Supervised Learning Lab/Coursework 1

17 October 2016
Due : 16 January, 2017

Abstract

[LAB]: Using a series of synthetic examples, in this exercise session you will acquaint yourselves with linear regression, regularization, and nonlinear regression using kernel methods.

[PROBLEMS]: Problems involving kernels, K-nn and online learning.

Submission

You may work in groups of up to two[1] if desired, only one report need be given for your group. You should produce a report about your results. You will not only be assessed on the correctness/quality of your experiments but also on clarity of presentation. Additionally make sure that your code is *well commented*. Please submit 1) the coursework to the department office on the due date at 12:00 noon with the following linked coversheet `http://www.cs.ucl.ac.uk/fileadmin/UCL-CS/students/documents/cwsheet.pdf` 2) on moodle please submit a zip file containing your coursework as well as any your source code.

PART I [50%]

Part I (assessment) : exercises 1-7 (40%), exercise 9 (30%), exercise 10 (30%)

1 Concepts

Basic idea of regression: we want to *fit* a function $g(\mathbf{x})$ through a set of samples $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_l, y_l)\}$, where (usually) $\mathbf{x}_i \in \Re^n$ and $y \in \Re$. Thus, we look for a function g such that $y_i \simeq g(\mathbf{x}_i)$ for all i.

There are two possible reasons why one may want to do regression:

- *Modeling* of the functional relation between points $\mathbf{x}_i$ and y_i by the function g.
- *Prediction* of the labels $y_{\text{test},i}$ corresponding to test points $\mathbf{x}_{\text{test},i}$ (drawn from the same population as the training set) as $g(\mathbf{x}_{\text{test},i})$.

The latter is the most common, however both goals are related.

The points $\mathbf{x}_i$ along with their given label y_i make up the *training set*; they are used during training to do the 'interpolation' or 'function estimation'. We use the term *test set* to denote the set of test points $\mathbf{x}_{\text{test},i}$ on which we afterwards make predictions. In practice, the true labels $y_{\text{test},i}$ of the test points are unknown. However, since we are interested in studying methods for regression and not yet in applying them to real problems, we will generate our data artificially, such that we do know the actual test set labels to compare with the predictions made by the regression algorithms.

[1] Affiliates students should work alone.

2 Linear regression

In linear regression, the functional relation we are looking for is of the form:

$$y_i \simeq g(\mathbf{x}_i) = \mathbf{x}_i'\mathbf{w} \tag{1}$$

where $\mathbf{w}$ is called the *weight vector*.

2.1 Least Squares Regression (LSR)

In LSR, in order to try to achieve that $g(\mathbf{x}_i) = \mathbf{x}_i'\mathbf{w} \simeq y_i$, we look for the weight vector that minimizes the mean of the squared errors on all training samples:

$$\mathbf{w}^* = \text{argmin}_{\mathbf{w}} \frac{1}{l}\sum_{i=1}^{l}(\mathbf{x}_i'\mathbf{w} - y_i)^2. \tag{2}$$

Using the notation $\mathbf{X} = (\mathbf{x}_1\mathbf{x}_2\cdots\mathbf{x}_l)'$, a matrix containing the training sample vectors as its rows, we can rewrite the cost function as:

$$\frac{1}{l}\sum_{i=1}^{l}(\mathbf{x}_i'\mathbf{w} - y_i)^2 = \frac{1}{l}(\mathbf{X}\mathbf{w} - \mathbf{y})'(\mathbf{X}\mathbf{w} - \mathbf{y}) = \frac{1}{l}\left(\mathbf{w}'\mathbf{X}'\mathbf{X}\mathbf{w} - 2\mathbf{y}'\mathbf{X}\mathbf{w} + \mathbf{y}'\mathbf{y}\right). \tag{3}$$

Taking the derivative with respect to $\mathbf{w}$ and setting this to $\mathbf{0}$ give the conditions for optimality:

$$\mathbf{X}'\mathbf{X}\mathbf{w}^* = \mathbf{X}'\mathbf{y}, \tag{4}$$

such that, for non-singular $\mathbf{X}'\mathbf{X}$:

$$\mathbf{w}^* = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}. \tag{5}$$

Note that in the special case where $\mathbf{x}_i \in \Re$ is one-dimensional, $\mathbf{X}$ is a column vector, and $\mathbf{X}'\mathbf{X}$ is a scalar then $\mathbf{w}$ will also be a scalar.

