Supervised Learning (COMP0078) Coursework 2

Student No. 22204524

December 15, 2022

PART I

Kernel perceptron (Handwritten Digit Classification)

For the question 1 – question 4, the One versus Rest (OvR) method is implemented to classify the digit pictures. According to the question, there are 10 classes (0-9) for the OvR method. The OvR algorithm would conduct binary perceptron 10 times for each data then choose the finalized label with the largest possibility. All value of epoch is 30. The whole algorithm could be displayed as follow.

Algorithm 1 One versus Rest generalized Kernel Perceptron

Input: $(x_1, y_1), \dots, (x_m, y_m) \in R^n \times \{-1, 1\}$
Kernel matrix K using the kernel function: $K_{ij} = K(x_i, x_j)$
k Classifiers, $C_1, \dots, C_k$
Ture class y $\in \{0, \dots, 9\}$
Initialed alpha $\alpha_1^{(k)} = 0$ for all k
Output: predict class $\hat{y}$

for *i in range(k)* **do**
  if *i == y* **then**
    $y_{current} = 1$
  else
    $y_{current} = -1$
  Prediction: $C_i(x) = K[i] * \alpha$
  if $sign(C_i(x)) \neq y_{current}$ **then**
    $\alpha[i] = \alpha[i] - sign(\hat{y})$
  else
    max value = $sign(\hat{y})$
    max index = **i**
$\hat{y}$ = max index
return $\hat{y}$

When the dataset is received, the 10 times binary perceptron will implement for each data, which means the data with one specified label would be positive data and others would be treated as negative data. If the prediction is wrong, the alpha would be updated. After the 10 times binary perceptron, the value of max index would be the

final predicted value for the data.

Basic Results

Perform 20 runs for d = 1, … , 7, each run should randomly split into 80% train and 20% test. Report the mean test and train error rates as well as standard deviations. The results could be shown in following table.

Degree d	Training error (%)	Test error (%)
1	8.674 ± 0.794	8.132 ± 0.673
2	5.200 ± 0.577	6.140 ± 0.630
3	4.888 ± 0.310	6.409 ± 0.729
4	4.231 ± 0.343	6.280 ± 0.632
5	3.241 ± 0.316	6.164 ± 0.562
6	2.215 ± 0.351	6.105 ± 0.631
7	1.402 ± 0.794	5.712 ± 0.524

Table 1: training error and test error rates for degree d

When using the 80% training data split from within to perform 5-fold cross-validation to select the “best” parameter d* then retrain on full 80% training set using d* and then record the test errors on the remaining 20%.

The best d per run	Test error rate
5	2.845
4	3.065
4	3.172
4	2.097
4	2.903
4	2.849
6	2.957
4	2.796
5	2.903
4	3.280
5	3.387
5	2.151
5	3.065
5	2.957
4	2.903
4	2.850
5	2.634
7	2.580
6	2.580
5	2.849

Table 2: Optimal d and test error rate for each d

Confusion matrix: Perform 20 runs: when using the 80% training data split that further to perform 5-fold cross-validation to select the "best" parameter d* retrain on the full "80%" training set using d* and then produce a confusion matrix. The mean value for 20 runs could be shown as below.

	0	1	2	3	4	5	6	7	8	9
0	0	0.0445	0.100	0.125	0.087	0.179	0.226	0.075	0.111	0.052
1	0.025	0	0.075	0	0.375	0.025	0.192	0.108	0.067	0.033
2	0.139	0.029	0	0.2025	0.270	0.01	0.06	0.215	0.074	0
3	0.068	0.034	0.139	0	0.010	0.387	0	0.123	0.196	0.043
4	0.017	0.138	0.193	0.024	0	0.074	0.209	0.111	0.028	0.206
5	0.165	0.044	0.095	0.221	0.136	0	0.165	0.023	0.099	0.053
6	0.266	0.123	0.136	0	0.188	0.117	0	0	0.103	0.018
7	0.008	0.101	0.195	0.046	0.239	0.045	0	0	0.082	0.284
8	0.182	0.074	0.111	0.202	0.099	0.221	0.013	0.061	0	0.036
9	0.122	0	0.053	0.018	0.332	0.044	0.011	0.312	0.058	0

Table 3: mean values of confusion matrix for 20 runs

The standard deviation of the confusion matrix could be shown as below.

