200,1);
all_gamma_5_10 = zeros(200,1);

all_gamma_6_100 = zeros(200,1);
all_gamma_6_10 = zeros(200,1);

%% Complete 200 independent trials
for j = 1:200

    w_j = randn(10,1);
    x_j = randn(600,10);
    n_j = randn(600,1);

```

```
47      %% Select best gamma by minimizing training error
48      [gamma_4_100,gamma_4_10,mse_test_4_100,mse_test_4_10] = ex7_4(x_j,w_j,n_j,
            gamma);
49      %% Select best gammma by minimizing validation set error
50      [gamma_5_100,gamma_5_10,mse_test_5_100,mse_test_5_10] = ex7_5(x_j,w_j,n_j,
            gamma);
51      %% Select best gamma using 5-fold cross-validation
52      [gamma_6_100,gamma_6_10,mse_test_6_100,mse_test_6_10] = ex7_6(x_j,w_j,n_j,
            gamma);
53
54      % Assigning resultant mse test values to their respective outputs
55      test_4_100(j) = mse_test_4_100;
56      test_4_10(j) = mse_test_4_10;
57
58      test_5_100(j) = mse_test_5_100;
59      test_5_10(j) = mse_test_5_10;
60
61      test_6_100(j) = mse_test_6_100;
62      test_6_10(j) = mse_test_6_10;
63
64      % Assignming optimal gamma values to vectors
65      all_gamma_4_100(j) = gamma_4_100;
66      all_gamma_4_10(j) = gamma_4_10;
67
68      all_gamma_5_100(j) = gamma_5_100;
69      all_gamma_5_10(j) = gamma_5_10;
70
71      all_gamma_6_100(j) = gamma_6_100;
72      all_gamma_6_10(j) = gamma_6_10;
73  end
74
75  %% Average MSE values
76  avg_test_4_100 = mean(test_4_100)
77  avg_test_4_10 = mean(test_4_10)
78
79  avg_test_5_100 = mean(test_5_100)
80  avg_test_5_10 = mean(test_5_10)
81
82  avg_test_6_100 = mean(test_6_100)
83  avg_test_6_10 = mean(test_6_10)
84
85  %% Average gamma values
86  avg_gamma_4_100 = mean(all_gamma_4_100)
87  avg_gamma_4_10 = mean(all_gamma_4_10)
88
89  avg_gamma_5_100 = mean(all_gamma_5_100)
90  avg_gamma_5_10 = mean(all_gamma_5_10)
91
92  avg_gamma_6_100 = mean(all_gamma_6_100)
```

```
93  avg_gamma_6_10 = mean(all_gamma_6_10)
94
95  %% Standard deviation calculation
96  sd_4_100 = standard_deviation(avg_test_4_100,test_4_100)
97  sd_4_10 = standard_deviation(avg_test_4_10,test_4_10)
98
99  sd_5_100 = standard_deviation(avg_test_5_100,test_5_100)
100 sd_5_10 = standard_deviation(avg_test_5_10,test_5_10)
101
102 sd_6_100 = standard_deviation(avg_test_6_100,test_6_100)
103 sd_6_10 = standard_deviation(avg_test_6_10,test_6_10)
104
105 %% plots
106 figure;
107 plot(test_4_100,'r*');
108 hold on;
109 plot(test_5_100,'b+')
110 hold on
111 plot(test_6_100,'go');
112 grid on;
113 set(gcf, 'Color', 'w');
114 xlabel('Trials','FontSize',15);
115 ylabel('Mean Square Error','FontSize',15);
116 leg=legend('MSE_{min-train}','MSE_{min-valid}','MSE_{min-cross}','Location','
        Best');
117 set(leg,'FontSize',15);
118 set(gca,'YMinorTick','on');
119 grid minor
120 axis tight;
121 print('ex7_mse_compare_tuning_100','-depsc');
122 close all;
123
124 figure;
125 plot(test_4_10,'r*');
126 hold on;
127 plot(test_5_10,'b+')
128 hold on
129 plot(test_6_10,'go');
130 grid on;
131 set(gcf, 'Color', 'w');
132 xlabel('Trials','FontSize',15);
133 ylabel('Mean Square Error','FontSize',15);
134 leg=legend('MSE_{min-train}','MSE_{min-valid}','MSE_{min-cross}','Location','
        Best');
135 set(leg,'FontSize',15);
136 set(gca,'YMinorTick','on');
137 grid minor
138 axis tight;
139 print('ex7_mse_compare_tuning_10','-depsc');
```