Exercise 1 (LSR — effect of the training set size.) *In this exercise we experimentally investigate the effect of the training set size. Make plots wherever this helps your understanding.*

a. *Pick a random value for $\mathbf{w} \in \Re$ from the standard normal distribution, and generate a noisy random data set, containing 600 samples, as $y_i = \mathbf{x}_i'\mathbf{w} + n_i$ where each $\mathbf{x}_i$ and n_i are drawn from the standard normal distribution. (To generate random $\mathbf{w}$, $\mathbf{x}_i$, and n_i, use the matlab function* `randn`*.) Split the data set into a training set of size 100 a test set of size 500.*

b. *Using equation (5), estimate $\mathbf{w}$ based on the training set. (Note that in Matlab you should not in general use the* `inv` *function (or A^{-1}) as the results can be numerically unstable and computationally inefficient. Instead use the left division operator:* `help mldivide`*.) Using equation (3) compute the mean squared error on both the training and test sets.*

c. *In order to investigate the influence of the training set size, repeat part b while using only a 10-sample training set and a 500-sample test set. Compute the mean squared error on the training and test set.*

d. *Repeat parts a, b and c, 200 times, each time selecting a different random weight vector and a different random data set. Average the mean squared errors for both the training and test sets, on both the 100 and 10 element-size training sets. Make a "2×2" table of these averages. What is the effect of training set size on the training and test set errors? What is larger, the training or the test set error? Try to explain.*

Exercise 2 (LSR — effect of the dimensionality.) *Whereas the previous exercise investigated the influence of the training set size on training set and test set error, here we investigate the influence of the dimensionality. We now use a data set with $\mathbf{x} \in \Re^{10}$.*

a. *Pick a random weight vector* $\mathbf{w} \in \Re^{10}$*, and generate a noisy random data set containing 600 samples as* $y_i = \mathbf{x}_i'\mathbf{w} + n_i$*, where each* n_i *and each component of the* $\mathbf{x}_i$ *and* $\mathbf{w}$ *are drawn from the standard normal distribution. Split the data set into a training set of size 100 a test set of size 500.*

b. *Repeat the tasks in b, c, d from the previous exercise, but with these (10-dimensional) data sets.*

c. *Provide an interpretation of your new table in light of the fact that the data is now 10-dimensional.*

2.2 Ridge Regression (RR)

As you should have noticed, for a small sample size as compared to the dimensionality, the test set performance may be poor even when the training set error is small. The reason for this is that the regression function will fit the noise too much, while the interesting part of the signal is too small. This phenomenon is called *overfitting.* In order to prevent this, a commonly used practice that can be theoretically motivated is called regularization: the freedom of the classifier is restricted, such that it is less susceptible to the influence of noise and generalization is better. The regularized version of LSR is Ridge Regression (RR).

2.3 Regularization

The way regularization is introduced here is by adding a so-called *complexity term* to the cost function to be minimized. This complexity term biases the solution to weight vectors that have a small norm, but on the other hand it reduces the noise sensitivity. As such, the optimal value $\mathbf{w}^*$ will trade off the mean squared error on the training set (also called the *empirical error*) with the complexity:

$$\mathbf{w}^* = \text{argmin}_{\mathbf{w}} \gamma \mathbf{w}'\mathbf{w} + \frac{1}{l}\sum_{i=1}^{l}(\mathbf{x}_i'\mathbf{w} - y_i)^2. \tag{6}$$

The weight of the empirical error as opposed to the complexity in this cost function can be adjusted with the *regularization parameter* γ.

Exercise 3 (Ridge Regression.)

1. *Prove that solving the optimization problem (6) yields:*

$$\begin{aligned} \mathbf{X}'\mathbf{X}\mathbf{w}^* + \gamma l \mathbf{w}^* &= \mathbf{X}'\mathbf{y} \\ \Rightarrow \mathbf{w}^* &= (\mathbf{X}'\mathbf{X} + \gamma l \mathbf{I})^{-1}\mathbf{X}'\mathbf{y}. \end{aligned} \tag{7}$$

2. *Prove that* $\mathbf{X}'\mathbf{X} + \gamma l \mathbf{I}$ *is a positive definite matrix.*

