COMP0078 Coursework 2

Student No: 18010577 18092642

December 20, 2021

1 PART I

In this section of the report we will work with the dataset 'zipcombo.dat', which is part of the well known MNIST handwritten digit database. There are 9298 digits in the given dataset, each has a resolution of 16 by 16 pixels. Below are some examples:

Figure 1: Sample figure for each digits

Typical perceptron is a binary classifier which linearly combines all misclassified data samples to form a linear decision boundary. Then by Novikoff's Theorem, the algorithm will converge and can partition the sample space given the data are linearly separable. However in majority of the circumstances, the data are not linearly separable, in which case using kernel trick to map the data into higher dimension is necessary to produce a non-linear decision boundary conveniently.

In addition, the dataset consist of digits from 0 to 9 and therefore we would need to generalise the binary classification algorithm to a multi-class setting. There are 2 of the most classic type of methods to generalise from binary to multiclass: 'One-versus-All' and 'One-versus-One', we will discuss these 2 methods separately.

1.1 Generalisation Methods

One-versus-One Classification

we first propose a one versus one method to generalise the binary classification. Suppose we have k classes, then under this setting, we need to construct $k(k-1)/2$ classifiers each trained using only the data from 2 of the k classes therefore each classifier is responsible for distinguish 2 class of data. And when it comes to prediction, we will employ a voting system: given an unseen data, we feed it into every classifier and record the result, the class that have the most +1 prediction is the final output.

Algorithm 1 One versus one generalised Kernel Perceptron

Input: $(x_1, y_1), \ldots, (x_m, y_m) \in \mathbb{R}^n \times \{-1, 1\}$
compute Gram matrix K using the kernel function: $K_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$
initialise $\boldsymbol{w}^{(p)} = \mathbf{0}$ for p $\in \{1, \cdots, k(k-1)/2\}$

for $t \leq$ max iteration **do**
  receive data $\boldsymbol{x}_t, y_t$
  calculate $sign(\sum_i \boldsymbol{w}_i^{(mn)} K(\boldsymbol{x}_i, \boldsymbol{x}_t))$ and obtain the prediction of each classifier
  count the predictions and assign the class with the most vote as $\hat{y}_t$
  for p from 1 to $k(k-1)/2$ **do**
    if the classifier involves y_t and the classifier made the wrong prediction **then**
      $\boldsymbol{w}_t^{(p)} = \boldsymbol{w}_t^{(p)} + 1$ if y_t is the positive class
      $\boldsymbol{w}_t^{(p)} = \boldsymbol{w}_t^{(p)} - 1$ if y_t is the negative class
    end ifconvergence

One-versus-All Classification

We first propose the one versus all setting to generalise the binary classification algorithm [1]. The method requires us to generate multiple binary classifiers each dedicated to separate one class of data from the rest. In the end we would have multiple decision boundaries, when making predictions on unseen data:

$$\hat{y}_i = \arg\max_k w_k \cdot x_i$$

we can consider the dot product as a degree of confidence, the data is classified based on whose weight generate the greatest confidence. These confidence can also be interpreted as distance from boundary in positive direction.

Algorithm 2 One versus Rest generalised Kernel Perceptron

Input: $(x_1, y_1), \ldots, (x_m, y_m) \in \mathbb{R}^n \times \{-1, 1\}$
compute Gram matrix K using the kernel function: $K_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$
initialise $\boldsymbol{w}_1^{(k)} = \mathbf{0}$ for all k

for $t \leq$ max iteration **do**
  receive data $\boldsymbol{x}_t, y_t$
  make prediction $\hat{y}_t = \arg\max_k \sum_i \boldsymbol{w}_i^{(k)} K(\boldsymbol{x}_i, \boldsymbol{x}_t)$
  compute $\hat{y}_t y_t$
  if $\hat{y}_t y_t \leq 0$: **then**
    $\boldsymbol{w}_t^{(y_t)} = \boldsymbol{w}_t^{(y_t)} + 1$
    $\boldsymbol{w}_t^{(\hat{y}_t)} = \boldsymbol{w}_t^{(\hat{y}_t)} - 1$
  end ifconvergence

1.2 Some Preliminary

Early stopping

We first investigate the learning or convergence pattern of the algorithm in order to determine a suitable early stopping scheme. polynomial kernel with degrees of 3 and 5 are experimented with, these 2 models are trained using 80% of the data and their test error is evaluated using the rest. their learning curve is shown below:

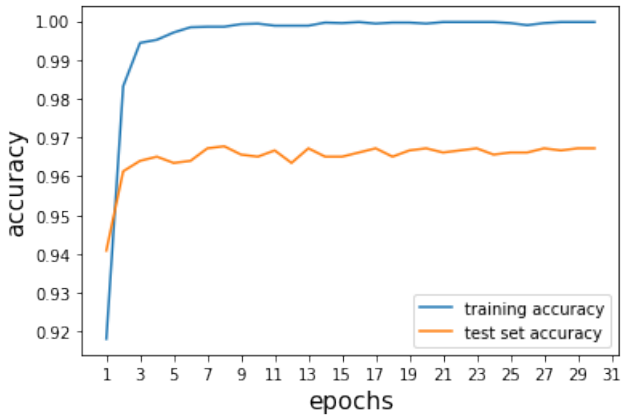


Figure 2: Sample figure for each digits

It shows that we can obtain adequate result after about 5 epochs, thus the minimum number of iteration is set to 5.

