Supervised Learning (COMP0078) – Coursework 2

Student1 (21118015)
Student2 (21181074)

December 18, 2021

Abstract

This is the answer report for Supervised Learning (COMP0078) Coursework 2. Answers are arranged in the order of Part I, Part II and Part III. All the code are attached at the end of this report with a single jupyter notebook file attached in the submission of work.

Contents

Packages

In this submission, our coding is done with jupyter notebook, using python 3.9 as the kernel language. As required, we have implemented all algorithms by writing our own functions and classes. To clarify that no packages that was not allowed were used in our work, here is a list of packages we used in this coursework:

- Numpy, for mathematical processing with data such as calculating, sorting and etc.;
- Numba.njit, for accelerating the speed of running the code in part I with lots for loops;
- matplotlib.pyplot, to give plots required in the coursework;

1 Part I

1.1 Introduction

Perceptron is a classical discriminant model for binary classification task. In this exercise, we aim to classify handwritten digits 1-9 pictures from 'zipcombo' dataset. This dataset includes 9298 samples, each with 256 attributes corresponding to every gray scale pixels of a 16*16 picture. So, we generalise the perceptron method to muti-classification task in two approaches: 'One Vs Rest' and 'One Vs One', and utilize the algorithm with two different kernels—Polynomial kernel and Gaussian kernel to separate 10 different classes with a nonlinear boundary. In this part, we analyze and compare the predicting performance with different classification strategies, and do more further study on he hyper-parameter's effect on kernel trick's performance.

Primal format perceptron (Online training setting)

Input: training set $D = \{(\boldsymbol{x}_k, y_k)\}_{k=1}^{m}, \boldsymbol{x}_k \in \{-1, 1\}^n, y_k \in \{0, 1...9\}$
Initialization: $\boldsymbol{w}_0 = [w_{01}, w_{02}, ..., w_{0n}]^T = [0, 0, ..., 0]^T$
Repeat:
Prediction: Once receiving tth sample $\{\boldsymbol{x}_t, y_t\}$, we make a prediction:$\hat{y}_t = sign\left(\boldsymbol{w}_t^T \boldsymbol{x}_t\right)$
Update: if $\hat{y}_t \neq y_t$, $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + y_t \boldsymbol{x}_t$
Until: No mistakes in the dataset

1.2 Theory

1.2.1 Perceptron algorithm

Perceptron algorithm is a kind of linear classification model with the goal of training a hyper-plane to classify +1 and -1 in a linearly separable space. The algorithm process of primal format is as below:

There is an intuitive explanation for the primal format. When the algorithm receives a wrong classified sample point, the hyper-plane will move closely to this point, consequently reduce the distance between the point and the hyper-plane[1].

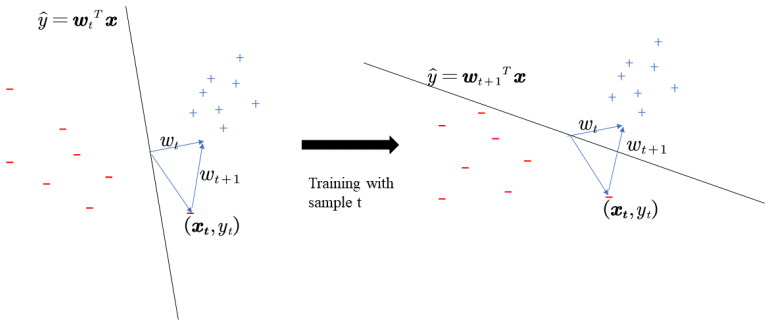

Figure 1: Training process of primal format perceptron

However, the primal perceptron only have the capacity of fitting linear functions. That means if our data space is not linearly divisible, this primal format does not work. So, we induce the kernel trick into to map the data features into a higher dimension, where the linear function can separate them. With the kernel trick, we can infer the dual format of perceptron.

Considering that when we traverse the data set for N epochs, each sample is traversed for N times. We set $\beta_i \geq 0, i = 1, 2..., m$, which represents number of misclassifications of ith sample through N epochs. Therefore,

$$\boldsymbol{w}_t = \sum_{i=1}^{t-1} \beta_i y_i \boldsymbol{x}_i := \sum_{i=1}^{t-1} \alpha_i \boldsymbol{x}_i$$

$$\hat{y}_t = sign\left(\boldsymbol{w}_t^T \boldsymbol{x}_t\right) = sign\left(\left(\sum_{i=1}^{m} \alpha_i \boldsymbol{x}_i\right)^T \boldsymbol{x}_t\right) = sign\left(\sum_{i=1}^{m} \alpha_i \boldsymbol{x}_i{}^T \boldsymbol{x}_t\right) = sign\left(\sum_{i=1}^{m} \alpha_i \boldsymbol{x}_i{}^T \boldsymbol{x}_t\right)$$

When we map $\boldsymbol{x}$ to the feature $\Phi(\boldsymbol{x}_i)$ in a higher dimensional space and then use the dual format perceptron, the kernel appears.

$$\hat{y}_t = sign\left(\sum_{i=1}^{m} \alpha_i \Phi(\boldsymbol{x}_i)^T \Phi(\boldsymbol{x}_t)\right) = sign\left(\sum_{i=1}^{m} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}_t)\right)$$

From the formula above, we find that kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_t) = \Phi(\boldsymbol{x}_t)^T \Phi(\boldsymbol{x}_t)$, so we can computing the kernel function value and store them into the Gram Matrix: $G = [K(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{m \times m}$, where m is the size of sample set, consequently when we need to update the classifier, we can get corresponding kernel value quickly from the Gram Matrix.

From the algorithm, we know that if we do not compute the Gram Matrix, we need to do vector product to calculate the kernel value for two samples for t times in every loops, which is time consuming. So with the use of Gram Matrix, we can save much more time in code running.

