COMP0078 Assignment 2

Student Numbers: 21168615 & 19004608

Dec 14, 2022

1 PART I

1.1 Kernel Perceptron

1.1.1 Experimental Results

1. To generalise the kernel perceptron to K classes:

Algorithm 1 An training algorithm for multi-class kernel perceptron

for $k \in \{1, 2, ..., K\}$ **do** ▷ Initialise weights for first training example
end for
for $i \in \{1, 2, ..., M\}$ **do**: ▷ Number of Epochs
  for $j \in \{2, ..., N\}$ **do** ▷ Number of training points
    for $k \in \{1, 2, ..., K\}$ **do** ▷ Number of classes
      $\hat{y}_k^{i,j} \leftarrow \sum_{i'=1}^{i-1} \sum_{j'=1}^{j-1} (w_k^{i,j} \cdot k(\mathbf{x}_{j'}, \mathbf{x}_j))$ ▷ Prediction for class k with kernel $k(\cdot, \cdot)$
      if $sign(\hat{y}_k^{i,j}) \neq sign(y_k^{i,j})$ **then** ▷ Compare predicted $\hat{y}_k^{i,j}$ and actual $y_k^{i,j}$)
        $w_k^{i,j} \leftarrow y_k^{i,j}$
      else
        $w_k^{i,j} \leftarrow 0$
      end if
    end for
  end for
end for

The number of epochs was determined by training on the mini train data set until the performance of the mini test set no longer changed. This was considered a reasonable number of epochs to use to train the actual data set.

For prediction, the k with the maximum value of $\hat{y}_k$ (the argmax of $\hat{y}$) is considered the class prediction by the model.

Basic Results:

	Number of Training Epochs	Train Error	Test Error
degree=1.0e+00	5	8.59%±1.10%	10.63%±1.21%
degree=2.0e+00	2	2.88%±0.58%	5.69%±0.62%
degree=3.0e+00	1	2.78%±0.72%	5.29%±0.80%
degree=4.0e+00	3	0.32%±0.14%	3.27%±0.44%
degree=5.0e+00	1	1.86%±0.38%	4.70%±0.59%
degree=6.0e+00	2	0.32%±0.07%	3.24%±0.36%
degree=7.0e+00	2	0.27%±0.08%	3.32%±0.45%

Figure 1: 20 Runs with a Polynomial Kernel

2. Cross Validation Results:

Run	Optimal degree	Number of Training Epochs	Train Error	Test Error
1	4.0e+00	3	0.31%	3.28%
2	4.0e+00	3	0.38%	3.28%
3	6.0e+00	2	0.03%	3.28%
4	7.0e+00	2	0.09%	2.80%
5	4.0e+00	3	0.13%	2.96%
6	4.0e+00	3	0.38%	2.96%
7	6.0e+00	2	0.11%	2.58%
8	4.0e+00	3	0.30%	3.28%
9	4.0e+00	3	0.31%	3.01%
10	4.0e+00	3	0.20%	3.60%
11	4.0e+00	3	0.26%	3.87%
12	6.0e+00	2	0.12%	2.90%
13	6.0e+00	2	0.16%	3.17%
14	6.0e+00	2	0.19%	2.53%
15	4.0e+00	3	0.22%	2.47%
16	4.0e+00	3	0.26%	4.03%
17	6.0e+00	2	0.07%	2.69%
18	4.0e+00	3	0.35%	3.82%
19	4.0e+00	3	0.38%	2.26%
20	7.0e+00	2	0.05%	3.82%
Across Runs	4.9e+00±1.1e+00	2.60±0.49	0.21%±0.11%	3.13%±0.50%

Figure 2: 5-fold Cross-Validation for 20 Runs with a Polynomial Kernel

3. Confusion Matrix:

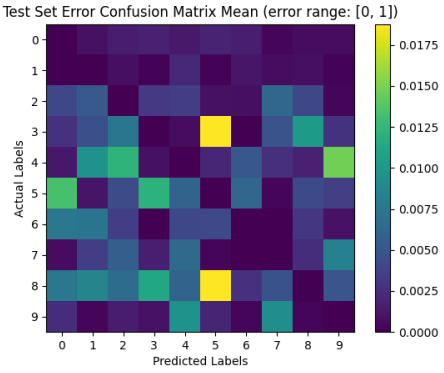


Figure 3: Polynomial Kernel Mean

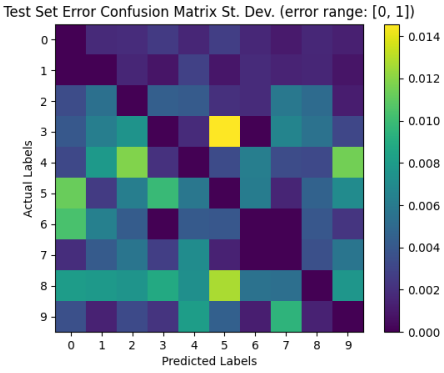


Figure 4: Polynomial Kernel St. Dev.