Prediction with randomised scheme

Traditionally, one-versus-all kernel perceptron makes prediction by finding the arg max of the output confidence matrix, but now we propose a randomised scheme to this classification methods. So alternatively we apply a softmax function to the output confidence to construct a discrete distribution, and prediction is made by random sampling from this distribution. However the resulting algorithm has shown no significant improvement and takes longer to converge.

Learning rate

Instead of updating the $\boldsymbol{w}$ by 1 per iteration, we can set a learning rate γ and update the weight by γ each time. However, after experimenting, setting a smaller learning rate makes no change to the speed of learning process apart from obtaining a smoother learning curve.

1.3 One-versus-rest polynomial kernel perceptron

basic result

To see how the algorithm performs using polynomial kernel of different degrees, we produce 7 separate models with different polynomial degree. Each of these models are trained and tested with a 80% and 20% random train test split for 20 runs, and the corresponding training, testing error are reported as averages over the 20 runs, with the same stopping criterion as earlier stated.
Generally speaking a higher degree will increase the model complexity and thus will yield a better accuracy. Furthermore, our experiment has not shown clear sign of overfitting as degree increases. The polynomial kernel with degree 1 has the highest mean error during both training and testing,

Polynomial degree d	Training error (%)	Test error (%)
1	5.864 ± 0.172	8.919 ± 1.172
2	0.996 ± 0.115	4.298 ± 0.513
3	0.382 ± 0.096	3.610 ± 0.577
4	0.218 ± 0.073	3.263 ± 0.429
5	0.161 ± 0.064	3.280 ± 0.444
6	0.130 ± 0.038	3.371 ± 0.476
7	0.102 ± 0.044	3.266 ± 0.428

Table 1: mean and standard deviation of training and test error rate over 20 runs with different polynomial kernel degree in percentage

following by kernel with degree 2. However the performance of other degrees are harder to distinguish and should all be suitable choices when training a kernel perceptron.

cross validation

In order to choose the most suitable polynomial degree, we perform 20 runs of model selection, each run consist of 5-fold cross-validating for each polynomial degree in order to choose a optimal degree, and retrain using the chosen degree on the holdout test set to generate test error. The result (also in percentage) is shown as follow:

Optimal d*	Test error rate in %
6	3.602
4	3.441
3	4.032
5	2.366
7	3.011
6	2.473
5	2.849
3	4.193
5	3.226
5	2.688
7	2.688
5	4.086
6	3.925
6	2.957
7	3.280
5	2.903
5	3.548
5	3.333
6	2.849
5	3.226

Table 2

From which we can calculate the the mean optimal polynomial degree is 5.30 ± 0.92 with a mean test error 3.228 ± 0.501. 5 is also the median within this 20 recorded degrees, therefore in future training, it should be the preferable choice for degree.

Confusion matrix

Another way of assessing the performance of kernel perceptron is through a confusion matrix, which can help us understand what error the algorithm is making and which pair of classes is harder for the algorithm to distinguish. The procedure is the same as before, we perform cross validation to select the optimal degree, retrain to compute the confusion matrix and average the entries over 20 runs.

TRUE \ PRED	0	1	2	3	4	5	6	7	8	9
0	0	0.009±0.03	0.118±0.178	0.118±0.241	0.102±0.229	0.18±0.233	0.322±0.288	0.048±0.12	0.065±0.194	0.038±0.119
1	0±0	0	0.088±0.241	0±0	0.372±0.372	0.027±0.109	0.208±0.269	0.006±0.019	0.144±0.241	0.006±0.027
2	0.107±0.146	0.05±0.15	0	0.168±0.259	0.232±0.241	0.069±0.19	0.05±0.143	0.185±0.204	0.099±0.163	0.041±0.125
3	0.114±0.218	0.013±0.037	0.153±0.209	0	0.025±0.109	0.363±0.283	0±0	0.07±0.153	0.232±0.26	0.03±0.075
4	0.03±0.109	0.178±0.242	0.127±0.149	0±0	0	0.045±0.119	0.137±0.193	0.08±0.179	0.028±0.109	0.326±0.296
5	0.162±0.214	0±	0.108±0.194	0.296±0.24	0.082±0.132	0	0.234±0.282	0.025±0.109	0.068±0.124	0.025±0.064
6	0.175±0.22	0.065±0.16	0.16±0.236	0±	0.341±0.294	0.132±0.22	0	0±	0.047±0.112	0.03±0.109
7	0.003±0.014	0.068±0.128	0.184±0.215	0.025±0.109	0.255±0.26	0.013±0.054	0±	0	0.068±0.178	0.385±0.332
8	0.081±0.12	0.032±0.084	0.078±0.147	0.355±0.212	0.06±0.126	0.167±0.204	0.102±0.179	0.059±0.125	0	0.066±0.129
9	0.037±0.117	0±	0±0.002	0.064±0.168	0.459±0.287	0.088±0.155	0±	0.337±0.284	0.014±0.032	0

Table 3: confusion matrix of optimal polynomial degree, entries average over 20 runs

The result is very interpretable, the algorithm is having a hard time telling apart digits that looks very similar, for example 3 and 5; 4 and 9; 7 and 9; 0 and 6. And vice versa, digits that are looks very different are much easier for the algorithm to classify and no misclassification is made between these pairs.

