Supervised Learning (COMP0078) – Coursework 2

Mark Herbster

Due : 14 December 2022.
`CW2-2223-v7`

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. Likewise any machine learning routines such as for example cross-validation should be implemented directly. Otherwise libraries are okay. Finally for extra emphasis some algorithms in this assignment will already have implementations on the internet, **you must not** "translate" such implementations to produce your code.

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.

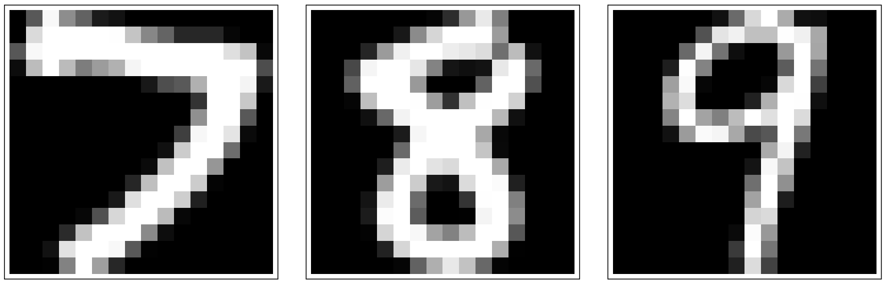

Figure 1: Scanned Digits

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 $d = 1, \dots, 7$ 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. In computing the error rate for a cell use

$$\frac{\text{“Number of times digit } a \text{ was mistaken for digit } b \text{ (test set)”}}{\text{“Number of digit } a \text{ points (test set)”}}.$$

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 $\{1, \dots, 7\}$ 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.
- Any table produced in 1-6 above should also have at least one sentence discussing the table.

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%]

Notation

Recall that $[m] := \{1, 2, \ldots, m\}$ and we also overload notation so that

$$[\texttt{pred}] := \begin{cases} 1 & \texttt{pred} \text{ is true} \\ 0 & \texttt{pred} \text{ is false} \end{cases} .$$

The vector *ith* "coordinate" vector is denoted $\mathbf{e}_i := ([j = i] : j \in [m])$. The notation M^+ denotes the pseudoinverse of M.

2.1 Semi-supervised Learning via Laplacian Interpolation

In this section, you are asked to implement two intimately connected semi-supervised learning methods over a graph. The two methods will be *Laplacian (semi-norm) interpolation (LI)* and *Laplacian kernel interpolation (LKI)*

Like the previous problem, on top of the lecture slides on semi-supervised learning, you may to use the web and other resources to complete this problem (but the code that you write must be your own). For Laplacian interpolation as well as the lecture slides you may find the paper (Zhu *et al.* Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions) valuable, which can be found at `https://www.aaai.org/Papers/ICML/2003/ICML03-118.pdf` . For Laplacian Kernel Interpolation, this is just a limiting case of Kernel ridge regression as the regularisation parameter goes to zero and we use the pseudo-inverse of the Laplacian as a kernel.

Semi-supervised learning is somewhere between unsupervised learning and supervised learning. In the context of the label prediction problems, the semi-supervised learning task is similar to supervised learning; to predict the labels of unlabeled data by using the label information. Supervised learning constructs a model using only labeled data points and then applies the model to predict on unlabelled data. On the other hand, in semi-supervised learning, a model is constructed using both labeled and unlabelled data to make predictions.

In the following, we focus on the Laplacian interpolation method, one of the established graph-based semi-supervised learning methods. The Laplacian interpolation method can be described as follows. We are given a data set which is represented by a data matrix $\mathbf{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m)$ where each datum $\boldsymbol{x}_i \in \Re^n$ is a vector. Let $\mathbf{y} \in \{-1, 1\}^m$ be a label vector, where $y_i \in \{-1, 1\}$ is the label of $\boldsymbol{x}_i$. We assume that we know some of the values of y_i, and the task is to predict the rest. More formally, we are given $\{(i, y_i) : i \in \mathcal{L}\}$, where $\mathcal{L} \subset [m]$ is a set of indices of the known labels, and the task is to predict $\{(i, y_i) : i \in [m] \backslash \mathcal{L}\}$. In general, we only know the very few labels, i.e., $|\mathcal{L}| \ll m$. We will build a graph using the 3-NN method (with the Euclidean distance). For the 3-NN graph, we create a weight "1" edge if i-th datum is among the 3-nearest neighbors of j-th datum or if j-th datum is among the 3-nearest neighbors of i-th datum; otherwise, we put weight 0. Thus, we produce an $m \times m$ weight matrix as

