Supervised Learning (COMP0078) CW2

Team(YES)
16040463
21070177

December 2021

1 Part 1

1.1 Kernel perceptron (Handwritten Digit Classification)

1.1.1 Introduction

In this part of the report, we will be covering the methodology to classify Handwritten Digit with multi-class kernel percepton. The Handwritten Digit data set can be represent as a 9298×257 matrix. It has 9298 sample data points and each represented in a row. The first column contains the class labels which denotes the digit. The rest 256 columns store the grey values scaled between -1 and 1. We will be implementing 6 experiments on this data set to discuss the following topics:
1. 2 methods chosen for generalising 2-class classifiers to k-class classifiers.

- The fist method is called 'One vs All' and we will be using it through Experimental Protocol 1 to 5. Implementing this method to perceptron will be talked in detail in Experimental Protocol 1.
- The second method is called One vs One. Implementing this method to perceptron will be talked in detail in Experimental Protocol 6.

A comparison of these two methods and discussion will be given after we implemented both methods and saw the results of Experimental Protocol 6.

2. How cross-validation improves our algorithm

- We implement a 5-fold cross-validation to the multi-class kernel perceptron algorithm. The detailed introduction of the code will be given in Experimental Protocol 2 and we use the same cross-validation method for Experimental Protocol 5 and 6.

A discussion of the effect of cross-validation will be given after we see the classification result in Experimental Protocol 2, 5 and 6.

3. Implementing polynomial kernel and Gaussian kernel to the perceptron

- Polynomial kernel will be used throughout Experimental Protocol 1-4 and 6. The detail of how to generalise the perceptron to use kernel will be described in Experimental Protocol 1.
- Gaussian kernel will be implemented in Experimental Protocol 5.

A comparison of these two kernels will be given after we see the result after Experimental protocol 5.

1.1.2 Experimental Protocol 1

One vs All method
Assuming the data are linearly separable, the 2-class perceptron learns a binary classifier

$$sign(\boldsymbol{w}(\boldsymbol{x}))$$

where $\boldsymbol{w}$ is the learned weight vector. The 2-class perceptron can be extend to a k-class perceptron by maintaining a weight for each class. This gives us k classifiers, each discriminating its corresponding class to all others and corresponds to a hyperplane $\boldsymbol{w}_c^T\boldsymbol{x} = 0$. The binary label is not sufficient to discriminant k-classes at once so here we compute the confidence of the example being in each classifier and the predicted class of this example will be the class with the highest confidence.The classifier then takes the form

$$argmax_c(\boldsymbol{w}_c^T\boldsymbol{x}), \text{for c = 1,...,k}$$

where $\boldsymbol{w}_c$ is the learned weight vector for each class.
We solved the prediction step and now we need to solve the update step. If the predicted class label matches the true class label then we keep the weight as it is. If the prediction is wrong then we update the weight of the true class and the weight of the predicted class by adding $\boldsymbol{x}$ to the weight of the true class and add $-\boldsymbol{x}$ to the weight of the predicted class. Weights of other classes remains unchanged, i.e.

$$\boldsymbol{w}_{c(t+1)} = \begin{cases} \boldsymbol{w}_{ct} + \boldsymbol{x}_t, c = \hat{y}_t \\ \boldsymbol{w}_{ct} - \boldsymbol{x}_t, c = y_t \\ \boldsymbol{w}_{ct}, c \in \{1, ..., k\} \setminus \{\hat{y}_t, y_t\} \end{cases}$$

Implementing Kernel
In the previous paragraph we made an assumption that the data are linearly separable but it is sometimes violated. We need to use feature map ϕ to map data to a space where data is linearly separable. If the feature map is carefully chosen then the inner product $\phi(\boldsymbol{x}_i)\cdot\phi(\boldsymbol{x}_j)$ can be computed by a kernel function

$K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Therefore, in practice, we sometimes choose kernel function directly without finding the actual feature map.
By implementing kernel on perceptron, our weight vectors at iteration 1 can be written as

$$\boldsymbol{w}_{c(1)} = \begin{cases} \phi(\boldsymbol{x}_1), c = \hat{y}_1 \\ -\phi(\boldsymbol{x}_1), c = y_1 \\ 0, c \in \{1, ..., k\} \setminus \{\hat{y}_1, y_1\} \end{cases}$$

Then in the prediction step, the confidence vector could be expressed as

$$\boldsymbol{w}_{c(1)}\phi(\boldsymbol{x}_2) = \begin{cases} \phi(\boldsymbol{x}_1) \cdot \phi(\boldsymbol{x}_2) = K(\boldsymbol{x}_1, \boldsymbol{x}_2), c = \hat{y}_1 \\ -\phi(\boldsymbol{x}_1) \cdot \phi(\boldsymbol{x}_2) = -K(\boldsymbol{x}_1, \boldsymbol{x}_2), c = y_t \\ 0, c \in \{1, ..., k\} \setminus \{\hat{y}_t, y_t\} \end{cases}$$