For kernel perceptron, there is also an intuitive explanation. The kernel value can be considered as the measurement of the similarity between two samples. In the algorithm, every time when we get a new sample, we

Kernel perceptron (Online training setting)

Input: training set $D = \{(\boldsymbol{x}_k, y_k)\}_{k=1}^{m}$, $\boldsymbol{x}_k \in \{-1, 1\}^n$, $y_k \in \{0, 1...9\}$
Initialization: $\boldsymbol{w}_0 = 0, \boldsymbol{\alpha}_0 = [\alpha_1, \alpha_2, ..., \alpha_m]^T = [0, 0, ..., 0]^T$
Repeat:
Prediction: Once receiving tth sample $\{\boldsymbol{x}_t, y_t\}$, we make a prediction:

$$\hat{y}_t = sign\left(\sum_{i=0}^{t-1} \alpha_i K\left(\boldsymbol{x}_i, \boldsymbol{x}_t\right)\right) = sign\left(\boldsymbol{\alpha}^{\boldsymbol{T}} G_t\right) \ , G = [G_1, G_2, ..., G_m]$$

Update: if $\hat{y}_t \neq y_t$, $\alpha_t = y_t$
Until: No mistakes in the dataset

need to compute the similarity between it and every misclassified points. It is obvious that the hardest predicted misclassified points are with larger α (without considering α's sign). So $\boldsymbol{w}_t\left(\boldsymbol{x}_t\right)$ is the linear combination of the similarity between $\boldsymbol{x}_t$ and all current misclassified points. If the new sample is more similar with one of the hardest predicted point $\boldsymbol{x}_i$, that means $\boldsymbol{x}_t$ is not far away from $\boldsymbol{x}_i$ in the feature space, so this sample is more likely to possess the same label with $\boldsymbol{x}_i$. So we conclude that the perceptron algorithm's performance is sensitive to the misclassified points, hence when we select the hyper-parameter of the algorithm in the kernel, we need to care more about information of the misclassified point. This will be clarified more clearly in later experiments of the exercise.

1.2.2 Generalization from binary-class to multi-class classification

The perceptron algorithm is designed to solve binary-class classifying problems. However, in this exercise, we have 0-9, 10 different labels for the dataset. So in order to deal with the task, we have to train multiple perceptrons with the same data sequence at the same time and then ensemble the predicting result of these perceptrons to achieve an multi-class classified result. There are two classical approaches: 'One Vs Rest' and 'One Vs One' . Here we discuss and compare such two approaches.

1.2.2.1 " One Vs Rest "

"One Vs Rest"('OvR') approach aims to ensemble N_c base binary classifiers together (N_c is the number of classes, N_c=10 in our task). Each classifier learns to separate only one of ten classes, i.e. the ith (i=j+1, $j = C_i \in \{0...9\}$) classifier will consider samples with label C_i as the positive sample, while other samples will be considered as negative samples. In the prediction stage, each binary classifier will provide a prediction result(so called confidence) for a specific class when the predictor get a new test sample, and we choose the classifier result with the highest confidence as the final label for such new sample.

1.2.2.2 " One Vs One "

Unlike 'One Vs Rest' strategy, we need to train $\frac{N_c(N_c-1)}{2}$ binary classifiers to generalize 10-class classification in " One Vs One "(OvO) strategy. Each classifier can only separate the class C_i and C_j, it considers samples with C_i label as positive samples, while those with C_j labels as negative samples. In predict stage, we can get $\frac{N_c(N_c-1)}{2}$ classification results for a new test sample, and the final label for the test sample will be voted by the results of $\frac{N_c(N_c-1)}{2}$ classifiers.

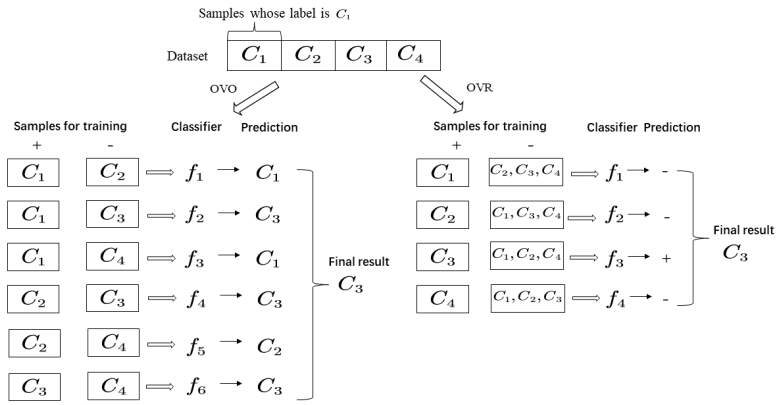

Figure 2: Schematic diagram for 'OvO' and 'OvR' [3]

From the schematic diagram, we find that the computational complexity of 'OvR' is $O(N_c)$,while that for 'OvO' is $O\left(N_c^2\right)$, which means 'OvO' could be more time consuming than 'OvR' in training and testing stage[3].

1.2.3 Experiment results and analysis

1.2.3.1 Epoch number selection

We perform a pre-experiment based on Polynomial Kernal with 'OvR' strategy, setting 20 epochs and 1 run. We get the pre-experiment result as below:

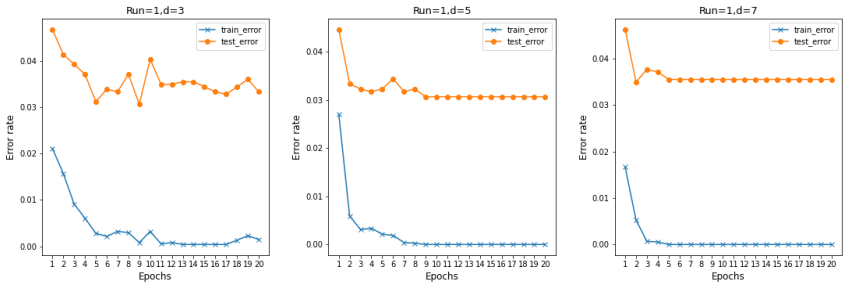

Figure 3: Error rate for training and testing among 20 epochs

We can easily find that the training error and testing error trend become stable after 10th epoch. So it is not necessary for us to waste time to run more epochs and set total epochs for every experiments in Part1 as 10.

1.2.3.2 Basic result for Polynomial kernel perceptron

Polynomial kernel's expression is $K_d(\boldsymbol{p}, \boldsymbol{q}) = (\boldsymbol{p}, \boldsymbol{q})^d$. When d=1, the kernel perceptron is equivalent to dual format of classical perceptron. In our experiment, we benchmark perceptron's average prediction performance both on train set and test set over 20 runs for every potential hyper-parameters.

