Machine Learning

Prof. Jose L. Mendoza-Cortes

Scientific Computing Department, Dirac Science Building
Materials Science and Engineering, High Performance Materials Institute
Florida State University
jmendozacortes@fsu.edu

Condensed Matter Theory, National High Magnetic Field Laboratory
Florida State University
mendoza@magnet.fsu.edu

Chemical and Biomedical Engineering
Florida State University — Florida A&M University — College of Engineering
mendoza@eng.famu.fsu.edu

Web: http://mendoza.eng.fsu.edu/

Matlab Code: Linear Fitting

You do not have to use my code. If you do, it is not recommended to copy-paste it. Instead type it all line by line, it will help you understand it. Alternatively, you can create your own code.

Code 1: Code used in class

```
clc; clear; close all;
%You can also import the data from a text file into MATLAB
datafromfile = importdata('forcevsvelocity.txt');
%or other ways: https://www.mathworks.com/help/matlab/text-files.html
%https://www.mathworks.com/help/matlab/ref/xlsread.html

format shortG
table = [datafromfile.data ...
    log10(datafromfile.data(:,1)) ...
    log10(datafromfile.data(:,2)) ...
    log10(datafromfile.data(:,1)).^2 ...
    log10(datafromfile.data(:,1)).*log10(datafromfile.data(:,2))];

%Using the formulas for linear fitting (page 338 of textbook)
%y = ao + ai*x
%a1 = (n*sum(xiyi)-Sum(xi)*Sum(yi))/(nSum(xi^2)-(Sum(xi))^2)
%ao = mean(y) - a1*mean(x)
n = length(datafromfile.data(:,1));
xi=log10(datafromfile.data(:,1));
yi=log10(datafromfile.data(:,2));
a1 = (n*sum(xi.*yi)-sum(xi)*sum(yi))/(n.*sum(xi.^2)-(sum(xi)).^2);
a0 = mean(yi) - a1.*mean(xi);
```

Matlab Code: Linear Fitting

Code 2: Code used in class (cont.)

```
x=log10(datafromfile.data(1,1)):0.01:log10(datafromfile.data(end,1));
%log(y) = a0 + a1*log(x)
y = a0 + a1.*x;

figure(1)
plot(xi,yi,'xb',x,y,'-b')
legend('data','fitting','location','best')
xlabel('log x','Interpreter','latex')
ylabel('log y','Interpreter','latex')
title('Transformed Data')


%If we transform back to the original equation
%y=alpha*x^(beta)
alpha = 10^(a0);
beta = a1; %We did not take the algorithm10 of this when transforming
x=datafromfile.data(1,1):2:datafromfile.data(end,1);
y =alpha.*x.^(beta);
figure(2)
plot(datafromfile.data(:,1),datafromfile.data(:,2),'xm',x,y)
legend('data','fitting','location','best')
xlabel('log x','Interpreter','latex')
ylabel('log y','Interpreter','latex')
title('Original Data (No transformation)')
```

Can you calculate the R^2 for this fitting? What is the value?

Matlab Code: Linear Fitting

Code 3: forcevsvelocity.txt

```
velocity, m/s   Force,N velocity, m/s
10 25
20 70
30 380
40 550
50 610
60 1220
70 830
80 1450
```

This is the data file called forcevsvelocity.txt what is needed for the first part of the code.
Look for the line of the code:
‘`datafromfile = importdata('forcevsvelocity.txt')`’

Up to here we have been using the formulas from page 338 of textbook

Matlab Code: Polynomial Regression

Code 4: Code used in class (cont.)

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% POLYNOMIAL REGRESSION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

datapoly=[
0 2.1
1 7.7
2 13.6
3 27.2
4 40.9
5 61.1
];

%Therefore to solve the fitting to
% y = a0 + a1*x + a2*x^2 + ... + am*x^m

%Matrix to solve: a = N^(-1)*r
%N = [
% n          sum(xi)    sum(xi^2)
% sum(xi)   sum(xi^2)  sum(xi^3)
% sum(xi^2) sum(xi^3)  sum(xi^4)]

% s = [
% sum(yi)
% sum(xiyi)
% sum(xi^2*yi)]
```


Matlab Code: Polynomial Regression

Code 5: Code used in class (cont.)