Most often misclassified data

We investigate further as when the algorithm makes mistake. This time the procedure consist of 40 runs, for each run model is trained using 80 % of the data and evaluated using the entire dataset, data which are misclassified are recorded during each run. In the end, the 5 data which are most often misclassified are displayed down below:

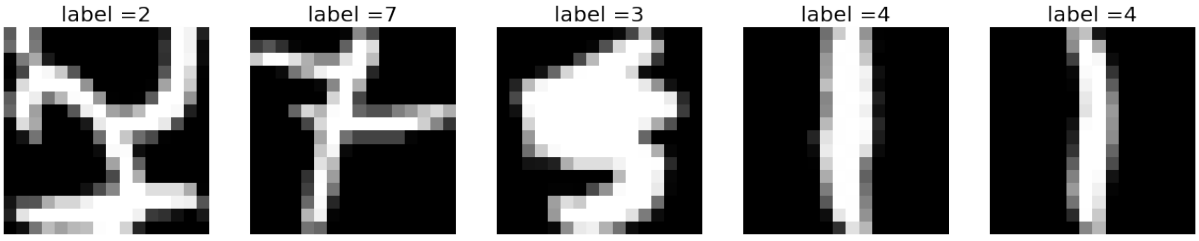


Figure 3: Sample figure for each digits

the twp digits 1 is never classified correctly since they are incorrectly labelled or written, their labels are both 4. On the other hand the 3 digits are handwritten in a messy manor, it is hard to recognise even for us. Therefore it is reasonable that our model is able to achieve high accuracy but never 100% as we can imagine there are more data like these 5.

1.4 One-versus-rest Gaussian Kernel Perceptron

Alternatively, we can use Gaussian kernel as our kernel function. An advantage of this particular kernel function is that it allows us to map the input into infinite dimensional feature space, allowing the algorithm to have high complexity whilst retaining computational efficiency. The kernel function is:

$$K(p, q) = exp(-c\|p - q\|^2)$$

Now c corresponds to $\frac{1}{2\sigma^2}$, which controls how influential a datapoint has on the prediction depending on its distance with the point of interest. The larger c is, data further away will have less relative influence.

Basic results

We investigate in what are the feasible range for parameter c. Initially we have experimented with $c \in \{10^0, \cdots, 10^{-7}\}$ and we found that for c smaller than 10^{-4}, test error rate become significantly high. Therefore the feasible range is adjusted to $\{3^{-1}, \cdots 3^{-7}\}$ and using the same procedure as before, the training and testing error rate is reported:

c	Training error (%)	Test error (%)
3^{-1}	0 ± 0	6.132 ± 0.649
3^{-2}	0.007 ± 0.012	6.1 ± 0.450
3^{-3}	0.029 ± 0.022	3.392 ± 0.358
3^{-4}	0.116 ± 0.038	3.097 ± 0.449
3^{-5}	0.842 ± 0.161	4.097 ± 0.694
3^{-6}	4.393 ± 0.267	6.656 ± 1.676
3^{-7}	8.778 ± 0.272	10.501 ± 4.141

Table 4: mean and standard deviation of training and test error rate over 20 runs with different c in percentage

Initially, width parameter c is a tab too large, generalisation error is slightly higher perhaps data further away contribute too less. As c decreases to 3^{-4}, we can see that generalisation error also decreases. And for even lower values of c, it is clear that the data further away contributes too much, causing predictions to be biased. This effect is similar to k-NN algorithm.

Cross validation

Using the same procedure, we search for the optimal value for c, from the table below, it is clear that 3^{-4} is the best with a mean test error rate of 3.228 ± 0.442. The result is strikingly similar to the one using polynomial kernel. Therefore for this data, we are unable to fully utilise the expressive potential of Gaussian kernel. Or perhaps the optimal feasible range for d is not found.

Optimal d*	Test error rate in %
3^{-4}	3.387
3^{-3}	3.064
3^{-4}	3.763
3^{-4}	2.795
3^{-4}	2.688
3^{-4}	3.494
3^{-4}	3.494
3^{-4}	3.279
3^{-3}	2.849
3^{-4}	3.763
3^{-4}	3.064
3^{-4}	3.655
3^{-4}	2.956
3^{-4}	3.763
3^{-4}	2.634
3^{-4}	2.903
3^{-3}	4.301
3^{-4}	2.903
3^{-4}	2.795
3^{-4}	3.010

Table 5

1.5 One-versus-one generalised polynomial kernel perceptron

Basic results

Procedures are repeated to check the performance of One-versus-one generalised polynomial kernel perceptron. And the results are very similar to that of one-versus-rest version.

