

Question 1. Learning real predictors–linear hypotheses

In this exercise you will write simple Matlab/Octave functions to learn minimum error linear functions and test them on simulated data.

You will need to be familiar with the `backslash` operator for solving linear systems, as well as `rand`, `randn`, `function`, `plot`, and `print`. Furthermore, you will need to solve linear programs by calling the built in solvers. (In Matlab you can use `linprog` to solve linear programs. In Octave, you can use `glpk` to solve linear programs.) Type `help <topic>` in Matlab/Octave if you need to learn about any function.

We will consider three ways of generating data X, y . All will involve the same target function defined by the weight vector $u = [0 \ 1 \dots 1]'$ corrupted by noise. The training data will have the form

$$X = \begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,n-1} \\ \vdots & & & \vdots \\ 1 & x_{t,1} & \dots & x_{t,n-1} \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_t \end{bmatrix}$$

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 pattern
```

```
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/Octave function `minL2(X, y)` which takes a $t \times n$ matrix X and $t \times 1$ vector of target values y and returns and $n \times 1$ vector of weights w_2 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 the “backslash” 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/Octave function `minL1(X, y)` which returns $n \times 1$ vector of weights w_1 corresponding to the minimum *sum of absolute errors* linear function. (Note: If you are using Matlab, then you can use Matlab’s built in `linprog` function to solve linear programs. If you are using Octave, then you can use Octave’s built in `glpk` function to solve linear programs.)
- (c) (1%) Write a Matlab/Octave function `minLoo(X, y)` which returns $n \times 1$ vector of weights w_{oo} corresponding to the minimum *max of absolute errors* linear function. (Note: If you are using Matlab, then you can use Matlab’s built in `linprog` function to solve linear programs. If you are using Octave, then you can use Octave’s built in `glpk` function to solve linear programs.)

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

A: Generate synthetic data and train. In other words, generate a random training set X, y using the model, and solve for each kind of linear function: $w_2 = \text{minL2}(X, y)$, $w_1 = \text{minL1}(X, y)$, and $w_{\infty} = \text{minL}_{\infty}(X, y)$

B: Produce a 2D plot of training data and the three hypotheses corresponding to w_2 , w_1 , and w_{∞} .

```
clf
plot((X(:,2)', y', 'k*'))
hold
plot([0 1], [w2(1) sum(w2)], 'r-')
plot([0 1], [w1(1) sum(w1)], 'g-')
plot([0 1], [w_{\infty}(1) sum(w_{\infty})], 'b-')
print -deps experiment.1.1.<m>.ps
```

where $m \in 1, 2, 3$ ranges over the 3 generative models.

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

		type of error		
		L_1 error	L_2 error	L_{∞} error
type	w1			
of	w2			
function	w_{\infty}			

D: Keeping w_2 , w_1 and w_{∞} fixed, generate another set of $t_e = 1000$ test examples from the same generative model, and report the matrix of error values made by the predictors on the test data.

		type of error		
		L_1 error	L_2 error	L_{∞} error
w1				
w2				
w_{\infty}				

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/Octave 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 the `polyval` and `axis` functions in Matlab/Octave.

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

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_t \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_t \end{bmatrix}$$

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

Generative model 1: `y = double(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) (1%) Write a Matlab/Octave function `minL2poly(x, y, d)` which takes a $t \times 1$ vector of training inputs \mathbf{x} and a $t \times 1$ vector of target values \mathbf{y} and returns a $(d+1) \times 1$ vector of weights \mathbf{c} corresponding to the minimum *sum of squared errors* polynomial of degree d . That is, \mathbf{c} should be a vector of weights $\mathbf{c} = [c_1, c_2, \dots, c_{d+1}]'$, which will be interpreted as specifying a polynomial

$$p(x) = c_1 x^d + c_2 x^{d-1} + \dots + c_d x + c_{d+1}.$$

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

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

and then solving a system of linear equations to obtain \mathbf{c} . You can do this using “backslash” operator in Matlab/Octave, but you cannot use the builtin `polyfit` function.

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

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

- A: Generate a random training set \mathbf{x}, \mathbf{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', y', 'k*')
xx = (0:1000)/1000
yy = <mean value of generative model at xx>
plot(xx, yy, 'k:')

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

where $\langle m \rangle \in \{1, 2\}$ ranges over the 2 generative models. Here “<mean value of generative model at xx>” means plot the step function or the cubic function, *without the noise*, depending on which generative model is being used.

- 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	
c5	
c9	

- D: Generate another set of $t_e = 1000$ *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 t_e) 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.

- (a) (1%) Prove that sum of squares error function is a convex function of weight parameter. You could either use the definition of a convex function, or show that the Hessian H is a symmetric positive semi-definite matrix.
- (b) (1%) Compute the gradient $\nabla f(\mathbf{x})$ and Hessian $\nabla^2 f(\mathbf{x})$ of the Rosenbrock function

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

where $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$. Show that $\mathbf{x}^* = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is the only local minimizer of this function.