	0	1	2	3	4	5	6	7	8	9
0	0	0.095	0.157	0.139	0.128	0.262	0.281	0.124	0.163	0.132
1	0.109	0	0.238	0	0.404	0.109	0.362	0.192	0.162	0.1
2	0.136	0.061	0	0.226	0.232	0.044	0.103	0.193	0.107	0
3	0.102	0.066	0.137	0	0.030	0.177	0	0.191	0.156	0.091
4	0.052	0.146	0.153	0.059	0	0.108	0.170	0.167	0.057	0.177
5	0.130	0.069	0.124	0.181	0.203	0	0.141	0.055	0.085	0.075
6	0.193	0.160	0.232	0	0.174	0.157	0	0	0.161	0.055
7	0.036	0.150	0.187	0.083	0.173	0.091	0	0	0.118	0.186
8	0.138	0.080	0.120	0.137	0.095	0.145	0.040	0.083	0	0.057
9	0.163	0	0.099	0.058	0.276	0.097	0.035	0.212	0.111	0

Table 4: standard deviation of the confusion matrix for 20 run

Within the dataset relative to your experiments there will be five hardest to predict correctly "pixelated images". Print out the visualization of these five digits along with their labels.

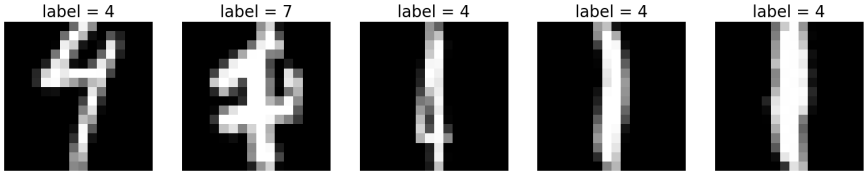

Figure 1: hardest 5 pixelated images

According to the five images, they may be incorrectly labelled of written because it is obvious that the shapes of the writing do not fit the labels.

One versus Rest method with Gaussian kernel

For the parameters, the default parameters are not suitable because of the high test error rate. After serval experiments, the parameters from 3^{-1} to 3^{-7} are selected for the OvR method with Gaussian kernel method. The basic results could be shown as follow:

Degree d	Training error (%)	Test error (%)
3^{-1}	0.101 ± 0.427	7.742 ± 0
3^{-2}	0.001 ± 0.004	6.409 ± 0.047
3^{-3}	0.007 ± 0.011	4.043 ± 0.040
3^{-4}	0.097 ± 0.281	4.027 ± 0.061
3^{-5}	0.393 ± 0.823	4.113 ± 0.237
3^{-6}	1.951 ± 1.861	5.255 ± 0.971
3^{-7}	7.497 ± 1.820	8.734 ± 2.857

Table 5: training and testing error rates for OvR method with Gaussian kernel

According to table 5, it is obvious that the training error rate increases significantly when the degree is larger than 3^{-5}. When the degree is low, the prediction is biased.

The optimal d which is found by cross validation could be shown as below.

The best d per run	Test error rate
3^{-4}	7.312
3^{-4}	6.720
3^{-4}	6.720
3^{-4}	7.258
3^{-4}	6.344
3^{-4}	5.860
3^{-4}	6.935
3^{-4}	6.720
3^{-4}	7.097
3^{-4}	6.774
3^{-4}	6.559
3^{-4}	7.043
3^{-4}	7.367
3^{-4}	6.936
3^{-4}	6.398
3^{-4}	7.419
3^{-4}	7.151
3^{-4}	7.419
3^{-4}	6.505
3^{-4}	6.452

