Supervised Learning (COMP0078) – Coursework 2

Mark Herbster

Due : 13 December 2021.
`CW2_2122v5`

Submission

You may work in groups of up to two. You should produce a report (this should be in .pdf format) about your results. You will not only be assessed on the **correctness/quality** of your answers but also on **clarity of presentation**. Additionally make sure that your code is *well commented*. Please submit on moodle i) your report as well as a ii) zip file with your source code. Finally, please ensure that if you are working in a group both of your student ids are on the coversheet. Regarding the use of libraries, you should implement regression using matrix algebra directly. Otherwise libraries are okay. **Note:** Both members of the group should submit on moodle and they both must submit the same report. You should produce a report about your results. You will not only be assessed on the **correctness/quality** of your answers but also on **clarity of presentation**. Additionally make sure that your code is *well commented*. Please submit on moodle i) your report as well as a ii) zip file with your source code. Regarding the use of libraries, you should implement regression using matrix algebra directly. Likewise any basic machine learning routines such as cross-validation should be implemented directly. Otherwise libraries are okay.

Questions please use the moodle forum or alternatively e-mail `sl-support@cs.ucl.ac.uk` .

1 PART I [40%]

1.1 Kernel perceptron (Handwritten Digit Classification)

Introduction: In this exercise you will train a classifier to recognize hand written digits. The task is quasi-realistic and you will (perhaps) encounter difficulties in working with a moderately large dataset which you would not encounter on a "toy" problem.

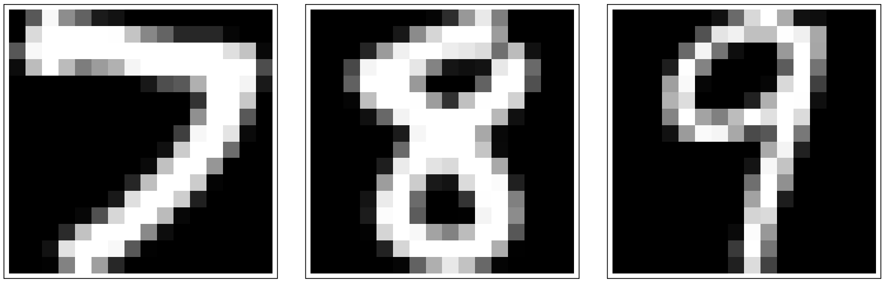

Figure 1: Scanned Digits

You may already be familiar with the perceptron, this exercise generalizes the perceptron in two ways, first we generalize the perceptron to use *kernel* functions so that we may generate a nonlinear separating surface and second, we generalize the perceptron into a majority network of perceptrons so that instead of separating only two classes we may separate k classes.

Adding a kernel: The *kernel* allows one to map the data to a higher dimensional space as we did with basis functions so that class of functions learned is larger than simply linear functions. We will consider a single type of kernel, the polynomial $K_d(\boldsymbol{p}, \boldsymbol{q}) = (\boldsymbol{p} \cdot \boldsymbol{q})^d$ which is parameterized by a positive integer d controlling the dimension of the polynomial.

Training and testing the kernel perceptron: The algorithm is *online* that is the algorithms operate on a single example $(\boldsymbol{x}_t, y_t)$ at a time. As may be observed from the update equation a single kernel function $K(\boldsymbol{x}_t, \cdot)$ is added for each example scaled by the term α_t (may be zero). In online training we repeatedly cycle through the training set; each cycle is known as an *epoch.* When the classifier is no longer changing when we cycle thru the training set, we say that it has converged. It may be the case for some datasets that the classifier never converges or it may be the case that the classifier will *generalize* better if not trained to convergence, for this exercise I leave the choice to you to decide how many epochs to train a particular classifier (alternately you may research and choose a method for converting an online algorithm to a batch algorithm and use that conversion method). The algorithm given in the table correctly describes training for a single pass thru the data (*1st epoch*). The algorithm is still correct for multiple epochs, however, explicit notation is not given. Rather, latter epochs (additional passes thru the data) is represented by repeating the dataset with the $\boldsymbol{x}_i$'s renumbered. I.e., suppose we have a 40 element training set $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_{40}, y_{40})\}$ to model additional epochs simply extend the data by duplication, hence an m epoch dataset is

