Supervised Learning (COMP0078) – Coursework 2

Student No: 18145399

December 12, 2021

Content

1 PART I 2

1.1 Introduction 2

1.2 Methodology 2

1.2.1 The choice of min_iteration 4

1.2.2 The choice of ω 4

1.3 Questions 4

1.3.1 Basic Results when testing with OVR method and polynomial kernel 4

1.3.2 Cross-validation when testing with OVR method and polynomial kernel 5

1.3.3 Confusion matrix when testing with OVR method and polynomial kernel 6

1.3.4 Five hardest to predict correctly pixelated images with OVR method and polynomial kernel 7

1.3.5 Basic Results and Cross Validation when testing with OVR method and Gaussian kernel 7

1.3.5.1 Basic Results 7

1.3.5.1 Cross Validation 8

1.3.6 Basic Results and Cross Validation when testing with OvO method and polynomial kernel 10

1.3.6.1 Basic Results 10

1.3.5.1 Cross Validation 10

2 PART II - Spectral Clustering 12

2.1 Introduction 12

2.2 Methodology 12

2.3 Experiment 12

2.3.1 Two moons cluster 12

2.3.2 Random generated data cluster 14

2.3.3 Real digit dataset cluster 15

2.4 Questions 16

2.4.1 CP(c) 16

2.4.2 Eigenvalue of the Laplacian 16

2.4.3 Why spectral clustering “works" 17
2.4.4 Parameter c 17
3 PART III - Spectral Clustering 17
3.1 Sparse Learning 17
(a) Complexity 17
(b) 19
(c) 19
(d) 21
Reference 21

1 PART I

1.1 Introduction

In this part, I train and test a classifier with the database “zipcombo.dat”, which is a MNIST handwritten digit dataset. There are 9298 figures with digits from 0 to 9 and a resolution of 16 by 16 pixels (as shown in Fig 1) in the database.

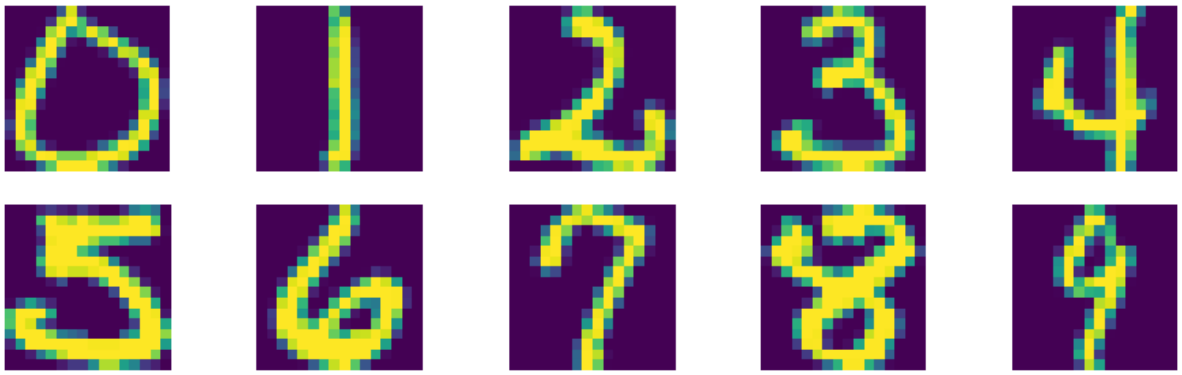


Fig 1. Sample figures for digits from 0 to 9 in database “zipcombo.dat”

1.2 Methodology

For effectiveness, I map the data to a higher dimensional space to produce a non-linear decision boundary for classification by adding a kernel to the perceptron with the equation

$$K_d(\boldsymbol{p}, \boldsymbol{q}) = (\boldsymbol{p} \cdot \boldsymbol{q})^d,$$

where d is the parameter representing the width of the kernel, it is chosen manually for reasonable error.

Alternatively, there is another kernel function called the Gaussian kernel. By adding this kernel, I can map the data into infinite dimensional feature space. It allows high-complexity algorithms to run with high computational efficiency. The equation of Gaussian kernel is

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

where c is the parameter representing the width of the kernel.
Then, I use two common ways to classify the MNIST figures to 10 classes with the idea of binary classifier generalisation, as there are more than 2 classes, which are 'One-versus-Rest'(OvR) and 'One-versus-One'(OvO):

Method 1	**OvR Generalised Kernel Perceptron**
Input:	1. $\{(x_1, y_1), \ldots, (x_m, y_m)\} \in (\Re^n, \{-1, +1\})^m$ 2. Kernel function and its parameter a. Polynomial kernel function $K_d(p, q) = (p \cdot q)^d$ and the parameter d b. Gaussian kernel function
Initialization:	${\omega_1}^{(k)} = 0$ **for all k**
Prediction:	**for t ≤ max_iteration:** 1. receive data x_t, y_t 2. make prediction $\widehat{y_t} = argmax_{\ k} \sum_i \ {\omega_i}^{(k)} K(x_i, x_t)$
Update:	**if** $\widehat{y_t} y_t \leq 0$: ${\omega_t}^{(y_t)} = {\omega_t}^{(y_t)} + 1$, ${\omega_t}^{(\widehat{y_t})} = {\omega_t}^{(\widehat{y_t})} - 1$ **end when** 1. **error converges** (i.e., the difference between latest generated accuracy and second latest generated accuracy smaller than 0.1) 2. **no. of iteration larger than the min_iteration** (i.e., the minimum no. needed for adequate result)

Table 1. Method 1. OvR Generalised Kernel Perceptron

Method 2	**OvO Generalised Kernel Perceptron**
Input:	1. $\{(x_1, y_1), \ldots, (x_m, y_m)\} \in (\Re^n, \{-1, +1\})^m$ 2. Kernel function $K_d(p, q) = (p \cdot q)^d$ and the parameter d
Initialization:	$\omega^{(p)} = 0$ **for** $p \in \{1, \ldots, k(k-1)/2\}$
Prediction:	**for t ≤ max_iteration:** 1. receive data x_t, y_t 2. obtain confidence = ${\omega_i}^{(p)} K(x_i, x_t)$ 3. define a voting scheme with $sign(\sum_i \ {\omega_i}^{(p)} K(x_i, x_t))$ and obtain the prediction of each classifier 4. make prediction $\widehat{y_t}$ with the most vote count
Update:	**for** $p \in \{1, \ldots, k(k-1)/2\}$**:** **if** $\widehat{y_t} \neq y_t$: ${\omega_t}^{(p)} = {\omega_t}^{(p)} + 1$ if y_t is the positive class

	$\omega_t{}^{(p)} = \omega_t{}^{(p)} - 1$ if y_t is the negative class **end when** 1. **error converges** (i.e., the difference between latest generated accuracy and second latest generated accuracy smaller than 0.01) 2. **no. of iteration larger than the min_iteration** (i.e., the minimum no. needed for adequate result)

Table 2. Method 2. OvO Generalised Kernel Perceptron

1.2.1 The choice of min_iteration

For this dataset, I find that when I choose kernel degrees of 3 and 5, after two algorithms trained and tested 7 epochs (80% data for training randomly and 20% for testing), their testing accuracy almost remained unchanged with a little neglectable fluctuation. So, the min_iteration I use for later stop criterion in two algorithms is 7. (It might be different when using different database)

1.2.2 The choice of ω

I choose ± 1 as an updated value for ω. This learning rate can be smaller for obtaining more details when training the algorithms, but it is small enough for this database as the accuracy did not improve significantly when I used learning rates of 0.5 and 0.8.