Table 6: the optimal d through 20 runs for OvR method with Gaussian kernel

According to the table 6, it is obvious that the optimal d is 3^{-4}, which is the same conclusion as the previous section

One versus One generalized polynomial kernel perceptron

In this section, the One versus One (OvO) generalized polynomial kernel perceptron method is conducted. For the OvO case, the number of the classifiers is k(k-1)/2. The alpha would be updated. For the prediction part, a voting method is introduced to test the data to every classifier, then choose the label with the largest possibility. The pseudocode could be displayed as follow:

Algorithm 2 One versus One generalized Kernel Perceptron

Input: $(x_1, y_1), \ldots, (x_m, y_m) \in R^n \times \{-1, 1\}$

Kernel matrix K using the kernel function: $K_{ij} = K(x_i, x_j)$

Initialed alpha $\alpha_1^{(n)} = 0$ for n $\in \{1, 2, \ldots, \frac{k(k-1)}{2}\}$

Output: predict class $\hat{y}$

for *m in range (k(k – 1)/ 2)* **do**
  calculate $\hat{y}_m = \arg max_k \sum_i \alpha_i^{(k)} K(x_i, x_m)$
  if $\hat{y}_m y_m \leq 0$ **then**
    $\alpha_m^{(y_m)} = \alpha_m^{(y_m)} + 1$
    $\alpha_m^{(\hat{y}_m)} = \alpha_m^{(\hat{y}_m)} - 1$
  end if converge

Basic Results

Perform 20 runs for d = 1, … , 7, each run should randomly split into 80% train and 20% test. Report the mean test and train error rates as well as standard deviations. The results could be shown in following table.

Degree d	Training error (%)	Test error (%)
1	6.217 ± 0.308	7.358 ± 0.729
2	2.211 ± 0.194	4.452 ± 0.619
3	1.432 ± 0.147	3.968 ± 0.594
4	1.605 ± 0.096	3.470 ± 0.295
5	0.871 ± 0.136	3.766 ± 1.346
6	0.715 ± 0.125	3.524 ± 0.434
7	0.736 ± 0.106	3.551 ± 0.406

Table 7: training error and test error rates for OvO method

The optimal d for 20 runs could be shown as below.

The best d per run	Test error rate
6	3.925
6	3.495
5	3.710
6	3.387
4	3.441
4	3.118
4	3.817
4	4.355
5	3.548
5	3.602
5	3.441
4	4.140
6	3.602
5	2.796
5	2.634
7	3.387
7	4.194
6	4.516
4	4.570
4	3.172

Table 8: optimal d for 20 runs of OvO method

Comparing the OvR method and OvO method, the OvR method needs lower classifiers than OvO method. What's more, there are some disadvantages for the OvO method, the computing time would increase a lot if the number of classes increase. The use of the voting function may cause the 'tied' situation which may lead to a wrong prediction.

PART II

In this part, two semi-supervised learning methods would be implemented and the corresponding errors and standard deviations with different datasets would be shown as below. In this part, the consensus algorithm is selected for Laplacian Interpolation method. To implement the consensus algorithm, a free vertex is chosen firstly firstly at random, then update the value of the vertex by average the sum of neighbors' values. To determine the neighbors, a 3-nn basedn on Euclidean distance is used. For the Laplacian Kernel Interpolation, the specific steps are shown in the question.

	know labels per class				
data points per label	1	2	4	8	16
50	18.37 ± 12.30	10.16 ± 6.95	5.65 ± 12.30	6.61 ± 4.25	4.19 ± 1.82
100	5.53 ± 4.56	5.41 ± 3.45	3.91 ± 0.10	3.23 ± 1.00	2.98 ± 1.00
200	4.10 ± 4.16	3.14 ± 2.10	2.42 ± 0.90	2.64 ± 1.81	1.86 ± 0.61
400	4.87 ± 4.11	2.73 ± 1.94	1.89 ± 1.60	1.39 ± 0.44	1.18 ± 0.45