$$W_{ij} := \begin{cases} \text{"+1"} & \boldsymbol{x}_i \text{ is 3-NN of } \boldsymbol{x}_j \text{ or } \boldsymbol{x}_j \text{ is 3-NN of } \boldsymbol{x}_i \\ \text{"0"} & \text{otherwise.} \end{cases}$$

Note that $W_{ii} = 0$. Then, we solve the following optimization,

$$\mathbf{v} := \operatorname*{argmin}_{\mathbf{u} \in \Re^n} \mathbf{u}^\top L \mathbf{u} \, : \, u_i = y_i, \forall i \in \mathcal{L} \, . \tag{1}$$

```
For each dataset ∈ {dtrain13_50.dat, dtrain13_100.dat, dtrain13_200.dat, dtrain13_400.dat}
    For each $\ell \in \{1,2,4,8,16\}$ do
        Do 20 runs ...
            $\mathcal{L}$ = random sample (w/o replacement) of size $\ell$ from each class ($|\mathcal{L}| = 2\ell$)
            errLI = LaplacianInterpolation(dataset, $\mathcal{L}$)
            errLKI = LaplacianKernelInterpolation(dataset, $\mathcal{L}$)
            The estimated generalisation error is
                the mean empirical error of these 20 runs
```

Figure 3: Experimental Protocol A

Where L is Laplacian matrix created from the matrix W. We will then predict by assigning $\hat{y}_i = \text{sign}(u_i)$ for $i \in [m] \backslash \mathcal{L}$ (as summarised Figure 2). For Laplacian kernel interpolation the set-up is the same but we will exploit the induced Laplacian differently than in Equation (1) instead see Figure 2 for details.

Inputs:	Data: $\mathbf{X} := (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m), \boldsymbol{y} := (y_1, \ldots, y_m)$ Observed label set : $\mathcal{L} \subset [m]$
Adjacency weight matrix W:	$W_{ij} := \begin{cases} \text{“+1”} & \boldsymbol{x}_i \text{ is 3-NN of } \boldsymbol{x}_j \text{ or } \boldsymbol{x}_j \text{ is 3-NN of } \boldsymbol{x}_i \\ \text{“0”} & \text{otherwise} \end{cases}$ Note that $W_{ii} = 0.$;
Degree Matrix D :	$D_{ii} := \sum_{j \neq i} W_{ij}$ and if $i \neq j$ then $D_{ij} := 0$.
Graph Laplacian L:	$L = D - W$,
Laplacian (semi-norm) Interpolation (LI) :	$\mathbf{v} := \text{argmin}_{\mathbf{u}} \mathbf{u}^\top L \mathbf{u} \, : \, u_i = y_i, \forall i \in \mathcal{L}$
Laplacian Kernel Interpolation (LKI):	Kernel Matrix $\boldsymbol{K} := (L^+_{ij} : i, j \in \mathcal{L})$ $\boldsymbol{y}_{\mathcal{L}} := (y_i : i \in \mathcal{L})$ $\boldsymbol{\alpha}^* := \boldsymbol{K}^+ \boldsymbol{y}_{\mathcal{L}}$ $\mathbf{v} := \sum_{i \in \mathcal{L}} \boldsymbol{\alpha}^*_i \mathbf{e}_i^\top L^+$
LKI implementation note:	Note: $\boldsymbol{K}, \boldsymbol{y}_{\mathcal{L}}, \boldsymbol{\alpha}^*$ are indexed by the elements of $\mathcal{L}$ and hence the indices are not likely to be consecutive.
Discrete prediction vector:	$\hat{\mathbf{y}} := (\text{sign}(v_i) : i \in [m])$
Empirical generalisation error:	$\text{err} = \sum_{i \in [m] \backslash \mathcal{L}} \frac{[\hat{y}_i \neq y_i]}{\lvert [m] \backslash \mathcal{L} \rvert}$