		Predicted Labels									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Actual Labels	**0**	0.0%±0.0%	0.08%±0.18%	0.14%±0.19%	0.17%±0.25%	0.13%±0.16%	0.19%±0.27%	0.16%±0.16%	0.04%±0.11%	0.06%±0.16%	0.06%±0.13%
	1	0.0%±0.0%	0.0%±0.0%	0.08%±0.16%	0.02%±0.08%	0.21%±0.28%	0.02%±0.09%	0.12%±0.18%	0.06%±0.14%	0.08%±0.16%	0.02%±0.08%
	2	0.4%±0.34%	0.52%±0.54%	0.0%±0.0%	0.31%±0.44%	0.34%±0.42%	0.08%±0.2%	0.08%±0.18%	0.63%±0.58%	0.41%±0.51%	0.03%±0.12%
	3	0.27%±0.4%	0.46%±0.62%	0.74%±0.75%	0.0%±0.0%	0.06%±0.18%	1.87%±1.46%	0.0%±0.0%	0.48%±0.66%	1.02%±0.55%	0.28%±0.31%
	4	0.12%±0.32%	0.96%±0.79%	1.22%±1.18%	0.09%±0.21%	0.0%±0.0%	0.2%±0.33%	0.51%±0.62%	0.26%±0.34%	0.17%±0.32%	1.48%±1.15%
	5	1.33%±1.13%	0.11%±0.25%	0.43%±0.62%	1.21%±0.99%	0.6%±0.58%	0.0%±0.0%	0.62%±0.61%	0.03%±0.15%	0.42%±0.46%	0.35%±0.7%
	6	0.74%±1.04%	0.73%±0.64%	0.35%±0.43%	0.0%±0.0%	0.41%±0.41%	0.41%±0.39%	0.0%±0.0%	0.0%±0.0%	0.29%±0.4%	0.09%±0.22%
	7	0.06%±0.19%	0.34%±0.42%	0.57%±0.57%	0.15%±0.27%	0.65%±0.71%	0.03%±0.14%	0.0%±0.0%	0.0%±0.0%	0.25%±0.36%	0.82%±0.57%
	8	0.75%±0.8%	0.86%±0.79%	0.67%±0.75%	1.12%±0.89%	0.6%±0.72%	1.88%±1.27%	0.26%±0.55%	0.47%±0.53%	0.0%±0.0%	0.5%±0.76%
	9	0.24%±0.36%	0.03%±0.14%	0.15%±0.33%	0.09%±0.22%	0.96%±0.81%	0.19%±0.44%	0.03%±0.12%	0.92%±0.95%	0.03%±0.14%	0.0%±0.0%

Figure 5: Polynomial Kernel Confusion Matrix Error Rate Table

4. The images with the most errors made by the 20 models trained with cross-validation were considered the hardest to predict and visualised. It is not surprising that these are to predict. We can see that in all five cases in Figure 6, the images do not look like their labels, in fact, they mostly look vertical or horizontal bars. Thus it would be unreasonable to expect that the model would be able to predict them correctly.

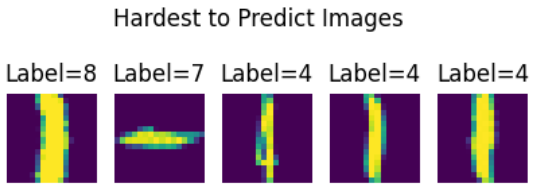


Figure 6: Images with the Most Errors from the Trained Models

5. To select a good range of values for kernel width sigma for the Gaussian kernel, the mini training and test set was used to quickly gauge the test error for different hyper-parameters. The hyper-parameter values above and below the best performing hyper-parameter (with respect to test set error) was chosen as the hyper-parameter range to define S, the set of values for cross-validation for repeating parts 1 and 2 with the Gaussian Kernel.