Table 1: Classification performance for polynomial kernel perceptron by 'OvR'

d	Train error(%)	Test error(%)
1	6.31487±1.485363	8.876344±1.525205
2	0.683652±0.456544	4.053763±0.552849
3	0.149234±0.104132	3.217742±0.412361
4	0.123017±0.089915	3.158602±0.355358
5	0.091422±0.142003	3.107527±0.369992
6	0.049072±0.063812	3.053763±0.392363
7	0.0847±0.234994	3.13172±0.505455

The performance for polynomial kernal perceptron by 'OvR' is shown on the table above, we find that both train error and test error experience a precipitous decline when hyper-parameter d increases from 1 to 2 , afterwards error rate gradually converges. This phenomena shows that the data set contains a lot of non-linearly divisible samples, which can not be separated by a linear hyper-plane. However, we can fit some more complicated nonlinear boundaries with d increasing, which could help classify those non-linearly divisible samples. From the table, we still find that when d increases from the 6 to 7, the error rate for both training set and test increases(over-fitting occurs). This situation attribute to the dimensional curse, i.e. when we kept increasing the features in feature space, the samples in the feature could be sparser, consequently the classifier over fit the noise points which are not necessary to fit.

The above table shows that the classification performance trend with d increasing in 'OvO' is similar to 'OvR' strategy, because it is the information of the data set that determines which hyper-parameter has a better performance. We also find that when feature space's dimension is more than 2, the 'OvR' strategy performs better that 'OvO', because unlike the "OvR" strategy, 'OvO' trains each binary classifier with specific train set, which contributes to that the distance between samples became really far, especially in higher dimension space. So the fitting hyper-plane seems to be with a weaker generalization capability because of sparsity of samples.

The question is that which hyper-parameter is most suitable for this data set? We perform a 5-fold cross validation to answer this question. From the table, 'OvR' and "OvO" strategy both give a hint that the best parameter is around 4-5. The best parameter for 'OvO' strategy seems smaller than that of 'OvR' because of the limited training data mentioned above, which causes that the 'OvO' only perform well when the dimension of feature space is not too high.

Table 2: Classification performance for polynomial kernel perceptron by 'OvO'

d	Train error(%)	Test error(%)
1	2.912073±0.646208	6.395161±0.743312
2	0.104867±0.077069	3.467742±0.405371
3	0.032267±0.030776	3.258065±0.350247
4	0.146545±0.540292	3.327957±0.637703
5	0.018822±0.011565	3.336022±0.404648
6	0.016133±0.013842	3.284946±0.398321
7	0.025545±0.048082	3.459677±0.420345

Table 3: Test error and optimized hyper-parameters' information over 20 runs

	OvR strategy(%)	OvO strategy(%)
Test error(%)	3.290322±0.409308	3.13172±0.312973
Optimal hyper-parameter	5.1±1.220655	4.4±1.280624

In this part, we analyze classification error using confusion matrix, which gives a intuitive format of showing the test error between each classes. The row index i of confusion mateix refers to the truth label, while column index j refers to prediction result, every cells store the mean error rate of misclassify i as j.

0±0	0±0	4.569892±4.585678	4.83871±5.349395	2.688172±3.606561	5.107527±4.648284	2.688172±3.180688	0±0	0.537634±1.612903	0.537634±1.612903
0±0	0±0	1.075269±2.150538	0.537634±1.612903	9.946237±26.463139	0.537634±1.612903	2.419355±3.978131	1.344086±2.882743	1.075269±2.741408	1.612903±3.839478
4.83871±4.465927	1.612903±2.46375	0±0	6.72043±5.342636	9.946237±7.066365	1.612903±2.46375	1.344086±2.328025	6.72043±5.606627	6.989247±5.913978	1.075269±2.741408
2.150538±3.13492	0.268817±1.171747	8.064516±5.240212	0±0	1.075269±2.150538	20.698925±9.507924	0.268817±1.171747	4.301075±4.686988	6.989247±4.83871	0.806452±1.919739
1.075269±2.741408	4.83871±6.106353	5.376344±6.800597	1.612903±2.46375	0±0	1.344086±2.882743	2.956989±3.169308	2.150538±2.63386	2.419355±3.596529	12.903226±12.189015
5.376344±4.808748	0.537634±1.612903	2.956989±3.596529	15.053763±11.096649	2.956989±3.169308	0±0	5.376344±5.100448	0.268817±1.171747	5.645161±3.596529	5.107527±4.94945
7.526882±5.986843	1.88172±3.076754	2.956989±4.326203	0±0	4.569892±4.8907	3.494624±4.585678	0±0	0±0	1.612903±3.44254	0.268817±1.171747
0.268817±1.171747	1.075269±2.150538	3.225806±4.624906	1.612903±2.993422	8.333333±6.688094	0.806452±1.919739	0±0	0±0	3.763441±3.839478	9.946237±5.708807
6.451613±6.715052	3.494624±3.515241	5.645161±5.759216	11.55914±9.807224	4.032258±6.557156	9.139785±6.606562	2.150538±3.566263	2.150538±3.566263	0±0	3.225806±3.566263
1.344086±2.328025	1.344086±2.882743	0.806452±1.919739	3.225806±3.566263	8.870968±6.42355	1.88172±3.076754	0±0	8.333333±6.240961	1.075269±2.150538	0±0

Figure 4: Confusion matrix

		Prediction									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Truth	**0**	0	0	4.57	4.839	2.688	5.108	2.688	0	0.538	0.538
	1	0	0	1.075	0.538	9.946	0.538	2.419	1.344	1.075	1.613
	2	4.83871	1.613	0	6.72	9.946	1.613	1.344	6.72	6.989	1.075
	3	2.150538	0.269	8.065	0	1.075	20.7	0.269	4.301	6.989	0.806
	4	1.075269	4.839	5.376	1.613	0	1.344	2.957	2.151	2.419	12.9
	5	5.376344	0.538	2.957	15.05	2.957	0	5.376	0.269	5.645	5.108
	6	7.526882	1.882	2.957	0	4.57	3.495	0	0	1.613	0.269
	7	0.268817	1.075	3.226	1.613	8.333	0.806	0	0	3.763	9.946
	8	6.451613	3.495	5.645	11.56	4.032	9.14	2.151	2.151	0	3.226
	9	1.344086	1.344	0.806	3.226	8.871	1.882	0	8.333	1.075	0