This can be extend to following iterations, and we spot that the confidence vector is sum of kernel functions. Therefore, we build a matrix $\boldsymbol{\alpha}$ with dimension $(k \times T)$, where T is the number of total iteration, and initially set to $\mathbf{0}$. When we failed to predict the true label y_t, we will add 1 to $\alpha_{y_t t}$ and add -1 to $\alpha_{\hat{y}_t t}$. In the prediction step, the confidence vector will be given by $\boldsymbol{\alpha}\boldsymbol{K}$ where $\boldsymbol{K}$ is the kernel matrix of the data i.e. $K_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
In practice, it is difficult to chose a appropriate kernel by hand so we will do it by computing the training and testing error generated by different kernels.

Online training
In online training we repeatedly cycle through the training set, and each cycle is called an epoch. For each epoch, we will be using the same training data, therefore we will be using the same kernel matrix. By observing this, we can use a small trick to save time, we could keep updating the alpha matrix we had in epoch 1 by adding and subtracting on the relevant entry according to the update rule we mentioned in 'One vs All method'. We do not need to build a giant alpha matrix and expand the training data set.

Combining the One vs All method and Implementing Kernel, we now have the algorithm as follow:

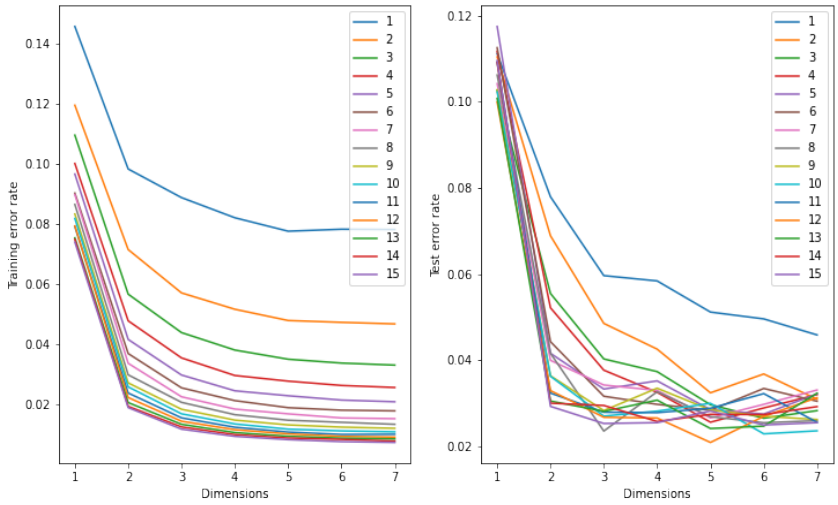
Training process:

	k-class Kernel Perceptron with One vs All Method (Online training)
Input:	$\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_m, y_m)\} \in (\mathbf{R}^n, \{0, ..., k\})^m, \mathbf{X} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_m)^T$
Initialization:	$\boldsymbol{\alpha}_{(k\times m)} = \mathbf{0}$
Repeat the following **Update** and **Prediction** steps for e times (e = number of epochs)	
Prediction:	Upon receiving the tth instance $\boldsymbol{x}_t$, predict $\hat{y}_t = argmax_c(\sum_{i=0}^{t-1} \alpha_{ci} K(\boldsymbol{x}_i, \boldsymbol{x}_j))$, $t = 1, ..., m$
Update:	if $\hat{y}_t = y_t$, then $\alpha_{ct} += 0, \forall c = 1, ..., k$ else $\alpha_{\hat{y}_t} = -1$, $\alpha_{y_t} += 1$, $\alpha_c += 0$ for $c \in \{1, ..., k\} \setminus \{\hat{y}_t, y_t\}$
Output:	$\boldsymbol{\alpha}$

Predicting Process

	k-class Kernel Perceptron with One vs All Method (Predicting)
Input:	Test data set:$\{(\boldsymbol{z}_1, y_1), ..., (\boldsymbol{z}_r, y_r)\} \in (\mathbf{R}^n, \{0, ..., k\})^r, \mathbf{Z} = (\boldsymbol{z}_1, ..., \boldsymbol{z}_n)^r$ Trained classifiers: $\boldsymbol{\alpha}$ Training data set: $\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_m, y_m)\} \in (\mathbf{R}^n, \{0, ..., k\})^m$, $\mathbf{X} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_m)^T$
Prediction:	For the tth test data point $\boldsymbol{z}_t$, predict $\hat{y}_t = argmax_c(\sum_{i=0}^{m} \alpha_{ci} K(\boldsymbol{x}_i, \boldsymbol{z}_r))$, $t = 1, ..., r$
Output:	Set of prediction of test data points: $\{\hat{y}_1, ..., \hat{y}_r\}$

Choice of epoch number One more thing left to do is to choose an appropriate number of epochs. Having too many epochs could cause overfitting and a small number of epochs could cause underfitting. We choose the optimal epoch number by running the algorithm with different epoch number and comparing the training error and testing error. We run the k-class Kernel Perceptron with One vs All Method for e = 1,...,15, where e is the number of epochs.