	Number of Training Epochs	Train Error	Test Error
sigma=1.0e-05	6	22.44%±4.69%	22.74%±6.04%
sigma=2.1e-05	4	11.24%±2.05%	11.94%±2.08%
sigma=4.3e-05	5	8.54%±1.79%	9.24%±1.89%
sigma=8.9e-05	11	2.65%±0.96%	4.20%±1.63%
sigma=1.8e-04	5	2.78%±0.84%	4.11%±1.54%
sigma=3.8e-04	10	2.62%±1.66%	4.04%±2.14%
sigma=7.8e-04	3	2.04%±0.95%	2.83%±1.55%
sigma=1.6e-03	5	0.56%±0.34%	2.36%±1.37%
sigma=3.4e-03	3	0.28%±0.33%	2.23%±1.00%
sigma=7.0e-03	2	0.17%±0.20%	1.91%±0.99%
sigma=1.4e-02	2	0.10%±0.15%	1.72%±0.99%
sigma=3.0e-02	2	0.03%±0.06%	1.59%±1.02%
sigma=6.2e-02	1	1.44%±0.43%	3.41%±1.36%
sigma=1.3e-01	2	0.06%±0.11%	3.60%±1.63%
sigma=2.6e-01	2	0.12%±0.14%	3.73%±1.48%
sigma=5.5e-01	1	0.62%±0.31%	6.40%±1.22%
sigma=1.1e+00	1	0.09%±0.08%	36.11%±3.68%
sigma=2.3e+00	1	0.15%±0.03%	57.32%±3.65%
sigma=4.8e+00	1	0.15%±0.03%	60.54%±3.61%
sigma=1.0e+01	1	0.15%±0.03%	60.70%±3.52%

Figure 7: Performance for mini data set (to narrow down a hyper-parameter range)

Basic Results:

	Number of Training Epochs	Train Error	Test Error
sigma=1.4e-02	2	0.31%±0.10%	3.37%±0.37%
sigma=2.2e-02	3	0.04%±0.02%	2.96%±0.31%
sigma=3.0e-02	2	0.18%±0.06%	3.28%±0.27%
sigma=3.8e-02	1	1.38%±0.17%	4.68%±0.45%
sigma=4.6e-02	3	0.05%±0.04%	3.75%±0.35%
sigma=5.4e-02	3	0.05%±0.03%	4.06%±0.44%
sigma=6.2e-02	1	1.77%±0.15%	5.61%±0.45%

Figure 8: 20 Runs with a Gaussian Kernel

Cross Validation Results:

Run	Optimal sigma	Number of Training Epochs	Train Error	Test Error
1	2.2e-02	3	0.05%	2.69%
2	2.2e-02	3	0.07%	2.63%
3	2.2e-02	3	0.04%	3.66%
4	2.2e-02	3	0.01%	2.69%
5	2.2e-02	3	0.04%	2.10%
6	2.2e-02	3	0.04%	2.37%
7	2.2e-02	3	0.07%	2.85%
8	2.2e-02	3	0.07%	2.69%
9	2.2e-02	3	0.05%	2.42%
10	2.2e-02	3	0.01%	2.47%
11	2.2e-02	3	0.00%	2.31%
12	2.2e-02	3	0.04%	2.90%
13	2.2e-02	3	0.08%	2.63%
14	2.2e-02	3	0.05%	2.90%
15	2.2e-02	3	0.05%	1.88%
16	2.2e-02	3	0.01%	2.63%
17	2.2e-02	3	0.08%	1.99%
18	2.2e-02	3	0.04%	2.85%
19	2.2e-02	3	0.04%	2.42%
20	2.2e-02	3	0.13%	3.49%
Across Runs	2.2e-02±3.5e-18	3.00±0.00	0.05%±0.03%	2.63%±0.42%