Figure 5: Colour scale chart of confusion matrix

From the colour scale chart, we can see that the classify has a related high possibility to misclassify 8 as 3, 5 as 3, 3 as 5 , 4 as 9 and 2 as 4. Also, we get the five hardest classified samples (samples whose prediction confidence range among different base classifiers are smallest) from the sample set, as the figure shown below:

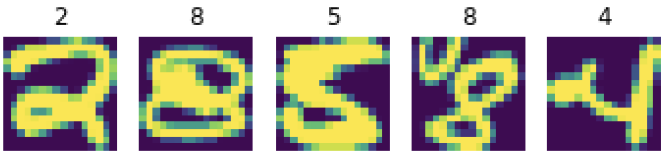

Figure 6: Five hardest classified samples

While for such samples, our prediction are 9,2,2,4,2 respectively. From the picture, we observe that such samples either have a really similar attributes with the predicted digits either really different from digit 0..9. So it is hard for our perceptron to make a correct prediction on such samples.

1.2.3.3 Basic result for Gaussian kernel perceptron

We also implement Gaussian kernel for this part. Gaussian kernel $K_c(\boldsymbol{p}, \boldsymbol{q}) = e^{-c\|\boldsymbol{p}-\boldsymbol{q}\|^2}$, we can also represent Gaussian kernel as $K_\sigma(\boldsymbol{p}, \boldsymbol{q}) = \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\|\boldsymbol{p}-\boldsymbol{q}\|^2}{2\sigma^2}}\right) \times \sqrt{2\pi}\sigma$, $c = \frac{1}{2\sigma^2}$. It is known that the perceptron's training process depends on the current misclassified points, and the kernal value measure the similarity between the new sample and those misclassified points.

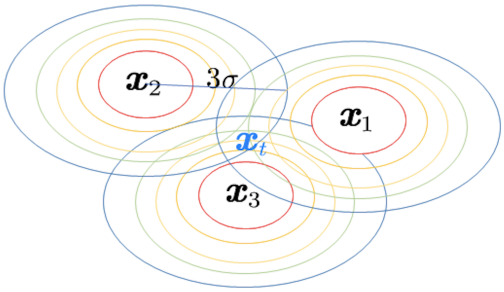

Figure 7: Gussian kernel perceptron

What we want is to make full use of every misclassified points, which means the new sample's kernel value with all current misclassified points should not be 0 as far as possible. From property of Gaussian distribution, more than 99% of possibilities laid in the interval $[\boldsymbol{x}_t - 3\sigma, \boldsymbol{x}_t + 3\sigma]$. So with the use of this property,it doesn't hurt to let

$$3\sigma \geq \|\boldsymbol{x}_t - \boldsymbol{x}_i\|^2 \ , for\ \boldsymbol{x}_i \in M$$

However, because the M set keeps changing with new training data fed, so it is hard to fix the σ's range. So we loose the condition as below:

$$3\sigma \geq \max_{\boldsymbol{x}_i} \left(\|\boldsymbol{x}_t - \boldsymbol{x}_i\|^2\right) \ , for\ \boldsymbol{x}_i \in D$$

With this condition, we can roughly estimate sigma's range using $3\sigma \geq 25.21$,where 25.21 is the maximum distance between any two samples in the data set, so we can induce the range of c as $0 \leq c \leq 0.007$. Because this is only a rough estimation, so we choose some values in the neighbourhood of 0.007 to search for a better parameter when we perform the experiment. Therefore, we create a hyper-value set:

$$S = \{0.0001, 0.001, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01, 0.011, 0.02, 0.05, 0.1\}$$

Then we benchmark algorithm's performance on this set:

Table 4: Classification performance for Gaussian kernel perceptron by 'OvR'

c	Train error(%)	Test error(%)
0.0001	13.985614±4.004971	14.806452±3.708368
0.001	5.379806±1.865041	7.739247±1.838873
0.005	0.399973±0.227481	3.879032±0.557503
0.006	0.219817±0.146987	3.446237±0.568699
0.007	0.198978±0.105742	3.325269±0.492831
0.008	0.108228±0.083364	3.217742±0.400267
0.009	0.055795±0.046432	3.107527±0.456071
0.01	0.044367±0.038989	2.946237±0.352386
0.011	0.046383±0.051102	3.123656±0.50317
0.02	0.026217±0.031379	3.094086±0.392317
0.05	0.006722±0.009971	3.94086±0.400925
0.1	0±0	5.075269±0.573129

From the train error and test error over S, we find that when c=0.01,the performance of kernel perceptron is the best, no matter in terms of training error or test error. Based on set S, we also implement a 5-fold cross validation to find the best hyper-parameter over the whole test set, the result is in the table below:

Table 5: Test error and optimized hyper-parameters' information on different kernel perceptrons

	Polynomial perceptron (OvR strategy)(%)	Gaussian kernel perceptron(%)
Test error(%)	3.290322±0.409308	3.077957 ±0.3622056
Optimal hyper-parameter	5.1±1.220655	0.00995 ±0.001359228

We observe that averge performance of the Guassian kernel perceptron is better in terms of test error, due to Gaussian kernel map the data sample into the infinity dimensional space, which means the gaussian kernel has a strong and flexible mapping capacity than polynomial kernel.

2 Part II

In this section we implement a basic spectral clustering method and apply it to given data twomoons.dat and dtrain123.dat.

In this section, we use the given procedure on Spectral Clustering, however there are few things changing when we adjust between different experiments:

Firstly, the data set is changing, for experiments 1 and 3 we are using the given data set twomoons.dat and dtrain123.dat; for experiment 2 we are using the randomly generated data from given distribution.

Secondly, the parameter α in the Gaussian Kernel we used in this model is changing in order to get the best clustering result, where represents the bandwidth of the Gaussian kernel, we will talk about the affects of α on the spectral clustering.

For each single experiment, we draw two figure representing the result:

First the figure of Correct points vs. value of c(or, α) we chosen, in order to find the best clustering bandwidth parameter;

Second the figure of one example of clustering result, showing how well the clustering results are, however, this wouldn't apply to experiment 3 since dtrain123.dat is not consist of 2-d plot.