```
close all;

figure;
plot(all_gamma_4_10,'r*');
hold on;
plot(all_gamma_5_10,'b+')
hold on
plot(all_gamma_6_10,'go');
grid on;
set(gcf, 'Color', 'w');
xlabel('Trials','FontSize',15);
ylabel('Mean Square Error','FontSize',15);
leg=legend('MSE_{min-train}','MSE_{min-valid}','MSE_{min-cross}','Location','
    Best');
set(leg,'FontSize',15);
set(gca,'YMinorTick','on');
grid minor
axis tight;
print('ex7_mse_compare_tuning_gamma_10','-depsc');
close all;

figure;
plot(all_gamma_4_100,'r*');
hold on;
plot(all_gamma_5_100,'b+')
hold on
plot(all_gamma_6_100,'go');
grid on;
set(gcf, 'Color', 'w');
xlabel('Trials','FontSize',15);
ylabel('Mean Square Error','FontSize',15);
leg=legend('MSE_{min-train}','MSE_{min-valid}','MSE_{min-cross}','Location','
    Best');
set(leg,'FontSize',15);
set(gca,'YMinorTick','on');
grid minor
axis tight;
print('ex7_mse_compare_tuning_gamma_100','-depsc');
close all;
```

**ex7_4.m**

```
function [best_gamma_100,best_gamma_10,result_mse_test_100,result_mse_test_10]
    = ex7_4(x,w,n,gamma)

addpath('../library')

mse_train_100 = [];
mse_train_10 = [];
% Evaluate performance of test set
```

```
for i=1: size(gamma,2)
    [result_w_100_j, result_mse_train_100_j, result_mse_test_100_j] =
        calculateLSRex4(x,w,n,100,500,gamma(i));
    mse_train_100 = [mse_train_100, result_mse_train_100_j];

    [result_w_10_j, result_mse_train_10_j, result_mse_test_10_j] =
        calculateLSRex4(x,w,n,10,500,gamma(i));
    mse_train_10 = [mse_train_10, result_mse_train_10_j];

end
% Find the gamma that minimizes the MSE of training set
[min_train_error_100,min_idx_100] = min(mse_train_100);
best_gamma_100 = gamma(min_idx_100);

[min_train_error_10,min_idx_10] = min(mse_train_10);
best_gamma_10 = gamma(min_idx_10);

% Evaluate performance on test vector
[result_w_100, result_mse_train_100, result_mse_test_100] = calculateLSRex4(x,
    w,n,100,500,best_gamma_100);
[result_w_10, result_mse_train_10, result_mse_test_10] = calculateLSRex4(x,w,n
    ,10,500,best_gamma_10);
```

**ex7_5.m**

```
function [best_gamma_100,best_gamma_10,result_mse_test_100,result_mse_test_10]
    = ex7_5(x,w,n,gamma)

addpath('../library')

mse_valid_100 = [];
mse_valid_10 = [];
% Evaluate performance of training and validation sets of data
for i=1: size(gamma,2)
    [result_w_100_j, result_mse_train_100_j,result_mse_valid_100_j,
        result_mse_test_100_j] = calculateLSRex5(x,w,n,80,20,500,gamma(i));
    mse_valid_100 = [mse_valid_100, result_mse_valid_100_j];

    [result_w_10_j, result_mse_train_10_j,result_mse_valid_10_j,
        result_mse_test_10_j] = calculateLSRex5(x,w,n,8,2,500,gamma(i));
    mse_valid_10 = [mse_valid_10, result_mse_valid_10_j];

end
% Find the gamma that minimizes the MSE of validation set
[min_mse_100,idx_min_100] = min(mse_valid_100);
best_gamma_100 = gamma(idx_min_100);

[min_mse_10,idx_min_10] = min(mse_valid_10);
best_gamma_10 = gamma(idx_min_10);

```

```
% Evaluate performance on test vector
[w_test_100, result_mse_test_100] = calculateLSRTest(x,w,n,80,20,500,
    best_gamma_100);
[w_test_10, result_mse_test_10] = calculateLSRTest(x,w,n,8,2,500,best_gamma_10
    );
```

**ex7_6.m**

```
function [best_gamma_100,best_gamma_10,result_mse_test_100,result_mse_test_10]
    = ex7_6(x,w,n,gamma)

addpath('../library')

result_mse_valid_100 = [];
result_mse_valid_10 = [];

% Perform 5-fold cross-validation
for j = 1:5

    mse_valid_100 = [];
    mse_valid_10 = [];

    for i=1: size(gamma,2)
        [result_w_100_j, result_mse_train_100_j,result_mse_valid_100_j] =
            calculateLSRex6(x,w,n,100,gamma(i),j);
        mse_valid_100 = [mse_valid_100, result_mse_valid_100_j];

        [result_w_10_j, result_mse_train_10_j,result_mse_valid_10_j] =
            calculateLSRex6(x,w,n,10,gamma(i),j);
        mse_valid_10 = [mse_valid_10, result_mse_valid_10_j];
    end

    result_mse_valid_100 = [result_mse_valid_100 ; mse_valid_100];
    result_mse_valid_10 = [result_mse_valid_10 ; mse_valid_10];
end