1.3 Questions

1.3.1 Basic Results when testing with OVR method and polynomial kernel

I run the algorithm OVR 20 times with different dimensions of polynomial kernel d from 1 to 7. For each run, the zipcombo dataset is randomly split into 80% for training and 20% for testing. The mean test and train error rates as well as standard deviations are shown below.

OvR Generalised Polynomial Kernel Perceptron		
Dimension of polynomial kernel d	**Training error rates $\pm$ std (%)**	**Testing error rates $\pm$ std (%)**
1	7.397 ± 0.288	9.452 ± 2.194
2	1.190 ± 0.160	4.317 ± 0.687
3	0.454 ± 0.109	3.444 ± 0.424
4	0.275 ± 0.095	3.089 ± 0.502
5	0.198 ± 0.083	3.089 ± 0.243
6	0.157 ± 0.046	3.446 ± 0.509
7	0.122 ± 0.081	3.226 ± 0.388

Table 3. Results of OvR Generalised Kernel Perceptron with d from 1 to 7

Discussion:

- The results almost remained the same when the value of d increased, so that there was no clear sign of overfitting because of the high dimension.
- All training and testing error values are acceptable with an order of magnitude of 10^{-2} or 10^{-3}.
- The polynomial kernel with degree of 1 has the highest mean error for both training and testing processes, and it with degree of 2 has the second highest.
- The performances of polynomial kernels with degree larger than 3 are similar, which indicates that OVR is a well-performed algorithm for those polynomial kernel perceptron (with d > 3). And the total mean testing error for them is 3.2588 ± 0.4132.

1.3.2 Cross-validation when testing with OVR method and polynomial kernel

Run the algorithm OVR 20 times again. For each run, perform 5-fold cross-validation for each polynomial degree to select the “best" parameter $d*$, then train the randomly split 80% training data with $d*$ and find the test error for 20% testing data. The optimal d and its corresponding mean test and train error rates are shown below.

Optimal d	Testing error rates (%)
5	2.527
6	2.849
4	2.903
6	2.903
7	3.011
7	3.172
6	3.226
6	3.333
7	3.333
4	3.387
7	3.441
5	3.495
5	3.495
4	3.602
4	3.656
6	3.656
7	3.710
6	3.925
6	4.032

6	4.301

Table 4. Results of OvR Generalised Kernel Perceptron with optimal d

Therefore, the overall mean test error ± std is 3.398 ± 0.423 and the mean d∗ ± its std is 5.737 ± 1.068. For each optimal d, its mean test error ± std are shown below.

Optimal d	Testing error rates ± std (%)	The existence frequency
4	3.432 ± 0.262	4
5	3.172 ± 0.456	3
6	3.528 ± 0.501	8
7	3.333 ± 0.238	5
Overall	3.398 ± 0.423	20

Table 5. The mean test error ± std for each optimal d

Discussion:

- The degree of 6 is the mode of above list of 20 optimal dimension. Although its mean test error ± std is 3.528 ± 0.501, which is highest among all, the highness is acceptable. Therefore, we can say OVR is a well-performed algorithm when dimension of the polynomial kernel perceptron is 6.

1.3.3 Confusion matrix when testing with OVR method and polynomial kernel

Confusion matrix is another way to assess the performance of kernel perceptron. It shows how the predictions for each label satisfied the true labels.

Pred y / Test y	0	1	2	3	4	5	6	7	8	9
0	0	**0**	0.192 ± 0.249	0.155 ± 0.216	0.073 ± 0.155	0.259 ± 0.311	0.189 ± 0.242	0.022 ± 0.066	0.055 ± 0.133	0.006 ± 0.027
1	**0**	0	0.031 ± 0.111	0.012 ± 0.054	0.361 ± 0.344	0.0 ± 0.0	0.191 ± 0.273	0.028 ± 0.109	0.077 ± 0.221	**0**
2	0.115 ± 0.200	0.031 ± 0.111	0	0.201 ± 0.243	0.322 ± 0.265	0.016 ± 0.055	0.041 ± 0.079	0.190 ± 0.192	0.076 ± 0.145	0.006 ± 0.019
3	0.030 ± 0.109	0.003 ± 0.014	0.149 ± 0.217	0	0.008 ± 0.028	0.421 ± 0.279	**0**	0.179 ± 0.248	0.191 ± 0.233	0.019 ± 0.055
4	0.014 ± 0.055	0.104 ± 0.160	0.200 ± 0.226	0.072 ± 0.148	0	0.028 ± 0.037	0.133 ± 0.193	0.123 ± 0.197	0.012 ± 0.054	0.313 ± 0.249
5	0.172 ± 0.206	**0**	0.076 ± 0.131	0.194 ± 0.203	0.133 ± 0.226	0	0.177 ± 0.217	0.013 ± 0.032	0.159 ± 0.197	0.077 ± 0.132
6	0.323 ± 0.330	0.081 ± 0.208	0.041 ± 0.069	0.006 ± 0.027	0.336 ± 0.320	0.093 ± 0.123	0	**0**	0.120 ± 0.196	**0**
7	**0**	0.044 ± 0.091	0.047 ± 0.106	0.111 ± 0.216	0.226 ± 0.225	0.042 ± 0.079	**0**	0	0.134 ± 0.209	0.396 ± 0.322
8	0.077 ± 0.161	0.112 ± 0.195	0.208 ± 0.209	0.123 ± 0.191	0.107 ± 0.155	0.225 ± 0.263	0.028 ± 0.109	0.098 ± 0.186	0	0.023 ± 0.059
9	0.013 ± 0.035	0.018 ± 0.051	0.025 ± 0.075	0.068 ± 0.169	0.332 ± 0.259	0.065 ± 0.155	0.006 ± 0.027	0.437 ± 0.338	0.036 ± 0.111	0

Table 6. The confusion matrix when testing with OVR method

Discussion:

- Confusion error rate matrix gives us an insight of classifier performance regarding the digits that figures representing to. For example, the MNIST figures with label 3 are much easier being classified as figures with label 5, which is reasonable.
- According to the values in the table, the digits are hard to be distinguished are (1,4), (2,4), (3,5), (4,9), (6,1), (6,4) and (7,9). For the other digits that look different, the confusion error rate is much lower, which means the algorithm can classify them easier.