We can see that the line for training errors with 11 to 15 epochs starts to overlapping that means the increase of number of epochs does not improve the classifier anymore. We can also see the lines for test errors with epoch number lager or equal to 11 starts to come up to the lines for test errors generated by lower epoch numbers which means the models has a worse generalization and starts to overfitting. Therefore, we chose to train the model with **10** epochs and stay with this epoch number for all the rest Experimental Protocols.

Data table
We then implement the algorithm on the Handwritten dataset with d = 1,...,7 and get the following table.

Mean train and test error of each dimension

d	1	2	3	4	5	6	7
Mean train error rate+std	8.16% ±0.004	2.59% ±0.001	1.67% ±0.002	1.32% ±0.0004	1.19% ±0.0005	1.1% ±0.0005	1.09% ±0.0004
Mean test error rate+std	10.25% ±0.016	4.39% ±0.012	3.40% ±0.009	3.50% ±0.008	3.27% ±0.009	3.30% ±0.012	3.76% ±0.011

1.1.3 Experimental Protocol 2

Here we are still using the 'k-class Kernel Perceptron with One vs All Method' we had in Experimental Protocol 1. We find the best dimension d^* by performing

cross-validation.

Mean test error rate ±std	Mean d^*±std
2.2%±0.0097	4.8 ± 1.08

1.1.4 Experimental Protocol 3

Each entry $\mathbf{M}_{ij}$ of the confusion matrix contains the mean rate and standard deviation of miss-classifying the digit i into the digit j. The mean error rate is the number of miss-classifying the digit i into the digit j over the number of digit i in test data set. We break the matrix into two parts to fit in the page.

Confusion Matrix

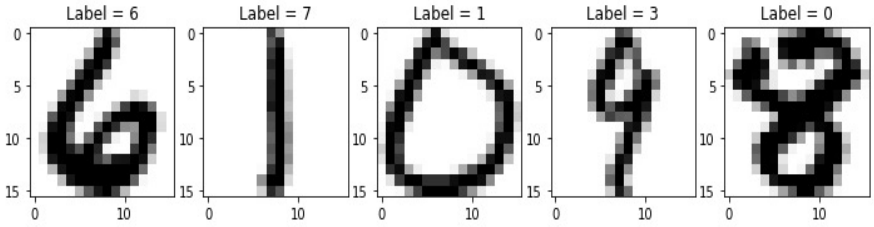
	0	1	2	3	4
0	0.0% ± 0.0	0.0 % ±0.0	0.36 % ±0.0029	0.23 % ±0.0024	0.07 % ±0.0017
1	0.0% ± 0.0	0.0 % ±0.0	0.1 % ±0.0018	0.0 % ±0.0	0.1 % ±0.0018
2	0.3 % ±0.0064	0.19 % ±0.003	0.0 % ±0.0	0.55 % ±0.0042	0.63 % ±0.0043
3	0.18 % ±0.0031	0.2 % ±0.0036	0.73 % ±0.0074	0.0 % ±0.0	0.02 % ±0.001
4	0.13 % ±0.0026	0.63 % ±0.004	0.55 % ±0.0044	0.05 % ±0.0014	0.0 % ±0.0
5	0.62 % ±0.0053	0.03 % ±0.0013	0.43 % ±0.0051	1.36 % ±0.0108	0.35 % ±0.0037
6	0.63 % ±0.0055	0.12 % ±0.003	0.29 % ±0.0044	0.0 % ±0.0	0.42 % ±0.0061
7	0.0 % ±0.0	0.2 % ±0.0043	0.44 % ±0.0061	0.05 % ±0.0017	0.61 % ±0.0119
8	0.85 % ±0.0061	0.52 % ±0.0058	0.39 % ±0.0061	1.31 % ±0.0094	0.71 % ±0.0062
9	0.12 % ±0.0026	0.1 % ±0.0036	0.22 % ±0.0042	0.16 % ±0.003	1.83 % ±0.0157