Exercise 4 (Effect of the regularisation parameter.) *The effect of regularisation should be clearly illustrated by:*

a. *Perform ridge regression on the data sets as generated in exercise 2 part a for* γ *ranging from* 10^{-6} *up to* 10^3*, and all powers of* 10 *in between. What are the training and test mean squared errors? Plot the training and test set errors as a function of* γ *(use a log scale for the* γ *axis).*

b. *Repeat the experiment but with 10 training samples.*

c. *Now perform the ridge regressions of parts a and b, 200 times and then create the plots of the two parts but with respect to the averaged "errors". Do your results suggest a method to set the regularisation parameter to minimise test set error? Write up your observations.*

2.4 Tuning the regularization parameter: using a validation set

In the exercise 4, we tried a series of values for γ, and observed the resulting training and test set performance. In this way, we discovered that the training set error is not a sufficiently good guidance to select the regularization parameter... We need a better method to make an estimate of the expected test set error for the different values of γ.

One way to proceed is by using a part of the training set for *validation*. I.e., we divide the training set in a (smaller) training set and a *separate* validation set. Then, for different regularization parameter values, we can compute $\mathbf{w}^*$, and evaluate the mean squared error on the validation set. Then, the regularization parameter leading to the smallest error on the validation set will be used to do RR on the original training set (including the validation set).

Exercise 5 (Tuning the regularization parameter using a validation set) *Use a validation set to tune the regularization parameter in the following two cases:*

Note: For a and b; do not plot the result of a single "run" but instead using 200 averaged runs.

a. *For the data set of exercise 2 part a, make a split of the training set in two parts, 80% for the new smaller training set and the remaining 20% for the validation set. Plot the validation error as a function of the regularization parameter, together with the training and test set error. Perform RR on the full training set (with 100 samples)* selecting *the regularization parameter that gives rise to the smallest validation set error.*

b. *Do the same with only 8 and 2 training and validation samples respectively.*

c. *In the experiments above each of the 200 runs will have a different* selected γ, *for each experiment compute an averaged $\bar{\gamma}$ by averaging the 200* selected *"γ"s, call them $\bar{\gamma}^{(100)}$ and $\bar{\gamma}^{(10)}$, respectively. Compare $\bar{\gamma}^{(100)}$ and $\bar{\gamma}^{(10)}$ In which case is it larger and why?*

d. *Think about what would happen with the one-dimensional datasets from exercise 1. If you don't know, try it. Write up your observations or predictions.*

2.5 Cross-validation

Of course, it's a pity that part of the training set has to be sacrificed. Additionally, a validation set of size 2 is actually very small, such that the result will be very noise sensitive.

An alternative approach that is used very often is called *k-fold cross-validation.* It divides the training set in k disjoint sets. Each of these k sets of samples is once lifted out as the validation set, and the remaining $k-1$ sets are used for training. As a result, we get k validation scores. The average of these scores is used as a good estimate of the test set performance.

Exercise 6 (Tuning the regularization parameter using cross-validation) *Use* 5*-fold cross validation to tune the regularization parameter.*

a. *For the data set of exercise 2 part a. Plot the cross-validation score on top of the training and test set errors for different values $\gamma = \{10^{-6}, 10^{-5}, \ldots, 10^{3}\}$ of the regularization parameter.*

b. *Do the same with the training set of size 10.*

Exercise 7 (Comparison of γ tuning methods) *Generate 200 such data sets as above, and tune the regularization parameter γ using each of these three methods:*

- *by minimizing the training error (exercise 4),*
- *by minimizing the validation error (based on a 80%/20% split) (exercise 5,*
- *by minimizing the* 5*-fold cross validation error (exercise 6).*

For each of these approaches, compute the average test set error and the standard deviation of the errors on both. Report on both the 100 sample set and the 10 sample set-up.

3 Boston housing and kernels

In this section we will use kernel methods to extend linear regression. Boston housing is a classic dataset where you are given 13 values and a goal is to predict the 14th which is the median house price. It is described in more detail at `http://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html`. There are 506 entries which we will split into train and test.