1.3.4 Five hardest to predict correctly pixelated images with OVR method and polynomial kernel

To obtain more accurate result, I run the algorithm OVR for 40 times to find the 5 hardest to distinguish figures within all 40 runs (the accumulation of 40 runs). From previous analysis, the kernel with dimension of 6 has the most well-performed algorithm, so I use 6 as the parameter of the algorithm. And for each run, the algorithm is trained with 80% dataset and tested with 100% dataset. The indices for those figures in my random selected dataset are 720, 8835, 2700, 8261, and 5296, with number of prediction failures of 14, 16, 17, 36, and 39.

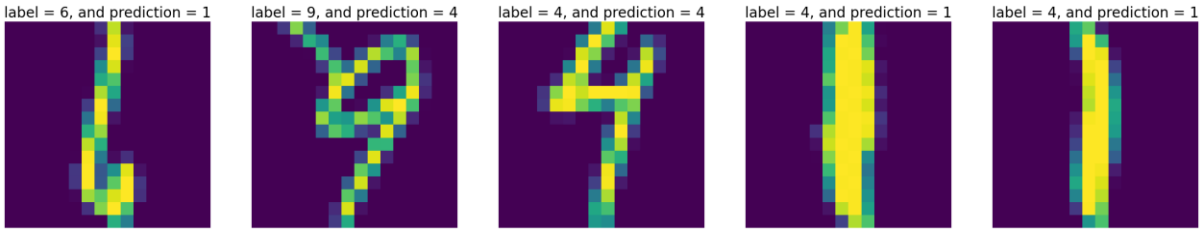


Fig 2. The 5 hardest to predict pixelated images

Discussion:

There are two main situations of the wrong prediction shown above.

- The original label in the dataset is unreasonable, and manual classification is similar as algorithm's classification.
- The original figure is vague, and it is hard to manually classify the figure.

Therefore, there are errors that cannot be eliminated because of the inaccuracy in database's labels, and the accuracy will never reach to 100%.

1.3.5 Basic Results and Cross Validation when testing with OVR method and Gaussian kernel

1.3.5.1 Basic Results

I run the algorithm OVR 20 times with different width of Gaussian kernel c from 10^{-1} to 10^{-7}, but the test error rate become unacceptable high when c is smaller than 10^{-4}. To decrease the error rate, I switched value of c into the power term of 2. (i.e., from 2^{-1} to 2^{-7}) For each run, the zipcombo dataset is randomly split into 80% for training and 20% for testing. The mean test and train error rates as well as standard deviations are shown below.

OvR Generalised Gaussian Kernel Perceptron

Width of Gaussian kernel c	Training error rates $\pm$ std (%)	Testing error rates $\pm$ std (%)
2^{-1}	0.0 ± 0.0	6.108 ± 0.449
2^{-2}	0.0 ± 0.0	5.831 ± 0.425
2^{-3}	0.001 ± 0.006	5.097 ± 0.528
2^{-4}	0.021 ± 0.017	4.390 ± 0.405
2^{-5}	0.030 ± 0.021	3.487 ± 0.390
2^{-6}	0.082 ± 0.046	2.833 ± 0.349
2^{-7}	0.250 ± 0.079	3.215 ± 0.319

Table 7. Results of OvR Generalised Gaussian Kernel Perceptron with c from 2^{-1} to 2^{-7}

Discussion:

- When the value of width of Gaussian kernel is smaller than 10^{-4}, the error is unacceptable high. This might because when the chosen width is too small, the data far away contribute too much.
- When the value of width of Gaussian kernel is in the range $[2^{-7}, 2^{-1}]$, the test error decrease with width value from 2^{-1} to 2^{-6}, which might because the wider the kernel width is, the more insufficient contribution of data far is. And the test error is increasing when width value is smaller than 2^{-6}, which might be resulted from the overdose contribution of data far.

1.3.5.1 Cross Validation

Similar as the procedure of 3.1.2, I run the two algorithms 20 times again. For each run, perform 5-fold cross-validation for each Gaussian width to select the “best" parameter $c*$, then train the randomly split 80% training data with selected $c*$ and find the test error for 20% testing data. The optimal width c* and its corresponding mean test and train error rates are shown below.

Optimal c	Testing error rates (%)
2^{-6}	2.634
2^{-7}	2.957
2^{-6}	3.011
2^{-7}	4.194
2^{-7}	3.172
2^{-6}	3.065
2^{-6}	3.817
2^{-6}	2.581

2^{-6}	3.387
2^{-6}	3.065
2^{-6}	3.387
2^{-6}	2.957
2^{-6}	2.419
2^{-6}	2.688
2^{-6}	2.634
2^{-6}	2.581
2^{-6}	3.011
2^{-6}	3.011
2^{-6}	3.172
2^{-6}	2.742

Table 8. The mean test error ± std for each optimal c

Optimal c	**Testing error rates ± std (%)**	**The existence frequency**
2^{-6}	2.951 ± 0.350	17
2^{-7}	3.441 ± 0.539	3
Overall	3.024 ± 0.423	20

Table 9. The mean test error ± std for each optimal c

Discussion:

- The best choice of Gaussian kernel width value is much clearer (17/20) than it of polynomial kernel degree value (8/20).
- From above, the error and std of the optimal OVR algorithm with Gaussian kernel is 3.024 ± 0.423, while the error and std of the optimal OVR algorithm with polynomial kernel is 3.398 ± 0.423. The value ranges of two results collapse a bit and the accuracy of optimal OVR algorithm with Gaussian kernel is higher. But it is not clear that gaussian kernel perform better, as the optimal d of polynomial kernel I obtained before might not be the most optimal one.
- The table above indicates that the width value of 2^{-6} is the best choice for the OVR algorithm with Gaussian kernel, with an average test error rate and std of 2.951 ± 0.350. This result is much lower than the result of optimal OVR algorithm with polynomial kernel (3.528 ± 0.501).

1.3.6 Basic Results and Cross Validation when testing with OvO method and polynomial kernel

1.3.6.1 Basic Results

I run the algorithm OvO 20 times with different degree of polynomial kernel d from 1 to 7.