$$\underbrace{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_{40}, y_{40})}_{\text{epoch 1}}, \underbrace{(\boldsymbol{x}_{41}, y_{41}), \ldots, (\boldsymbol{x}_{80}, y_{80})}_{\text{epoch 2}}, \ldots, \underbrace{(\boldsymbol{x}_{(m-1)\times 40+1}, y_{(m-1)\times 40+1}), \ldots, (\boldsymbol{x}_{(m-1)\times 40+40}, y_{(m-1)\times 40+40})}_{\text{epoch m}}$$

where $\boldsymbol{x}_1 = \boldsymbol{x}_{41} = \boldsymbol{x}_{81} = \ldots = \boldsymbol{x}_{(m-1)\times 40+1}$, etc. Testing is performed as follows, once we have trained a classifier $\boldsymbol{w}$ on the training set, we simply use the trained classifier with only the *prediction* step for each example in test set. It is a mistake when ever the prediction $\hat{y}_t$ does not match the desired output y_t, thus the test error is simply the number of mistakes divided by test set size. Remember in testing the *update* step is never performed.

	Two Class Kernel Perceptron (training)
Input:	$\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\} \in (\Re^n, \{-1, +1\})^m$
Initialization:	$\boldsymbol{w}_1 = \boldsymbol{0}$ $(\alpha_0 = 0)$
Prediction:	Upon receiving the tth instance $\boldsymbol{x}_t$, predict $\hat{y}_t = \text{sign}(\boldsymbol{w}_t(\boldsymbol{x}_t)) = \text{sign}(\sum_{i=0}^{t-1} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}_t))$
Update:	`if` $\hat{y}_t = y_t$ `then` $\alpha_t = 0$ `else` $\alpha_t = y_t$ $\mathbf{w}_{t+1}(\cdot) = \mathbf{w}_t(\cdot) + \alpha_t K(\boldsymbol{x}_t, \cdot)$

Generalizing to k classes: Design a method (or research a method) to generalise your two-class classifier to k classes. The method should return a vector $\boldsymbol{\kappa} \in \Re^k$ where κ_i is the "confidence" in label i; then you should predict either with a label that maximises confidence or alternately with a randomised scheme.

I'm providing you with mathematica code for a 3-classifier and a demonstration on a small subset of the data. First, however, my mathematica implementation is flawed and is relatively inefficient for large datasets. One aspect of your goals are to improve my code so that it can work on larger datasets. The mathematical logic of the algorithm should not change, however either the program logic and/or the data structures will need to change. Also, I suspect that it will be considerable easier to implement sufficiently fast code in Python (or the language of your choice) rather than Mathematica.

Files: From `http://www0.cs.ucl.ac.uk/staff/M.Herbster/SL/misc/`, you will find files relevant to this assignment. These are,

`poorCodeDemoDig.nb`	demo mathematica code
`dtrain123.dat`	mini training set with only digits 1,2,3 (329 records)
`dtest123.dat`	mini testing set with only digits 1,2,3 (456 records)
`zipcombo.dat`	full data set with all digits (9298 records)

each of the data files consists of records (lines) each record (line) contains 257 values, the first value is the digit, the remaining 256 values represent a 16×16 matrix of grey values scaled between -1 and 1.

In attempting to understand the algorithms you may find it valuable to study the mathematica code. However, remember the demo code is partial (it does not address model selection, and though less efficient implementations are possible it is not particularly efficient.) Improving the code may require thought and observation of behaviour on the given data, there are many distinct types of implementations for the kernel perceptron.