```

n=length(datapoly);
xi=datapoly(:,1);
yi=datapoly(:,2);
%Thus
N = [
 n           sum(xi)     sum(xi.^2)
 sum(xi)     sum(xi.^2)  sum(xi.^3)
 sum(xi.^2) sum(xi.^3)   sum(xi.^4)];

s = [
 sum(yi)
 sum(xi.*yi)
 sum(xi.^2.*yi)];

%a= N\s
%or
%a=inv(N)*s
%or
a = GaussPivot(N,s)

%Therefore
a0 = a(1); % 1st element of the array
a1 = a(2); % 2nd element of the array
a2 = a(3); % 3rd element of the array
%Now to get the r^2 we use:
%Sr = sum((yi-a0-a1*xi).^2)
```

Matlab Code: Polynomial Regression

Code 6: Code used in class (cont.)

```
%St = sum ((yi-mean(y)).^2)
%r2 = (St-Sr)/St

Sr = sum((yi-a0-a1*xi-a2*xi.^2).^2);
St = sum ((yi-mean(yi)).^2);
r2 = (St-Sr)/St;

x = datapoly(1,1):0.1:datapoly(end,1);
y = a0 + a1.*x + a2*x.^2;
figure(3)
plot(xi,yi,'xr',x,y,'-r')
legend('data',strcat('fit, R^2=',num2str(r2)),'location','best')
xlabel('x','Interpreter','latex')
```

Up to here we have been using the formulas from page 362 of textbook

You do not have to use my code. If you do, it is not recommended to copy-paste it. Instead type it all line by line, it will help you understand it. Alternatively, you can create your own code.

Matlab Code: Multiple Linear Regression

Code 7: Code used in class (cont.)

```



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MULTIPLE LINEAR REGRESSION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%format long

%x1 x2 y
multi_data=[
0.0  0  5.1
2.0  1  10
2.5  2  9.3
1.0  3  0.1
4.0  6  3.3
7.0  2  27.2
]; % Defines the data input

%Therefore to solve the fitting to
% y = a0 + a1*x1 + a2*x2 + ... + am*xj^m

%Matrix to solve: a = N^(-1)*r
%N = [
% n          sum(x1i)     sum(x2i)
% sum(x1i)   sum(x1i^2)   sum(x1i*x2i)
% sum(x2i)   sum(x1i*x2i) sum(x2i^2)]
```

Matlab Code: Multiple Linear Regression

Code 8: Code used in class (cont.)

```
% s = [
% sum(yi)
% sum(x1i*yi)
% sum(x2i*yi)]

n=length(multi_data); % Length of data
x1i=multi_data(:,1); % Call collumn 1
x2i=multi_data(:,2); % Call collumn 2
yi=multi_data(:,3);% Call collumn 3

%Substituting these values
N = [
 n          sum(x1i)     sum(x2i)
 sum(x1i)  sum(x1i.^2)   sum(x1i.*x2i)
 sum(x2i)  sum(x1i.*x2i) sum(x2i.^2)]; % Defines matrix after minimization

s = [
sum(yi)
sum(x1i.*yi)
sum(x2i.*yi)]; % RHS of the equation

%a= N\r
%or
%a=inv(N)*r
%or
a = GaussPivot(N,s) % Solves for a
```


Matlab Code: Multiple Linear Regression

Code 9: Code used in class (cont.)

```
%Therefore
a0 = a(1); % 1st element of the array
a1 = a(2); % 2nd element of the array
a2 = a(3); % 3rd element of the array
%Now to get the r^2 we use:
%Sr = sum((yi-a0-a1*x1i-a2*x2i).^2)
%St = sum ((yi-mean(y)).^2)
%r2 = (St-Sr)/St

Sr = sum((yi-a0-a1*x1i-a2*x2i).^2); % Square of the residual with respect to the model
St = sum ((yi-mean(y)).^2); % Square of the residual with respect to the mean
r2 = (St-Sr)/St; % Coefficient of determination
% r = correlation coefficient(sqrt(r^2))

x1 = multi_data(1,1):0.5:multi_data(end,1); % linspace in 2 dimensions
x2 = multi_data(1,1):0.5:multi_data(end,1); % linspace in 2 dimensions
[x1,x2] = meshgrid(x1, x2); % linspace in 2 dimensions
y = a0 + a1*x1 + a2*x2; % plot