	5	6	7	8	9
0	0.3% ± 0.0028	0.19 % ±0.0019	0.02 % ±0.0008	0.13 % ±0.0028	0.01 % ±0.0006
1	0.02% ± 0.001	0.07 % ±0.002	0.05 % ±0.0013	0.06 % ±0.0018	0.05 % ±0.0018
2	0.11% ± 0.003	0.12 % ±0.003	0.47 % ±0.0049	0.43 % ±0.0066	0.05 % ±0.002
3	1.8% ± 0.0168	0.0 % ±0.0	0.49 % ±0.0065	0.88 % ±0.0071	0.13 % ±0.0027
4	0.31% ± 0.0052	0.56 % ±0.0054	0.23 % ±0.0042	0.3 % ±0.004	0.77 % ±0.0071
5	0.0% ± 0.0	0.67 % ±0.0093	0.13 % ±0.0027	0.71 % ±0.0054	0.51 % ±0.006
6	0.53% ± 0.0073	0.0 % ±0.0	0.0 % ±0.0	0.43 % ±0.0045	0.0 % ±0.0
7	0.17% ± 0.0026	0.0 % ±0.0	0.0 % ±0.0	0.14 % ±0.0025	1.15 % ±0.0091
8	0.76% ± 0.0069	0.3 % ±0.006	0.59 % ±0.0056	0.0 % ±0.0	0.22 % ±0.0037
9	0.13% ± 0.0026	0.0 % ±0.0	1.25 % ±0.0117	0.09 % ±0.0024	0.0 % ±0.0

1.1.5 Experimental Protocol 4

According to the confusion matrix we got from Experimental Protocol 4, digits 6, 1, 0, 9 and 8 are the 5 hardest to classify for our algorithm. This is reasonable

since we can see from the graph that digit 7 has overlapping pixels with both digit 0, 9 and 8. Digit 1 has overlapping pixels with digit 9 . When two images have overlapping pixels, the entries of the vector, which stores their pixel information, will have some values that are equal. When classifying those data points with perceptron, our decision rule is $\hat{y}_t = argmax_c(\sum_{i=0}^{t-1} \alpha_{ci} K(\boldsymbol{x}_i, \boldsymbol{x}_j))$. The value $\sum_{i=0}^{t-1} \alpha_{ci} K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ will be larger if $\boldsymbol{x}_t$ has overlapping pixels with the digit of class c. In practice, this might affected by the data quality, we can also see in some images for digit 1, the digit lying on the diagonal which makes it difficult to recognise.

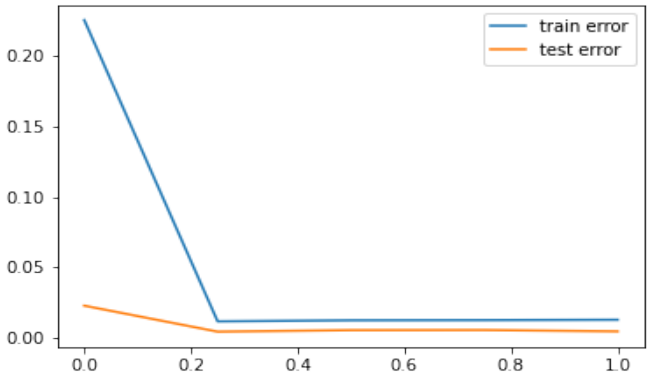


1.1.6 Experimental Protocol 5

In this section, we will be using the same algorithm as Experimental Protocol 1&2 with Gaussian Kernel.

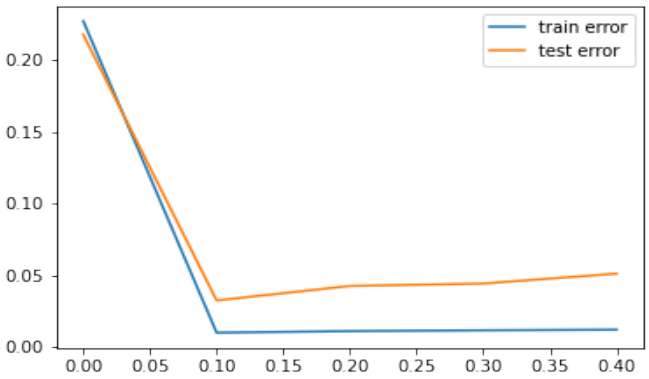
Experiments on S

We first do some experiments on S. We start with taking 5 values from the range $[0, 1]$. The mean train and test error rate of 20 runs are shown as follow. We can see the train and test error rate decreases until 0.4.



Therefore, we narrow S to the range of $[0, 0.4]$ and take 5 numbers from it. The train and test error rate are shown as following. There is a obvious increase in error rates after 0.15. Therefore we will do cross validate over $S = [0, 0.20]$ to

find the best c.