Experimental Protocol: Your report on the main results should contain the following (errors reported should be percentages not raw totals):

1. Basic Results: Perform 20 runs for $d = 1, \ldots, 7$ each run should randomly split `zipcombo` into 80% train and 20% test. Report the mean test and train error rates as well as well as standard deviations. Thus your data table, here, will be 2×7 with each "cell" containing a mean±std.

2. Cross-validation: Perform 20 runs : when using the 80% training data split from within to perform 5-fold cross-validation to select the "best" parameter d^* then retrain on full 80% training set using d^* and then record the test errors on the remaining 20%. Thus you will find 20 d^* and 20 test errors. Your final result will consist of a mean test error±std and a mean d^* with std.

3. Confusion matrix: Perform 20 runs : when using the 80% training data split that further to perform 5-fold cross-validation to select the "best" parameter d^* retrain on the full "80%" training set using d^* and then produce a *confusion matrix*. Here the goal is to find "confusions" thus if the true label (on the test set) was "7" and "2" was predicted then a "error" should recorded for "(7,2)"; the final output will be a 10×10 matrix where each cell contains a confusion error *rate* and its standard deviation (here you will have averaged over the 20 runs). Note the diagonal will be 0.

4. Within the dataset relative to your experiments there will be five hardest to predict correctly "pixelated images." Print out the visualisation of these five digits along with their labels. Is it surprising that these are hard to predict?

5. Repeat 1 and 2 (d^* is now c and $\{1, \ldots, 7\}$ is now S) above with a Gaussian kernel

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

   c the width of the kernel is now a parameter which must be optimised during cross-validation however, you will also need to perform some initial experiments to a decide a reasonable set S of values to cross-validate c over.

6. Choose (research) an alternate method to generalise the kernel perceptron to k-classes then repeat 1 and 2.

Assessment: In your report you will not only be assessed on the correctness/quality of your experiment (e.g., sound methods for choosing parameters, reasonable final test errors) but also on the clarity of presentation and the insightfulness of your observations. Thus the aim is that your report is sufficiently detailed so that the reader could largely repeat your experiments based on the description in your report alone. The report should also contain the following.

- A discussion of any parameters of your method which were not cross-validated over.
- A discussion of the two methods chosen for generalising 2-class classifiers to k-class classifiers.
- A discussion comparing results of the Gaussian to the polynomial Kernel.
- A discussion of your implementation of the kernel perceptron. This should at least discuss how the sum $\boldsymbol{w}(\cdot) = \sum_{i=0}^{m} \alpha_i K(\boldsymbol{x}_i, \cdot)$ was i) represented, ii) evaluated and iii) how new terms are added to the sum during training.

Note: (further comments on assessment) : Your score in Part I is not the "percentage correct," rather it will be a qualitative judgement of the report's *scientific excellence.* Thus a report that is merely correct/good as a baseline can expect 24-32 points out of 40. An excellent report will receive 32-40 points. Regarding page limits the expectation is that an excellent report will be approximately no more of three pages of text (this does not include tables, extra blank space on page, repetition of text from the assignment). There is no strict limit, however, as some writers are more or less concise than others (thus one page may be sufficient) and there are a range of formatting possibilities.

2 PART II [20%]

2.1 Spectral Clustering

Introduction: In this exercise you will apply *spectral clustering* to a variety of datasets. Note you are not expected to be already familiar with spectral clustering beyond the semi-supervised learning lecture. It is expected that you will be able to use the *web* and other resources to research that which you need to know to complete the assignment. A very comprehensive tutorial on spectral clustering can be found at (U. von Luxburg. A Tutorial on Spectral Clustering): `https://arxiv.org/pdf/0711.0189.pdf`

Clustering is a term which encompasses a variety of heterogeneous methodologies whose aims are to divide data into a series of natural groupings (*clusters*). A key feature of spectral clustering is that it can separate very irregularly shaped groupings of data into clusters as opposed to the more classical *k-means* clustering which has difficulty with "irregular" data clusters. Spectral clustering works by representing the data as a (weighted) graph. Two data points are close in the graph if the weight between them is large. The goal then is two separate/partition the graph into two "clusters" such that total weight of edges between the "clusters" is "small" and each cluster has roughly the same number of data points.