figure(4)
%hold on
scatter3(x1i,x2i,yi,'filled') % plots scatter data (data)
```

Up to here we have been using the formulas from page 366 of textbook

Contents

Motivation

Antoine de Saint-Exupery

Perfection is achieved, not when there is nothing left to add, but when there is nothing left to take away

Least Squares

Recall, for polynomial interpolation, given

$$x_0, x_1, \cdots, x_i, \cdots x_n$$

and the "values" at those points

$$f_0, f_1, \cdots, f_i, \cdots f_n$$

we sought an order n polynomial, which passed exactly though all the $\{x_i, f_i\}$

$$f_i = \sum_{j=0}^{n} a_j x_i^j$$

Least Squares

- Written in full, we wanted a polynomial $p_n(x)$ to pass through the $n+1$ points by solving the linear system

$$\begin{aligned} p_n(x_0) = &\quad a_0 + a_1 x_0 + \cdots + a_n x_0^n = &\quad f_0 \\ p_n(x_1) = &\quad a_0 + a_1 x_1 + \cdots + a_n x_1^n = &\quad f_1 \\ &\vdots \\ p_n(x_n) = &\quad a_0 + a_1 x_n + \cdots + a_n x_n^n = &\quad f_n \end{aligned}$$

- In all, $n+1$ equations, $n+1$ unknowns (the a_i)

$$\mathbf{Xa} = \mathbf{f}$$

where $\mathbf{X}$ was the (square) Vandermonde matrix.

Least Squares: Motivation

- As n increases, the interpolating function becomes more complex
  - often picks up undesirable features
  - oscillations (Runge's phenomenon)
  - begins fitting noise, instead of the signal
- In least squares approximation
  - we do not require the approximating function to pass through points
  - we require it to lie as close as possible to the data "in some sense"
  - the approximating function is "coarser" than the data

Coarser Object

Suppose we wanted to fit a lower order polynomial p_m through $n+1$ points such that $m < n$.

Written in full, we seek a polynomial $p_m(x)$ that "passes" through the $n+1$ points by solving the linear system

$$\begin{aligned}
a_0 + a_1 x_0 + \cdots + a_m x_0^n &= f_0 \\
a_0 + a_1 x_1 + \cdots + a_m x_1^n &= f_1 \\
&\vdots \\
a_0 + a_1 x_n + \cdots + a_m x_n^n &= f_n
\end{aligned}$$

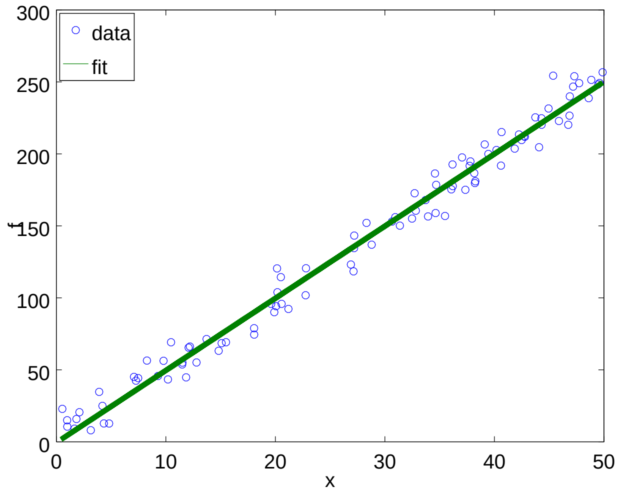
In all, $n+1$ equations, $m+1$ unknowns (the a_i)

$$\mathbf{Aa} = \mathbf{f}.$$

Q: If $n+1 = 100$ and $m+1 = 2$, what is the size of $\mathbf{A}$?

Least Squared Error

- In this case, we have too many equations, and too few unknowns.



- We can't "pass" or interpolate a straight-line through the 100 points!
- We can, however, try to "minimize" the distance between the straight line and the scatter of points.

Matlab code used

Code 10: One way of doingthe fitting and obtain the plot above

```
clc; close all; clear;
%if(0)
n = 100;
x = rand(100,1)*50;
m = 5;
c = 1;
y = m*x + c + 10*randn(n,1);