% Find the gamma that minimizes the mean cross-validation MSE
[min_mse_100,min_idx_100] = min(mean(result_mse_valid_100));
best_gamma_100 = gamma(min_idx_100);

[min_mse_10,min_idx_10] = min(mean(result_mse_valid_10));
best_gamma_10 = gamma(min_idx_10);

% Evaluate performance on test vector
[result_w_100, result_mse_train_100,result_mse_test_100] = calculateLSRex4(x,w
    ,n,100,500,best_gamma_100);
[result_w_10, result_mse_train_10,result_mse_test_10] = calculateLSRex4(x,w,n
    ,10,500,best_gamma_10);
```

## Exercise 9: Code

**Ex9_a.m**

```
%-------------------------------------------------------------------------%
% Module: GI01 - Supervised Learning
% Assignment : Coursework 1
% Author : Russel Stuart Daries, Nitish Mutha
% Student ID: 16079408 ,15113106
% Question: 9a
% Section: Part 1
% Description: Baseline versus full linear regression.
% -------------------------------------------------------------------------%

close all;
clear all;
addpath('../library')

load('boston.mat')

%% Ex9(a)

% 20 trails
for j=1:20

    data_train = datasample(boston,round(2/3*size(boston,1)),1,'Replace',false);
    % Stored left over rows in the test set 1/3
    data_test = setdiff(boston, data_train, 'rows');

    x_train_v = ones(round(2/3*size(boston,1)),1);
    y_train = data_train(:,14);

    x_test_v = ones(round(1/3*size(boston,1)),1);
    y_test = data_test(:,14);

    % Calculate weight vector w
    w_star = learnModel(x_train_v, y_train);
    % Calculate MSE
    mse_train(j) = meanSquareError(w_star,x_train_v,y_train,size(x_train_v,1));
    mse_test(j) = meanSquareError(w_star,x_test_v,y_test,size(x_test_v,1));

end
% Calculate mean and standard deviation of trian and test set.
avg_mse_train = mean(mse_train)
std_train = std(mse_train)

avg_mse_test = mean(mse_test)
std_test = std(mse_test)
```

```
45
46 mse = [avg_mse_train; avg_mse_test]';
```

**Ex9_b.m**

```
1  %-------------------------------------------------------------------------%
2  % Module: GI01 - Supervised Learning
3  % Assignment : Coursework 1
4  % Author : Russel Stuart Daries, Nitish Mutha
5  % Student ID: 16079408 ,15113106
6  % Question: 9b
7  % Section: Part 1
8  % Description: Baseline versus full linear regression.
9  % -------------------------------------------------------------------------%
10
11 close all;
12 clear all;
13 addpath('../library')
14
15 load('boston.mat')
16
17 %% Ex 9(b)
18
19 % 20 trials
20 for j=1:20
21
22     % Radomly withdraw 2/3 data from Boston for training samples
23     data_train = datasample(boston,round(2/3*size(boston,1)),1,'Replace',false
           );
24     % Stored left over rows in the test set 1/3
25     data_test = setdiff(boston, data_train, 'rows');
26
27     % Segment the columns for training attributes
28     x_train = data_train(:,1:13);
29     % Take the last column as the true value we are trying to predict
30     y_train = data_train(:,14);
31     % Perform same mapping for test set
32     x_test = data_test(:,1:13);
33     y_test = data_test(:,14);
34
35     % Loop through each individual attribute with bias and try to predict
36     % outcome
37     for i=1:size(x_train,2)
38         x_train_v = [x_train(:,i), ones(size(x_train,1),1)];
39         w_star = learnModel(x_train_v, y_train);
40         mse_train(j,i) = meanSquareError(w_star,x_train_v,y_train,size(
               x_train_v,1));
41
42         x_test_v = [x_test(:,i), ones(size(x_test,1),1)];
```