In the following exercises we will limit ourselves to clustering into two groups/clusters: cluster "-1" and cluster "$+1$". The spectral clustering algorithm can be described as follows. We are given a data set which is represented by a data matrix $\mathbf{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_\ell)$ where each datum $\boldsymbol{x}_i \in \Re^n$ is a vector. First, produce an $\ell \times \ell$ *weight matrix*,

$$W = (e^{-c\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2})_{i,j=1}^{\ell} \ . \tag{1}$$

To understand this weight matrix, we intuitively assume if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are adjacent (that is $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ is small), then as a "rule of thumb" they should be in the same cluster. Therefore, roughly $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are in the same cluster if the weight W_{ij} is big. Secondly, we construct the graph Laplacian $L := D - W$ where D is a diagonal matrix with $D_{ii} = \sum_{j=1}^{\ell} W_{ij}$, with the off-diagonal elements equal to zero. We denote the eigensystem of L as

$$\{(\mathbf{v}_i, \lambda_i)\}_{i=1}^{\ell} = \{(\mathbf{v}_1, \lambda_1), (\mathbf{v}_2, \lambda_2), \ldots, (\mathbf{v}_\ell, \lambda_\ell)\} \qquad (\lambda_1 \le \lambda_2 \le \ldots \le \lambda_\ell) \ .$$

We will use the eigenvector $\mathbf{v}_2$ (the eigenvector with second smallest eigenvalue) of the graph Laplacian L to perform the clustering as follows. Define the function

$$\operatorname{sign}(x) = \begin{cases} +1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$$

and denote the ith component of a vector $\mathbf{v}$ as $\mathbf{v}(i)$. Now assign $\boldsymbol{x}_i$ to cluster "$+1$" if $\operatorname{sign}(\mathbf{v}_2(i)) \in \{0, 1\}$ otherwise assign $\boldsymbol{x}_i$ to cluster "-1" and do this for $i = 1, 2, \ldots, \ell$ and you will have clustered the data (also see Figure 2).

Inputs:	Data: $\mathbf{X} := (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_\ell)$ Weight matrix parameter: $c > 0$
Adjacency weight matrix W**:**	$W = (e^{-c\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2})_{i,j=1}^{\ell}$
Graph Laplacian L**:**	$L = D - W$,
Cluster vector:	Let $\mathbf{v}_2$ denote the eigenvector with second smallest eigenvalue.
Clustering:	$\hat{y}(i) = \text{cluster}(\boldsymbol{x}_i) = \begin{cases} \text{"+1"} & \operatorname{sign}(\mathbf{v}_2(i)) = +1 \\ \text{"−1"} & \operatorname{sign}(\mathbf{v}_2(i)) = -1 \end{cases}$

Figure 2: Spectral clustering (two clusters)

Files: From `http://www0.cs.ucl.ac.uk/staff/M.Herbster/SL/misc/`, you will find files relevant to this assignment. These are,

`dtrain123.dat`	digits 1,2,3 (329 examples)
`twomoons.dat`	two moons (200 examples)

Experiments[12pts]: In these experiments we will "cheat" in so far as we already will know what constitutes a good clustering. In a more realistic situation we would not already know what constitutes a good clustering.

1. Implement the spectral clustering on `twomoons`. Plot both the original data and the now correctly (or almost correcly) clustered data which should be similar to Figure 3. What is a value of c that will correctly cluster the data? As a range to search over try

$$c \in \{2^i : i \in \{-10.0, -9.9, -9.8, \ldots, 0, 0.1, \ldots, 9.8, 9.9, 10.0\}\}.$$

2. Implement spectral clustering on the following random data set:

   (a) For class "-1" generate 20 data 2-d data points for the isotropic Gaussian centered at $(-0.3, -0.3)$ with a standard deviation of $\sigma = 0.2$, i.e., the covariance matrix is $0.04I$ where I is the 2×2 identity matrix.

