COMP0078 Coursework 2

Student Number: 19002505

Date: 14/12/2022

1 PART I

Kernel Perceptron

In this section, we generalized the two class kernel perceptron algorithm to performing multi-class classification tasks. The dataset used consists of 9298 16×16 hand-written digits each represented by a 1×256 vector.

1.1 The One-vs-All Kernel Perceptron

A the name suggested, a two class kernel perceptron algorithm can only handle binary classification tasks. A simple way to generalize it into a multi-class classifier is to apply the One-vs-All scheme. To do so, we train k binary kernel perceptron classifiers corresponding to K distinct classes. For the Handwritten Digits dataset, we have $K = 10$. Each binary classifier C_j treats the j^{th} class as the positive and all the other classes as negative. The output of the classifier C_j on a test sample x_t gives the confidence that x_t belongs to the j^{th} class:

$$Confidence_j = C_j(x_t) = \sum_{i=1}^{m} \alpha_i^{(j)} K(x_i, x_t)$$

where m is the number of samples the algorithm has received so far, α is a $K \times m$ matrix storing the values of the learnable parameter and $\alpha_i^{(j)}$ is the i^{th} element of the j^{th} row of α. The final classification result is then the class with the maximum confidence:

$$\hat{y}_t = \underset{j}{\operatorname{argmax}}(C_j(x_t))$$

The pseudo-code demonstrating the implementation of the One-vs-All kernel perceptron algorithm is given as following:

Algorithm 1: One-vs-All Kernel Perceptron

```
1. **Input**: $\{(x_1, y_1), \dots (x_m, y_m)\} \in (R^n, \{-1, +1\})^m$
2. **Initialize** $\alpha = 0^{K,m}$ (a $K \times m$ matrix of zeros)
3. **for** $epoch = 1,2,\dots,5$ **do**
4.     **for** $t = 1,2,\dots,m$ **do**
5.         **for** $j = 0,1,\dots,K$ **do**
6.             compute confidence for the $j^{th}$ class: $\sum_{i=1}^{m} \alpha_i^{(j)} K(x_i, x_t)$
7.         **end**
8.         make prediction $\hat{y}_t = \underset{j}{\operatorname{argmax}}(C_j(x_t))$
9.         **if** $\hat{y}_t \neq y_t$ **then**
10.            $\alpha^{(\hat{y}_t)} = \alpha^{(\hat{y}_t)} - 1$
11.            $\alpha^{(y_t)} = \alpha^{(y_t)} + 1$
12.    **end**
13. **end**
```

1.2 Polynomial Kernel

The generalized multi-class classifier was first implemented with the polynomial kernel:

$$K(x_i, x_t) = (x_i \cdot x_t)^d$$

where d is the dimension of the polynomial.

We performed 20 repeated experiments with 7 different values of $d = 1,2,\dots,7$. In each run, the dataset was randomly shuffled and split into 80% training data and 20% test data. The algorithm was first trained on the training set and then evaluated on both the training and test sets. The mean train and test error rates (with standard deviations) are as following:

d	Mean Train Error Rates (%)	Mean Test Error Rates (%)
1	9.0239±0.2944	9.6317±1.8428
2	4.7681±0.2877	6.9086±1.5810
3	2.5746±0.1838	4.9274±0.6437
4	1.6510±0.1692	4.7527±0.7367
5	1.1119±0.1261	4.0484±0.6295
6	0.8120±0.1401	3.7876±0.6084
7	0.6460±0.1170	3.5618±0.4026

Table 1. Mean train and test error (%) for different dimensions of polynomial kernel

From table 1, we observe that the algorithm performs better as d increases. The best performance was observed when $d = 7$. However, it can be seen that the improvement in test error rate in decreasing as d gets very large, suggesting a tendency towards overfitting. We will further verify this using cross-validation in the next section.

1.3 Optimal Dimension for Polynomial Kernel

In this section, we carried out 20 more experiments with 5-fold cross-validation to find the optimal value of d. In each run, the dataset was first randomly spilt into 80% training data and 20% test data. Then, a 5-fold cross-validation was carried out on the training data to find the optimal d^*. Finally, the algorithm was re-trained over the entire training data and evaluated on the test data. Noted that the shuffling was not done in cross-validation to ensure the algorithm was train on the same subsets but only with a different d value to allow comparison.