Figure 9: 5-fold Cross-Validation for 20 Runs with a Gaussian Kernel

6. An alternative method to generalise the kernel perceptron to k-classes:

Algorithm 2 Another training algorithm for multi-class kernel perceptron

for $k \in \{1, 2, ..., K\}$ **do** ▷ Initialise weights for first training example
  $w_k^{1,1} \leftarrow 0$
end for
for $i \in \{1, 2, ..., M\}$ **do**: ▷ Number of Epochs
  for $j \in \{2, ..., N\}$ **do** ▷ Number of training points
    for $k \in \{1, 2, ..., K\}$ **do** ▷ Number of classes
      $\hat{y}_k^{i,j} \leftarrow \sum_{i'=1}^{i-1} \sum_{j'=1}^{j-1} (w_k^{i,j} \cdot k(\mathbf{x}_{j'}, \mathbf{x}_j))$ ▷ Prediction for class k with kernel $k(\cdot, \cdot)$
    end for
    $\hat{c} \leftarrow argmax_k \hat{y}_k^{i,j}$ ▷ Predicted class
    $c \leftarrow argmax_k y_k^{i,j}$ ▷ Actual class
    if $\hat{c} \neq c$ **then** ▷ Compare predicted and actual class
      $w_c^{i,j} \leftarrow y_c^{i,j}$ ▷ Non-zero weights only for actual and predicted classes
      $w_{\hat{c}}^{i,j} \leftarrow y_{\hat{c}}^{i,j}$
      $w_k^{i,j} \leftarrow 0 \forall k \in \{1, 2, ..., K\}, k \neq c, k \neq \hat{c}$
    else ▷ All zero weights if actual and predicted classes match
      $w_k^{i,j} \leftarrow 0 \forall k \in \{1, 2, ..., K\}$
    end if
  end for
end for

Basic Results:

	Number of Training Epochs	Train Error	Test Error
degree=1.0e+00	5	9.47%±2.42%	11.66%±2.36%
degree=2.0e+00	3	3.46%±1.15%	6.44%±1.29%
degree=3.0e+00	2	2.33%±0.66%	5.35%±0.69%
degree=4.0e+00	2	1.52%±0.45%	4.56%±0.71%
degree=5.0e+00	2	0.98%±0.34%	4.10%±0.39%
degree=6.0e+00	2	0.62%±0.14%	3.68%±0.44%
degree=7.0e+00	2	0.59%±0.20%	3.79%±0.49%

Figure 10:

Cross Validation Results:

Run	Optimal degree	Number of Training Epochs	Train Error	Test Error
1	7.0e+00	2	0.69%	3.60%
2	6.0e+00	2	0.62%	3.66%
3	6.0e+00	2	0.75%	4.19%
4	5.0e+00	2	0.99%	4.30%
5	7.0e+00	2	0.77%	3.87%
6	7.0e+00	2	0.77%	4.09%
7	5.0e+00	2	0.95%	4.03%
8	7.0e+00	2	0.55%	3.87%
9	7.0e+00	2	0.50%	3.98%
10	6.0e+00	2	0.93%	3.92%
11	6.0e+00	2	0.59%	3.71%
12	7.0e+00	2	0.62%	4.19%
13	6.0e+00	2	0.67%	4.14%
14	7.0e+00	2	0.63%	3.71%
15	6.0e+00	2	0.67%	3.60%
16	6.0e+00	2	0.95%	3.82%
17	7.0e+00	2	0.65%	3.17%
18	7.0e+00	2	0.66%	4.57%
19	5.0e+00	2	0.78%	3.92%
20	6.0e+00	2	1.26%	5.05%
Across Runs	6.3e+00±7.1e-01	2.00±0.00	0.75%±0.18%	3.97%±0.39%

Figure 11:

1.1.2 Discussions

Choice of Parameters that weren't cross validated

Generalisation to k-classifiers

Kernel Comparison

Kernel Perceptron Implementation

2 PART II

2.1 Semi-supervised Learning via Laplacian Interpolation

Experimental report for the laplacian interpolation approach:

		# of known labels (per class)				
		1	2	4	8	16
accuracy	50	0.82±0.0927	0.88±0.1119	0.93±0.0607	0.95±0.0136	0.95±0.0343
	100	0.92±0.0917	0.94±0.015	0.93±0.0475	0.94±0.0151	0.94±0.0286
	200	0.88±0.1482	0.96±0.0765	0.97±0.0119	0.98±0.0092	0.98±0.0098
	400	0.93±0.1331	0.97±0.0386	0.99±0.0041	0.98±0.0069	0.99±0.0051

Figure 12:

And for the laplacian kernel method:

		# of known labels (per class)				
		1	2	4	8	16
accuracy	50	0.92±0.0752	0.93±0.0693	0.93±0.0777	0.96±0.0143	0.96±0.0174
	100	0.92±0.0391	0.91±0.0439	0.93±0.0199	0.94±0.0271	0.95±0.0164
	200	0.98±0.0235	0.98±0.014	0.98±0.0081	0.98±0.0084	0.98±0.01
	400	0.99±0.006	0.98±0.0188	0.99±0.0053	0.99±0.0036	0.98±0.0043

Figure 13:

Some observations to make here are:

a) Both models seem to perform fairly well, even in the low-data setting.