The boston housing data set as ".mat" file is located at

`http://www.cs.ucl.ac.uk/staff/M.Herbster/boston`

Exercise 8 (Unassesed: *Load the data*) *Load the file* '`boston.mat`' *into Matlab. We will be using the first 13 attributes of the data as our independent variables (regressors) and the 14th as the dependent variable.*

Exercise 9 (Baseline versus full linear regression.) *Rather than use all of our attributes for prediction it is often useful to see how well a baseline method works for a problem. In this exercise, we will compare the following:*

1. *Predicting with the mean y-value on the training set.*
2. *Predicting with a single attribute and a bias term.*
3. *Predicting with all the attributes*

The training set should be 2/3, and the test set should be 1/3, of your data in (a)-(c). In the following average your results over 20 runs (each run based on a different (2/3,1/3) random split).

a. *Naive Regression. Create a vector of ones that is the same length as the training set using the function* `ones`*. Do the same for the test set. By using these vectors we will be learning the simple equation* $\hat{y} = b$*. Perform linear regression on the training set. Calculate the MSE on the training and test sets and note down the results.*

b. *Linear Regression with single attributes. For each of the thirteen attributes, perform a linear regression using only the single attribute but incorporating a bias term so that the inputs are augmented with an additional* 1 *entry,* $(\mathbf{x}_i, 1)$*, so that we learn a weight vector* $\mathbf{w} \in \Re^2$*.*

c. *Linear Regression using all attributes. Now we would like to perform linear regression using all of the data attributes at once.*

   Perform linear regression on the training set using this regressor, and incorporate a bias term as above. Calculate the MSE on the training and test sets and note down the results. You should find that this method outperforms any of the individual regressors.

3.1 Duality

The optimal value $\mathbf{w}^*$ as obtained by Ridge Regression was given by (7). From this we can derive that:

$$\begin{aligned}
\mathbf{X}'\mathbf{X}\mathbf{w}^* + \gamma l \mathbf{w}^* &= \mathbf{X}'\mathbf{y} \\
\gamma l \mathbf{w}^* &= \mathbf{X}'\mathbf{y} - \mathbf{X}'\mathbf{X}\mathbf{w}^* \\
\mathbf{w}^* &= \frac{1}{\gamma l}\mathbf{X}'\left[(\mathbf{y} - \mathbf{X}\mathbf{w}^*)\right], \qquad (8)
\end{aligned}$$

showing that $\mathbf{w}^*$ can be expressed as a linear combination of the columns of $\mathbf{X}'$:

$$\mathbf{w}^* = \mathbf{X}'\alpha^*. \qquad (9)$$

The vector $\alpha^* \in \Re^n$ will be referred to as the *dual weight vector*.

Using this dual vector, the so-called dual version of RR can be derived as follows:

$$\begin{aligned}\alpha^* &= \frac{1}{\gamma l}(\mathbf{y} - \mathbf{X}\mathbf{w}^*) \\ \Rightarrow \gamma l \alpha^* &= (\mathbf{y} - \mathbf{X}\mathbf{X}'\alpha^*) \\ \Rightarrow (\mathbf{X}\mathbf{X}' + \gamma l \mathbf{I}_l)\alpha^* &= \mathbf{y} \\ \Rightarrow \alpha^* &= (\mathbf{X}\mathbf{X}' + \gamma l \mathbf{I}_l)^{-1}\mathbf{y} \qquad (10)\end{aligned}$$

3.2 Nonlinear regression

For nonlinear regression, the dual version will prove important. Obviously, linear regression is not capable of achieving a good predictive performance on a nonlinear data set. Here, the dual formulation will prove extremely useful, in combination with the *kernel trick*.

This kernel trick is based on nothing more than the observation that equation to compute α^* (equation (10)) only contains the vectors $\mathbf{x}_i$ in inner products with each other. Therefore, it is sufficient to know these inner products only, instead of the actual vectors $\mathbf{x}_i$.

As a result, we can also work with inner products between *nonlinear* mappings $\phi : \mathbf{x}_i \rightarrow \phi(\mathbf{x}_i) \in \mathcal{F}$ of $\mathbf{x}_i$ into a so-called *feature space* $\mathcal{F}$, as long as the inner product $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)'\phi(\mathbf{x}_j)$ can be evaluated efficiently. In many cases, this inner product (from now on called the *kernel function*) can be evaluated much more efficiently than the feature vector itself, which can even be of infinite dimensionality in principle. A commonly used kernel function for which this is the case is the Gaussian kernel, which is defined as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right) \qquad (11)$$

Using the notation $\mathbf{K} = \mathbf{X}\mathbf{X}'$ for the *kernel matrix* or *Gram matrix* containing the inner products between all training points, we can rewrite equation (10) in its kernel formulation (KRR):

$$\alpha^* = (\mathbf{K} + \gamma l \mathbf{I}_l)^{-1}\mathbf{y} \qquad (12)$$

The evaluation of the regression function on a test point can be reformulated as:

$$y_{\text{test}} = \sum_{i=1}^{l} \alpha_i^* K(\mathbf{x}_i, \mathbf{x}_{\text{test}}) \qquad (13)$$

where the K is the kernel function.