The optimal d^* values and the corresponding test error rates (with standard deviations) are given as following:

run	d*	Test Error Rates (%)
1	5	3.3871
2	7	2.8495
3	7	3.2796
4	7	3.1183
5	7	3.4946
6	7	4.086
7	7	3.4946
8	5	3.4946
9	6	3.2258
10	7	3.4409
11	6	4.5161
12	7	4.086
13	6	3.6559
14	6	3.1183
15	6	4.3011
16	4	4.6237
17	6	3.3333
18	7	2.5806
19	4	4.3548
20	7	2.6344

Table 2. Optimal dimensions and the corresponding test error rates over 20 runs.

From Table 2, we can obtain a mean test error rate and a mean d^* value:

$$mean\ optimal\ d = 6.2 \pm 1.0$$

$$mean\ test\ error = (3.55 \pm\ 0.58)\%$$

Besides test error rates, we also obtained an averaged confusion matrix along the 20 repeated experiments:

	0	1	2	3	4	5	6	7	8	9
0	0.000±0.000	0.050±0.119	0.223±0.318	0.356±0.385	0.164±0.164	0.408±0.407	0.370±0.332	0.000±0.000	0.143±0.236	0.097±0.148
1	0.000±0.000	0.000±0.000	0.020±0.089	0.097±0.206	0.436±0.398	0.100±0.216	0.237±0.315	0.097±0.206	0.100±0.173	0.038±0.115
2	0.244±0.398	0.184±0.342	0.000±0.000	0.803±0.658	0.929±0.602	0.320±0.398	0.293±0.396	0.580±0.601	0.839±0.743	0.189±0.399
3	0.177±0.329	0.029±0.127	0.906±0.766	0.000±0.000	0.147±0.363	2.060±1.932	0.000±0.000	0.472±0.493	1.656±1.659	0.181±0.342
4	0.145±0.313	0.686±0.967	0.609±0.630	0.029±0.125	0.000±0.000	0.469±0.713	0.732±0.720	0.405±0.884	0.302±0.604	1.510±1.501
5	0.698±0.851	0.032±0.138	0.704±0.578	1.572±1.206	0.457±0.410	0.000±0.000	1.318±0.925	0.099±0.235	0.605±1.172	0.493±0.792
6	0.304±0.402	0.277±0.368	0.352±0.658	0.030±0.131	0.708±0.839	0.369±0.480	0.000±0.000	0.000±0.000	0.331±0.444	0.000±0.000
7	0.000±0.000	0.221±0.419	0.463±0.541	0.190±0.351	0.553±0.674	0.062±0.187	0.000±0.000	0.000±0.000	0.663±0.674	1.782±1.554
8	0.471±0.476	0.462±0.986	0.598±0.724	1.360±1.321	0.594±0.508	1.481±1.353	0.313±0.839	0.520±0.602	0.000±0.000	0.269±0.331
9	0.087±0.209	0.091±0.218	0.158±0.274	0.180±0.275	1.422±1.272	0.211±0.289	0.000±0.000	0.960±0.990	0.413±0.564	0.000±0.000

Table 3. An average confusion matrix showing the misclassification rates for each class.

In Table 3, the rows correspond to the true labels and the columns correspond to the predictions. The element of the confusion matrix at the i^{th} row and the j^{th} column represents the percentage of times that samples of the i^{th} class had been misclassified as the j^{th} class. It can observed that samples with label 3 and 5 had been misclassified as the each other very often.

1.4 Five Hardest to Predict Samples

It is worth noticing that there exist some hardest to predict samples within the dataset. The following are the top 5 samples with the highest misclassification rates over 100 runs:

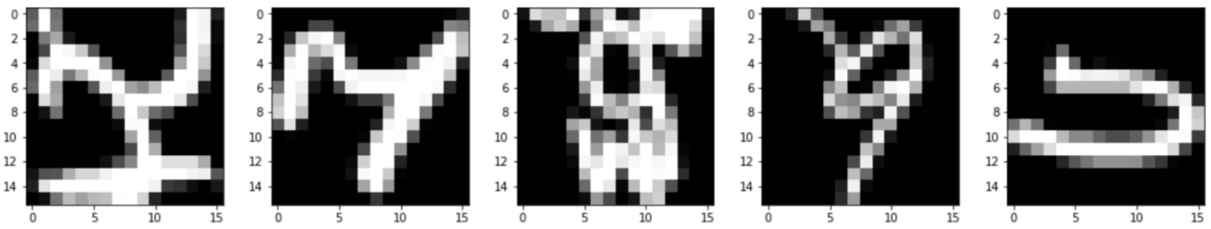


Figure 1. Top 5 hardest to predict samples with labels (left to right): 2, 7, 9, 9 5

In this section, a 100 distinct classifiers were trained on a unique subset of the data and each sample were predicted 100 times (when they are not used in training). It was not a surprised to obtain such results since these 5 hand-written digits merely resemble the actual digits.