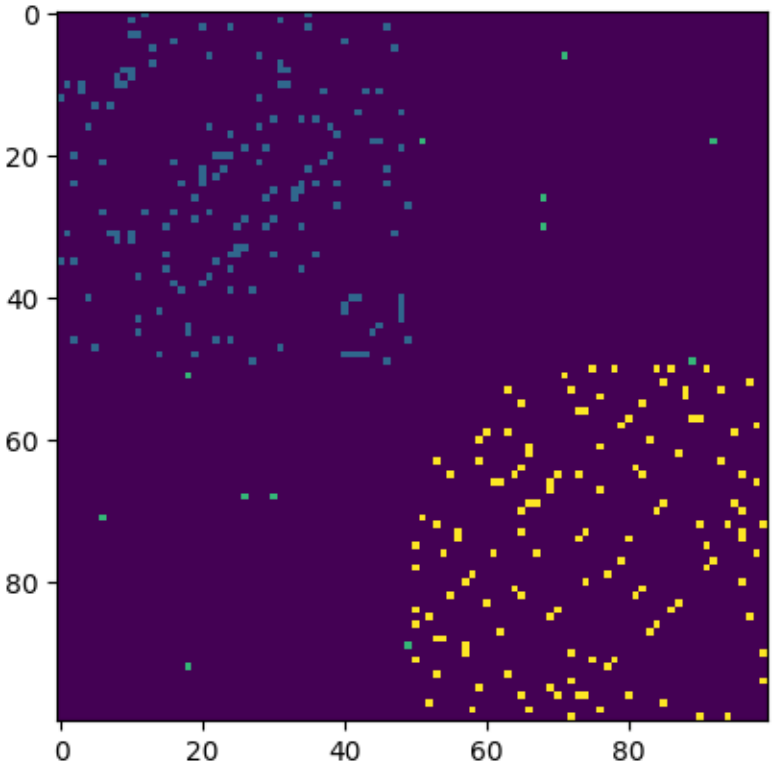
b) Increating the number of known labels increases the accuracy and decreases the variance of the predictors.

c) Increasing the total number of datapoints generally increases both model accuracies, and decreases variance.

d) The laplacian kernel method outperforms the vanilla inerpolation approach considerably, on both accuracy and variance.

The main reason for the success of this algorithm is the high degree of cluster separation observed in the dataset.

For illustration, here is a diagram representing the graph adjacency matrix, with colours showing the labelling of an edge. Here blue edges are where both labels are -1, teal when both are different, and yellow when both are +1.



We see through this that the vast majority of datapoints are connected only to points of the same labelling, and that the graph is highly separated into 2 clusters. Hence with minimal training information, both methods are able to predict with a high degree of accuracy.

Note that error variance decreases as a function of the number of known labels, but has a muhc more significant impacgt for the vanilla interpolation method.

We see that the kernel approach consistently outperforms the simple laplacian interpolation method. A reasonable explanation for this is that the laplacian interpolation approach utilises local information to 'diffuse' labels through the graph. This means that only datapoints close to labelled data receive information from the labels themselves, and accuracy is likely reduced for datapoints far from the labels. In contrast, the kernel interpolation method takes global information from all the labelled datapoints and weights them according to their proximity and connectedness to eachother to predict labels.

3 PART III

3.1 Questions

a. Sample complexity plots for the four algorithms: perceptron, winnow, least squares, and 1-nearest neighbours.