```
43          mse_test(j,i) = meanSquareError(w_star,x_test_v,y_test,size(x_test_v
                ,1));
44
45      end
46  end
47
48  % Calculate mean and standard deviation for each of the 20 trials for the
49  % 13 attributes for train and test set
50  avg_mse_train = mean(mse_train)
51  std_train = std(mse_train)
52
53  avg_mse_test = mean(mse_test)
54  std_test = std(mse_test)
55
56  mse = [avg_mse_train; avg_mse_test]';
57
58  %% plots
59  figure;
60  plot(avg_mse_train,'r+');
61  hold on
62  plot(avg_mse_test,'bs');
63  set(gcf, 'Color', 'w');
64  xlabel('Attributes','FontSize',15);
65  ylabel('Mean Square Error','FontSize',15);
66  leg=legend('Train','Test','Location','Best');
67  set(leg,'FontSize',15);
68  set(gca,'YMinorTick','on');
69  grid minor
70  axis tight;
71  print('ex9b_avg_mse','-depsc');
72  close all;
73
74  figure;
75  boxplot(mse_train);
76  set(gcf, 'Color', 'w');
77  xlabel('Attributes','FontSize',15);
78  ylabel('Mean Square Error','FontSize',15);
79  leg=legend('Train','Test','Location','Best');
80  set(leg,'FontSize',15);
81  set(gca,'YMinorTick','on');
82  grid minor
83  axis tight;
84  print('ex9b_mse_box_train','-depsc');
85  close all;
86
87  figure;
88  boxplot(mse_test);
89  set(gcf, 'Color', 'w');
90  xlabel('Attributes','FontSize',15);
```

```
ylabel('Mean Square Error','FontSize',15);
leg=legend('Train','Test','Location','Best');
set(leg,'FontSize',15);
set(gca,'YMinorTick','on');
grid minor
axis tight;
print('ex9b_mse_box_test','-depsc');
close all;
```

**Ex9_c.m**

```
%-------------------------------------------------------------------------%
% Module: GI01 - Supervised Learning
% Assignment : Coursework 1
% Author : Russel Stuart Daries, Nitish Mutha
% Student ID: 16079408 ,15113106
% Question: 9c
% Section: Part 1
% Description: Baseline versus full linear regression.
% -------------------------------------------------------------------------%


close all;
clear all;
addpath('../library')

load('boston.mat')

%% Ex9(c)

% 20 trials
for j=1:20

    % Radomly withdraw 2/3 data from Boston for training samples
    data_train = datasample(boston,round(2/3*size(boston,1)),1,'Replace',false
        );
    % Stored left over rows in the test set 1/3
    data_test = setdiff(boston, data_train, 'rows');
    % Segment the columns for training attributes
    x_train = data_train(:,1:13);
    % Take the last column as the true value we are trying to predict
    y_train = data_train(:,14);

    x_test = data_test(:,1:13);
    y_test = data_test(:,14);

    % Train the model vector w
    x_train_v = [x_train, ones(size(x_train,1),1)];
    w_star = learnModel(x_train_v, y_train);
```

```
38      mse_train(j) = meanSquareError(w_star,x_train_v,y_train,size(x_train_v,1))
            ;
39
40      % Test the learnt vector w
41      x_test_v = [x_test, ones(size(x_test,1),1)];
42      mse_test(j) = meanSquareError(w_star,x_test_v,y_test,size(x_test_v,1));
43
44  end
45
46  avg_mse_train = mean(mse_train)
47  std_train = std(mse_train)
48
49  avg_mse_test = mean(mse_test)
50  std_test = std(mse_test)
51
52  mse = [avg_mse_train; avg_mse_test]';
53
54  %% plots
55  figure;
56  boxplot(mse_train);
57  set(gcf, 'Color', 'w');
58  % xlabel('Attributes','FontSize',15);
59  ylabel('Mean Square Error','FontSize',15);
60  leg=legend('Train','Test','Location','Best');
61  set(leg,'FontSize',15);
62  set(gca,'YMinorTick','on');
63  grid minor
64  axis tight;
65  print('ex9c_mse_box_train','-depsc');
66  close all;
67
68  figure;
69  boxplot(mse_test);
70  set(gcf, 'Color', 'w');
71  % xlabel('Attributes','FontSize',15);
72  ylabel('Mean Square Error','FontSize',15);
73  leg=legend('Train','Test','Location','Best');
74  set(leg,'FontSize',15);
75  set(gca,'YMinorTick','on');
76  grid minor
77  axis tight;
78  print('ex9c_mse_box_test','-depsc');
79  close all;
```

## Exercise 10: Code

### Ex10_RR.m

```
%-------------------------------------------------------------------------%
% Module: GI01 - Supervised Learning
% Assignment : Coursework 1
% Author : Russel Stuart Daries, Nitish Mutha
% Student ID: 16079408 ,15113106
% Question: 10
% Section: Part 1
% Description: Kernel Ridge Regression.
% -------------------------------------------------------------------------%

close all;
clear;
clc;

addpath('../library')

load('boston.mat');

%% Ex10(c) and (d)

% Create gamma vector
gamma_exp = -40:-26;
gamma = 2.^gamma_exp;

% Create sigma vector
sigma_exp = 7:0.5:13;
sigma = 2.^sigma_exp;

% 20 trials
for j=1:20

    % Random sampling of data
    data_train = datasample(boston,round(2/3*size(boston,1)),1,'Replace',false
        );
    data_test = setdiff(boston, data_train, 'rows');