Best c found by cross-validation

Mean test error rate ±std	Mean c^*±std
2.5%±0.0124	0.015 ± 0.0056

Comparison of Polynomial kernel and Gaussian kernel

	Mean test error rate ±std	Mean c^*&d^*±std
OvA Gaussian	2.5%±0.0124	0.015 ± 0.0056
OvA Polynomial	2.8%±0.0097	4.8 ± 1.08

Gaussian kernel turns out to have a lower error rate compared to the Polynomial kernel trained with same number of epochs and same runs. The result given by multiplying the alpha matrix with $\boldsymbol{x}_t$ increases exponentially if two images have more similar pixels. Therefore with enough epochs Gaussian kernel promises to converge but Polynomial kernel does not.

1.1.7 Experimental Protocol 6

One vs One Method

In this method, we split the k-class classification problem into $\frac{k(k-1)}{2}$ 2-class classification problems by pairing each class with every other classes then we will be training $\frac{k(k-1)}{2}$ classifiers. For each classification problem, we assign one class with label -1 and the other one with label 1, we call this binary label of each class in their related classifier y_c^p, where $c = 1, ..., k$ is the class and $p = 1, ..., \frac{k(k-1)}{2}$ denotes the 2-class classifier. Each classifier only relates to 2

classes, if c is related to the classifier p then y_c^p will be either 1 or -1 depends on your setup, otherwise it will be 0. The classifer for each 2-class classification problem is

$$sign(\boldsymbol{w}(\boldsymbol{x}))$$

We will then record the class c according to y_c^p. After getting the results of $\frac{k(k-1)}{2}$ classifiers, we implement a majority vote and take the class that gets the maximal votes to be the predicted class of sample $\boldsymbol{x}$.
If the predicted class equals to the true class, then weights $\boldsymbol{w}_c$ for all classifiers remain unchanged. If the predicted class does not equal to the true class, we need to add $y_c^p \boldsymbol{x}$ to the classifiers related to the true class. The other classifiers remains the same.

Mean train and test error of each dimension

d	1	2	3	4	5	6	7
Mean train error rate+std	9.62% ±0.005	5.39% ±0.002	2.88% ±0.002	2.08% ±0.001	1.66% ±0.001	1.4% ±0.001	1.25% ±0.001
Mean test error rate+std	9.3% ±0.016	3.70% ±0.012	2.76% ±0.009	2.80% ±0.008	2.36% ±0.009	2.33% ±0.012	3.11% ±0.011

Data Table

Cross-validation

Mean test error rate ±std	Mean d^*±std
2.5 %±0.0092	4.1 ± 1.02

The Mean test error here is the average test error generated by the model trained with the best dimension in each run. This is lower than the error we get before cross-validation. We are actually getting a range for the Mean d^*, since there are some randomness in our experiments.

Comparison of OvO and OvA

	Mean test error rate ±std	Mean c^*&d^*±std
OvO Polynomial	2.5%±0.0092	4.1 ± 1.02
OvA Polynomial	2.8%±0.0097	4.8 ± 1.08

The OvO method trains more classifiers than OvA method, and OvO requires less complex hypothesis, it only assumes linearly separable within each pair of 2 classes while OvA assumes every class to be linearly separable with all other

classifiers at once. Therefore, it is not surprising to see that OvO gives lower mean test errors.

2 Part 2

2.1 Spectral Clustering

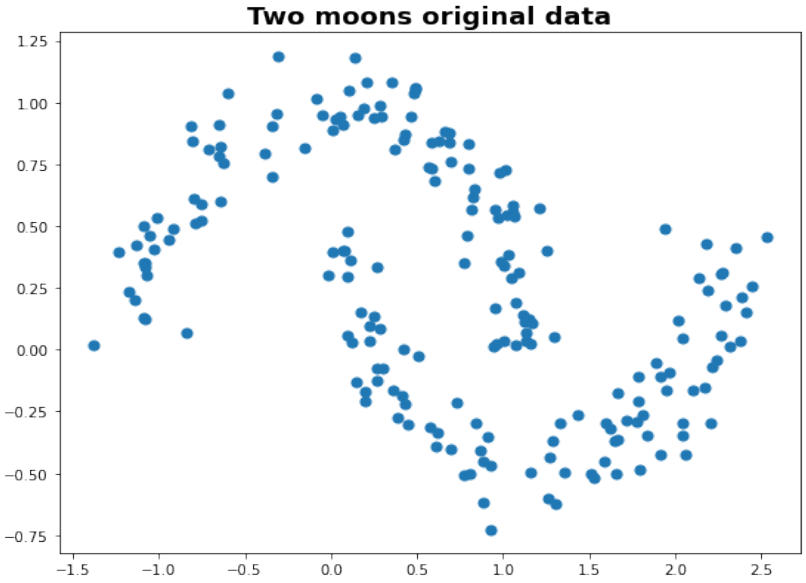
2.1.1 Introduction

In Part 2, we implemented Spectral Clustering algorithm on 3 different data sets. There are mainly 4 steps to perform spectral clustering:
1. Create a similarity fully connected graph between N objects to cluster, where $s(xi, xj) = exp(-c||xi - xj||^2)$
2. Determine the Adjacency matrix W, Wij $\rightarrow$ 1 when the points are close and Wij $\rightarrow$ 0 of the points are far apart. Degree Matrix D, and the unormalized Laplacian matrix L.
3. Compute the eigenvectors of the matrix L.
4. Using the second smallest eigenvector to classify data under