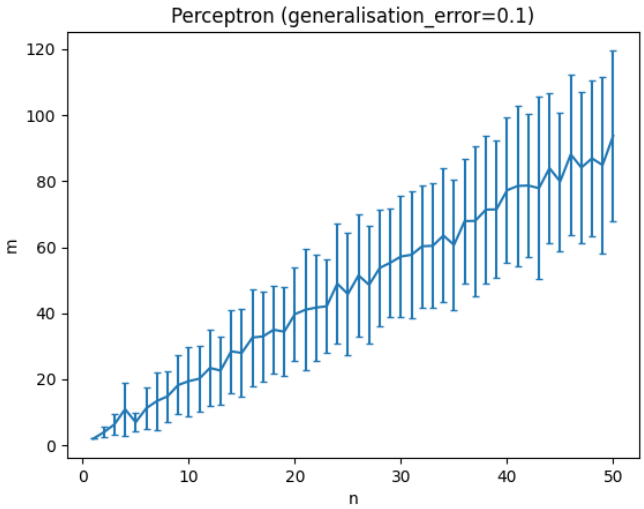


Figure 14: Perceptron Sample Complexity

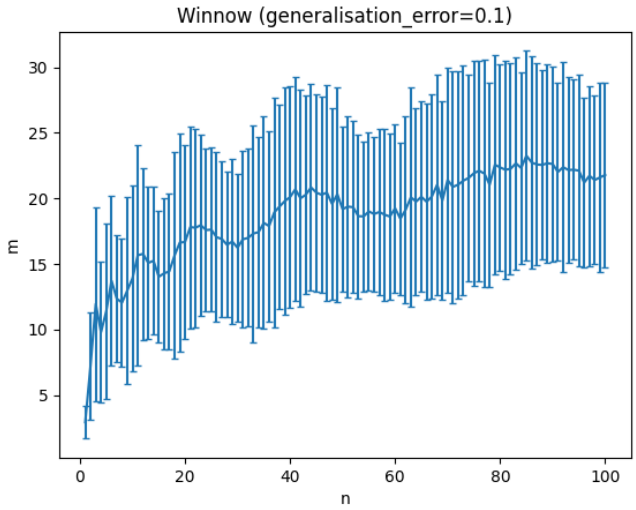


Figure 15: Winnow Sample Complexity

‘

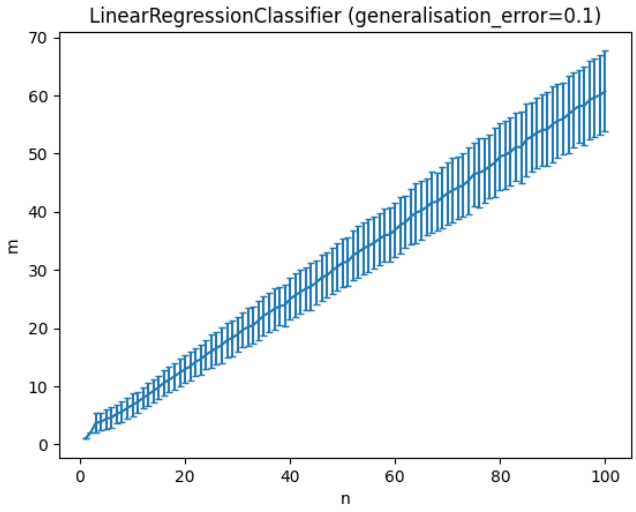


Figure 16: Least Squares Sample Complexity

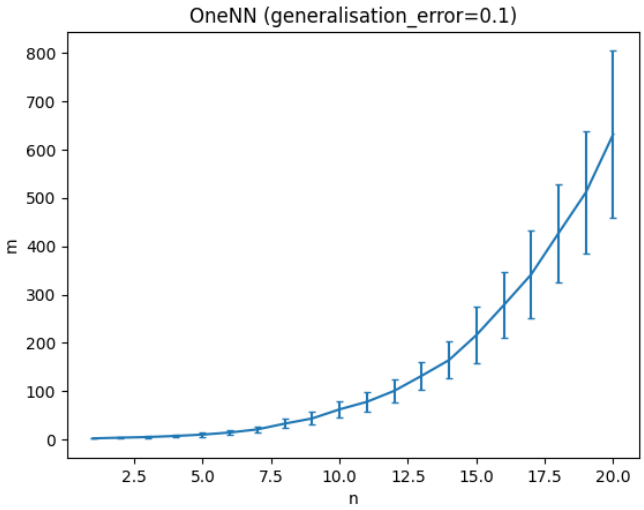


Figure 17: 1-Nearest Neighbours Sample Complexity

b. To estimate sample complexity,

   Tradeoffs and biases of this method of sample complexity estimation include

c. To estimate how m grows as a function of n as $n \to \infty$ for each of the four algorithms, we first visualise lines of best fit of the sample complexities for different function classes (i.e. polynomial, exponential, logarithm):