Table 9: error and standard deviation for Laplacian Interpolation

	know labels per class				
data points per label	1	2	4	8	16
50	14.23 ± 23.84	12.50 ± 16.71	27.17 ± 18.58	52.38 ± 21.85	80.88 ± 25.67
100	25.25 ± 22.86	53.06 ± 20.52	6.25 ± 17.85	7.07 ± 25.27	2.98 ± 30.33
200	23.62 ± 15.88	31.07 ± 21.19	5.36 ± 22.84	1.56 ± 22.53	45.38 ± 25.23
400	23.43 ± 20.00	8.17 ± 14.70	8.21 ± 35.93	15.69 ± 18.11	4.04 ± 16.31

Table 10: error and standard deviation for Laplacian Kernel Interpolation

According to the table 9 and table 10, it is obvious that the consensus algorithm has a better performance than Laplacian Kernel Interpolation. For the Laplacian Kernel Interpolation method, there is a relative large error rate and standard deviation, which means the error rate fluctuates a lot each run. However, for the consensus algorithm, It is clear that the error rate decrease with the increase of the data points given. What's more, the increase of known labels per class also contributes to the decrease of the error rate.

PART III

Sparse learning. In the following problem we will consider the sample complexity of the perceptron, winnow, least square, and 1-nn algorithms

In this part, you will implement the four classification algorithms and then use them to estimate the sample complexity of these algorithms. Please include sample complexity plots for all four algorithms.