The most efficient way to do this is to compute the kernel function for all the data (train and test), and then select the appropriate portion of the kernel matrix as the testing data. If the train and test data were contiguous, this would be the lower left portion of the matrix as follows:

$$\bar{\mathbf{K}} = \left[\begin{array}{ll} K(\mathbf{x}_{\text{train}}, \mathbf{x}_{\text{train}}), & K(\mathbf{x}_{\text{train}}, \mathbf{x}_{\text{test}}) \\ K(\mathbf{x}_{\text{test}}, \mathbf{x}_{\text{train}}), & K(\mathbf{x}_{\text{test}}, \mathbf{x}_{\text{test}}) \end{array}\right] \qquad (14)$$

The kernel for the test points is then:

$$\mathbf{K}_{\text{test}} = \bar{\mathbf{K}}_{(\text{test, train})}$$

The cost function (MSE) can then be calculated as follows:

$$\frac{1}{l}(\mathbf{K}_{\text{test}}\alpha - y)'(\mathbf{K}_{\text{test}}\alpha - y) \qquad (15)$$

Effectively, by using such a kernel function corresponding to a nonlinear feature map, a linear regression method such as RR can be carried out in the feature space, amounting to a nonlinear regression function (equation (13)) in the original input space.

Exercise 10 (Kernel Ridge Regression) *In this exercise we will perform kernel ridge regression (KRR) on the data set. For this exercise, you will hold out* 2/3 *of data for training and report the test results on the remaining 1/3.*

a. Create a function called `kridgereg.m` *to perform kernel ridge regression using equation (12). It should take in a kernel matrix* `K`, `y` *vector, and scalar ridge parameter* γ *as inputs and return a vector* α *of dual weights. Note: do not explicitly use a matrix inverse as in* (12) *instead use the matrix left division operator:* `help mldivide`. *Please explain why your implementation correctly computes* α.

b. Create a function called `dualcost.m` *to calculate the Mean Squared Error (MSE) using equation (15). It should take in a kernel matrix* `K`, `y` *vector, and dual weight vector* α *as inputs and return a single value.*

c. Create a vector of γ *values* $[2^{-40}, \ldots, 2^{-26}]$ *and a vector of* σ *values* $[2^7, 2^{7.5}, \ldots, 2^{12.5}, 2^{13}]$ *(recall that* σ *is the variance parameter in the Gaussian kernel see equation* (11)*). Perform kernel ridge regression on the training set using five-fold cross-validation to choose among all pairing of the values of* γ *and* σ. *Choose the indices of the* γ *and* σ *values that perform the best to compute the predictor that you will use to report the test and training error.*

   1. Plot the "cross-validation error" (mean over folds of validation error) as a function of γ *and* σ.

   2. Calculate the MSE on the training and test sets for the best γ *and* σ.

d. Repeat a-c over 20 random $(2/3, 1/3)$ *splits of your data record the train/test error and the standard deviations of the train/test errors and summarise your results for exercise 9 and 10 in the following type of table.*

Method	**MSE train**	**MSE test**
Naive Regression	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Linear Regression (attribute 1)	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Linear Regression (attribute 2)	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Linear Regression (attribute 3)	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Linear Regression (attribute 4)	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Linear Regression (attribute 5)	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Linear Regression (attribute 6)	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Linear Regression (attribute 7)	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Linear Regression (attribute 8)	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Linear Regression (attribute 9)	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Linear Regression (attribute 10)	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Linear Regression (attribute 11)	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Linear Regression (attribute 12)	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Linear Regression (attribute 13)	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Linear Regression (all attributes)	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$
Kernel Ridge Regression	$??.?? \pm \sigma?$	$??.?? \pm \sigma?$

In summary, by using the kernel trick, we can apply simple techniques for linear regression in the *feature space* to perform nonlinear regression in the *input space.*

PART II [50%]

Questions

1. **[5 pts]** Suppose we perform linear regression with a Gaussian kernel $K_\beta(\mathbf{x}, \mathbf{t}) = \exp(-\beta\|\mathbf{x} - \mathbf{t}\|^2)$ to train a classifier for two-class data (i.e., $y \in \{-1, 1\}$). This classifier depends on the parameter β selected for the kernel. How should one choose β so that the learned linear classifier simulates a 1-Nearest Neighbor classifier? Explain your reasoning.