OvO Generalised Polynomial Kernel Perceptron		
Dimension of polynomial kernel d	**Training error rates ± std (%)**	**Testing error rates ± std (%)**
1	6.081 ± 0.225	7.438 ± 0.791
2	2.233 ± 0.173	4.422 ± 0.640
3	1.381 ± 0.112	3.634 ± 0.435
4	1.113 ± 0.165	3.723 ± 0.518
5	0.912 ± 0.121	3.589 ± 0.399
6	0.818 ± 0.092	3.508 ± 0.433
7	0.752 ± 0.117	3.790 ± 0.479

Table 7. Results of OvO Generalised Polynomial Kernel Perceptron with d from 1 to 7

Discussion:

- Compared to results of OvR, test errors of both have the same regularity that they decrease to a point then increase. And the smallest test errors of them are close to each other.
    - The error of OvR decreases from 9.452 to 3.089 when d = 4,5 and then increases. While the error of OvO decreases from 7.438 to 3.508 when d = 6 and then increases.

1.3.5.1 Cross Validation

I run the algorithm OvO 20 times again. For each run, perform 5-fold cross-validation for each polynomial dimension to select the “best" parameter d∗, then train the randomly split 80% training data with selected d∗ and find the test error for 20% testing data. The optimal width d* and its corresponding mean test and train error rates are shown below.

Optimal d	**Testing error rates (%)**
6	3.817
4	3.871
5	3.118
4	3.656
4	2.849

4	3.710
5	3.441
4	3.118
4	3.118
6	3.065
6	3.495
5	3.172
3	3.172
5	3.763
5	2.581
6	3.333
6	3.441
4	3.011
4	2.742
5	3.011

Table 8. The mean test error ± std for each optimal d

Therefore, the overall mean test error ± std is 3.274 ± 0.357 and the mean d∗ ± its std is 4.750 ± 0.887. And for each optimal d, its mean test error ± std are shown below.

Optimal d	**Testing error rates ± std (%)**	**The existence frequency**
3	3.172	1
4	3.259 ± 0.399	8
5	3.181 ± 0.365	6
6	3.430 ± 0.244	5
Overall	3.274 ± 0.357	20

Table 9. The mean test error ± std for each optimal d

Discussion:

- The mean test errors for the overall optimal d and most frequently chosen d* within the OvO algorithm (3.259 ± 0.399 and 3.274 ± 0.357) are close to errors within the OvR algorithm (3.528 ± 0.501 and 3.398 ± 0.423).

Comparation between OvR and OvO:

- The time taken:
  During the experiment, the time taken by OVO algorithm in each run is longer than it taken by OvR. This might be resulted from the number of classifiers the algorithm needed and the structure of algorithms. The number of classifiers of OvR will be ten as the same as the number of classes with complexity of O(n), while the number of classifiers of OvO will be $\frac{10\times9}{2} = 45$ with complexity of $O(n^2)$. Although the number of data one classifier of OvO used to train is less than it of OVR used, the high computational complexity still leads to high time consumption.
- The accuracy:
  From above tables, the accuracy is similar for both algorithms with their optimal choice of parameters, while the accuracy of OVO is slightly higher. This might because it is the true type being used for comparison in each dichotomy, without confusing the other classes. This class imbalance in OvR might be reduced by a random sampling scheme selecting data for the rest categories.
- Which one is better:
  For this dataset, both algorithms can be chosen, as the test accuracy are similar, and the time taken are similar. But if the number of classes in the dataset became larger, the computational complexity of OvO will become higher, which means the longer time the OvO will take.

2 PART II - Spectral Clustering

2.1 Introduction

Spectral clustering is the division of very irregularly shaped groups of data. Spectral clustering works by representing the data as a (weighted) graph. If the weights between two data points are large, they are close in the graph. Spectral clustering is the splitting of the graph into two 'clusters' where the total weight of the borders between the two clusters is small and the number of data points in each cluster is approximately the same. In this experiment, I implement spectral clustering on “twomoons.dat” dataset and “dtrain123.dat” dataset.

2.2 Methodology

First, I produce an ‘ $l \times l$’ weight matrix $W = exp(-c||p-q||^2)$ where c > 0. Then I construct the graph Laplacian $L \coloneqq D - W$ where D is a diagonal matrix with $D = \sum_{j=1}^{l} W_{ij}$. Let v_2 denote the eigenvector with second smallest eigenvalue of L, then use the sign of v_2 to cluster the data by

$$\hat{y}(i) = cluster(\boldsymbol{x_i}) = \begin{cases} +1\,, & sign\big(\mathrm{v}_2(i)\big) = +1{,}0 \\ -1, & sign\big(\mathrm{v}_2(i)\big) = -1 \end{cases}$$

.

2.3 Experiment

2.3.1 Two moons cluster

The original data graphs are shown below for the understanding.

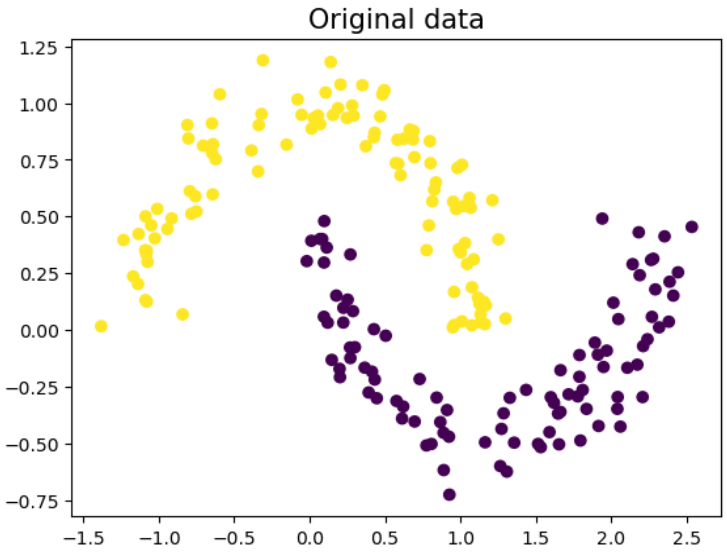


Fig 3. The original data graph of two moons

And the result of classification when using c = 60 is shown in Fig 4.