A = [ones(n,1) x];
a = (A'*A)\(A'*y)
f = A * a;

figure (1)
set(gca,'FontSize',18);
plot(x,y,'o',x,f,'LineWidth',4)
legend('data','fit','Location','NorthWest');
xlabel('x');
ylabel('f');
axis([0 50 0 300]);
%print -dpdf -FHelvetica:16  overdetermined.pdf
%end
```

Least Squared Error

For any line (defined by $\mathbf{a}$), we define the squared-error as:

$$\epsilon^2 = ||\mathbf{A}\mathbf{a} - \mathbf{f}||_2^2$$

Written out more explicitly,

$$\begin{aligned}\epsilon^2 &= (\mathbf{A}\mathbf{a} - \mathbf{f})^T(\mathbf{A}\mathbf{a} - \mathbf{f}) \\ &= \mathbf{a}^T\mathbf{A}^T\mathbf{A}\mathbf{a} - 2\mathbf{f}^T\mathbf{A}\mathbf{a} + \mathbf{f}^T\mathbf{f}\end{aligned}$$

We want to minimize the squared error, so we set:

$$\frac{\partial \epsilon^2}{\partial \mathbf{a}} = 0.$$

This yields:

$$2\mathbf{A}^T\mathbf{A}\mathbf{a} - 2\mathbf{A}^T\mathbf{f} = \mathbf{0}$$

Least Squared Error

The least-squares solution $\hat{\mathbf{a}}$ can be obtained by solving the so-called "normal equations":

$$\boxed{\mathbf{A}^T\mathbf{A}\hat{\mathbf{a}} = \mathbf{A}^T\mathbf{f}}$$

Question:
If $n + 1 = 100$ and $m = 2$ as before, what is the size of:

- $\mathbf{A}^T\mathbf{A}$?
- $\mathbf{A}^T\mathbf{f}$?

Are the normal equations over-determined?

Summary: Discrete Polynomial Approximation

- Given an over-determined system,

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^m \\ 1 & x_1 & x_1^2 & \dots & x_1^m \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ a_m \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ \dots \\ f_n \end{bmatrix}$$

$$\boxed{\mathbf{A}_{(n+1)\times(m+1)}\mathbf{a}_{(m+1)\times 1} = \mathbf{f}_{(n+1)\times 1}}$$

- In typical regression problems, $n \gg m$, and hence the matrix $\mathbf{A}$ is tall.
- In the usual sense, this corresponds to a case with too many equations $(n+1)$, and too few unknowns $(m+1 < n+1)$

Discrete Polynomial Approximation

- The normal equations balance this mismatch by seeking to minimize the squared-error:

$$\mathbf{A}^T\mathbf{A}\hat{\mathbf{a}} = \mathbf{A}^T\mathbf{f}$$

- Essentially, by pre-multiplying both sides of the original equation by $\mathbf{A}^T$, we get a “square” linear system, where the number of equations is $m+1$
- This can be solved using methods from linear algebra.
- Let us consider a simple example.

Example

Problem: Consider the following data generated by adding "white noise" according to the equation

$$f = 5x + 1 + 2.5N(0,1)$$

x	f
1.00	3.97
2.00	9.66
3.00	14.41
4.00	19.38
5.00	22.10
6.00	31.00
7.00	38.70
8.00	35.53
9.00	44.99
10.00	54.54

Code

```
n = 9;
x = (1:1:n+1)';
f = 5*x + 1 + 2.5 * randn(n+1,1); % adds white noise
```

We want to set the matrix $\mathbf{A}$ as

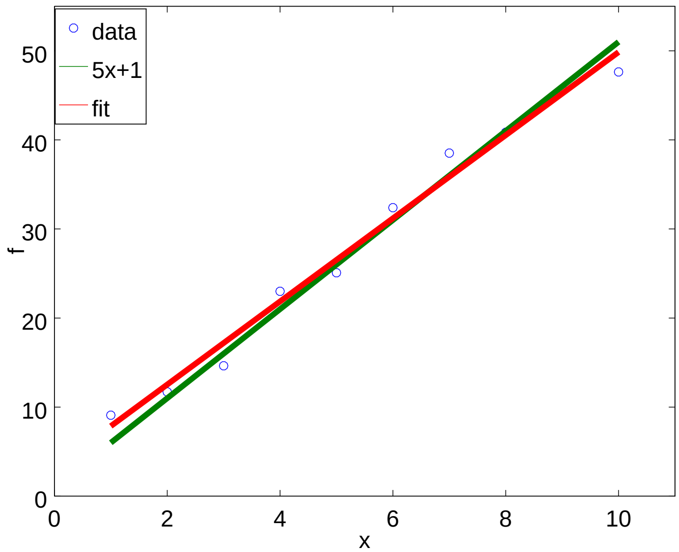
$$\mathbf{A} = \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \\ \dots & \\ 1 & x_n \end{bmatrix}$$

and solve the linear system $\mathbf{A}^T\mathbf{A}\hat{\mathbf{a}} = \mathbf{A}^T\mathbf{f}$.