2. **[10 pts]** (*Using a perceptron to learn a linear classifier with a bias*)

$$f_{\mathbf{w},b}(\mathbf{x}) := \operatorname{sign}(\mathbf{w}^\top \mathbf{x} + b)\,.$$

(a) Describe how we may change the perceptron to learn linear classifier with bias $b \in \mathbb{R}$.

(b) How does incorporating the bias term change the guarantee (perceptron convergence theorem [Novikoff]) on the number of mistakes incurred by the perceptron algorithm? Specifically, will the bound on the number of mistakes

i. Increase.

ii. Decrease.

iii. May increase or decrease depending on the dataset.

Discuss your answer.

3. **[10 pts]** *Kernel modification* Consider the function $K_c(\mathbf{x}, \mathbf{z}) := c + \sum_{i=1}^{n} x_i z_i$ where $\mathbf{x}$, $\mathbf{z} \in \mathbb{R}^n$.

(a) For what values of $c \in \mathbb{R}$ is K_c a positive semidefinite kernel? Give an argument supporting your claim.

(b) Suppose we use K_c as a kernel function with linear regression (least squares). Explain how c influences the solution.

4. **[75 pts (a,b,c,d 45pts, e 30pts)]** *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 $\mathbf{x}_1, \ldots \mathbf{x}_m$ 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 $\mathbf{x}$ 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[2] of estimated sample complexity for least squares. Please include sample complexity plots for all four 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 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 $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{s-1}, y_{s-1})$. The derived upper bound $\hat{p}_{m,n}$ should be function of only m and n (i.e., it is independent of s). Justify your derivation, analytically.

(e) **[Challenge]** Find a 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 $m = \Omega(f(n))$.

[2] 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.

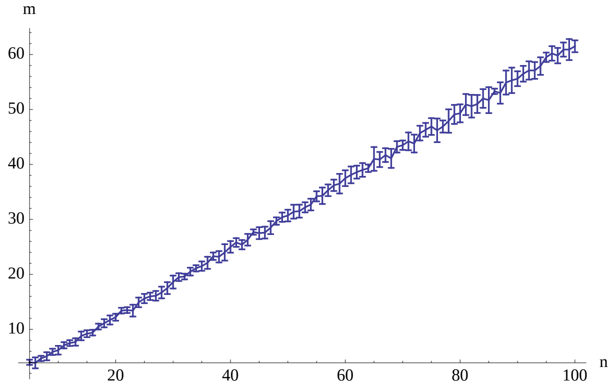

Figure 1: Estimated number of samples (m) to obtain 10% generalisation error versus dimension (n) for least squares.

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 $\mathbf{w}$ to define a classifier $f_{\mathbf{w}}(\mathbf{x}) := \text{sign}(\mathbf{w}^\top \mathbf{x})$.

   (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 in PMML (GI07) [Massi's Lecture 2, see `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 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 sequence of m patterns drawn uniformly at random from $\{-1,1\}^n$ with their derived labels. Then let $\mathcal{A}_{\mathcal{S}}(\mathbf{x})$ denote the prediction of an algorithm $\mathcal{A}$ trained from data sequence $\mathcal{S}$ on pattern $\mathbf{x}$. Thus the generalisation error is then

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

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

$$\mathcal{C}(\mathcal{A}) := \operatorname*{argmin}_m \mathbb{E}[\mathcal{E}(\mathcal{A}_{\mathcal{S}_m}) \leq 0.1] \,.$$

   With sufficient computational resources we may compute this exactly via,

$$\mathcal{C}(\mathcal{A}) = \operatorname*{argmin}_m 2^{-nm} \sum_{S \subseteq \{-1,1\}^{nm}} \mathcal{E}(\mathcal{A}_{\mathcal{S}}) \leq 0.1 \,.$$