Polynomial degree d	Training error (%)	Test error (%)
1	6.246 ± 0.216	7.384 ± 0.942
2	2.287± 0.206	4.387± 0.487
3	1.442± 0.16	3.817± 0.425
4	1.036 ± 0.104	3.481 ± 0.518
5	0.881 ± 0.145	3.476 ± 0.735
6	0.789 ± 0.105	3.548 ± 0.469
7	0.743 ± 0.1	3.651 ± 0.538

Table 6: mean and standard deviation of training and test error rate over 20 runs with different polynomial kernel degree in percentage

Cross validation

Again the results are almost identical to one-versus-rest version with mean optimal degree remain to be 5.1 with standard deviation of 0.84.

Optimal d*	Test error rate in %
5	2.849
6	3.387
5	3.548
6	4.032
4	3.871
5	2.849
6	3.710
4	3.279
5	3.172
4	3.387
6	3.225
5	3.064
4	3.225
5	3.225
4	3.763
6	3.548
4	3.494
6	3.118
6	4.247
5	3.763

Table 7

Evaluation between OvR and OvO

comparing these 2 approaches, it is obvious that they consist of different number of classifiers. The handwritten digits contains 10 digits, therefore One-versus-All approach need 10 classifiers, One-versus-One on the other hand need a whopping 45 classifier. However each classifier in One-versus-One will take less data to train. In the experiment I have found that each epoch of OvO approach takes longer to train, however it converges very fast: 3 epochs already yield adequate result. Therefore, in our case these 2 approaches are all suitable.

One drawback of one-versus-one method is that because we are using a voting system, there can potentially be tied voting. Furthermore, if there are more classes, we can expect the computation time will increase further for OvO approach, since counting the vote to produce a prediction takes a long time already.

On the other hand, there is a problem when training a OvR version of perceptron: each classifier need to deal with severe class imbalance , thus we might need to employ a random sampling scheme to select data in the "rest" category, in restore balance.

2 PART II

2.1 Spectral Clustering Implementation

First, we implement the algorithm to the classical 2 moons dataset. After experimenting, we have found that a suitable c parameter is greater or equal to $2^{4.3}$, and the algorithm has managed to separate the class perfectly. This is probably because the distances between different group is consistently farther than within group neighboring distances.

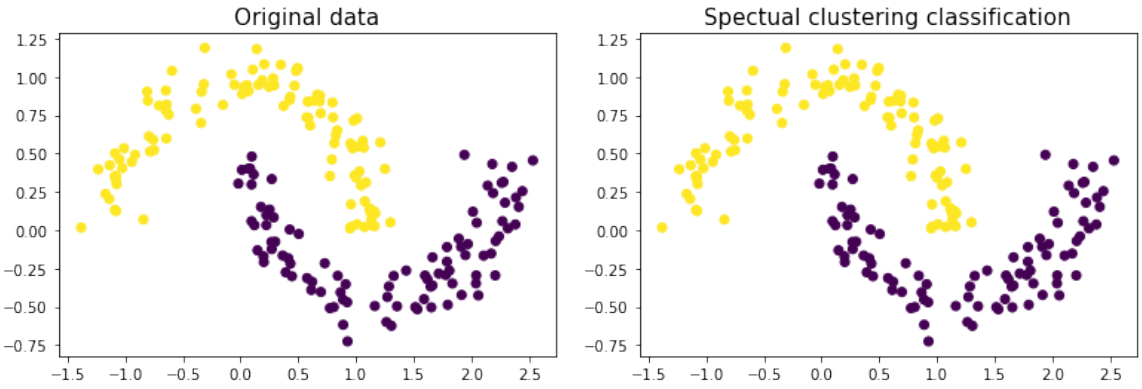


Figure 4: twomoon dataset

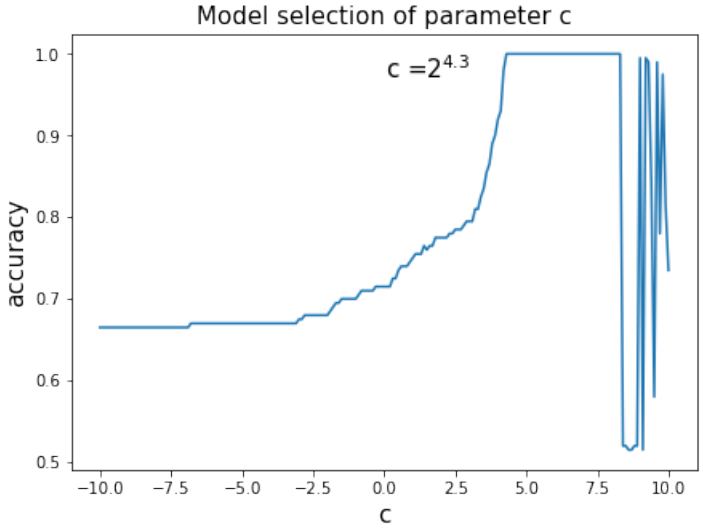


Figure 5

However, if c gets too large the performance becomes unstable, probably because the weights of datapoints further away are too similar to neighboring data's.