The Correct Points are defined following the coursework2, which is:

$$CP(c) = \frac{max(\text{number of correct labelled points, number of wrongly labelled points})}{\text{number of all data points}}$$

2.1 Experiments

1.Two Moons

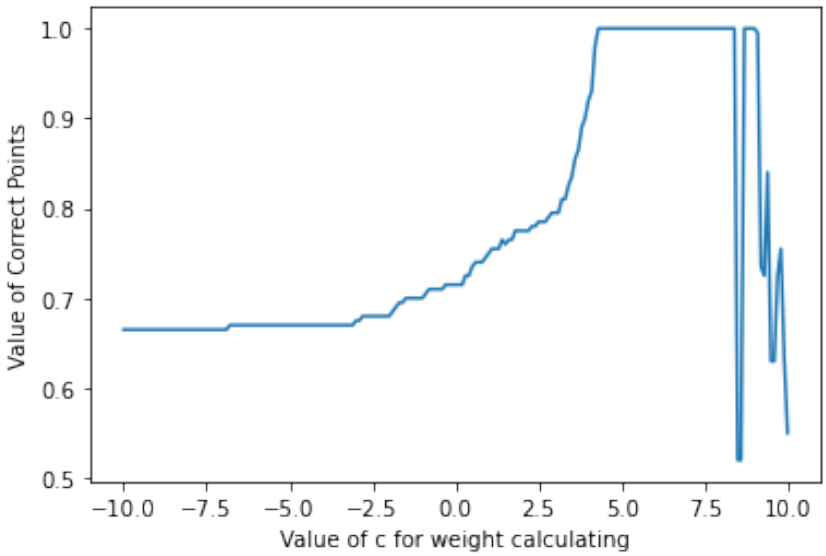

Figure 8: Correct Points for twomoons clustering, it is seen that the number of c that correctly cluster the data is $2^{\alpha}, \alpha = [4.3 - 9.0]$ without $[8.5, 8.6]$

Figure 9: Plot for twomoons clustering

2.Gaussian Distribution data

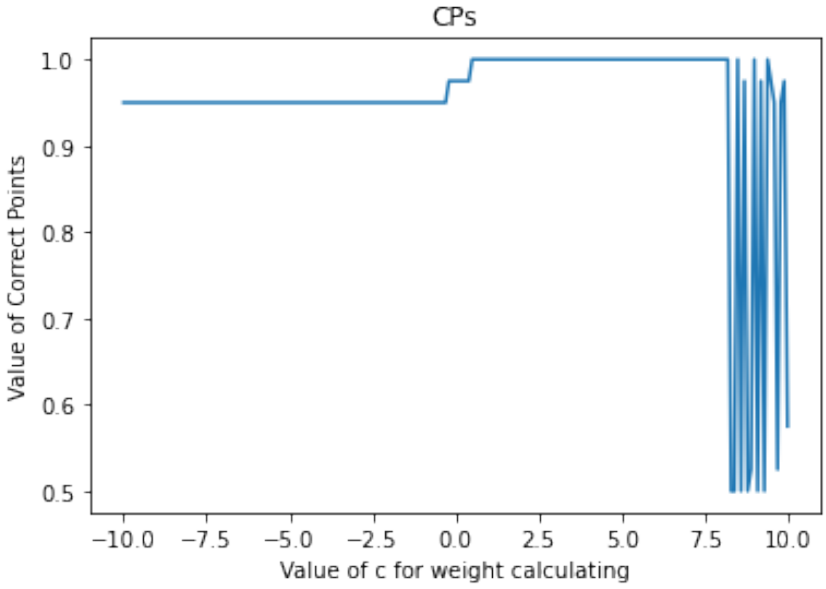

Figure 10: Correct Points for Gaussian Distributed data clustering, it is seen that the number of c that correctly cluster the data is $2^{\alpha}, \alpha$ is almost all the values from -10 to 10, except for some values beyond 8.0

Figure 11: Plot for Gaussian clustering with $c = 2^{\alpha}$, $\alpha = 5.0$

3.Given data dt123.dat

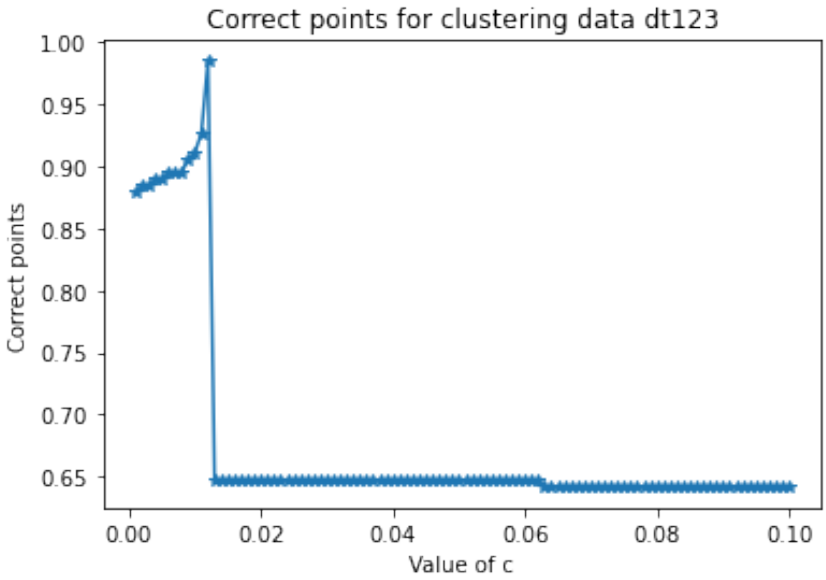

Figure 12: Correct points for Clustering data from dt123.dat, the best clustering parameter is $c = 0.012$

2.2 Questions

1. CP(c) is reasonable for there are two groups to be separated, so all in all there are 4 possibilities of assigning the label:
   i) Origin label +1, predict label +1;
   ii) Origin label +1, predict label -1;
   iii) Origin label -1, predict label -1;
   iv) Origin label -1, predict label +1.
   While in either ways of clustering the data input, the max function would yield the maximum number of points that are either correctly predicted or reversely predicted, those are the separation of different clusters. So the CP(c) is is level of separation between groups predicted comparing to the origin clusters, which is reasonable for bench-marking the outcome of clustering.
2. Because the Laplacian is a singular matrix which is the result of linear operation on Weight matrix W, so that the determinant of the singular matrix is zero which yields the multiple of all eigenvalues of such matrix is 0. On the other hand, the Laplacian is a PSD for $\forall f$,
   $f^T L f = \sum \frac{1}{2}\omega_{i,j}(f_i - f_j)^2 >= 0$
   with conclusion, the eigenvalue of Laplacian matrix is non negative and with multiple 0, which yields there is at least one 0 eigenvalue here.