1.5 Gaussian Kernel

To understand how well the polynomial kernel did compare to other kernels, we repeated the experiments with a Gaussian kernel:

$$K(x_i, x_t) = e^{-c||x_i - x_t||}$$

where c is the width of the kernel which will be optimized during cross-validation. Some initial experiments were carried out to determine a search space S for the parameter c. It was found that a small c $(< 10^{-4})$ to lead to under-fitting a large c (> 5) tends to cause over-fitting. Therefore, a refined search space was set to be $S = \{0.001, 0.005,\ 0.010, 0.015, 0.020, 0.050, 0.100\}$. The above experiments done with the polynomial kernel were repeated and we obtained the following results:

c	Mean Train Error Rates (%)	Mean Test Error Rates (%)
0.001	7.3420±0.3039	8.7124±1.9736
0.005	1.1690±0.1853	4.0296±0.5973
0.01	0.4228±0.1045	3.3548±0.5175
0.015	0.2084±0.0547	3.0887±0.5244
0.02	0.1560±0.0694	3.0108±0.3666
0.05	0.0471±0.0277	3.8844±0.3728
0.1	0.0255±0.0175	4.8737±0.5252

Table 4. Mean train and test error (%) for different widths of gaussian kernel.

run	c*	Test Error Rates (%)
1	0.01	2.9032
2	0.015	3.0645
3	0.02	3.2796
4	0.02	2.5269
5	0.015	2.7957
6	0.02	3.3871
7	0.01	3.4409
8	0.015	2.7957
9	0.02	3.0108
10	0.015	2.7957
11	0.01	2.957
12	0.01	3.2796
13	0.015	3.2796
14	0.02	3.3333
15	0.02	3.172
16	0.015	3.4409
17	0.015	2.7419
18	0.015	2.9032
19	0.015	3.2796
20	0.015	2.6344

Table 5. Optimal widths and the corresponding test error rates over 20 runs.

From Table 5, we can obtain a mean test error and a mean c^* value:

$$mean\ optimal\ c = 0.0155 \pm 0.0035$$

$$mean\ test\ error\ rate = (3.0511 \pm\ 0.2751)\%$$

From Table 4, it can be observed that the performance of the algorithm was improved as the width of the gaussian kernel c increases. However, overfitting occurs when the width becomes too large where we can observed an improvement in training error rate but with a worse test error rate. From the result of the cross-validation, the gaussian kernel slightly overperforms the polynomial kernel. However, this not ensure as the bounds of uncertainty of the mean test error rates for the two kernel overlap.

1.6 One-vs-One Kernel Perceptron

In this section, another generalization method called the One-vs-One scheme was studied.

The One-vs-One Kernel Perceptron trains N binary classifiers where $N = \frac{K(K-1)}{2}$ for each pair of classes. In the case of the Hand-Written Digits dataset, we will be training 45 classifiers in total. Each classifier gives a voting on one of the two classes it compares based on the sign of the output. Then, the class with the maximum amount of votes out of the 45 classifiers becomes the final prediction.

The implementation of the one-vs-one kernel perceptron algorithm is demonstrated as following:

Algorithm 2: One-vs-One Kernel Perceptron

```
1. Input: $\{(x_1, y_1), \ldots (x_m, y_m)\} \in (R^n, \{-1, +1\})^m$
2. Initialize $\alpha = 0^{N,m}$ (a $N \times m$ matrix of zeros where $N = \frac{K(K-1)}{2}$)
3. for $epoch = 1,2, \ldots ,5$ do
4.     for $t = 1,2, \ldots , m$ do
5.         for $j = 0,1, \ldots , N$ do
6.             evaluate the vote of the $j^{th}$ classifier: $sign(\sum_{i=1}^{m} \alpha_i^{(j)} K(x_i, x_t))$
7.         end
8.         make prediction $\hat{y}_t$ based on maximum voting
9.         for $i = 0,1, \ldots , N$ do
10.            if the $i^{th}$ classifier involved label $y_t$ AND $\hat{y}_t \neq y_t$ then
11.                $\alpha^{(i)} = \alpha^{(i)} + 1$ if $y_t$ is the positive class
12.                $\alpha^{(i)} = \alpha^{(i)} - 1$ if $y_t$ is the negative class
12.    end
13. end
```