In**experiment 2**, dataset generated with isotropic Gaussians, in this case we cannot find an optimal c, probably because of the small sample size, we cannot obtain a stable distribution pattern, every time the data looks very different. And now misclassification becomes very likely since it is the 2 Gaussian distributions overlap and we cannot find an optimal parameter c.

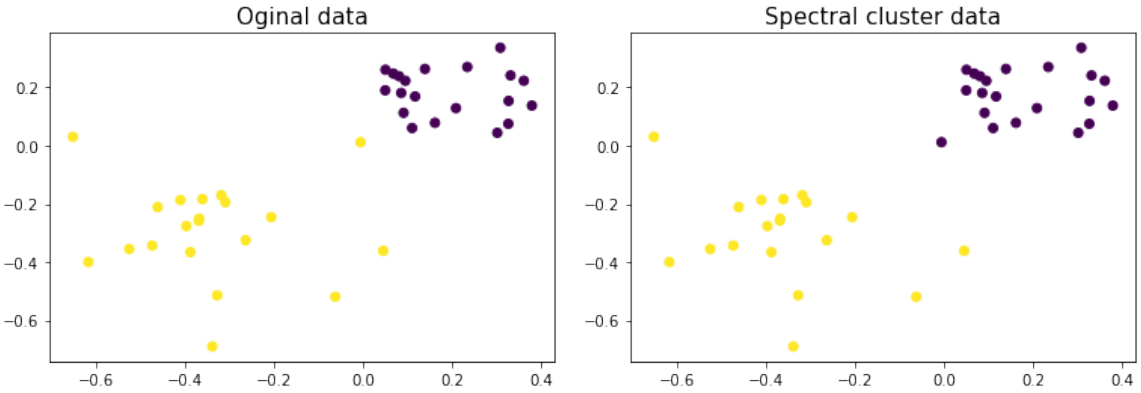


Figure 6: Isotropic Gaussian dataset

In **experiment 3**, the algorithm is implemented to a higher dimensional data handwritten digits 1 and 3. Now, the optimal parameter c is 0.0112, any values greater than 0 and less than 0.0112 is acceptable, and greater values generate worse result, probably because Gaussian kernels are too small and relative similar in values.

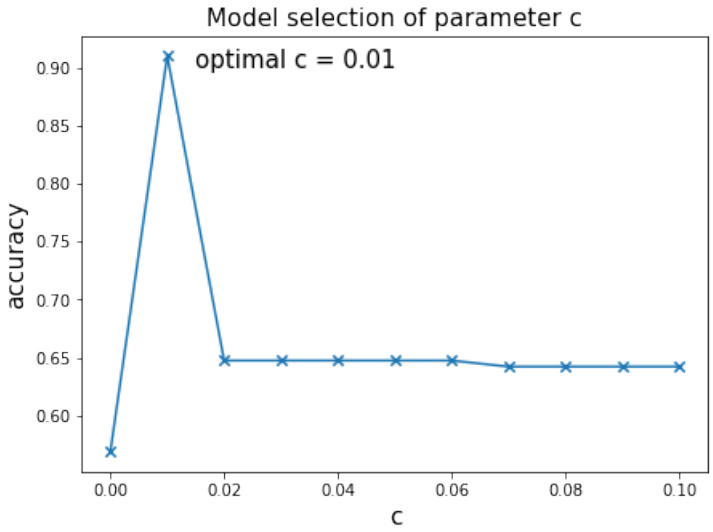


Figure 7

2.2 Questions

1. Since this implementation of spectral clustering algorithm makes prediction by extracting signs of entries in the second eigenvector. And according to the definition of eigenvectors: v is equivalent to $-v$ and are both considered as the same eigen vector. Therefore although the algorithm can distinguish 2 classes, it does not necessarily produce the correct label since vector $v \equiv -v$, they are merely in different directions. And therefore the algorithm might have correctly classified the data but with completely opposite label In which case, to calculate the correct accuracy we need

to flip the signs of output. This is equivalent to the function:

$$CP(c) := \frac{\max(\ell_-, \ell_+)}{\ell}$$

2. Since (by article)[2] we can show that:

$$\begin{aligned} f^T L f = f^T D f - f^T W f &= \sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j=1}^{n} f_i f_j w_{ij} \\ &= \frac{1}{2}(\sum_{i=1}^{n} d_i f_i^2 - 2\sum_{i,j=1}^{n} f_i f_j w_{ij} + \sum_{i=1}^{n} d_i f_i^2) \\ &= \frac{1}{2}(\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2) \geq 0 \qquad \text{for all } f \in \mathbb{R}^n \end{aligned}$$

therefore the Laplacian matrix is positive semi-definite and all its eigenvalues are positive. In addition observe that $d_i = \sum_j w_{ij}$ and $L = D - W$, meaning every row of L sum to zero and therefore:

$$Lv = 0 \cdot v \quad for \;\; v = (1, 1, \cdots, 1)^T$$

thus 0 and $\underline{\mathbf{1}}$ is an eigenvalue and eigenvector pair of L. And since L is positive semi-definite, its eigenvalues are greater or equal to 0. Thus 0 must be its smallest eigenvalue whose eigenvector is a constant vector.

