CMPUT 466/551—Machine Learning

Assignment 1

Winter 2004
Department of Computing Science
University of Alberta

Due: in class, *Tuesday, January 27*
Worth: 15% of final grade
Instructor: Dale Schuurmans, Ath409, x2-4806, dale@cs.ualberta.ca

---

Note These questions require you to write small Matlab programs which are to be submitted by email. When finished, please send a *single* tar file containing all of your .m files **to the TA, Qiongyun Zhang, at qiongyun@cs.ualberta.ca** with a subject heading "CMPUT 466/551 A1 solutions".

Question 1 (Learning real predictors—linear hypotheses)

In this exercise you will write simple MATLAB functions to learn minimum error linear functions and test them on simulated data. You will need to be familiar with `ops`, `slash`, `rand`, `randn`, `function`, `linprog`, `plot`, and `print`. (Type `help` ⟨topic⟩ in MATLAB if you need to learn about any of these.)

We will consider three ways of generating data **X**, `y`. All will involve the same target function defined by the weight vector `u`= [01...1]′. The training data will have the form

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,n-1} \\ \vdots & & & \vdots \\ 1 & x_{t,1} & \cdots & x_{t,n-1} \end{bmatrix} \qquad \mathtt{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_t \end{bmatrix}$$

Data generation:

```
n = 2                              % dimension
t = 10                             % training size
u = [0; ones(n-1,1)]               % target weights
sigma = 0.1                        % noise level
X = [ones(t,1) rand(t, n-1)]       % training patterns
```

Generative model 1: `y = X*u + randn(t,1)*sigma`
Generative model 2: `y = X*u + randn(t,1)./randn(t,1)*sigma`
Generative model 3: `y = X*u + randn(t,1).*randn(t,1)*sigma`

(a) (1%) Write a MATLAB function `minL2(X,y)` which takes a $t \times n$ matrix **X** and $t \times 1$ vector of target values `y` and returns an $n \times 1$ vector of weights `w2` corresponding to the minimum *sum of squared errors* linear function.

Note: You must implement the algorithm as shown in class. This involves solving a system of linear equations. You can do this using MATLAB's `slash` operator, but you *cannot* use the built in `regress` function.

Your function must be able to handle arbitrary n and t.

(b) (1%) Write a MATLAB function `minL1(X,y)` which returns $n \times 1$ vector of weights `w1` corresponding to the minimum *sum of absolute errors* linear function. (**Note**: You can use MATLAB's built in `linprog` function to solve linear programs.)

(c) (1%) Write a MATLAB function `minLoo(X,y)` which returns $n \times 1$ vector of weights `woo` corresponding to the minimum *maximum absolute error* linear function. (**Note**: You can use MATLAB's built in `linprog` function to solve linear programs.)

(d) (1%) For each of the generative models 1, 2 and 3:

A: Generate a random training set $\mathbf{X}$, `y` using the model, and solve for each kind of linear function: `w2 = minL2(X,y)`, `w1 = minL1(X,y)`, and `woo = minLoo(X,y)`.

B: Produce a 2D plot of the training data and the three hypotheses corresponding to `w2`, `w1`, and `woo`.

```
clf
plot(X(:,2)',y','k*')
hold
plot([0 1],[w2(1) sum(w2)],'k-')
plot([0 1],[w1(1) sum(w1)],'k-.')
plot([0 1],[woo(1) sum(woo)],'k:')
print -deps experiment.1.1.<k>.ps
```

C: Report the three different kinds of error each of the three hypotheses obtained on the training data:

		type of error		
		L_1 error	L_2 error	L_∞ error
type	w1			
of	w2			
function	woo			

D: Generate `te = 1000` test examples from the same generative model and report the same matrix of results for the test data.

Hand in a plot and two tables for each generative model.

(e) (1%) For each generative model: Repeat (d) parts A, C and D 100 times and accumulate the sum of each kind of error for each kind of function in two matrices: one for the training errors and one for the testing errors. Report the *averages* for each kind of error and each kind of function in two tables (one training error and the other testing error).

Question 2 (Learning real predictors—polynomial bases)

In this exercise you will write a MATLAB function to compute minimum error polynomials and test it on simulated data. This will demonstrate overfitting dramatically! For this exercise it will be useful to know about MATLAB's `polyval` and `axis` functions.

We will consider two ways of generating the data, but will only consider one dimensional input so the data will (initially) have the form

$$\mathrm{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_t \end{bmatrix} \qquad \mathrm{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_t \end{bmatrix}$$

Data generation:

```
t = 10          % training size
sigma = 0.1     % noise level
x = rand(t, 1)  % training patterns
```

Generative model 1: `y = x > 0.5` (step function)

Generative model 2: `y = 0.5 - 10.4*x.*(x-0.5).*(x-1) + sigma*randn(t,1)` (noisy cubic)

(a) (2%) Write a MATLAB function `minL2poly(x,y,d)` which takes a $t \times 1$ vector of training inputs `x` and a $t \times 1$ vector of target values `y` and returns a $(d+1) \times 1$ vector of weights `c` corresponding to the minimum *sum of squared errors* polynomial of degree `d`. That is, `c` should be a vector of weights $\mathtt{c} = [c_1, ..., c_{d+1}]'$, which will be interpreted as specifying a polynomial $p(x) = c_1 x^d + c_2 x^{d-1} + \cdots + c_d x + c_{d+1}$.