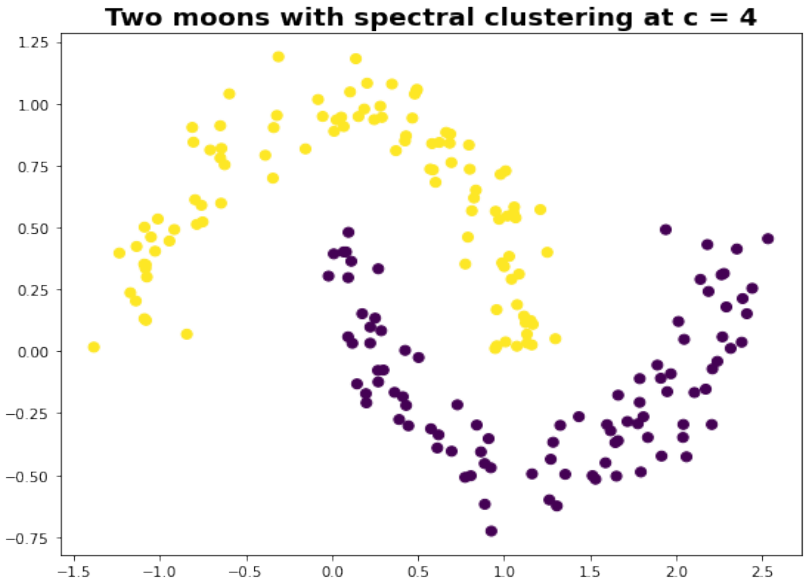
$$\text{cluster(xi)} = \begin{cases} +1 & \text{if} \quad v2(i) \geq 0 \\ -1 & \text{if} \quad v2(i) < 0 \end{cases}$$

2.1.2 Experiments

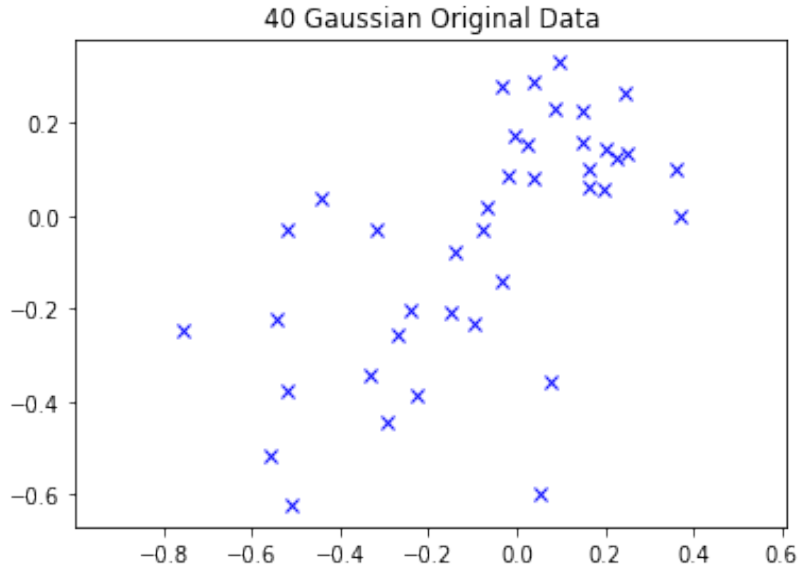
1.



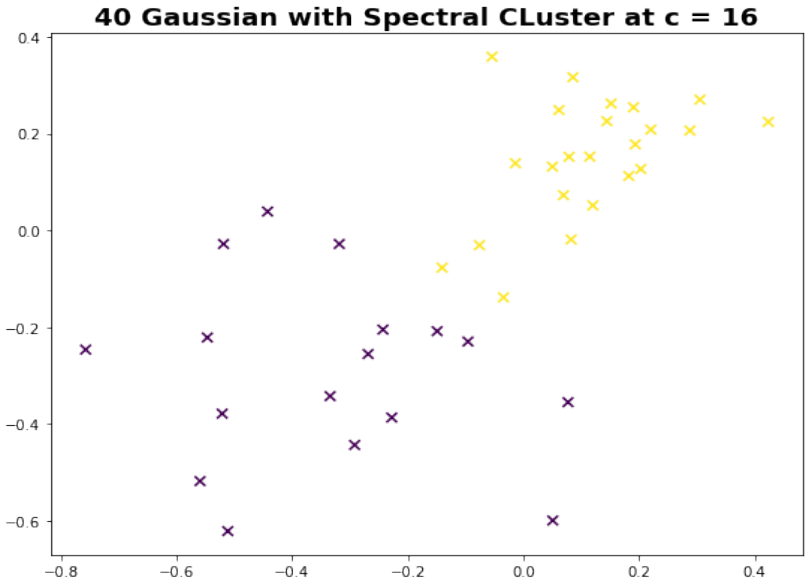
After spectral clustering, the data is correctly labelled at c = 2^2.