3. We consider the entire dataset as a fully connected graph $G(V, E)$ with similarities between datapoints assigned to edges [2]. If we wish to bipartite the graph, the most sensible way of partitioning is the sparsest cut [3]. In the article, this is referred to partitioning the graph which minimise the $RatioCut = \sum_i^k \frac{cut(A_i, \overline{A}_i)}{|A_i|}$. It can be shown that the above problem is equivalent to minimising $\frac{\alpha^T L \alpha}{\alpha^T \alpha}$ [4] where α contains partitioning information and L is the graph laplacian. This is the min or maximisation problem of quadratic form on unit space which we have seen in PCA and CCA etc. Since this is a minimisation problem, the solution is the smallest eigenvalue-vector pair, but since the smallest eigenvalue carry no significant information, we use the second smallest or k smallest pairs to cluster of data instead of in original dimensions.

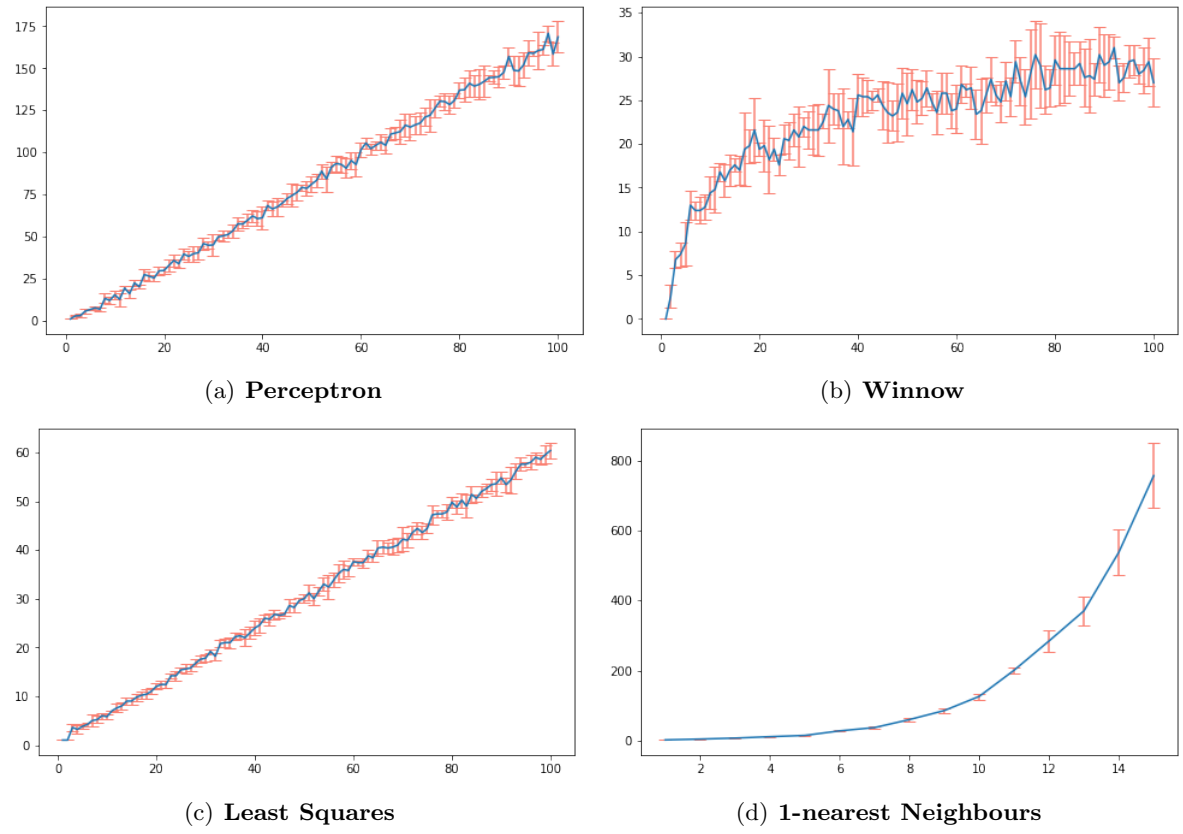
4. Since this is a gaussian kernel, c correspond to $\frac{1}{2\sigma^2}$. Therefore the value of c determines the spread of the gaussian distribution around the point of interest. If another point lies outside the standard dedviation of the spread the similarity value w_{ij} will be small. Therefore a relatively larger c means we expect a tighter spread, and data further are more likely to be have lower similarity and we can expect a better result. Conversely smaller c could cause data from different clusters or different distances to have similar weights, this may result in a lower accuracy. However c cannot be too large neither since it will make everything converge to 0, this effect is shown in figure 5: when c exceeds 2^8, the performance varies.