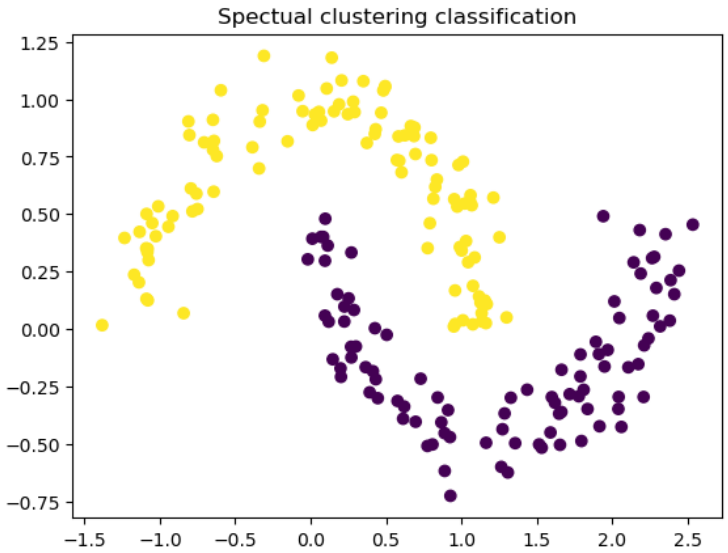


Fig 4. The classification result of two moons

I select c in range [-10,10] with gap of 0.1 and find the correct classification, and the relationship between the accuracy and value of c is shown below. In the range of [-10,10], c with value in range $[2^{4.3},2^{8.2}]$ can achieve a relatively high accuracy.

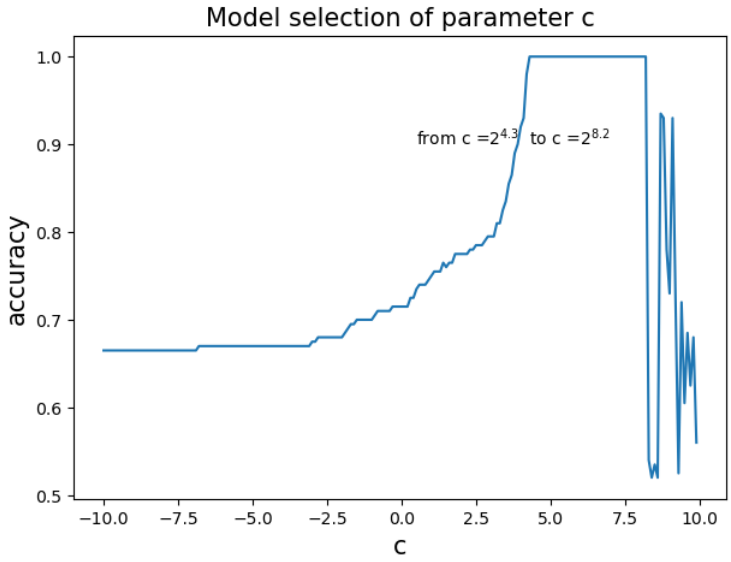


Fig 5. the relationship between the accuracy and value of c in range [-10,10]

When c becomes larger, the performance of classifier becomes unstable. This probably because high c blurs the difference between width and results in the similarity of weights of datapoints further away and in the neighborhood.

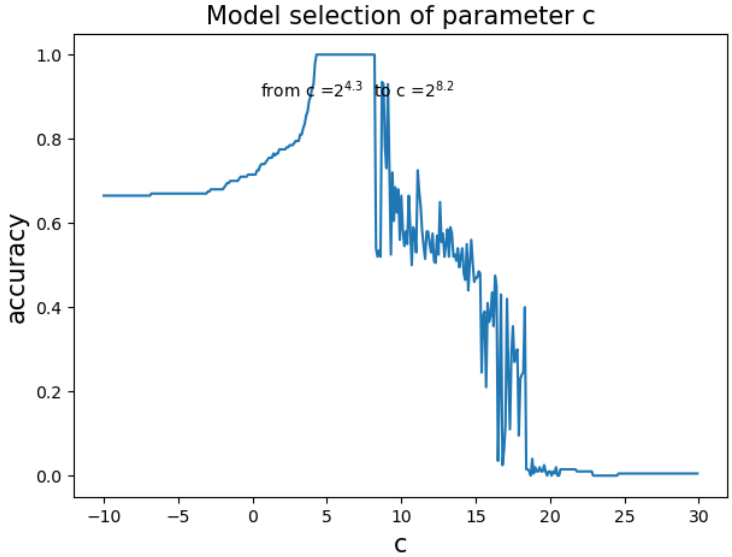


Fig 5. The relationship between the accuracy and value of c in range [-10,30]

2.3.2 Random generated data cluster

With the same procedure as first experiment. We analysis the algorithm of spectral clustering with a random generated isotropic Gaussians dataset. There still are two cluster with label of -1 and 1. The original data is plotted below with its cluster prediction result.

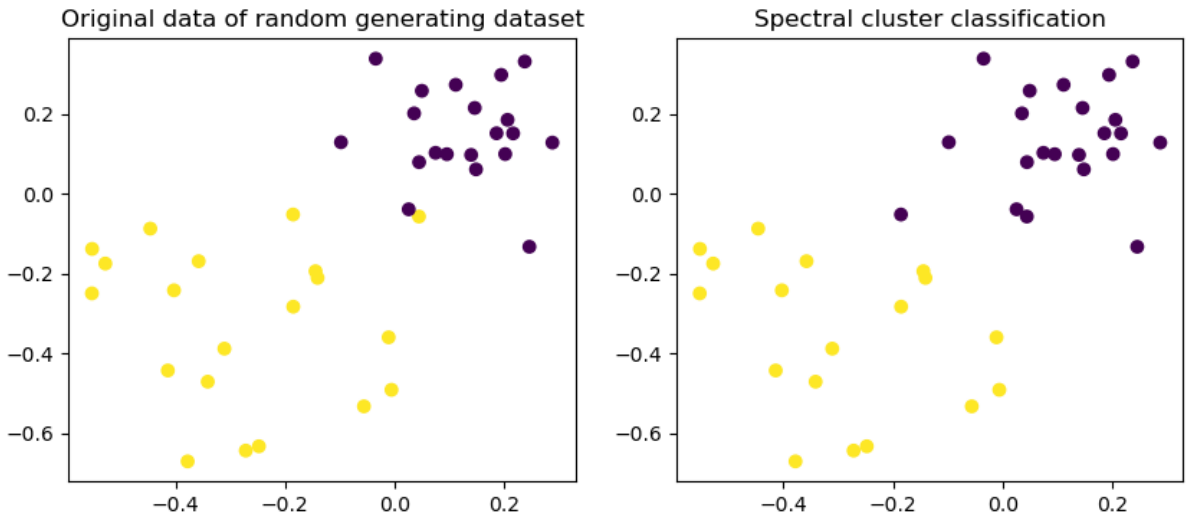


Fig 6. The origin and prediction of random generated isotropic Gaussians dataset

It is hard to justify the most optimal parameter as the lack of dataset. In each run of the codes, the dataset will change, and the distribution pattern will change. Also, the two datasets are slight overlapping in the original plot, so there must be some of data that hard to be distinguished.