Similarly, the above experiments were again repeated with the polynomial kernel to evaluate the one-vs-one kernel perceptron algorithm. The results obtained are as following:

d	Mean Train Error Rates (%)	Mean Test Error Rates (%)
1	10.4006±0.2609	11.1801±1.3514
2	8.2751±0.2533	9.3441±0.7706
3	6.4843±0.2309	7.9355±0.7961
4	5.1573±0.1596	7.1559±0.6411
5	3.8129±0.2346	6.4516±0.4705
6	2.7494±0.2016	6.0054±0.7436
7	1.9293±0.2232	5.6290±0.4916

Table 6. Mean train and test error (%) for different dimensions of polynomial kernel.

run	d*	Mean Test Error Rates (%)
1	7	6.2903
2	7	4.7312
3	7	6.129
4	7	5.6452
5	7	6.6667
6	7	6.0753
7	7	4.5699
8	7	6.0753
9	7	5.6989
10	7	5.6452
11	7	6.4516
12	6	6.1828
13	7	5.7527
14	6	6.6129
15	6	6.1828
16	7	6.2366
17	7	5.8602
18	6	5.4301
19	7	5.3763
20	7	6.4516

Table 7. Optimal dimensions and the corresponding test error rates over 20 runs.

From Table 7, we can obtain a mean test error and a mean d^* value:

$$mean\ optimal\ d = 6.8 \pm 0.4$$

$$mean\ test\ error\ rate = (5.90 \pm\ 0.55)\%$$

From Table 6, it can be observed that the performance of the algorithm improves as the dimension of the polynomial kernel increases. Similar to the One-vs-All kernel perceptron, diminishing improvements in test error rates can be seen suggesting a tendency towards overfitting.

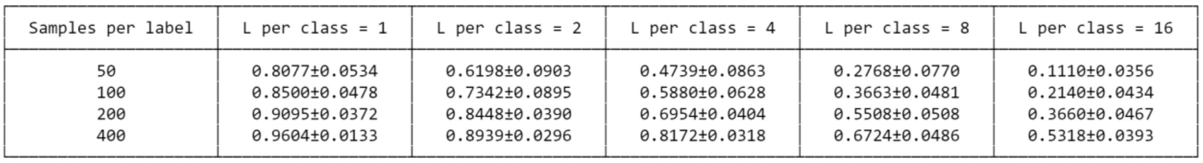
2 PART II

Semi-supervised Learning vis Laplacian Interpolation

In this section, we implemented Laplacian Interpolation (LI) and Laplacian Kernel Interpolation (LKI) and evaluated their performances in binary classification tasks.

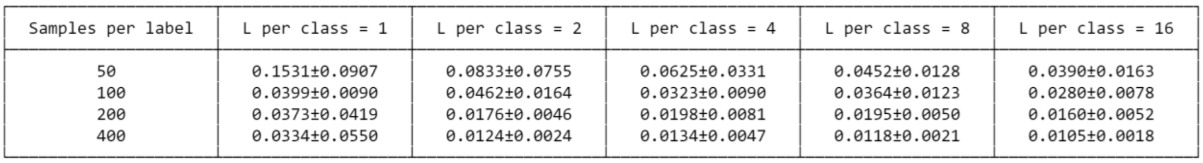
The datasets used in this experiment were four distinct subsets of the USPS dataset which were randomly sampled with sizes $m \in \{100, 200, 400, 800\}$. In each dataset, the number of samples per class are equivalent. For each dataset, the LI and LKI algorithms were ran 20 times to obtain averaged results. In each run, we randomly labelled $L \in \{1, 2, 4, 8, 16\}$ samples from each class and the algorithms were trained on the entire dataset. Then, the generalization error rates were computed by evaluating the algorithms over the unlabelled samples.

The averaged generalization error rates (with standard deviations) over the 20 runs for each dataset with different numbers of labelled samples were recorded in the following tables:



Samples per label	L per class = 1	L per class = 2	L per class = 4	L per class = 8	L per class = 16
50	0.8077±0.0534	0.6198±0.0903	0.4739±0.0863	0.2768±0.0770	0.1110±0.0356
100	0.8500±0.0478	0.7342±0.0895	0.5880±0.0628	0.3663±0.0481	0.2140±0.0434
200	0.9095±0.0372	0.8448±0.0390	0.6954±0.0404	0.5508±0.0508	0.3660±0.0467
400	0.9604±0.0133	0.8939±0.0296	0.8172±0.0318	0.6724±0.0486	0.5318±0.0393

Table 8. Experiment results using the Laplacian Interpolation algorithm.