Note: You must implement the algorithm as shown in class. This involves expanding each training input x_i into a vector $(x_i^d, x_i^{d-1}, ..., x_i, 1)$ to obtain

$$\mathbf{X} = \begin{bmatrix} x_1^d & x_1^{d-1} & \cdots & x_1 & 1 \\ \vdots & & & & \vdots \\ x_t^d & x_t^{d-1} & \cdots & x_t & 1 \end{bmatrix} \qquad \mathrm{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_t \end{bmatrix}$$

and then solving a system of linear equations to obtain `c`. You can do this using MATLAB's `slash` operator, but you *cannot* use the built in `polyfit` function.

Your function must be able to handle arbitrary t and d.

(b) (2%) For each of the generative models 1 and 2:

A: Generate a random training set `x`, `y` using the model, and solve for the best fit polynomials of degree 1, 3, 5, and 9: `c1 = minL2poly(x,y,1)`, `c3 = minL2poly(x,y,3)`, `c5 = minL2poly(x,y,5)`, `c9 = minL2poly(x,y,9)`.

B: Produce a 2D plot of the training data and the four hypotheses corresponding to `c1`, `c3`, `c5`, and `c9`.

```
clf
axis([0 1 -0.5 1.5]);
hold
plot(X(:,d)',y','k*')
xx = (0:1000)/1000;
yy = <mean value of generative model at xx>
plot(xx,yy,'k:')

plot(xx,polyval(c1,xx),'k-')
plot(xx,polyval(c3,xx),'k-.')
plot(xx,polyval(c5,xx),'k-')
plot(xx,polyval(c9,xx),'k-.')
print -deps experiment.1.2.<k>.ps
```

C: Report the *mean* sum of squares error (*i.e.*, the sum of squares error divided by t) that each of the four hypotheses obtained on the training data:

		L_2 error
	c1	
	c3	
polynomial	c5	
	c9	

D: Generate `te = 1000` test examples from the same generative model and report the mean sum of squares error on the *test* data (*i.e.*, the sum of squares error divided by `te`) in the same form as above.

Hand in a plot and two tables for each generative model.

(c) (1%) For each generative model: Repeat (b) parts A, C and D 100 times and accumulate the sum of mean squared errors for each degree in two matrices: one for the training errors and one for the testing errors. Report the *averages* for each kind of mean error at each degree in two tables (one training error and the other testing error).

Question 3 (Learning real predictors—regularization and kernels)

In this exercise you will tackle a real world prediction problem (which is discussed on pages 3 and 4 of the course textbook). The problem is to learn how to predict the log volume of a prostate cancer tumor (*lcavol*) from 7 clinical measurements: total log weight of the prostate (*lweight*), age of the patient (*age*), log of benign prostatic hyperplasia amount (*lbph*), seminal vesicle invasion (*svi*), log of capsular penetration (*lcp*), Gleason score (*gleason*), percent Gleason scores 4 or 5 (*pgg45*).

On the course webpage, download the data file data1.mat. Then type "`load data1.mat`" in MATLAB. This will load the training data into a matrix `X` and a vector `y` and the test data into a matrix `Xtest` and a vector `ytest`. Each row of the matrices corresponds to a 7 dimensional vector containing the clinical measurements for a patient, and the corresponding entry in the associated y-vector gives the discovered log volume of the patient's prostate cancer tumor. The goal is to learn a function that can predict the log volume of the prostate cancer tumor from the vector of 7 clinical measurements. Here, functions will be learned on the training data and then tested on the separate test data.

(a) (1%) Write a MATLAB function `minRegL2(X,y,lambda)` which takes a $t \times n$ matrix of training inputs `X`, a $t \times 1$ vector of target values `y`, and a regularization parameter `lambda`, and returns a $n \times 1$ dimensional vector of weights `w` corresponding to the minimum *penalized* sum of squared errors linear function. Specifically, `w` is the solution to $(\mathtt{X}^\top \mathtt{X} + \lambda I)\mathtt{w} = \mathtt{X}^\top \mathtt{y}$.

(b) (2%) Write a MATLAB function `minRegL2RadialKernel(X,y,sigma,lambda)` which returns a $t \times 1$ vector of training example weights `u` corresponding to the minimum *penalized* sum of squared errors radial kernel generalized linear function. Specifically, use the Gaussian kernel function $g(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{1}{2}\left\|\frac{\mathbf{x}_i - \mathbf{x}_j}{\sigma}\right\|^2}$ on input vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. Thus, given the $t \times n$ matrix of training inputs `X`, calculate the $t \times t$ matrix G such that $g_{ij} = g(\mathbf{x}_i, \mathbf{x}_j)$, and then solve for the example weights `u` such that $(G^\top G + \lambda I)\mathtt{u} = G^\top \mathtt{y}$.

(c) (1%) Write a MATLAB function `RadialKernelPredict(X,u,sigma,Xtest)` which returns a vector `yhat` of predictions made by the radial kernel generalized linear function on the matrix of test inputs `Xtest`. That is, given a $t \times 1$ vector of example weights `u` and $t \times n$ matrix of training inputs `X`, calculate the prediction $\hat{y}$ on a test row vector $\mathbf{x}$ according to $\hat{y} = \sum_{i=1}^{t} u_i \, g(\mathbf{x}_i, \mathbf{x})$, where the training row vectors $\mathbf{x}_i$ are chosen from `X`.

(d) (1%) Use the following parameters to learn different prediction functions:

```
w0 = minRegL2(X,y,0)                      % no regularization
w1 = minRegL2(X,y,24)                     % regularization
u0 = minRegL2RadialKernel(X,y,14,0)       % no regularization
u1 = minRegL2RadialKernel(X,y,14,0.002)   % regularization
```

Report the mean sum of squares error that each of these four functions make on both the training and the test data (in a 4×2 table).