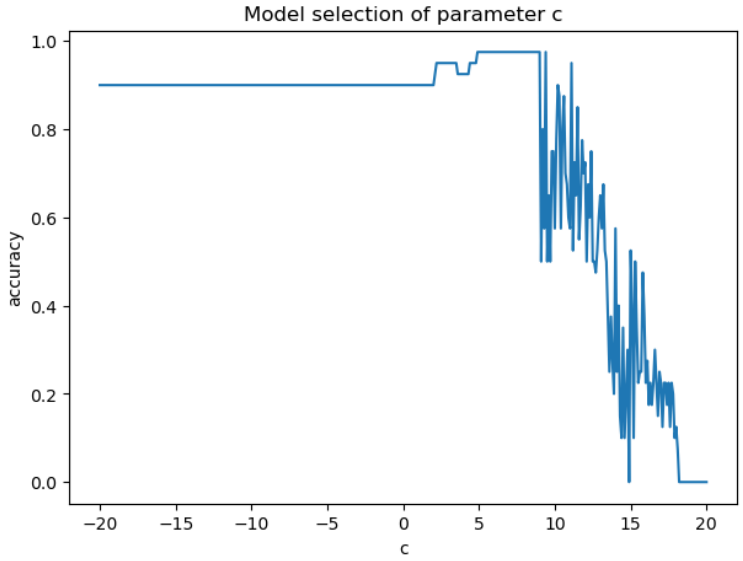


Fig 7. The relationship between the accuracy and value of c in range [-20, 20]

2.3.3 Real digit dataset cluster

Repeat the procedure above on the real digit dataset, which has larger scale, and use the below algorithm quality of spectral clustering CP to analysis the performance of algorithm.

$$CP(c) := \frac{\max\{l_-, L_+\}}{l}$$

Within the value range of [0, 0.10], the nest performed c is 0.01. For weight with c larger than 0.01 or smaller than 0.01, the performance of algorithm is unexpected. This might be resulted from the small Gaussian kernel with close value of each data.

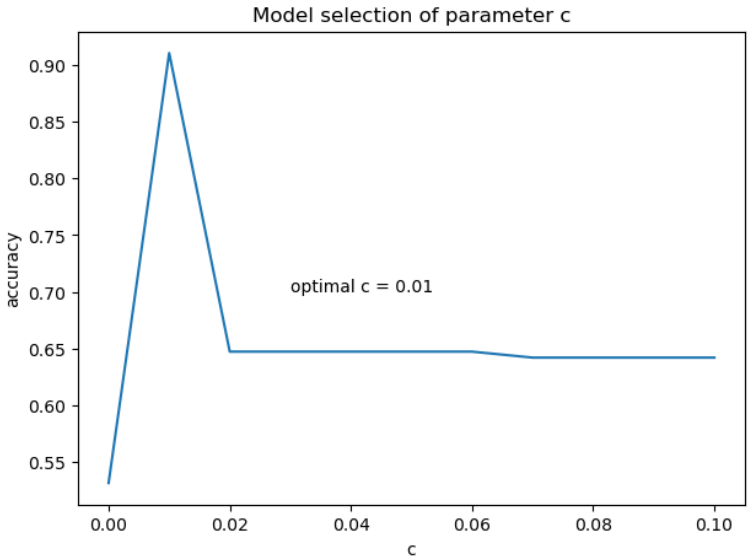


Fig 8. The relationship between the algorithm quality and value of c in range [0, 0.1]

2.4 Questions

2.4.1 CP(c)

CP(c) can minimise the influence of the calculation of eigenvector by flipping the sign of output into the correct one with the below equation

$$CP(c) \coloneqq \frac{\max\{l_-, L_+\}}{l}$$

In the calculation of eigenvectors, the vectors with the same modulus values and different sign will be considered as the same eigenvectors, which means the sign of eigenvectors might be incorrect. This results in the wrong prediction of label as the spectral clustering assigning label based on the sign of second eigenvector in eigensystem of the graph Laplacian L.

2.4.2 Eigenvalue of the Laplacian

The proof that the smallest eigenvalue of Laplacian is 0 and the corresponding eigenvector is the constant one vector is shown below.(1)

Known $L = D - W$ where $W = exp(-c||p-q||^2)$ and D is a diagonal matrix with $D = \sum_{j=1}^{l} W_{ij}$.

We can write that

$$f'Lf = f'Df - f'Wf = \sum_{i=1}^{n} d_i f_i^2 - \sum_{i.j=1}^{n} f_i f_j w_{ij}$$

$$= \frac{1}{2}\left(\sum_{i=1}^{n} d_i f_i^2 - 2\sum_{i,j=1}^{n} f_i f_j w_{ij} + \sum_{j=1}^{n} d_i f_j^2\right) = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2$$

So that L is positively semi-definite, and its eigenvalue are non-negative for all $f \in \mathbb{R}^n$. As 0 and $\underline{1}$ is an eigen pair of L, 0 is the smallest eigenvalue of L and its corresponding eigenvector $\underline{1}$ is a constant vector.

2.4.3 Why spectral clustering “works"

Spectral clustering works by representing the data as a (weighted) graph and the differences between data are represented by the weight of the borders between the points from different clusters. To bipartite the graph, sparsest cut is a good choice. To find best line of bipartite is equivalent to minimize the $\frac{f'Lf}{f'f}$ where L is the graph Laplacian. This is similar to the minimization and maximization quadratic problem on unit space. To find the solution, it is needed to obtain the minimum eigen pair. In above experiments, the smallest eigenvalue does not provide adequate information, so second smallest pair might be a good choice.