Samples per label	L per class = 1	L per class = 2	L per class = 4	L per class = 8	L per class = 16
50	0.1531±0.0907	0.0833±0.0755	0.0625±0.0331	0.0452±0.0128	0.0390±0.0163
100	0.0399±0.0090	0.0462±0.0164	0.0323±0.0090	0.0364±0.0123	0.0280±0.0078
200	0.0373±0.0419	0.0176±0.0046	0.0198±0.0081	0.0195±0.0050	0.0160±0.0052
400	0.0334±0.0550	0.0124±0.0024	0.0134±0.0047	0.0118±0.0021	0.0105±0.0018

Table 9. Experiment results using the Laplacian Kernel Interpolation algorithm.

As shown in Table 8, the generalization error rates of the LI algorithm increases as the number of labelled samples increases and decreases as the size of the dataset increases. This is as expected since more labelled data means more information provided and a large dataset means the algorithm were evaluated over more unlabelled samples. The improvement in generalization error seems to be proportional to the ratio $\frac{L}{m}$ where for a fixed size of dataset a larger increase in labelled samples results in a large improvement in performance. Overall, the generalization error rates of the LI algorithm were extremely high when there was a very small number of labelled data and the improvements were huge when the number of labelled data increases. In short, the performance of the LI algorithm seems to be highly dependent on the number of labelled data.

A similar trend was observed in Table 9 for the LKI algorithm that the performance increases as the number of labelled samples increases. However, the LKI algorithm demonstrated better performances on larger sizes of datasets which is the complete opposite to that observed for the LI algorithm. Furthermore, the generalization error rates of the LKI algorithms were observed to be much smaller compare to that of the LI algorithm in all situations experimented. It also seems like that the LKI algorithm is less dependent on the number of labelled data.

In this experiment, we implemented the LI algorithm with the Harmonic solution where the prediction for a sample is based on the averaged value of its neighbours' labels. Therefore, more labelled data will no doubt result in more accurate predictions (i.e. highly dependent on the number of labelled data). Therefore, with a small number of labelled data used ($L \in \{1, 2, 4, 8, 16\}$), the accuracy of the LI algorithm was observed to be relatively low.

Back to the tables, it was observed that the LKI algorithm outperformed the LI algorithm in all the situation experimented. However, we consider that the LI algorithm will have a better performance than the LKI algorithm when we have a small dataset with a large number of labelled data (i.e. a large ratio of $\frac{L}{m}$). This is because that the LI algorithm performs better when there is a large number of labelled data and the LKI algorithm performs less accurate when the size of the dataset is small. Experiments were carried out to verify this inference and the results are as shown as below:

Samples per label	L = 30	L = 35	L = 40	L = 45	L = 48
50	0.0563±0.0386	0.0500±0.0289	0.0275±0.0295	0.0400±0.0663	0.0250±0.0750

Samples per label	L = 30	L = 35	L = 40	L = 45	L = 48
50	0.0512±0.0311	0.0567±0.0410	0.0325±0.0327	0.0500±0.0742	0.0375±0.0893

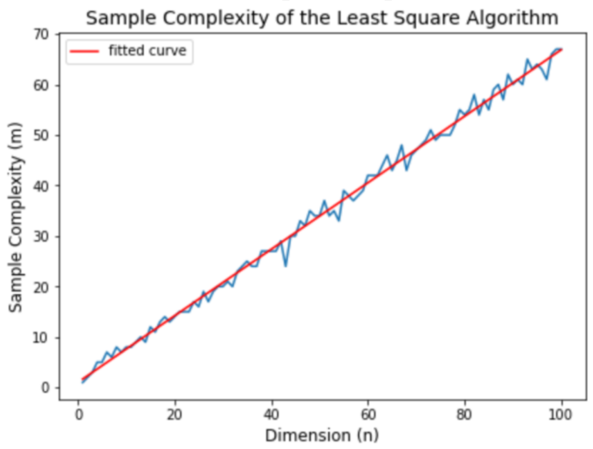
Table 10. Experiment results with a small ($m = 50$) dataset with a large number of labelled data ($L \geq 30$).