   (b) For class "+1" generate 20 data 2-d data points for the isotropic Gaussian centered at $(0.15, 0.15)$ with a standard deviation of $\sigma = 0.1$, i.e., the covariance matrix is $0.01I$ where I is the 2×2 identity matrix.

   Plot both the original data and the clustered data in different figures. For this random data set, spectral clustering typically yields very good results. The figure should be similar to Figure 4 which is well-separated. What is a value of c that will correctly (or almost correctly) cluster the data?

3. Now, test the spectral clustering on real digit data. Choose the corresponding input data of digits "1" and "3" from data set `dtrain123`. We know the true labels of the data set `dtrain123`, denote them as $\{y_i : i = \{1, 2, \ldots, \ell\}\}$ where either $y_i = 1$ or $y_i = 3$.

   What is a value of c that will perform well? The choice of the parameter c in spectral clustering in general is a challenging problem. However, as we know the true labels of our data, we can introduce a method to measure the quality of clustering as a function of the free parameter c, comparing the true labels to the results of the spectral clustering algorithm. To measure the quality of the clustering we can use the following method:

   (a) Create a variable $classy_i = 2 * (y_i == 1) - 1$ which maps values $1, 3$ to $1, -1$, respectively.

   (b) Denote ℓ_+ as the number of i such that $classy_i = \hat{y}_i$ and ℓ_- the number of those such that $classy_i \neq \hat{y}_i$

   (c) Then, we can use the correct cluster percentage to measure the quality of spectral clustering with regard to c:
$$\mathrm{CP}(c) := \frac{\max\{\ell_-, \ell_+\}}{\ell}\ .$$

   Now your task is to implement the method and to produce a plot (correctness vs $c \in \{0.00, 0.01, 0.02, \ldots, 0.10\}$) similar to Figure 5. What c works best?

Questions[8 pts]:

1. Explain why $\mathrm{CP}(c)$ is a reasonable measure of cluster correctness.

2. Explain why the first eigenvalue of the Laplacian is zero and the corresponding eigenvector is the constant vector.

3. In your own words (please explicitly cite any reference), provide a tentative explanation why spectral clustering "works" (1 paragraph only).

4. Recall the parameter c in the definition (see equation (1)) of the weight matrix. Explain why and how c influences the quality of the clustering.

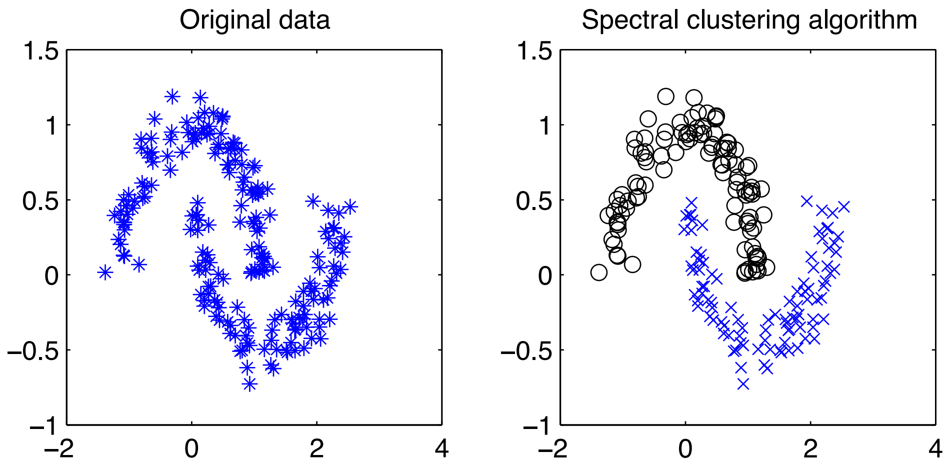

Figure 3: Two moons dataset

Assessment: The part of your report for Part II should be divided into two sections covering **experiments, questions**. Thus it will include

1. A short paragraph overviewing your implementation of spectral clustering.
2. Answers to any question in the experimental subsection.
3. All plots requested
4. Give any references used to answer the questions.
5. Answers to the four questions in **questions**.