2.4.4 Parameter c

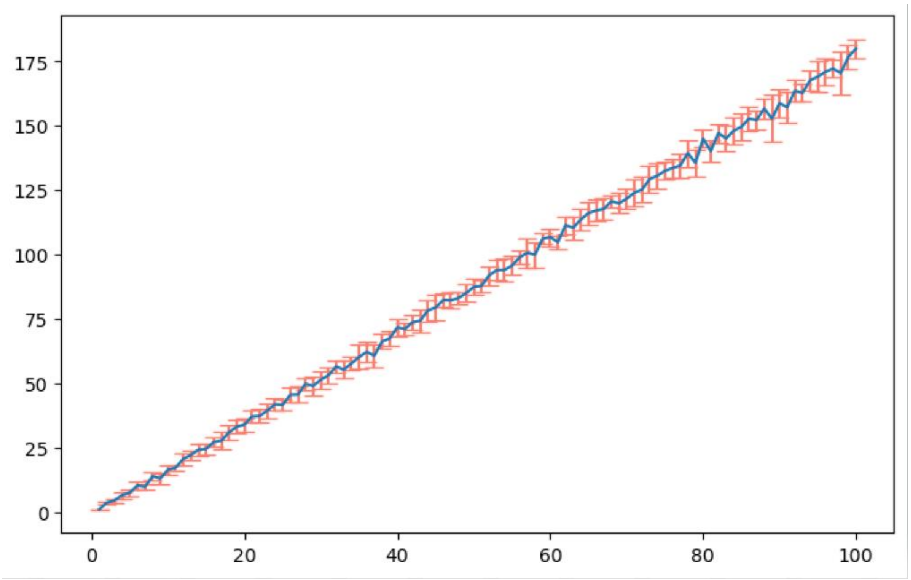
The parameter c in this algorithm is corresponding to $\frac{1}{\alpha^2}$, which is inversely linear with the spread of the Gaussian distribution. If the points lie outside the standard deviation range of the distribution (i.e., α is large), the relevant weight will be small. So, the higher c is, the higher similarity weight is, the more accuracy the algorithm will archive. In addition, c should be smaller than $2^{8.2}$ to avoid the convergence of accuracy to 0 as shown in Fig 5, 6 and 7.

3 PART III - Spectral Clustering

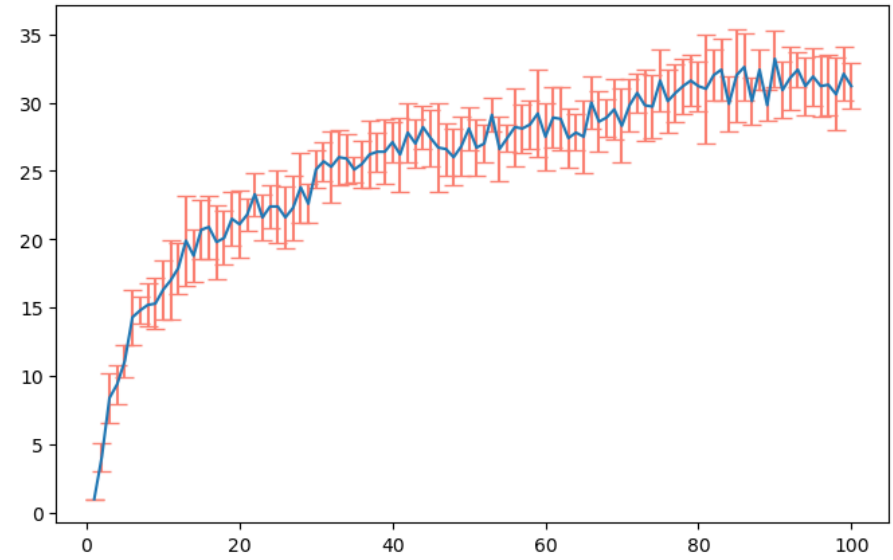
3.1 Sparse Learning

(a) Complexity

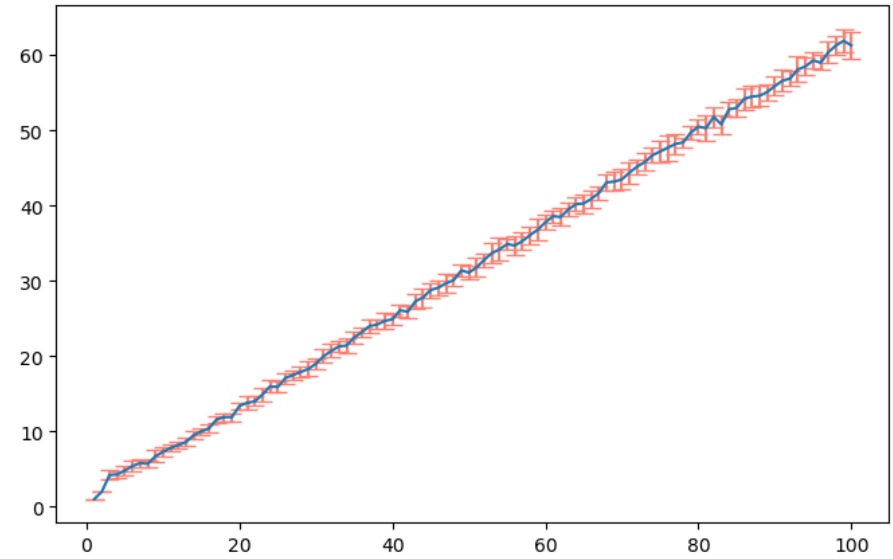
implement the classification algorithms of perceptron, winnow, least squares, and 1-nearest neighbours and obtain their sample complexity. (because of the long time taken of 1-NN, I only run it with dimension from 1 to 15.)