   And the zero eigenvalue does not yield any special pattern of the eigenvector, for all vectors multiple the 0 is zero vector, so the corresponding eigenvector is arbitrary constant vector.
3. Spectral clustering works for it has reduced the dimension of weight matrix into a eigen system of eigen values and eigen vectors, which is at most reduced the dimension of previous n(points) * n dimension data into a n(points) * m(groups or clusters) dimension of data, hence the number of clusters m is significantly smaller than number of points n in the clustering problems, this is relatively high performance for clustering of high dimensional data. The eigen value represents the similiarty of each other points hence the clustering onto eigen systems still works as the weight matrix. This is why spectral clustering works.
4. Parameter c in the parameter representing the bandwidth coefficient of a Gaussian Kernel, which represents likewise the 'priority' of how much distance between the data point are considered valuable for valuating the weights in the weight matrix. By changing the bandwidth of Gaussian Kernel, different range of points are likely to be considered in the same group of points hence the clustering result varies. It likes the k-Means Clustering for choosing different c in the Gaussian Kernel, for different c means by the end of the separation we consider k(c) of neighbour points as the same group of points. Where k(c) is the number of neighbour points that are considered the same group for the middle point which is the first term in the Gaussian kernel and k(c) is a function of c, that is how c is affecting the result's quality of clustering, for larger c would yields smaller variance of data and the cluster would be more significant, but too large c would yield some kind of

over fitting. At the same time, too small c meant too large variance of Gaussian Kernel which would yield high value of k and the result would be under-fitting.

3 Part III

3.1 Questions

1. Sparse Learning

- (a) The four plot for the four algorithm of sample complexity is:

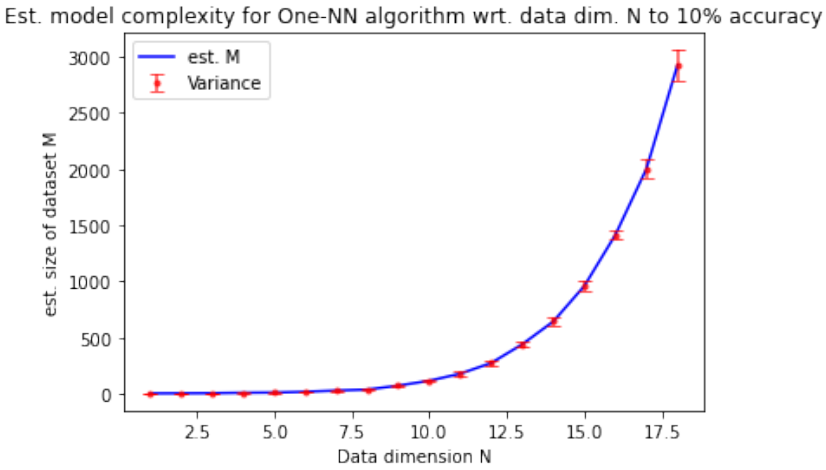

Figure 13: 1-NN

Figure 14: Least Regression

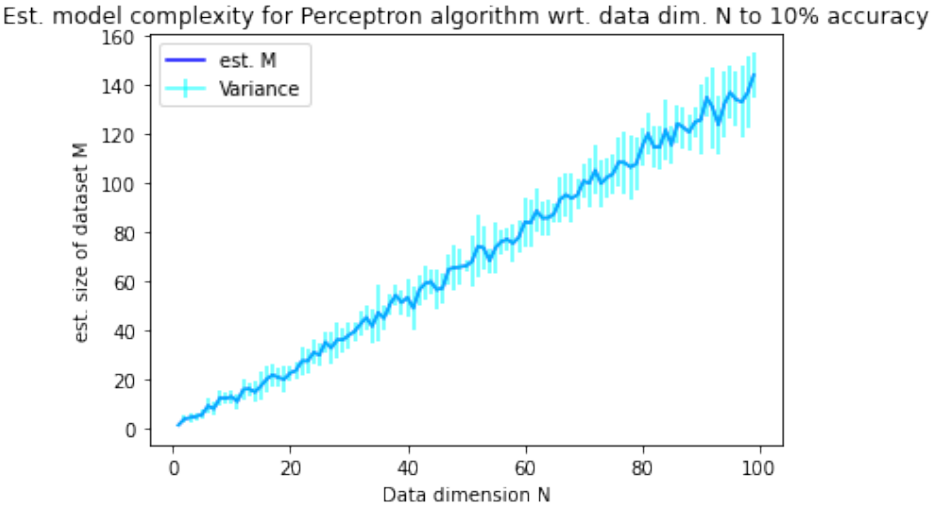

Figure 15: Perceptron

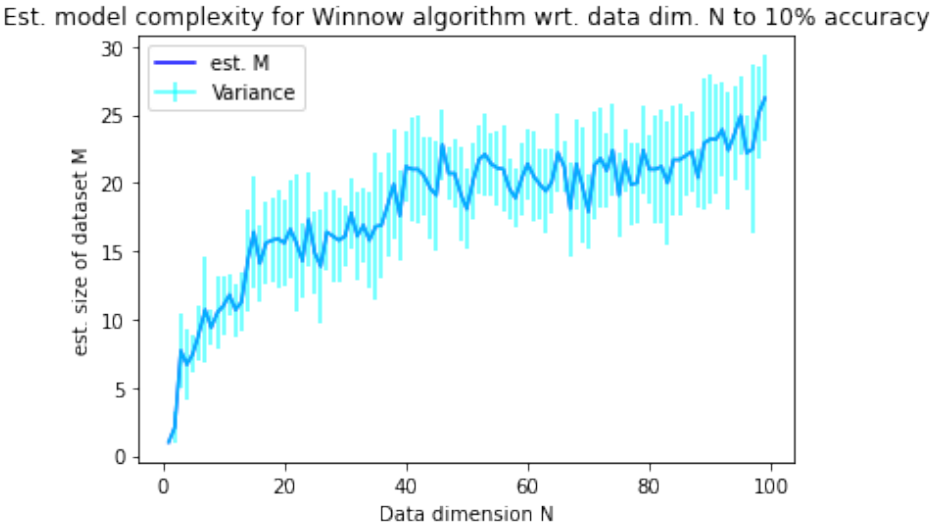

Figure 16: Winnow

- (b) Techniques used for estimating the generalization error:

  1.A number of relatively large test set with size = 10000 points are generated for testing

  2.For each iteration of n, the m are calculated under 10 runtimes such that the mean value and standard deviation are calculated from 10 trials.