```
A = [ones(n+1,1) x];
ahat = (A'*A)\(A'*f)
fhat = A * ahat;
```

where we have also finally set $\hat{\mathbf{f}} = \mathbf{A}\hat{\mathbf{a}}$

Solution



Best fit $f = 5.30x - 1.74$

Matlab code used (alternative)

Code 11: Another way of doing the fitting and obtain the plot above

```
%if(1)
n = 10;
x = (1:1:n)';
m = 5;
c = 1;
y = m*x + c + (m/2)*randn(n,1);

A = [ones(n,1) x];
a = (A'*A)\(A'*y)
f = A * a;

figure (2)
set(gca,'FontSize',18);
%plot(x,y,'o',x,m*x +c,'LineWidth',4,x,f,'LineWidth',4)
plot(x,y,'o');
hold on
plot(x,m*x +c,'LineWidth',4)
plot(x,f,'LineWidth',4)
legend('data','5x+1','fit','Location','NorthWest');
xlabel('x');
ylabel('f');
axis([0 11 0 55]);
%print -dpdf -FHelvetica:16 linregex.pdf
%end
```

Normal Equation (Polynomial): Matlab Code

Code 12: Most important Code of this class

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MATRIX FORM
% POLYNOMIAL REGRESSION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
datapoly=[
0 2.1
1 7.7
2 13.6
3 27.2
4 40.9
5 61.1
];

x=datapoly(:,1);
y=datapoly(:,2);

Z = [ones(size(x)) x x.^2];
% Solving a = {[Z^T][Z]}^-1 {[Z^T]{y}}
a = GaussPivot(Z'*Z,Z'*y)
%Compare parameters with lines 55-110

% Now getting r^2
r2 = 1 - ( sum((y-Z*a).^2) ) / ( sum((y-mean(y)).^2) );
```


Normal Equation (Multiple Linear): Matlab Code

Code 13: Most important Code of this class

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MATRIX FORM
% MULTIPLE LINEAR REGRESSION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%format long

%x1 x2 y
multi_data=[
0.0  0  5.1
2.0  1  10
2.5  2  9.3
1.0  3  0.1
4.0  6  3.3
7.0  2  27.2
];

x1i=multi_data(:,1);
x2i=multi_data(:,2);
y=multi_data(:,3);

Z = [ones(size(x)) x1i x2i];
% Solving a = {[Z^T][Z]}^-1 {[Z^T]{y}}

a = GaussPivot(Z'*Z,Z'*y)
%Compare parameters with lines 130-206

% Now getting r^2
r2 = 1 - ( sum((y-Z*a).^2) ) / ( sum((y-mean(y)).^2) )
```

Normal Equation (Multiple Linear): Daily Habits Are More Important Than Big, Infrequent Home Runs. - Nicolas Cole.

Anyone can talk the talk. Not many people can walk the walk.

A terrible habit quite a few people fall into is believing that "one day" it'll all come together. What does that even mean, "one day"? What are you going to do, wake up and find yourself in a $5-million mansion with two Ferraris parked outside? What, is it just going to "appear" out of nowhere?

"One day" is today. "One day" is right now. You're not going to "be patient one day." You're going to be patient NOW. You're not going to "start" doing things differently one day. "You're going to start doing things differently NOW. You're not going to "finally make it work one day." You're going to make it work right NOW.

Big leaps happen by adding lots of tiny steps up over a long period of time. If you think you can skip that process, you're wrong. Whatever it is you want to become, become that to the best of your ability right now. Whatever it is you want to do, do that to the best of your ability right now. In weightlifting we would call this "training until failure."

Every day, everything you do, train until failure.

Same goes for learning. Do not study for the exam or the quiz, study for learning and understand. Daily habits are more important than long days before the examination.

Appendix: Scripts included

Try these commands in your own workstation, i.e. have the lectures on one half side of your screen and Matlab/Octave-GUI on the other half.

Check the scripts/functions under the directory for this note number (X): /NX_Notes_directory