    % Training set
    x_train_j = data_train(:,1:13);
    y_train_j = data_train(:,14);

    %Test set
    x_test_j = data_test(:,1:13);
    y_test_j = data_test(:,14);

    %Dimension of training set x values
    [dim1, dim2] = size(x_train_j);
```

```
46
47      k_mse = [];
48      % Perform 5-fold cross validation
49      for k=1:5
50
51          k_width = round(dim1/5);
52
53          % Segment x-train data for training
54          x_train = cat( 1, x_train_j(1:(k-1)*k_width, :), x_train_j(k*k_width
                +1 :end,:));
55          % Segment x-train data for validation
56          x_valid = x_train_j((k-1)*k_width + 1:k*k_width, :);
57
58          % Segment x-train data for training
59          y_train = cat( 1, y_train_j(1:(k-1)*k_width, :), y_train_j(k*k_width
                +1 :end,:));
60          % Segment x-train data for validation
61          y_valid = y_train_j((k-1)*k_width + 1:k*k_width, :);
62
63          [train_dim1, train_dim2] = size(x_train);
64
65          K = zeros(dim1,dim1,size(sigma,2),1);
66          % From Gram matrix
67          for s=1:size(sigma,2)
68              K(:,:,s) = calK(x_train,x_valid,sigma(s));
69          end
70
71          %Vary selection of gamma
72          for g=1:size(gamma,2)
73              %Vary selection of sigma
74              for ss=1:size(sigma,2)
75                  current_k = K(:,:,ss);
76                  alpha = kridgereg(current_k(1:train_dim1,1:train_dim1),y_train
                        ,gamma(g));
77                  %Calculate resultant mse for each fold,each gamma and each
78                  %sigma selection and store it into 3D variable
79                  k_mse(g,ss,k) = dualcost(current_k(train_dim1+1:end,1:
                        train_dim1) ,y_valid,alpha);
80              end
81          end
82      end
83      % Calculating the mean MSE over the 5 folds of data
84      mean_k_mse = mean(k_mse,3);
85
86      %% Ex10 (c)
87      %plot the cross validation error (once as asked in 10_c)
88      if(j == 1)
89          surf(sigma',gamma',mean_k_mse)
90          colorbar;
```

```
91          set(gcf, 'Color', 'w');
92          xlabel('\sigma','FontSize',15);
93          ylabel('\gamma','FontSize',15);
94          zlabel('MSE_{cross-validaion}','FontSize',15);
95          set(gca,'YMinorTick','on');
96          grid minor
97          axis tight;
98          print('ex10_cv_mse','-depsc');
99          close all;
100     end
101
102     % Find the combination of gamma and sigma that produces smalled mse
103     [min_gamma_indx, min_sigma_indx] = find(mean_k_mse == min(min(mean_k_mse))
            );
104
105     % Using optimal gamma and sigma values to calculate optimal alpha
106     % values
107     K_optimal = calK(x_train_j,x_test_j,sigma(min_sigma_indx));
108     alpha_optimal = kridgereg(K_optimal(1:dim1,1:dim1),y_train_j,gamma(
            min_gamma_indx));
109
110     %training error
111     mse_training(j) = dualcost(K_optimal(1:dim1,1:dim1) ,y_train_j,
            alpha_optimal);
112
113     %testing error (Using K_test from lower left hand corner of gram matrix)
114     mse_testing(j) = dualcost(K_optimal(dim1+1:end,1:dim1) ,y_test_j,
            alpha_optimal);
115
116 end
117 % Calculate the mean and standard deviation of resultant mse of training
118 % and test set
119 avg_mse_training = mean(mse_training)
120 avg_mse_testing = mean(mse_testing)
121
122 std_mse_training = std(mse_training)
123 std_mse_testing = std(mse_testing)
```

**dualcost.m**

```
1 function mse = dualcost(K,y,a)
2
3 mse = (K*a - y)'*(K*a - y) / size(y,1);
```

**kridgereg.m**

```
1 function a = kridgereg(K, y, gamma)
2
3 a = (K + (gamma*size(y,1)*eye(size(y,1))) ) \ y;
```

## Question 4 (Part II): Code

**question4_1.m (least squares)**

```
%-------------------------------------------------------------------------%
% Module: GI01 - Supervised Learning
% Assignment : Coursework 1
% Author : Russel Daries, Nitish Mutha
% Student ID: 16079408, 15113106
% Question: 4 (Least squares)
% Section: Part 2
% Description: Least Squares sample complexity
% -------------------------------------------------------------------------%

clear all
close all

% N feaures
N = 100;
% M samples
M = 100;
testIterations = 100;
testSize = 500;

avg_missclassified = zeros(testIterations,N);
m_error = zeros(N,1);
%least square
for n = 1:N
    for m = 1:M