Figure 2: Laplacian Interpolation and Laplacian Kernel Interpolation

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

`dtrain13_50.dat`	subset of USPS digits 1,3 (50 examples per class)
`dtrain13_100.dat`	subset of USPS digits 1,3 (100 examples per class)
`dtrain13_200.dat`	subset of USPS digits 1,3 (200 examples per class)
`dtrain13_400.dat`	subset of USPS digits 1,3 (400 examples per class)

Experiments [10pts]:

In this experiment, you will implement Laplacian interpolation and Laplacian kernel interpolation (see Figure 2). This experiment uses four datasets in the **Files**; these are the subsets of the USPS dataset which were created by randomly sampling 50, 100, 200, 400 examples per class ($|m| \in \{100, 200, 400, 800\}$. For each dataset, we will experiment with label sets $\mathcal{L}$ that contain 1,2,4,8 or 16 labels per class. The set

$\mathcal{L}$ is formed by sampled uniformly at random without replacement from $[m]$ per class. so that we given $\{1,2,4,8,16\}$ labels per class thus as a consequence $|\mathcal{L}| \in \{2,4,8,16,32\}$. The task is to predict the labels of $\{1,\ldots,m\}\backslash\mathcal{L}$. You will repeat each experiment 20 times per dataset paired with label set size. The protocol for the experiments is detailed in Figure 3.

You will include your results of your experiments in the report requested below rather than separately.

		# of known labels (per class)				
		1	2	4	8	16
	50	??? ± ???	??? ± ???	??? ± ???	??? ± ???	??? ± ???
	100	??? ± ???	??? ± ???	??? ± ???	??? ± ???	??? ± ???
# data points per label	200	??? ± ???	??? ± ???	??? ± ???	??? ± ???	??? ± ???
	400	??? ± ???	??? ± ???	??? ± ???	??? ± ???	??? ± ???

Table 1: Errors and standard deviations for semi-supervised learning via Laplacian (kernel) interpolation.

Experimental Report [10pts]:. You should write a brief experimental report (1 to 2 pages expected length). The report should include the following :

1. You should give two tables with your results each in the form of Table 1 where one table is for Laplacian interpolation and the other for Laplacian kernel interpolation
2. You should give observations about each table that you believe of are of interest.
3. You should compare the performances of the two methods. If in your opinion one method significantly performed better than the other than the other in some aspects you are then encouraged propose reasons why this occurred. *Note :* It is better to say that one cannot explain than to give unreasonable explanations.
4. If there is a significant difference in the performance of the two algorithms and you believe those reasons depend in someway on the datasets, you are encouraged to speculate (with reasons) about datasets where the weaker method might outperform the other method.

Note: As in Part I, your report will be assessed on both its clarity and scientific excellence.

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. 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.

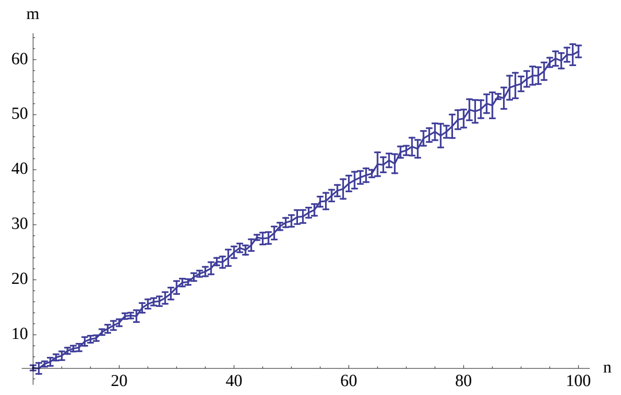

Figure 4: Estimated number of samples (m) to obtain 10% generalisation error versus dimension (n) for least squares.

(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 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

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

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.