3 PART III

3.1 Sparse Learning

In the following problem we will consider the sample complexity of the **perceptron, winnow, least squares**, and **1-nearest neighbours** algorithms for a specific problem.

(a) Implement the four classification algorithms and then use them to estimate the sample complexity of these algorithms.
Because **1-nearest neighbours** algorithm still runs very slowly after 'simplifying' the testing process, we only used dimension n from 1 to 15 for this algorithm (Figure 8 (d)).



(a) **Perceptron** (b) **Winnow**

(c) **Least Squares** (d) **1-nearest Neighbours**

Figure 8: Estimated number of samples (m) to obtain 10% generalisation error versus dimension (n) for these algorithms.

(b)

- Please describe your method for estimating sample complexity in detail.

  Because the sample complexity would be extremely expensive computationally, we want to 'simplify' the testing process to approximate the value. From the note given, the generalisation error is $\varepsilon(\mathcal{A}_S) := 2^{-n} \sum_{x \in \{-1,1\}^n} I[\mathcal{A}_S(x) \neq x_1]$
  We use $\varepsilon(\mathcal{A}_S) := \frac{1}{S} \sum_{x \in \{-1,1\}^n} I[\mathcal{A}_S(x) \neq x_1]$ to estimate generalisation error, S is the test size.
  The sample complexity on average at 10% generalisation error is $\mathcal{C}(\mathcal{A}) := min\{m \in \{1, 2, \dots\} : \mathbb{E}[\varepsilon(\mathcal{A}_{S_m})] \leq 0.1\}$

First we implement the four classification algorithms and the functions return the predicted value on test data set. With fixed sizes of train data sets and testing data sets, we runs for 10 times to calculate the average generalisation error.
After that, we start from $m = 1$ for each dimension n and increase m by 1 each time until the average generalisation error for current m is smaller or equal to 0.1. The sample complexity will be a list containing the latest m for each dimension n.
To get the expected sample complexity and its standard deviation (to plot error-bar plots), we repeat the previous steps for 10 runs and calculate the mean and standard deviation.
In practice, we set the test size to be 2000 for 1-nearest Neighbours and 5000 for others. The runs to getting mean and std we chose is 5 for 1-nearest Neighbours and 10 for others. The dimension n we chose is from 1 to 15 for 1-nearest Neighbours and from 1 to 100 for others.
Each setting for 1-nearest Neighbours is smaller because it runs very slowly under larger quantities and we are able to see the trend for 1-nearest Neighbours under this setting.

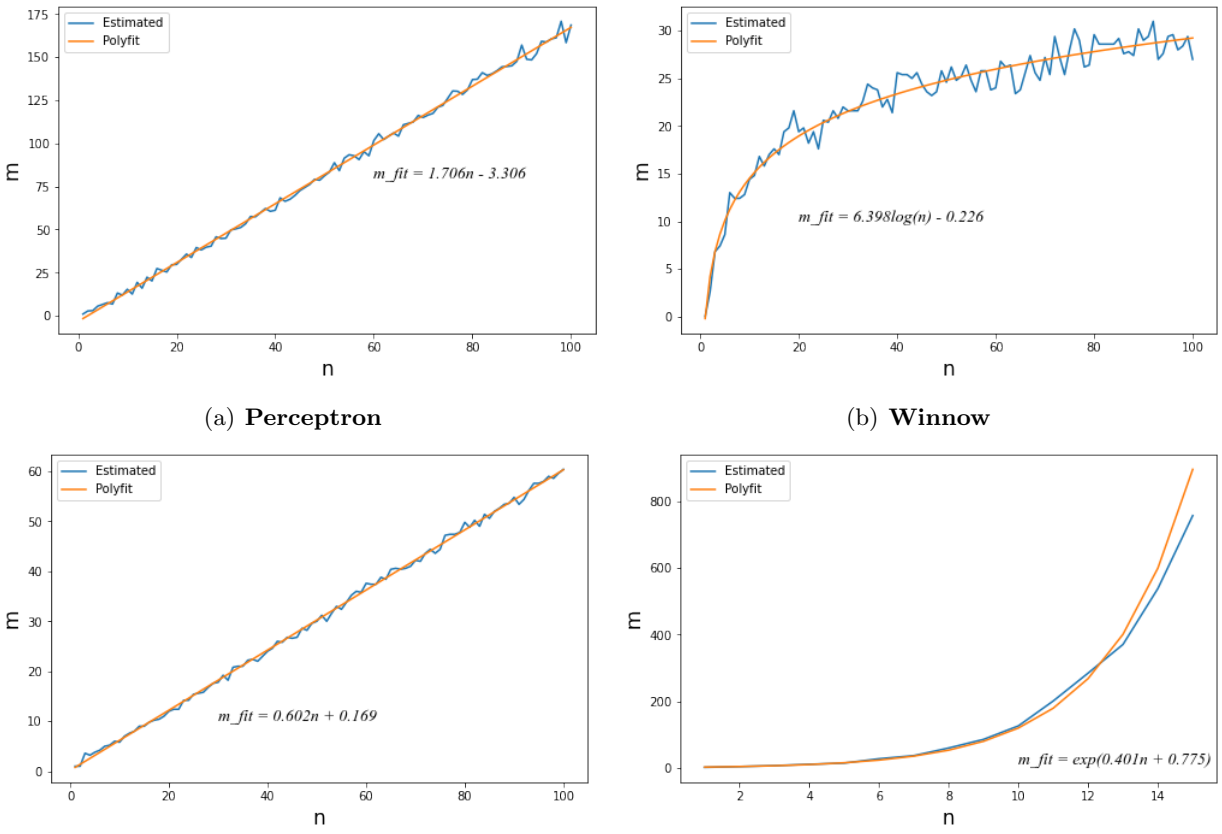
- Please discuss the tradeoffs and biases of your method.