        %train
        X = 2*randi([0 1], m, n) - 1;
        y = X(:,1);
        W = pinv(X)*y;

        %test
        missClassified = zeros(1,testIterations);
        for i = 1:testIterations
            X_test = 2*randi([0 1], testSize, n) - 1;
            y_test = X_test(:,1);
            y_predict = sign(X_test*W);
            missClassified(1,i) = sum(y_predict~=y_test);
        end
        percentError = mean(missClassified*100/testSize);
        if percentError <= 10
            m_error(n) = m;
            avg_missclassified(:,n) = missClassified';
            break;
        end
    end
```

```
end

figure;
plot(m_error);
set(gcf, 'Color', 'w');
xlabel('n features','FontSize',15);
ylabel('m samples','FontSize',15);
leg=legend('m sample < 10% generalisation error','Location','Best');
set(leg,'FontSize',15);
set(gca,'YMinorTick','on');
grid minor
axis tight;
print('ex4_least_sq','-depsc');
```

**question4_2.m (perceptron)**

```
%-------------------------------------------------------------------------%
% Module: GI01 - Supervised Learning
% Assignment : Coursework 1
% Author : Russel Daries, Nitish Mutha
% Student ID: 16079408, 15113106
% Question: 4 (Perceptron)
% Section: Part 2
% Description: Perceptron sample complexity
% ------------------------------------------------------------------------%

clear all
close all

N = 100;
M = 500;
testIterations = 100;
testSize = 500;

m_error = zeros(N,1);
%perceptron
for n = 1:N
    for m = 1:M

        %train
        X = 2*randi([0 1], m, n) - 1;
        y = X(:,1);
        W = zeros(1,n);
        for j=1:size(X,1)
            y_pred = sign(X(j,:)*W');
            if y_pred*y(j,:) <= 0
                W = W + y(j,:)*X(j,:);
            end
        end
```

```
34
35          %test
36          missClassified = zeros(1,testIterations);
37          for i = 1:testIterations
38              X_test = 2*randi([0 1], testSize, n) - 1;
39              y_test = X_test(:,1);
40              y_predict = sign(X_test*W');
41              missClassified(1,i) = sum(y_predict~=y_test);
42          end
43          percentError = mean(missClassified)*100/testSize;
44          if percentError <= 10
45              m_error(n) = m;
46              break;
47          end
48      end
49  end
50
51  figure;
52  plot(m_error);
53  set(gcf, 'Color', 'w');
54  xlabel('n features','FontSize',15);
55  ylabel('m samples','FontSize',15);
56  leg=legend('m sample < 10% generalisation error','Location','Best');
57  set(leg,'FontSize',15);
58  set(gca,'YMinorTick','on');
59  grid minor
60  axis tight;
61  print('ex4_perceptron','-depsc');
```

**question4_3.m (winnow)**

```
1  %-------------------------------------------------------------------------%
2  % Module: GI01 - Supervised Learning
3  % Assignment : Coursework 1
4  % Author : Russel Daries, Nitish Mutha
5  % Student ID: 16079408, 15113106
6  % Question: 4 (Winnow)
7  % Section: Part 2
8  % Description: Winnow sample complexity
9  %-------------------------------------------------------------------------%
10
11 clear all
12 close all
13
14 N = 100;
15 M = 1000;
16 testIterations = 300;
17 testSize = 500;
18
```

```
m_error = zeros(N,1);
%winnow
for n = 1:N
    for m = 1:M
        X = randi([0 1], m, n);
        y = X(:,1);

        %train
        W = ones(1,n);
        for j=1:size(X,1)
            if X(j,:)*W' < n
                y_pred = 0;
            else
                y_pred = 1;
            end
            if sum(y_pred~=y(j,:)) > 0
                W = W .* power(2,(y(j,:)-y_pred)*X(j,:));
            end
        end

        %test
        missClassified = zeros(1,testIterations);
        for i = 1:testIterations
            X_test = randi([0 1], testSize, n);
            y_test = X_test(:,1);
            y_predict = X_test*W';
            missClassified(1,i) = sum(y_predict>=n ~= y_test);
        end
        percentError = mean(missClassified)*100/testSize;
        if percentError <= 10
            m_error(n) = m;
            break;
        end
    end
end

figure;
plot(m_error);
set(gcf, 'Color', 'w');
xlabel('n features','FontSize',15);
ylabel('m samples','FontSize',15);
leg=legend('m sample < 10% genralisation error','Location','Best');
set(leg,'FontSize',15);
set(gca,'YMinorTick','on');
grid minor
axis tight;
print('ex4_winnow','-depsc');
```

**question4_4.m (1NN)**