Note: Under the same guidance about reports in Part I, the reporting for Part II is normally expected to be less than $1\frac{1}{2}$ pages.

3 PART III [40%]

3.1 Questions

1. [**(a,b,c,d 24pts, e 16pts)**] *Sparse learning:*

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

   Problem ('just a little bit'): The m patterns $\boldsymbol{x}_1, \ldots \boldsymbol{x}_m$ are sampled *uniformly* at random from $\{-1,1\}^n$, and each label is defined as $y_i := x_{i,1}$, i.e., the label of a pattern $\boldsymbol{x}$ is just its first coordinate.

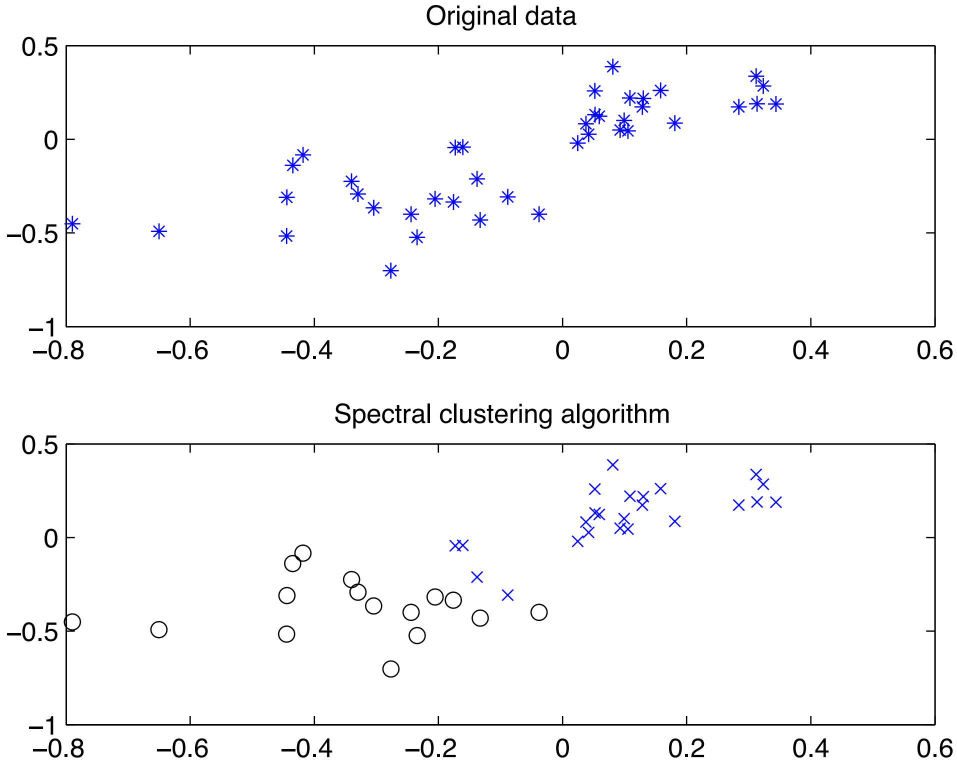

Figure 4: Gaussian clusters

Thus for example here is a typical data set with $m = 4$ examples in $n = 3$ dimensions,

$$X = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & 1 \end{pmatrix} \quad Y = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix}$$

We are concerned with estimating the sample complexity as a function of the dimension (n) of the data of this problem. Where our “working definition” of sample complexity is the minimum number of examples (m) to incur no more than 10% generalisation error (on average).

(a) In this part, you will implement the four classification algorithms and then use them to estimate the sample complexity of these algorithms. Here you will plot m (left axis) versus n (bottom axis). As an illustration I include an example plot[1] of estimated sample complexity for least squares. Please include sample complexity plots for all four algorithms.