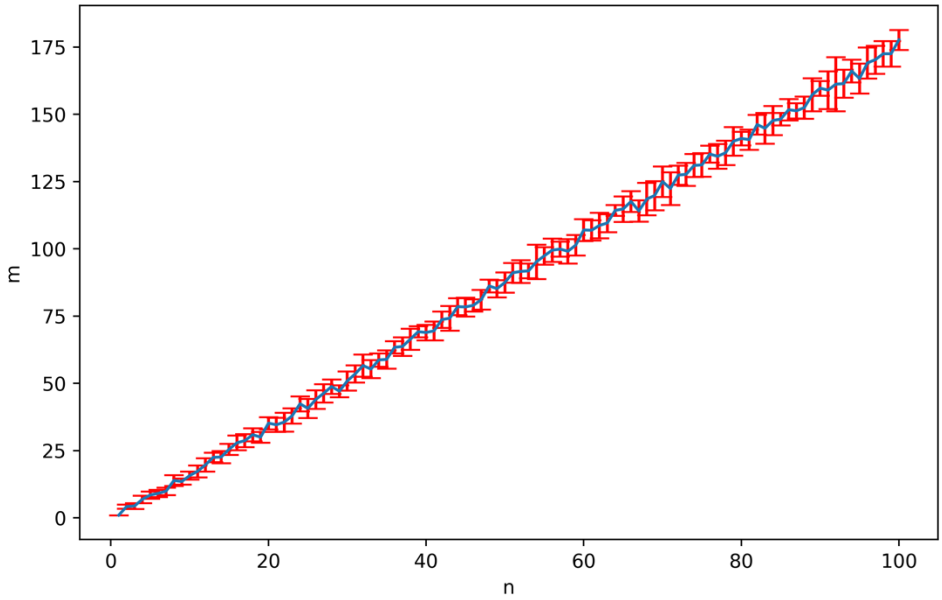

Figure 2: sample complexity plot for perceptron algorithm

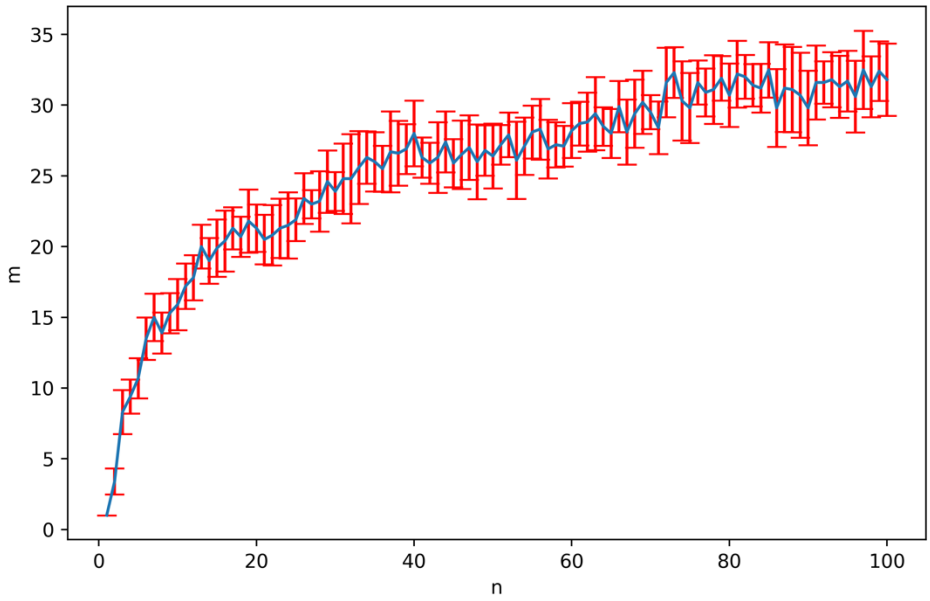

Figure 3: sample complexity plot for winnow algorithm

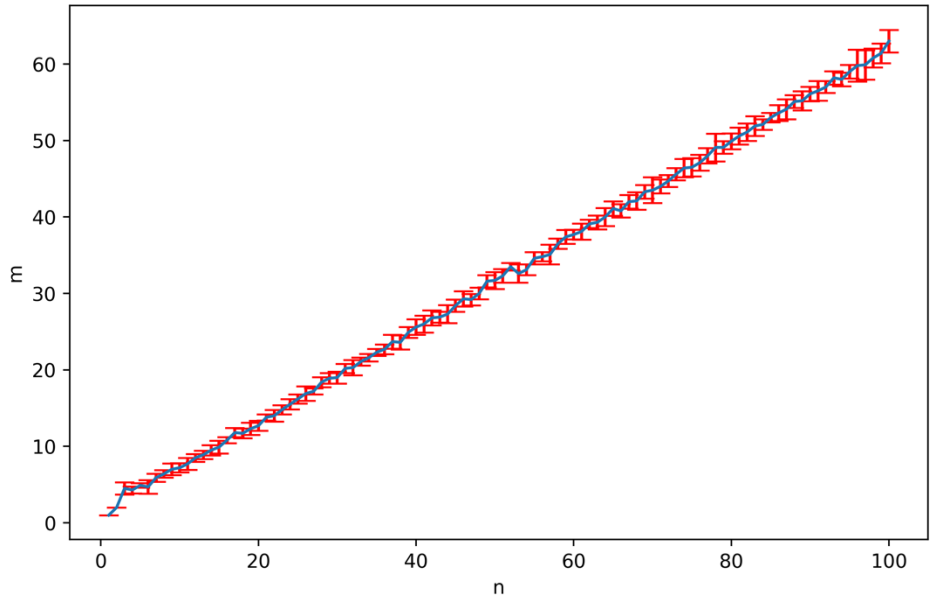

Figure 4: sample complexity for least square algorithm

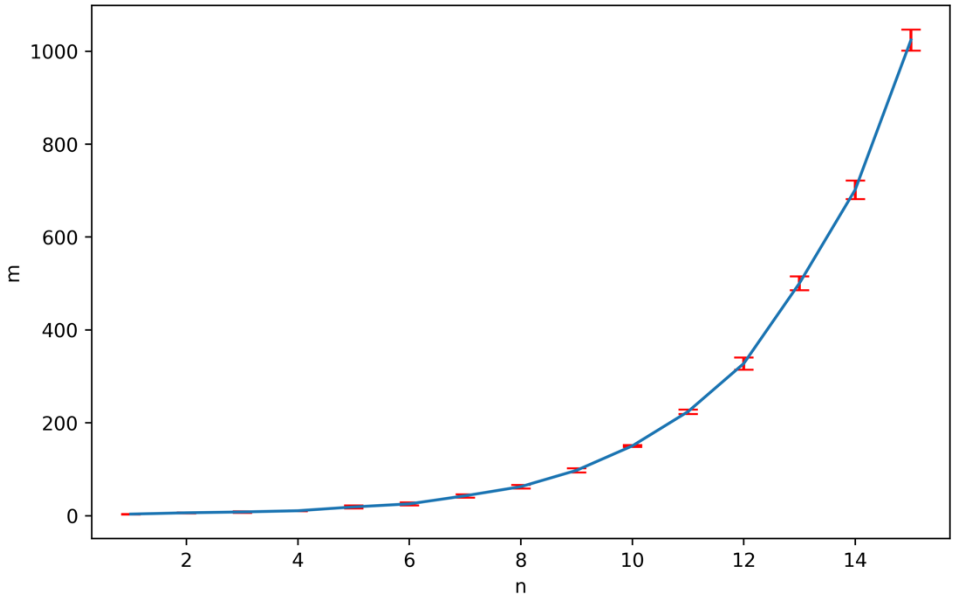

Figure 5: sample complexity for one-nn algorithm

In this question, the exactly generalization error is $\varepsilon(A_S) \coloneqq 2^{-n}\sum_{x\in\{-1,1\}^n} I[(A_S) \neq x_1]$. However, this method is expensive computationally, an estimate method is use alternatively. The estimated generalization error could be displayed as $\varepsilon(A_S) \coloneqq \frac{1}{S}\sum_{x\in\{-1,1\}^n} I[(A_S) \neq x_1]$, which S is the size of the test data.

Based on the estimate generalization error, the sample complexity could be derived as $C(A) = \min\{m \in \{1, 2, \ldots.\}: E[\varepsilon(A_S)] \leq 0.1$.
For the whole process, for each n the m starts from 1 each time until the corresponding generalization error for the current m and n less than 0.1. Then the whole process would repeat I times to derive the average value and error bar.
For the perceptron, winnow and least square, the size of n is 100, the size of test data is 5000 and the iterate times is 10. However, for the one-nn method, it takes much longer time when using these parameters. After experiments, the n for one-nn method is 15, the size of test data is 2000 and the iterate times is 5.