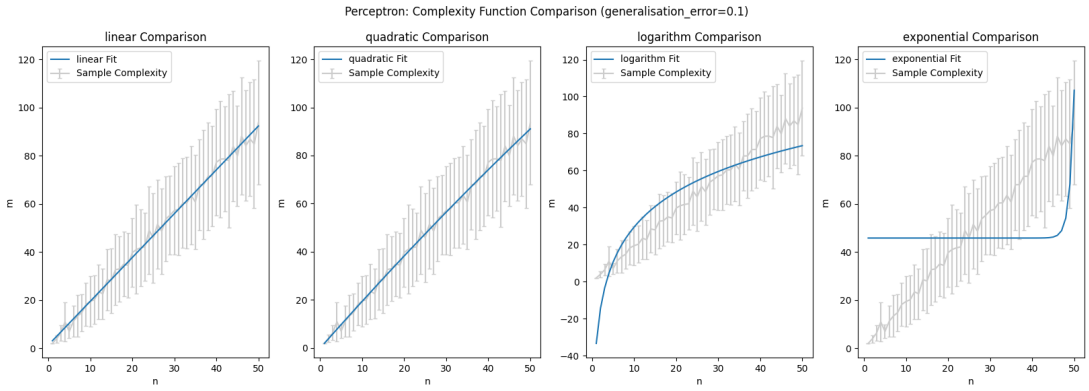


Figure 18: Perceptron Sample Complexity vs Function Classes

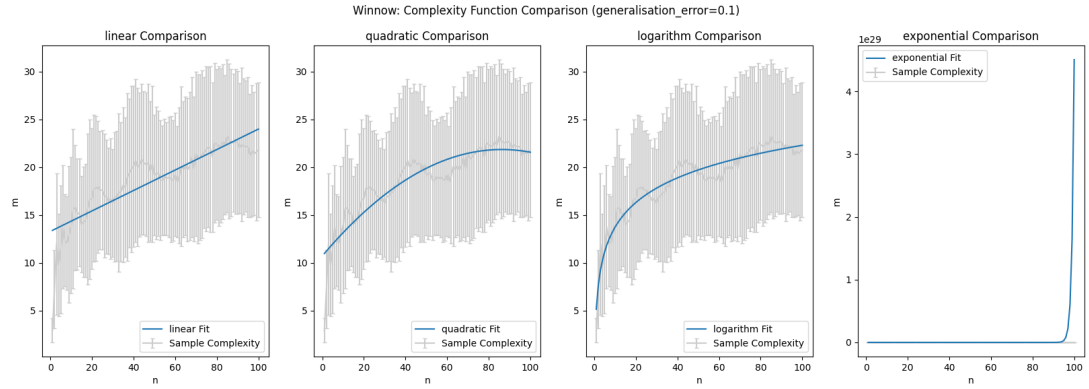


Figure 19: Winnow Sample Complexity vs Function Classes

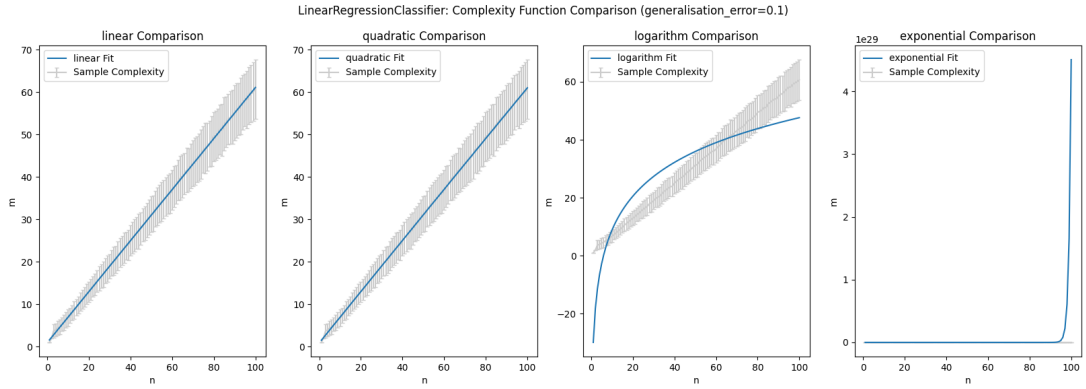


Figure 20: Least Squares Sample Complexity vs Function Classes

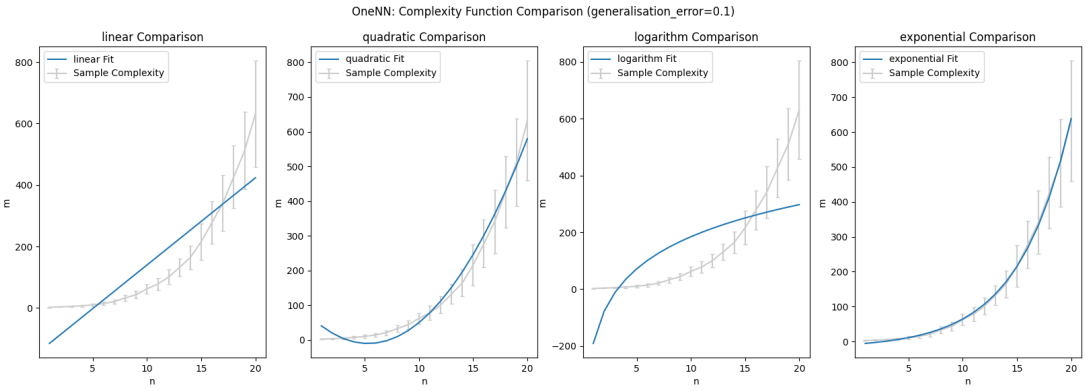


Figure 21: 1-Nearest Neighbours Sample Complexity vs Function Classes

From these plots we can see

Comparing the performance of the four algorithms

d. We note that since for any X_t , $y_t = X_{1,t}$, we have the following expression representing the linear separability of our dataset:

$$(v \cdot x_t) y_t \geq 1$$

Where $v = (1, 0, \ldots, 0)^T$. Note that $\|v\| = 1$.
Hence, in our online mistake bound for the perceptron, we have that $\gamma = 1$. Further note that $\forall t$, $x_t \in -1, 1^n \implies \|x_t\|^2 = n$. This gives us that $R = max_t \|x_t\| = \sqrt{n}$. Hence our mistake bound for the perceptron algorithm is given by:

$$M \leq n$$

.

Using the theorem on page 60 of the online learning notes, we arrive at:

$$Prob(\mathcal{A}_S(x') \neq y') \leq \frac{n}{m}$$

e. From our experimental observations we may expect the sample complexity of 1-NN to be lower bounded by some exponential function. We formalise this in the following proposition:

Proposition:

Following the data-generating distribution described above, we have that our sample complexity given by:

$m(n) = \Omega(n)$
Proof:

Suppose we sample a training set $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ uniformly from the set $\{-1, 1\}^n$, with associated labels defined by the rule $y|x = x_1$, and use this training set for inference on an arbitrary test point (x, x_1), sampled uniformly from the same set.