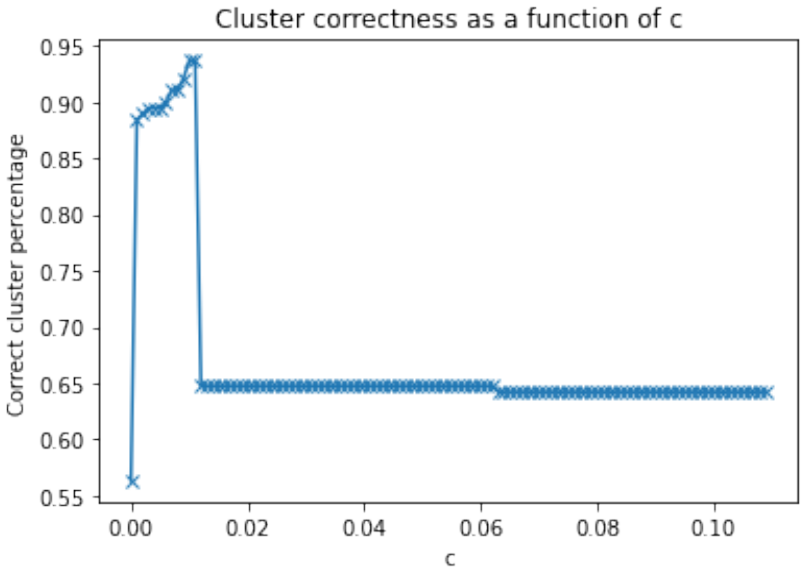
2.



After spectral clustering, the data is almost correctly labelled at c = 2^4, with 3 mislabelled data.



3.



As we can see from the above plots, when c = 0.01, the quality of the clustering is the best, with CP = 0.94.

2.1.3 Questions

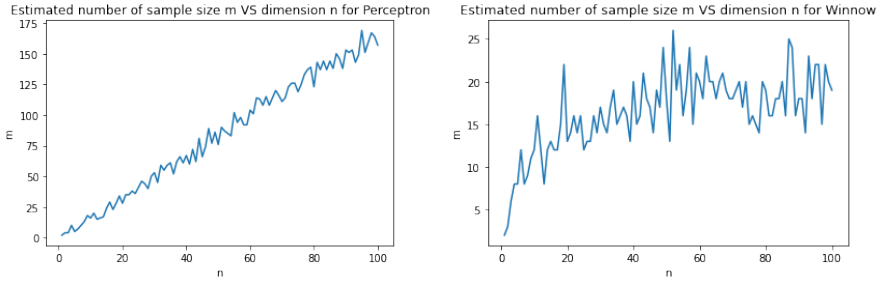
1.
CP(c) measures the purity of clustering. It always measures how separable the data sets are. Since this is unsupervised leaning, the labels are unknown before the experiment. So we only have to cluster the points, it only matters how separable the data sets are. It doesn't matter which label we assign to each group as long as they have the same label within each cluster.
2.
$L * v = (D - W) * v = \lambda * v$
Since L is a matrix where all rows and columns sum to zero, for any constant eigenvector $L * v = 0$, there will be a scale $\lambda 1 = 0$. The corresponding eigenvector then is always the constant vector with any scalar multiplication.
3.
In spectral clustering, the data points are treated as nodes of a graph. Thus, clustering is treated as a graph partitioning problem. In our case, we connected all points with each other, and weighted all edges by similarity. Then by computing Graph Laplacian, we can find the eigenvalues and eigenvectors which helps to transform the data points into a low-dimensional space so that when the 2 points are close, they are always in the same cluster, and when they are far apart, they are in different clusters. In the end, since the 2nd eigenvalue indicates how tightly connected the nodes are in the graph, we choose the eigenvector corresponding to the 2nd eigenvalue to assign values of each node.
4.
From the above experiment and plots, we can see the quality of clustering increases while c increases. Till c reaches a optimal point, the quality of clustering remains.