  3.From 1 and 2, we estimate the sample complexity by approximately the mean value of 10 runs under strategy of bounding the test error, since the generalization error are computationally hard to calculate.

  (1) The value of generalization error is larger than the estimate value of test error, so the number of sample complexity calculated is smaller than the true value, which indicates that there is a loss of accuracy for computing with exchange of feasible running time for the process.

  (2) Although the test error cannot represent the actual value of generalization error, due to the high volume of test set, it is likely that the test error would converge to the generalization error within some level.

  Trade-off

  (i) The value of generalization error is larger than the estimate value of test error, so the number of sample complexity calculated is smaller than the true value, which indicates that there is a loss of accuracy for computing, this is the trade-off of less computing time

  (ii) The error cannot represents the actual value of generalization error since the test error are bounded by limited number of testing set size, hence the generalization level is less than the true situation, which is the second trade-off of less computing time.

  Bias

As discussed above, the used strategy would always yield a smaller sample complexity than the actual value, which is the bias of the such techniques we chosen for this 'just a little bit' problem.

- (c) By curve-fitting, the functions for $m = f(n)$ is shown below:

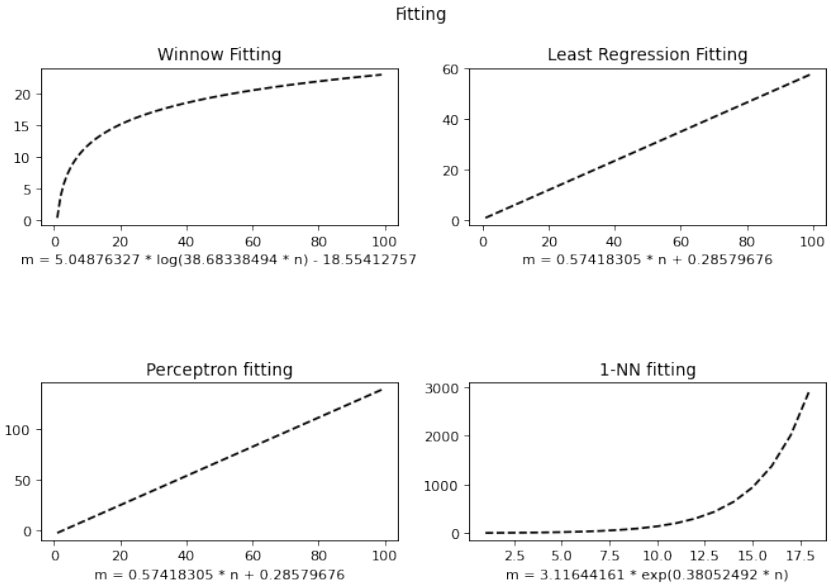

Figure 17: Fitting

K-nn: $m = 3.11644161 * exp(0.38052492 * n)$
Least regression: $m = 0.57418305 * n + 0.28579676$
Perceptron: $m = 1.45972418 * n - 4.3619666$
Winnow: $m = 5.04876327 * log(38.68338494 * n) - 18.55412757$
It is obvious that all the fittings are with limited variations, so the overall relationship for m and n are
For Linear Regression and Perceptron, which is of linear relationship:

$$m = \Theta(n) \tag{1}$$

For Winnow, with log relationship:

$$m = \Theta(log(n)) \tag{2}$$

For One-NN, with exp relationship:

$$m = \Theta(exp(n)) \tag{3}$$

- (d) We consider the possibility of misclassifying s th sample by perceptron being trained over s-1 samples in advance as generalization error $\mathcal{E}(h)$. Here, we try to give a non-trival upper bound $\hat{p}_{m,n}$ for $\mathcal{E}(h)$.

Lemma 1: Novikoff Bound [1]

Given that the data set $D = \{(\boldsymbol{x}_k, y_k)\}_{k=1}^{m}$, $\boldsymbol{x}_k \in \{-1, 1\}^n, y_k \in \{-1, 1\}$is linearly separable, then:
(1) There exists a hyper-plane $\hat{\boldsymbol{w}}_{opt}{}^T \hat{\boldsymbol{x}} = 0, \|\hat{\boldsymbol{w}}_{opt}\| = 1$ separating all training data correctly; Also there exists $r > 0$ making make sure that

$$y_i \left(\hat{\boldsymbol{w}}_{opt}{}^T \hat{\boldsymbol{x}}_i \right) \geq r, for\ all\ i = 1...m$$

(2) Setting $R = \max\limits_{1 \leq i \leq m} \|\hat{\boldsymbol{x}}_i\|$, the number of perceptron's misclassification number k on the training dataset satisfies the inequality[2]:

$$k \leq \left(\frac{R}{r} \right)^2$$

Lemma 2: Hoeffding inequality [2]

Let m random variables $Z_1, ..., Z_m, Z_1, ..., Z_m \overset{i.i.d}{\sim}$ *Bernoulli distribution* ϕ i.e. $P(Z_i = 1) = \phi, and\ P(Z_i = 0) = 1 - \phi$. Let $\hat{\phi} = \frac{1}{m} \sum_{i=1}^{m} Z_i$ be the mean of those random variables, and also let any $\gamma > 0$ be fixed[3]. Then

$$P\left(\left| \phi - \hat{\phi} \right| > \gamma \right) \leq 2e^{-2\gamma^2 m}$$

The outcome of perceptron classifier can be considered as a r.v. obeying Bernoulli distribution, it doesn't hurt to set

$$\begin{cases} Z_j = 1, if\ h_i(\boldsymbol{x}_j) \neq y(\boldsymbol{x}_j) \\ Z_j = 0, if\ h_i(\boldsymbol{x}_j) = y(\boldsymbol{x}_j) \end{cases}, h_i(\boldsymbol{x}) = \boldsymbol{w}_i{}^T \boldsymbol{x}$$