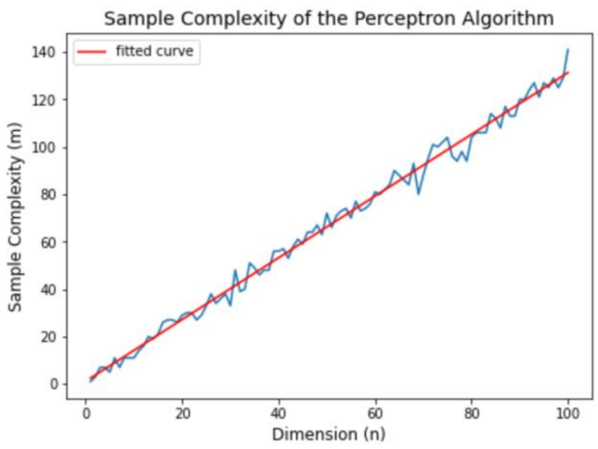
The above results in Table 10 were the averaged generalizations rates over 20 repeated runs. As shown, the performance of the LI algorithm was still less accurate than the LKI algorithm when $L = 30$. However, the LI algorithm started to outperform the LKI algorithm as the number of labelled data gets larger (i.e. for $L \geq 35$).

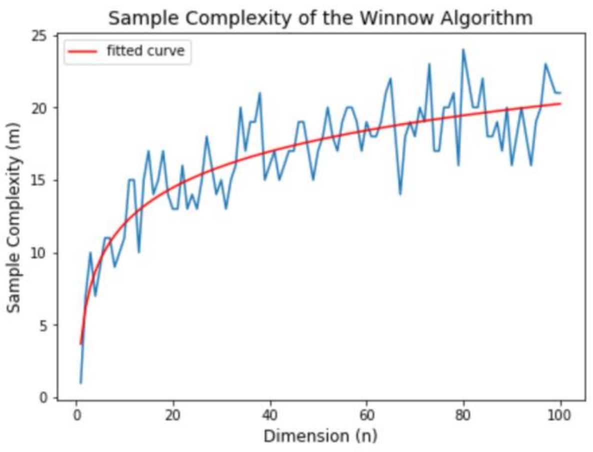
3 PART III

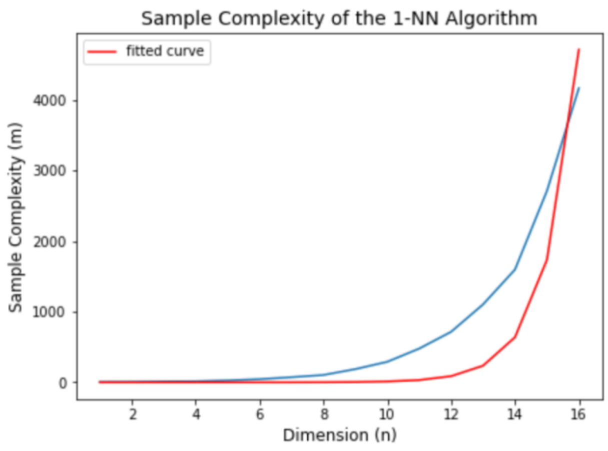
Sparse Learning

(a) In this part, 4 classification algorithms were implemented and the following diagrams demonstrate their sample complexities at different dimensions of data:









(b) Computing the "exact" sample complexity through simulation involves training the algorithms over multiple training sets at a specific level of dimension. This is computationally expensive and therefore the accuracy of the results presented in section (a) was traded-off for a faster computation time.

First of all, we used the validation set bound to bound the generalization error $\varepsilon(A_S)$ of an algorithm A trained on some dataset S as:

$$\varepsilon(A_S) \leq L_v(A_S) + \sqrt{\frac{\ln\left(\frac{1}{\delta}\right)}{2m_{test}}}$$

where $L_v(A_S)$ is the empirical error evaluated over the test set, m_{test} is the size of the test set and δ is the confidence of the bound. In section (a), we adopted a 95% confidence ($\delta = 0.05$) validation set bound. By setting the test set size as $m_{test} = 2000$, we have:

$$\varepsilon(A_S) \leq L_v(A_S) + 0.027367$$

Since the sample complexity was defined as the minimum amount of training data to achieve an averaged generalization no more than 10% (i.e. $E[\varepsilon(A_S)] \leq 0.1$), the expected empirical error $E[L_v(A_S)]$ needs to be bounded by:

$$E[L_v(A_S)] + 0.02737 \leq 0.1$$
$$E[L_v(A_S)] \leq 0.07263$$

The biggest bias of the validation set bound is that we are only obtaining a probabilistic bound for the generalization error (with a confidence of 95%) instead of a guarantee result.
To find the sample complexity in section (a), for each dimension of data, we carried out 5 repeated runs with distinct training (of size m) and test sets to obtain an estimation of the expected empirical error by averaging the results. For each dimension of data, we start with $m = 1$ and iteratively increasing m by 1 until the estimated expected empirical error is less than the calculated threshold. If a more accurate estimation of the expected empirical error was desired, we just have to increase the number of runs. Here we traded-off some accuracy for a faster search.