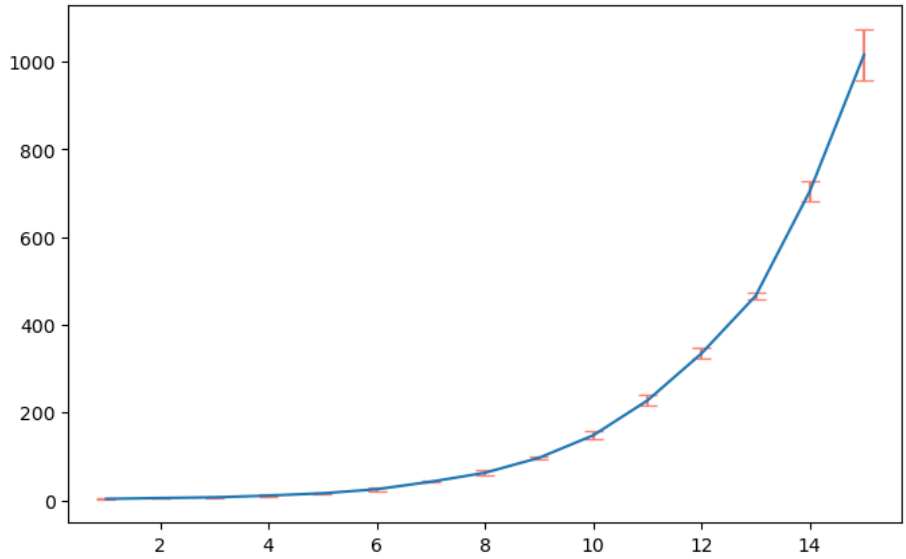
(1) Perceptron



(2) Winnow



(3) Least Squares



(4) 1-NN

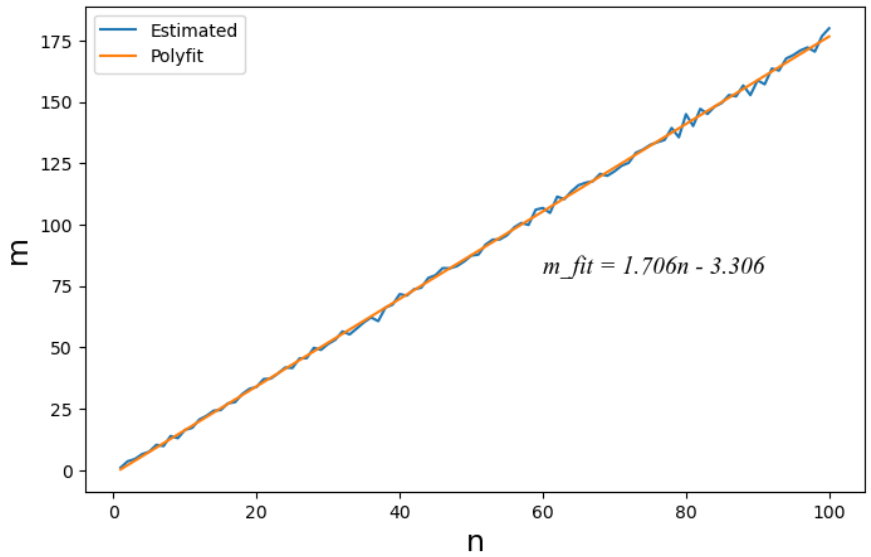
Fig 8. The relationship between predicted no. of samples to obtain 10% generalisation error and dimension (n) of the algorithm.

(b)

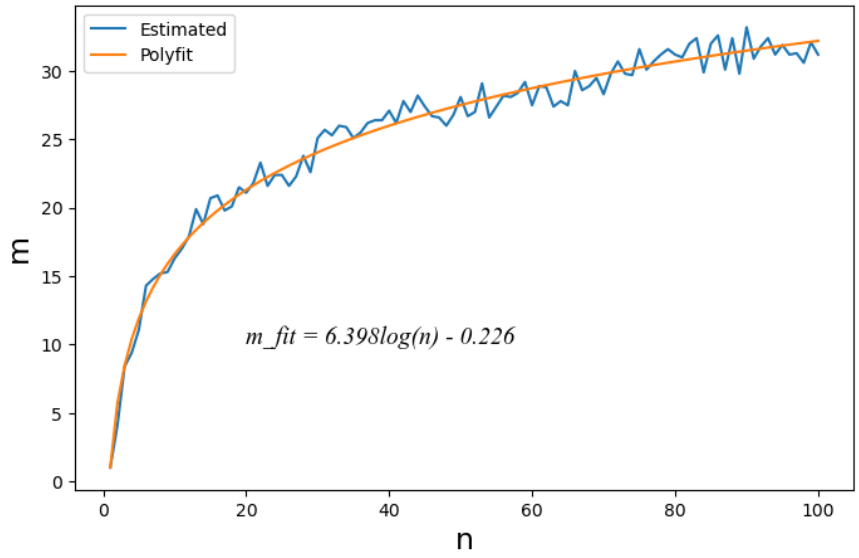
(1) describe your method for estimating sample complexity in detail.

(2) discuss the tradeoffs and biases of your method

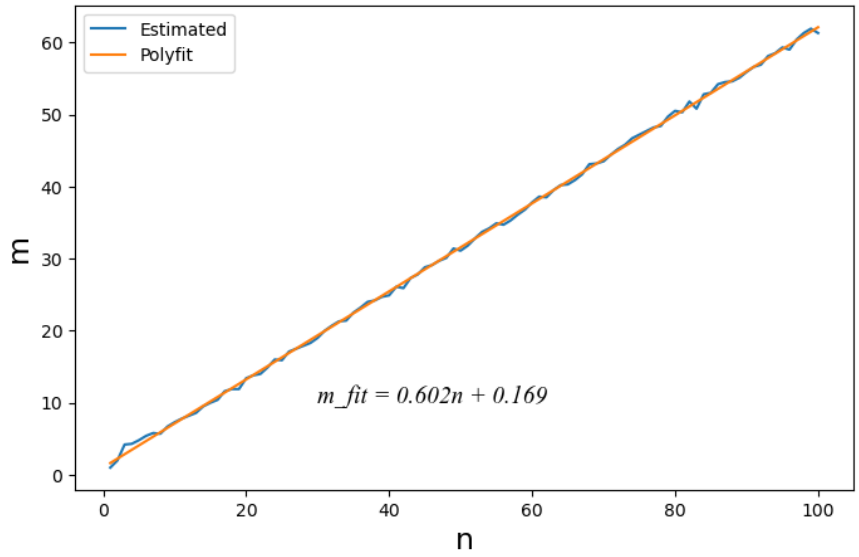
(c)



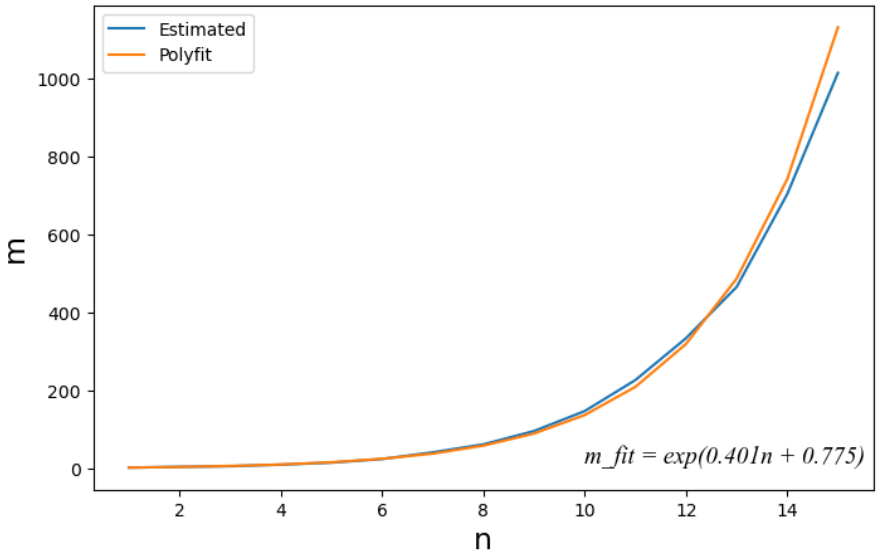
(1) Perceptron



(2) Winnow



(3) Least Squares



(4) 1-NN

Fig 9. Comparation of predicted and fitted no. of samples (m)

(d)

Reference

1. Von Luxburg, U. A tutorial on spectral clustering. *Statistics and Computing* 17, 395–416. https://arxiv.org/pdf/0711.0189.pdf (Aug. 2007).