```
%-------------------------------------------------------------------------%
% Module: GI01 - Supervised Learning
% Assignment : Coursework 1
% Author : Russel Daries, Nitish Mutha
% Student ID: 16079408, 15113106
% Question: 4 (1 nearest neighbour)
% Section: Part 2
% Description: 1 nearest neighbour sample complexity
% -------------------------------------------------------------------------%

clear all;
close all;

% N features
N = 100;
% M samples
M = 10000;
testIterations = 100;

avg_missclassified = zeros(N, testIterations);
m_error = zeros(N,1);
%1-NN
for n = 1:N
    for m = 1:M
        X = 2*randi([0 1], m, n) - 1;
        y = X(:,1);

        %test
        missClassified = zeros(1,testIterations);
        for i = 1:testIterations
            X_test = 2*randi([0 1], 100, n) - 1;
            y_test = X_test(:,1);
            closest_Id = 0;
            y_predict = zeros(size(X_test,1),1);
            for j = 1:size(X_test,1)
                dist = sqrt(sum((X - X_test(j,:)).^2,2));
                [val, closest_Id] = min(dist);
                y_predict(j,:) = X(closest_Id,1);
            end

            missClassified(1,i) = sum(y_predict~=y_test);
        end
        percentError = mean(missClassified*100/(100));
        if percentError <= 10
            m_error(n) = m;
            break;
        end
```

```
48      end
49  end
50
51  figure;
52  plot(m_error);
53  set(gcf, 'Color', 'w');
54  xlabel('n features','FontSize',15);
55  ylabel('m samples','FontSize',15);
56  leg=legend('m sample < 10% generalisation error','Location','Best');
57  set(leg,'FontSize',15);
58  set(gca,'YMinorTick','on');
59  grid minor
60  axis tight;
61  print('ex4_1nn-final','-depsc');
```

## Library Functions

Following are the list of API's used in multiple exercises.

### calculateLSR.m

```
function lsr = calculateLSR(x,w,n,train_len,test_len)
% Function to calculate y_true, weight vector (w) and resultant MSE for
% train and test vectors.

y = x*w + n;

x_train = x(1:train_len);
x_test = x((length(x)-test_len)+1 : end);

y_train = y(1:train_len);
y_test = y(length(x)-test_len+1 : end);

w_estimator = (x_train'*x_train)\(x_train'*y_train);

mse_train = meanSquareError(w_estimator,x_train,y_train,train_len);
mse_test = meanSquareError(w_estimator,x_test,y_test,test_len);

lsr = [w_estimator, mse_train, mse_test];

end
```

### calculateLSRex2.m

```
function [w_estimator, mse_train, mse_test] = calculateLSRex2(x,w,n,train_len,
    test_len)

y = x*w + n;

% Data segmentation and slicing
x_train = x(1:train_len, :);
x_test = x((length(x)-test_len)+1 : end, :);

y_train = y(1:train_len , :);
y_test = y(length(x)-test_len+1 : end, :);

w_estimator = (x_train'*x_train)\(x_train'*y_train);

% Compute MSE for train and test sets
mse_train = meanSquareError(w_estimator,x_train,y_train,train_len);
mse_test = meanSquareError(w_estimator,x_test,y_test,test_len);


```

```
19 end
```

**calculateLSRex4.m**

```
1  function [w_estimator, mse_train, mse_test] = calculateLSRex4(x,w,n,train_len,
       test_len, gamma)
2
3  y = x*w + n;
4
5  % Data segmentation and slicing
6  x_train = x(1:train_len, :);
7  x_test = x((length(x)-test_len)+1 : end, :);
8
9  y_train = y(1:train_len , :);
10 y_test = y(length(x)-test_len+1 : end, :);
11
12 % Compute weight vectors
13 w_estimator = (x_train'*x_train + gamma*train_len*eye(size(w,1)))\(x_train'*
       y_train);
14
15 % Compute MSE for train and test sets
16 mse_train = meanSquareError(w_estimator,x_train,y_train,train_len);
17 mse_test = meanSquareError(w_estimator,x_test,y_test,test_len);
18
19
20 end
```

**calculateLSRex5.m**

```
1  function [w_estimator, mse_train, mse_valid, mse_test] = calculateLSRex5(x,w,n
       ,train_len,validation_len,test_len, gamma)
2  % Create y vector
3  y = x*w + n;
4
5  %Segment data sets
6  % Training data
7  x_train = x(1:train_len, :);
8  % Validation set
9  x_valid = x(train_len+1 : train_len+1+validation_len, :);
10 % Test Set
11 x_test = x((length(x)-test_len)+1 : end, :);
12
13 % Training set
14 y_train = y(1:train_len , :);
15 % Validation set
16 y_valid = y(train_len+1 : train_len+1+validation_len, :);
17 % Test set
18 y_test = y(length(x)-test_len+1 : end, :);
```