(c) By looking at the shapes of the fitted curves shown above, we can infer the following relationships between the sample complexity m and the data dimension n:

$$Least\ Squares: \ m = \Theta(\text{n}) \quad grows\ linearly$$
$$Perceptron: \quad m = \Theta(\text{n}) \quad grows\ linearly$$
$$Winnow: \quad m = \Theta(\log(\text{n})) \ \ grows\ logarithmically$$
$$1 - NN: \ m = \Theta(e^n) \quad grows\ exponentially$$

Among all four algorithms evaluated, the Winnow algorithm performed the best overall with a logarithmic relationship between m and n suggesting a stable (slowly growing) sample complexity as $n \to \infty$. However, its sample complexity growth rapidly at small values of n compares to the others. On the contrary, the 1-NN algorithm performed the worst with a exponentially growing sample complexity since the distances between nearest neighbours grow rapidly as the dimension increases (the curse of dimensionality). While both the Least Squares and the Perceptron algorithms demonstrate linearly growing sample complexity, the Perceptron algorithm has a higher rate of increase.

(d) For an online learning perceptron algorithm trained on a dataset $S(X, y)$, Novikoff's Theorem suggests that the total number of mistakes during training is bound by:

$$B \leq \frac{R^2}{\gamma^2}$$

where R is the Euclidean distance such that $\forall x: ||x|| \geq R$ and γ is the margin of the hyperplane that linearly separates the data.
Since the classification of a data point x depends entirely on its first element $x_1 \in \{-1,1\}$, the hyperplane separating difference classes of data point has margin $\gamma = 1$.
For an n-dimensional space with $x \in \{-1,1\}$, we have:

$$||x|| = \sqrt{\sum_{i=1}^{n}(\pm 1)^2} = \sqrt{n} \geq R$$

Thus, the upper bound of the total number of mistakes in this case becomes:

$$B \leq \frac{(\sqrt{n})^2}{1^2} = n$$

Since s is sampled uniformly at random from $\{1, \dots, m\}$, the upper bound of the probability that the algorithm will misclassify the s^{th} sample after receiving $(s-1)$ samples is given as:

$$\hat{P}_{m,n} \leq \frac{B}{m} \leq \frac{n}{m}$$

(e) For the 'just a little bit' binary classification problem, we have the input space $X = \{-1,1\}^n$ and the labels $y = \{-1,1\}$. Then, we can model the conditional probability of y given a sample x as a function $\eta(x)$:

$$\eta(x) = P(y = 1|x)$$

Since the label of a sample is always the first element, we have $\eta(x)$ deterministic:

$$\eta(x) = \begin{cases} 0 & when\ x_1 = -1 \\ 1 & when\ x_1 = +1 \end{cases}$$

Frist of all, we find the minimum value of c that makes $\eta(x)$ c-Lipschitz. That is, we wish to have $c > 0$ such that for all x and x':

$$|\eta(x) - \eta(x')| \leq c||x - x'||$$

$$c \geq \frac{|\eta(x) - \eta(x')|}{||x - x'||}$$

$$\min(c) = \frac{\max(|\eta(x) - \eta(x')|)}{\min(||x - x'||)}$$

Since $\eta(\cdot)\epsilon\{0,1\}$, we have $|\eta(x) - \eta(x')|\epsilon\{0,1\}$ and therefore $\max(|\eta(x) - \eta(x')|) = 1$.

For $|\eta(x) - \eta(x')| = 1$, we need to have at least $x_1 \neq x_1'$ (while all the other elements can be equivalent) and therefore $\min(||x - x'||) = 2$.

Now we can compute the minimum value of c as $\frac{1}{2}$ and thus for x and x':

$$|\eta(x) - \eta(x')| \leq \frac{1}{2}\ ||x - x'||$$

Secondly, we look for a way to bound the expected generalization error rate $E[\varepsilon(A_{s_m})]$ of the algorithm A trained on the dataset S_m. For the 1-NN algorithm, the error rate $\varepsilon(A_{s_m})$ is the probability that, for a sample (x, y) in the test set, the label of the nearest neighbour (in the training set) to x is not equal to y. Denoting x'as the nearest neighbour to a sample x, the probability can be expressed as:

$$\begin{aligned} &P_{y\sim\eta(x),y'\sim\eta(x')}(y \neq y') \\ &= \eta(x')\big(1 - \eta(x)\big) + \big(1 - \eta(x')\big)\eta(x) \\ &= 2\eta(x)\big(1 - \eta(x)\big) + \big(\eta(x) - \eta(x')\big)(2\eta(x) - 1) \end{aligned}$$

$$= \left|2\eta(x)\big(1-\eta(x)\big) + \big(\eta(x)-\eta(x')\big)(2\eta(x)-1)\right|$$
$$= |\eta(x)-\eta(x')| \le \frac{1}{2}\,||x-x'||$$

given that $\eta(x)\big(1-\eta(x)\big)=0$ and $|2\eta(x)-1|=1$ since $\eta(\cdot)\epsilon\{0,1\}$.