We assume set $T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_{s-1}, y_{s-1})\}$, so after training on set T, we can get training error $\hat{\mathcal{E}}(h_i) = \frac{1}{s-1}\sum_{i=1}^{s-1} Z_{s-1}$. Then, with the help of Hoeffding inequality, we can get

$$P\left(\left|\mathcal{E}(h_i) - \hat{\mathcal{E}}(h_i)\right| > \gamma\right) \leq 2e^{-2\gamma^2(s-1)}$$

We want to make sure for all $h_i \in H$, where H refers to the hypothesis space($|H|$denotes the size of hypothesis space), $P\left(\left|\mathcal{E}(h_i) - \hat{\mathcal{E}}(h_i)\right| > \gamma\right)$ constantly holds. Hence,

Denote $P\left(\exists h \in H : |\mathcal{E}(h) - \hat{\mathcal{E}}(h)| > \gamma\right) = \theta$

$$\begin{aligned}
&\theta \\
=&P\left(\left(|\mathcal{E}(h_1) - \hat{\mathcal{E}}(h_1)| > \gamma\right) \vee \left(|\mathcal{E}(h_2) - \hat{\mathcal{E}}(h_2)| > \gamma\right) \vee ... \vee \left(|\mathcal{E}\left(h_{|H|}\right) - \hat{\mathcal{E}}\left(h_{|H|}\right)| > \gamma\right)\right) \\
\leq&\sum_{i=1}^{|H|} P\left(|\mathcal{E}(h_i) - \hat{\mathcal{E}}(h_i)| > \gamma\right) \\
=&2|H|e^{-2\gamma^2(s-1)} \leq 2|H|e^{-2\gamma^2 m}, \textit{where } |H| \textit{ denotes size of hypothesis space}
\end{aligned}$$

Let $2|H|e^{-2\gamma^2 m} = \delta$, so

$$\gamma = \sqrt{\frac{\ln|H| + \ln(2/\delta)}{2m}}$$

Hence, we deduce that

$$P\left(|\mathcal{E}(h) - \hat{\mathcal{E}}(h)| \leq \sqrt{\frac{\ln|H| + \ln(2/\delta)}{2m}}\right) \geq 1 - \delta$$

This means that for any hypothesises, following inequality hold with at least $1-\delta$ possibility

$$\mathcal{E}(h) \leq \sqrt{\frac{\ln|H| + \ln(2/\delta)}{2m}} + \hat{\mathcal{E}}(h)$$

Also, we find that $R = \max_{1\leq i\leq m} \|\hat{\boldsymbol{x}}_i\| \leq \left\|(1, 1, ..., 1)^T\right\| = \sqrt{n}$ and $y_i\left(\hat{\boldsymbol{w}}_{opt}{}^T \hat{\boldsymbol{x}}_i\right) \geq 1, i = 1...m$, so we can control the upper bound of $\hat{\mathcal{E}}(h)$ according to Novikoff Bound Theorem.

$$\hat{\mathcal{E}}(h) \leq \frac{1}{m}\left(\frac{R}{r}\right)^2 \leq \frac{1}{m}\left(\frac{\sqrt{n}}{1}\right)^2 = \frac{n}{m}$$

Therefore, we reason out the upper bound $\hat{p}_{m,n} = \sqrt{\frac{\ln|H|+\ln(2/\delta)}{2m}} + \frac{n}{m}$, where $|H| = 3^n$

(e) As is known to us, the computational complexity of k-NN algorithm for n dimensional samples is O(knm), then for 1-NN situation, the complexity becomes O(nm). And also, the size of sample space is 2^n due to n attributes of each sample. So, we can deduce that 1-NN algorithm is PAC learnable. For 1-NN algorithm, we find its training error is always 0, so any of hypothesises of 1-NN algorithm belongs to its version space. In order to figure out the sample complexity of 1-NN, i.e. to find a suitable m to let the leaner learn target concept efficiently, we carry out our derivation process as following.

Solution: We use S to denote the training set, and assume that the possibility of hypothesis h making an error on distribution D is larger than ε,$0 < \varepsilon < 1$. hence,

$$P(h(\boldsymbol{x}) = y) = 1 - P(h(\boldsymbol{x}) \neq y) \ = 1 - \mathcal{E}(h) \ < 1 - \varepsilon$$

Considering that the training set includes m samples derived from distribution D i.i.d, we can get the possibility

$$P((h(\boldsymbol{x}_1) = y_1) \wedge (h(\boldsymbol{x}_2) = y_2) \wedge ... \wedge (h(\boldsymbol{x}_m) = y_m)) = (1 - P(h(\boldsymbol{x}) \neq y))^m < (1-\varepsilon)^m$$

hence,

$$P\left(h \in H : \mathcal{E}(h) > \varepsilon \wedge \hat{\mathcal{E}}(h) = 0\right) < |H|(1-\varepsilon)^m < |H|e^{-m\varepsilon}$$

We induce a small value $\delta, 0 < \delta < 1$ to control the possibility above, so we let

$$|H|e^{-m\varepsilon} \leq \delta$$

Hence,

$$m \geq \frac{1}{\varepsilon}\left(\ln|H| + \ln\frac{1}{\delta}\right)$$

In this problem, $|H| = 3^n$ and we tend to set $\varepsilon = 0.1$ in real life. If we want to assure that the version space of 1-NN only contains hypotheses with $\mathcal{E}(h) \leq 0.1$ with a possibility of 95%, so $\delta = 0.05$ in this case. Consequently,

$$m \geq 10\left(3\ln(n) + \ln\left(\frac{1}{0.05}\right)\right) m \geq 10(3\ln(n) + 3\ln(2.7)) m \geq 30\ln(2.7n)$$

Therefore, we can conclude that $m = \Omega(\ln(2.7n))$

Q.E.D

References

[1] Hang Li. Perceptron. In *Statistical learning methods*, pages 25–35. Tsinghua University Press, 2012.

[2] Eric Xing. Introduction to machine learning. In *Introduction to Machine Learning, CMU 10701 PPT*, pages 16–32. 2020.

[3] Zhi-Hua Zhou. Linear models. In *Machine learning*, pages 64–65. Springer, 2021.