  We need to trade-off accuracy and computation time. If we choose more test size, we will get more accurate results and lower generalisation error but it is time-costing at the same time. If we increase the number of repeating runs, we will get lower standard deviations.
  From the errorbar plots above, we see the standard deviations for the Least Squares and Perceptron are small and stable. However, for winnow, the standard deviations have obvious fluctuations. For 1-nearest Neighbours, though we only chose n from 1 to 15, we can observe that the standard deviations increase with the increasing dimension.
  Since the sample size we chose is relatively small comparing to 2^n, variance might be higher than it should be and there might be a departure from the true curve. If computational power allowed, we could reduce this variance and eliminate potential biases in algorithm output.

(c) Estimate how m grows as a function of n as $n \to \infty$ for each of the four algorithms based on experimental or any other "analytical" observations.
The figures below show the estimated number of samples and their fitted lines with formulas.
The sample complexity grows linearly as dimension increases for both Perceptron and Least Squares, however comparing to perceptron least square algorithm requires less training samples to achieve sufficient accuracy. The sample complexity grows logarithmically for Winnow and grows exponentially for 1-nearest Neighbour. Therefore overall **Winnow** requires least samples to achieve 10% error rate, possibly because that the weight for correct expert is doubled each time. However the drawback is that there is high fluctuation as to how many samples are required for higher dimensions, since there is a random factor during training.

On the other end, 1-nn algorithm is the most sensitive to dimensionality increase, the minimum requirement blows up rapidly.

- Perceptron: $\Theta(n)$
- Winnow: $\Theta(log(n))$
- Least Squares: $\Theta(n)$
- 1-nearest Neighbours: $\Theta(exp(0.401)^n)$



(a) **Perceptron** (b) **Winnow**

(c) **Least Squares** (d) **1-nearest Neighbours**

Figure 9: Estimated and fitted number of samples (m) to obtain 10% generalisation error versus dimension (n) for these algorithms.

(d) Derive a non-trivial upper bound $\hat{p}_{m,n}$ on the probability that the perceptron will make a mistake.
From the Theorem Perceptron Bound (Novikoff on notes), we have the mistakes of the perceptron is bounded by $M \leq \left(\frac{R}{\gamma}\right)^2$.
$R := max_t \|x_t\| = \sqrt{\sum_{i=1}^{n} x_{t,i}^2} = \sqrt{n}$ since $x_{t,i} \in \{-1, 1\}$.
By definition, $x_i \in \{-1, 1\}^n$ and $y_i \in \{-1, 1\}$, we have $M \leq \left(\frac{\sqrt{n}}{\gamma}\right)^2 = \frac{n}{\gamma^2} = n$
$\implies \hat{p}_{m,n} \leq \frac{n}{m}$ ($Prob(A_s(x') \neq y') \leq \frac{B}{m}$ from notes)

(e) Find a simple function $f(n)$ which is a good lower bound of the sample complexity of 1-nearest neighbor algorithm for the 'just a little bit' problem. Prove $m = \Omega(f(n))$.

References

1. *Bandit Multiclass Linear Classification: Efficient Algorithms for the Separable Case* http://proceedings.mlr.press/v97/beygelzimer19a/beygelzimer19a-supp.pdf (2021).
2. Von Luxburg, U. A tutorial on spectral clustering. *Statistics and Computing* **17,** 395–416. https://arxiv.org/pdf/0711.0189.pdf (Aug. 2007).
3. Contributors, W. *Cut (graph theory)* Wikipedia, Aug. 2019. https://en.wikipedia.org/wiki/Cut_(graph_theory).
4. *Principals of Spectral Clusteringcsdn blog* blog.csdn.net, 111 2019. https://blog.csdn.net/june_young_fan/article/details/103051259 (2021).