Thus, we have the expected generalization error rate bounded by:

$$E\big[\varepsilon\big(A_{S_m}\big)\big] = E[P(y \ne y')] \le \frac{1}{2}\,\mathrm{E}\big[||x-x'||\big]$$

Next, we need to bound the expected distance $\mathrm{E}\big[||x-x'||\big]$ between a sample x (in the test set) and its nearest neighbour x' (in the training set). There exists two cases to be considered:

Case 1: $x = x'$, then we have $\big||x-x'|\big| = 0$, however $P(x = x') = 0$ because x is in the training set and x' is in the test set.

Case 2: $x \ne x'$, then we have $\big||x-x'|\big|$ bounded by the maximum possible distance between any two points drawn from the input space $X = \{-1,1\}^n$:

$$\big||x-x'|\big| \le \sqrt{\sum_{i=1}^{n}(x_i - x_i')^2} = \sqrt{\sum_{i=1}^{n} 2^2} = \sqrt{4n} = 2\sqrt{n}$$

Now we have to find $P(x \ne x')$ which is the probability that a randomly sampled data point x is not in the training set S_m. Let $C_1, C_2, \dots, C_r$ be the distinct subsets of the input space X each containing exactly one unique data point in X. Since $X = \{-1,1\}^n$, the number of unique data points are $r = 2^n$. Then, we can express the probability as:

$$\begin{aligned}
&P(x \ne x') \\
&= E\Big[\sum_{i:\, C_i \cap S_m = \emptyset} P(x \in C_i)\Big] \\
&= \sum_{i=1}^{2^n} P(x \in C_i)\cdot P(C_i \cap S_m = \emptyset) \\
&= \sum_{i=1}^{2^n} P(x \in C_i)\cdot [1 - P(x \in C_i)]^m \\
&\le \sum_{i=1}^{2^n} P(x \in C_i)\cdot e^{-P(x\in C_i)\cdot m} \\
&= 2^n \cdot \frac{1}{2^n}\cdot e^{-\frac{m}{2^n}} = e^{-\frac{m}{2^n}}
\end{aligned}$$

The expected distance is then given as:

$$\begin{aligned}
&\mathrm{E}\big[||x-x'||\big] \\
&= P(x = x')\cdot 0 + P(x \ne x')\cdot 2\sqrt{n}
\end{aligned}$$

$$\leq 2\sqrt{n}e^{-\frac{m}{2^n}}$$

Previously, we have the expected generalization error rate bound by:

$$E\left[\varepsilon\left(A_{s_m}\right)\right] \leq \frac{1}{2}\ \mathrm{E}\left[\left|\left|x - x'\right|\right|\right]$$

Thus, we have:

$$E\left[\varepsilon\left(A_{s_m}\right)\right] \leq \sqrt{n}e^{-\frac{m}{2^n}}$$

In this problem, the sample complexity was defined as the minimum m that gives an expected generalization error rate no more than 10%. Therefore, we set:

$$\begin{aligned}
&\sqrt{n}e^{-\frac{m}{2^n}} \leq 0.1 \\
&e^{\frac{m}{2^n}} \geq 10\sqrt{n} \\
&\frac{m}{2^n} \geq \log\left(10\sqrt{n}\right) \\
&m \geq 2^n \log\left(10\sqrt{n}\right) = 2^n \log(10) + 2^n \log\left(\sqrt{n}\right)
\end{aligned}$$

Thus, the lower bound for the sample complexity is:

$$\begin{aligned}
&m = \Omega\left(2^n \log(10) + 2^n \log\left(\sqrt{n}\right)\right) \\
&= \Omega\left(2^n \log\left(\sqrt{n}\right)\right) \\
&= \Omega(2^n \log(n)) = \Omega(f(n))
\end{aligned}$$

A simple function which is good lower bound for the sample complexity is then:

$$f(n) = 2^n \log(n)$$