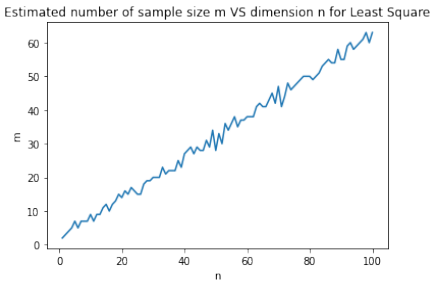
3 Part 3

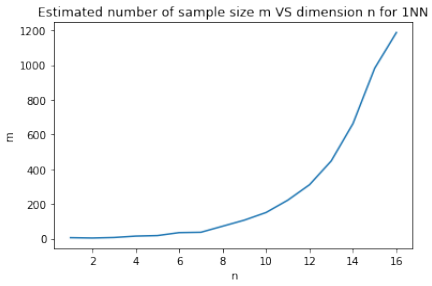
3.1 Questions

3.1.1 a.

In this section, we plotted estimated sample complexity for **Perceptron, Winnow, Least Squares, and 1NN** separately.





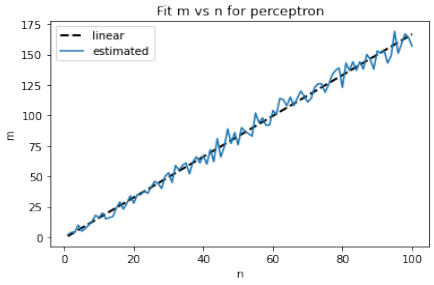


3.1.2 b.

i) For estimating time complexity, we make dimension (n) as independent variable, and sample size(m) as dependent variable. The sample complexity here can be expressed as the minimum sample size(m) which can incur 10 % generalisation error. We trained our model with n in range [0,100], m starting from 1. We re-sampled training data 20 times for each combination of m and n, and got the average generalisation error from test data (1000*n).
ii) From the method above, instead of training models with all uniformed data with size m*n, we randomly sampled data for each combination of m and n. By doing this, computing efficiency increases, while the accuracy decreases. In order to balance the trade-off, we re sampled training data 20 times for every m*n and used the average error instead.

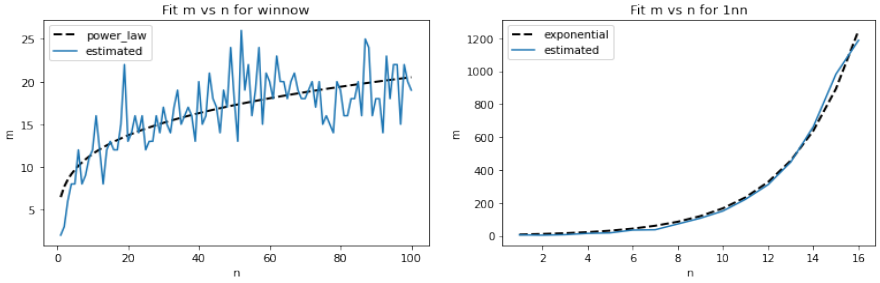
3.1.3 c.

From the plots above in part a, it seems that sample complexity grows linearly as a function of dimension for **Perceptron and Least Square**.
Perceptron: Θ(n), m = 1.67n − 0.82.
Least Square: Θ(n), m = 0.61n + 1.73



3_3c.png

For **Winnow**, m and n fit under power-law. And for **1NN**, though there is a limit number of dimensions, the trend is quite obvious and m fits n under exponential function.
Winnow: Θ(log n), m = 6.46 $n^{0.25}$
1NN: $\Theta(e^n)$, m = 5.88 $e^{0.33n}$



3.1.4 d.

The mistakes of perceptron is bounded by M $\leq \frac{R^2}{\gamma^2}$, where R is the distance of furthest instance, and γ is the distance of closest instance to the hyperplane. Since $x_i \in \{-1, +1\}^n$, the distance of all sample x_i to the center of hypersphere is the same. R = $\sqrt{1^2 + 1^2 + ... + 1^2} = \sqrt{n}$. Based on our question, since the first column of data set is the labels, there will always be a hyperplane which can separate the data vertically along axis = 0. So $\gamma = \sqrt{1^2 + 0^2 + ..0^2 + ... + 0^2} = 1$. M $\leq \frac{R^2}{\gamma^2} = \frac{n}{1}$ = n. From this formula, we know the largest number of mistake will be n. To solve the upper bound $\widehat{p}_{m,n}$ on the probability that the perceptron will make a mistake on the sth example, let S be consist of s-1 examples sampled iid from P, and let (x_s,y_s) be an additional example sampled from P. Then by theorem, $\widehat{p}_{m,n} = \frac{M}{m} = \frac{n}{m}$.

3.1.5 e.

Merry Christmas:)