```
19
20 w_estimator = (x_train'*x_train + gamma*train_len*eye(size(w,1)))\(x_train'*
       y_train);
21
22 % Calculation for MSE of each data set
23 mse_train = meanSquareError(w_estimator,x_train,y_train,train_len);
24 mse_valid = meanSquareError(w_estimator,x_valid,y_valid,validation_len);
25 mse_test = meanSquareError(w_estimator,x_test,y_test,test_len);
26
27
28 end
```

**calculateLSRex6.m**

```
1 function [w_estimator, mse_train, mse_valid] = calculateLSRex6(x,w,n,train_len
      ,gamma, k)
2
3 y = x*w + n;
4
5 x = x(1:train_len, :);
6 y = y(1:train_len, :);
7
8 [dim1, dim2] = size(x);
9
10 k_width = dim1/5;
11
12 % K-Fold Slicing and segmentation
13 x_valid = x((k-1)*k_width + 1:k*k_width, :);
14 x_train = cat( 1, x(1:(k-1)*k_width, :), x(k*k_width +1 :end,:));
15
16 y_valid = y((k-1)*k_width + 1:k*k_width, :);
17 y_train = cat( 1, y(1:(k-1)*k_width, :), y(k*k_width +1 :end,:));
18
19 % Compute weight vector
20 w_estimator = (x_train'*x_train + gamma*train_len*eye(size(w,1)))\(x_train'*
       y_train);
21
22 % Compute MSE for train and test sets
23 mse_train = meanSquareError(w_estimator,x_train,y_train,(train_len-k_width));
24 mse_valid = meanSquareError(w_estimator,x_valid,y_valid,k_width);
25
26
27 end
```

**calculateLSRTest.m**

```
1 function [w_estimator, mse_test] = calculateLSRTest(x,w,n,train_len,
      validation_len,test_len, gamma)
```

```
2
3  y = x*w + n;
4
5  x_train = x(1:(train_len+validation_len), :);
6  x_valid = x(train_len+1 : train_len+1+validation_len, :);
7  x_test = x((length(x)-test_len)+1 : end, :);
8
9  y_train = y(1:(train_len+validation_len) , :);
10 y_valid = y(train_len+1 : train_len+1+validation_len, :);
11 y_test = y(length(x)-test_len+1 : end, :);
12
13 w_estimator = (x_train'*x_train + gamma*train_len*eye(size(w,1)))\(x_train'*
       y_train);
14
15 %mse_train = meanSquareError(w_estimator,x_train,y_train,train_len);
16 %mse_valid = meanSquareError(w_estimator,x_valid,y_valid,validation_len);
17 mse_test = meanSquareError(w_estimator,x_test,y_test,test_len);
18
19
20 end
```

**meanSquareError.m**

```
1  function mse = meanSquareError(w, x, y, len)
2      %Compute MSE for input vectors
3      mse = (1/len) * sum(power((x*w - y),2));
4      %mse = (1/len) * ((w'*x'*x*w) - (2*y'*x*w) + (y'*y));
5  end
```

**standard_deviation.m**

```
1  function sd = standard_deviation(avg_error,test_errors)
2  %Compute the standard deviation of inputs
3
4  [dim1, dim2] = size(test_errors);
5
6  sum_of_sq = 0;
7  for i=1:dim1
8      sum_of_sq = sum_of_sq + power((test_errors(i) - avg_error),2);
9  end
10 %Output the resultant standard deviation
11 sd = sqrt(sum_of_sq / dim1);
```

**calK.m**

```
1  function K = calK(x_train,x_test,sigma)
2
3  x = [x_train;x_test];
```

```
4
5 for ii = 1 : size(x,1)
6     for jj = 1 : size(x,1)
7         K(ii,jj) = exp(-(norm(x(ii,:)-x(jj,:),2))^2/(2*sigma^2));
8     end
9 end
```

**learnModel.m**

```
1 function w_estimator = learnModel(x,y)
2
3 w_estimator = (x'*x)\(x'*y);
4
5 end
```

# References

[1] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 1 ed., 2000.

[2] MATHWORKS, *MATLAB Matrix Inverse*, 2016 (accessed January 7, 2017).

[3] U. of Pennsylania, *CIS520:Machine Learning*, 2016 (accessed January 5, 2017).

[4] M. Herbster, "Online learning," October 2016.

[5] M. Collins, "Convergence proof of perceptron algorithm," 2010. University of Columbia.

[6] A. Ng, "The perceptron and large margin classifiers," 2003. Slides of lecture CSS 229, Stanford University.

[7] V. N. Vapnik, *Statistical learning theory*. Adaptive and learning systems for signal processing, communications, and control, New York: Wiley, 1998.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2009.