For the tradeoffs and biases of the method, when a larger test size is chosen, the result would be more accurate, but it will take more time. If the iterate time I increases, the error bar would be smaller than before. According to the graphs above, we can conclude that the standard deviations for perceptron and least square is small and stable. However, the standard deviation of winnow still fluctuates for the given parameter. For the one-nn method, the error increase although there number of data is not enough large.

For the observations for the m, the situations of perceptron and least square methods could be easily observed that m = $\Theta(n)$. For the winnow method, it is also obvious that the m follow the m = $\Theta(\log(n))$. For the one-nn method, the method would find the nearest point by calculating the distance. It could be concluded that the sample complexity would be bounded by 2^n if the generalized error to be 0.
As a result, the m for four methods could be displayed as below:
Perceptron: m = $\Theta(n)$
Least square: m = $\Theta(n)$
Winnow: m = $\Theta(\log(n))$.
One-nn: m = $\Omega(2^n)$

Derive a non-trivial upper bound on the probability that the perceptron will make a mistake on the sth example after being trained on examples.

For the mistake of perceptron, we have the mistake is bounded by $M \leq (\frac{R}{\gamma})^2$, where R = $max_t||x_t|| = \sqrt{\sum_{i=1}^{n} x_{t,i}^2} = \sqrt{n}$. According to the question, $x_i \in \{-1, 1\}^n$ and $y_i \in \{-1, 1\}$, so the $M \leq \left(\frac{R}{\gamma}\right)^2 = \left(\frac{\sqrt{n}}{\gamma}\right)^2 = \frac{n}{\gamma^2} = n$