(b) Computing the sample complexity “exactly” by simulation would be extremely expensive computationally. Thus for your method in part (a) it is necessary to trade-off accuracy and computation time. Hence, i) Please describe your method for estimating sample complexity in detail. ii). please discuss the tradeoffs and biases of your method.

(c) Please estimate how m grows as a function of n as $n \to \infty$ for each of the four algorithms based on experimental or any other “analytical” observations. Here the use of $O(\cdot), \Omega(\cdot), \Theta(\cdot)$ will be useful. For example experimentally from the plot given for least squares it seems that

[1] In the figure I have included “error bars” which indicate the standard deviation for the estimates of m, it is not necessary for you to include them.

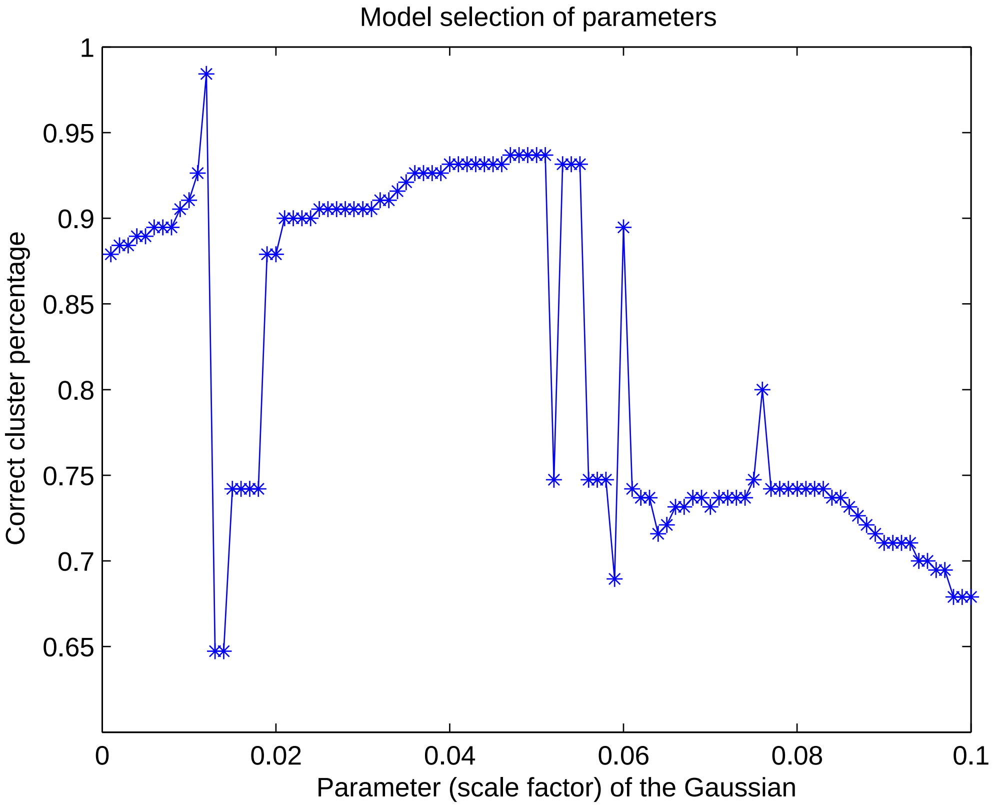

Figure 5: Cluster correctness as a function of c.