We note the following observation:

Obs:
if $x \in S$, then $\mathcal{A}_S(x)) = y$, where y is the true label for x.

To justify this, observe that our dataset represents a realiseable learning problem, and as such, if (x, y), $(x\prime, y\prime)$ are datapoints sampled from our distribution, $x = x\prime \implies y = y\prime$. Trivially, x is the 1-nearest neighbour to itself, so the algorithm makes a correct prediction for any datapoint present in our training set.

Hence, $\mathcal{A}_S$ makes an error on x $\implies x \notin S$.

Hence, the set of all training sets that make an error on x is contained in the set of all training sets not containing x.

Hence, for a given x, $P_S(\mathcal{A}_S(x) \neq y) \leq P_S(x \notin S)$.

Since S is a collection of points sampled iid from our data generating distribution,
$P_S(x \notin S) = \prod_{i=1}^{m} P(x_i \neq x) = \prod_i (1 - P(x_i = x)) = (1 - 2^{-n})^m$

We note that our choice of x was arbitrary, and since we sampled x uniformly, we arrive at the following generalisation error bound:

$\mathbb{E}_{S\sim\mathcal{D}^m, x\sim\mathcal{D}}[\mathcal{L}_S(x)] \leq (1 - 2^{-n})^m$
Using the identity $1 - x \leq e^{-x}$ provided frequently in the notes, we simplify this bound to give:

$\mathbb{E}_{S\sim\mathcal{D}^m, x\sim\mathcal{D}}[\mathcal{L}_S(x)] \leq exp(-2^n m)$
Suppose we seek m such that our generalisation error is less than some fixed ϵ.

Hence we require $\mathbb{E}_{S\sim\mathcal{D}^m, x\sim\mathcal{D}}[\mathcal{L}_S(x)] \leq exp(-2^n m) \leq \epsilon$

$\implies -m2^{-n} \leq log(\epsilon)$

Hence, $m \geq -log(\epsilon)2^n = log(\frac{1}{\epsilon})2^n$

$\implies m = \Omega(2^n)$

□