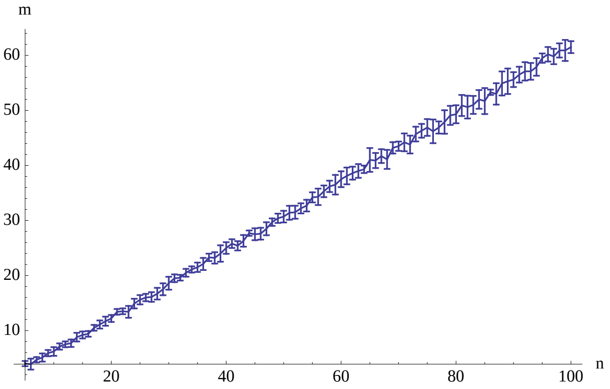

Figure 6: Estimated number of samples (m) to obtain 10% generalisation error versus dimension (n) for least squares.

sample complexity grows linearly as a function of dimension, i.e., it is $m = \Theta(n)$. Discuss your observations and compare the performance of the four algorithms.

(d) Now additionally, suppose we sample an integer $s \in \{1, \ldots, m\}$ uniformly at random. Derive a non-trivial upper bound $\hat{p}_{m,n}$ on the probability that the perceptron will make a mistake on the sth example after being trained on examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_{s-1}, y_{s-1})$. The derived upper bound $\hat{p}_{m,n}$ should be function of only m and n (i.e., it is independent of s) and should be with respect to the dataset decribed above. Justify your derivation, analytically.

(e) **[Challenge]** Find a *simple* function $f(n)$ which is a *good* lower bound of the sample complexity of `1-nearest neighbor` algorithm for the 'just a little bit' problem. Prove $m = \Omega(f(n))$. *There is no partial credit for this problem.*

Notes

1. `Winnow:` Use $\{0,1\}^n$ for the patterns and $\{0,1\}$ for the labels. This follows our presentation in the notes.

2. `Linear Regression:`

   (a) We use regression vector $\boldsymbol{w}$ to define a classifier $f_{\boldsymbol{w}}(\boldsymbol{x}) := \text{sign}(\boldsymbol{w}^\top \boldsymbol{x})$.

   (b) For this problem we are usually in the *underdetermined* case. We use the convention that the linear regression solution is a limiting case of the ridge regression solution. A technical presentation of this is given `http://www.cs.ucl.ac.uk/staff/M.Pontil/courses/2-gi07.pdf`, equation (6) and details on p25-p26]. The practical "take away," is that in matlab we may use $w = pinv(X) * y$ to compute the "w" of minimal norm that is consistent with the data. This is contrary to the usual advice to use the `left division` operator in matlab for regression. This is so that we have consistent definition of linear regression across programming libraries/languages. Finally, note computing pseudoinverse is still inefficient as a method to solve for the minimal norm solution in the underdetermined case, however for this exercise the efficiency of the implementation is not the focus.

3. **Sample Complexity:** For this problem, let $\mathcal{S}_m$, denote a set of m patterns drawn uniformly at random from $\{-1,1\}^n$ with their derived labels. Then let $\mathcal{A}_{\mathcal{S}}(\boldsymbol{x})$ denote the prediction of an algorithm $\mathcal{A}$ trained from data sequence $\mathcal{S}$ on pattern $\boldsymbol{x}$. Thus the generalisation error is then

$$\mathcal{E}(\mathcal{A}_{\mathcal{S}}) := 2^{-n} \sum_{\boldsymbol{x} \in \{-1,1\}^n} I[\mathcal{A}_{\mathcal{S}}(\boldsymbol{x}) \neq x_1]$$

and thus the sample complexity on average at 10% generalisation error is

$$\mathcal{C}(\mathcal{A}) := \min\{m \in \{1,2,\ldots\} : \mathbb{E}[\mathcal{E}(\mathcal{A}_{\mathcal{S}_m})] \leq 0.1\}.$$

With sufficient computational resources we may compute this exactly via,

$$\mathcal{C}(\mathcal{A}) = \min\{m \in \{1,2,\ldots\} : (2^{-nm} \sum_{S \subseteq \{-1,1\}^{nm}} \mathcal{E}(\mathcal{A}_{\mathcal{S}})) \leq 0.1\}.$$

Note: Under the same guidance about reports in Part I, the reporting for Part III (parts b,c) is normally expected to be less than one page.