Contents

Chapter 1

Week 1

1.1 Supervised and Unsupervised Learning

The most basic thing to remember is that we already know what out correct output should look like in Supervised Learning. But, we have little or no idea about what out results should look like.

Supervised Learning:

- Classification: Spam/Not-spam.
- Regression: Predicting age.

Unsupervised Learning:

- Clustering: Grouping based on different variables.
- Non Clustering: Finding structue in chaotic environment.

1.2 Linear Regression with one variable

Regression being a part of Supervised Learning is used for estimatng data (Real-valued output).

1.2.1 Cost Function

This function measures the performance of a Machine Learning model for given data.

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$
Parameters: θ_0, θ_1
Cost Function:

$$J(\theta_0, \theta_1) = 1/2m \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{1.1}$$

Goal: Minimize cost function with θ_0, θ_1 as parameters.

1.2.2 Gradient Descent

Basic idea:

- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we end up at minima.

Algorithm: repeat until convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_j} \tag{1.2}$$

(for $j = 0, 1$,here).

Intution: If α is too small, descent can be slow and if too large, descent may fail to converge or even diverge. Gradient descent can converge to a local minimum, even with fixed learning rate α. As we approach local mimimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.

1.2.3 Gradient Descent for linear regression

Combining gradient descent algorithm with linear regression model, we get:

$$j = 0 : \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0} = 1/2 \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \tag{1.3}$$

$$j = 1 : \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_1} = 1/2 \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}).x^{(i)} \tag{1.4}$$

Now, we can repeat 1.3 and 1.4 until convergence to obtain the minima.

"Batch" gradient descent: Each step of gradient descent uses all the training examples. For eq. "m" batches in equation 1.1.

Chapter 2

Week 2

2.1 Multivariate Linear Regression

Linear regression involving more than one variable. For eq., Predicting price of a house based on parameters "Plot Area", "No. of Floors", "Connectivity with markets", etc.

2.1.1 Multiple Features

The multivariable form of the hypothesis is as follows:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + ... + \theta_n x_n. \tag{2.1}$$

This hypothesis funtion can be concisely represented as:

$$h_\theta(x) = \theta^T x \tag{2.2}$$

where, θ^T is a 1xn matrix consisting of $\theta_0, \theta_1, \theta_2 ... \theta_n$.

2.1.2 Gradient Descent for Multiple Variables

Gradient descent formula for Multiple variable will be similar to that of single variable.

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}).x_j^{(i)} \tag{2.3}$$

Repeating this equation until convergence will give the minima. [1]

Feature Scaling

Feature Scaling is used to reduce the number of iterations in Gradient Descent. Basic idea of feature scaling is to bring all the features on the same scale. (in general we try

[1] $x_0 = 1$ in equation 2.3

to approximate every feature in the range $-1 < x_i < 1$)

Reducing the number of iteration doesn't mean making computation of each step easier. And also it does not effect comtational efficiency of Normal Equation.

Mean Normalisation

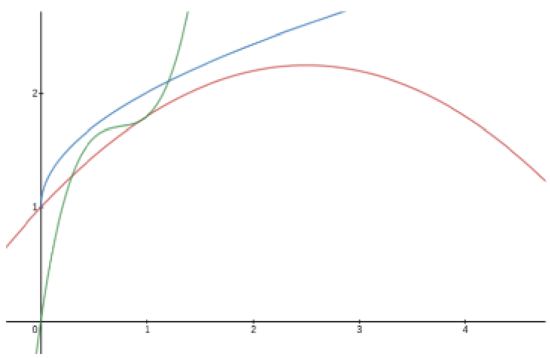
Mean Normalisation makes features to have approximately zero mean.

Learning Rate

If α is too small: slow convergence.
if α is too large: $J(\theta)$ may not decrease on every iteration, or may not converge.

Polynomial Regression

Selecting proper polynomial for fitting data is very important.



Red: Quadratic
Blue: Square root funtion $\theta_0 + \theta_1 x + \theta_2 \sqrt{x}$
Green: Cubic function

2.2 Normal equation

Normal Equation is a method to solve for θ_T analytically, by creating a $m \times (n+1)$ matrix X and another $m \times 1$ matrix Y.[2]

Mathematically θ is given as:

$$\theta = (X^T X)^{-1} X^t y \tag{2.4}$$

[2] Every element of first column of matrix X is 1 and other are the feature's coefficient

Gradient Descent	Normal Equation
Need to choose α	No need to choose α
Needs many iteration	Don't need to iterate
Works well with large n	Slow for large n

Reasons for non-invertiblity of $X^T X$

- Redundant features (linear dependence) [3]
- Too many features (m <= n)

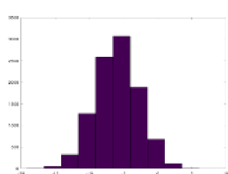
2.3 Octave/MATLAB commands

Basic Operations

```
octave:1> a = pi
a =  3.1416
octave:2> disp(sprintf('6 decimals: %0.6f', a))
6 decimals: 3.141593
octave:3> a
a =  3.1416
octave:4> format long
octave:5> a
a =  3.141592653589793
octave:6> format short
octave:7> a
a =  3.1416
octave:8> v = 1:0.1:2
v =

    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000
1.7000    1.8000    1.9000    2.0000

octave:9> v = 1:0.1:2
v =

 Columns 1 through 8:

    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000
1.7000

 Columns 9 through 11:
```

[3] Eg. Using both m^2 & $(feet)^2$ features

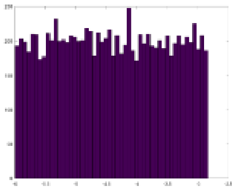
```
   1.8000   1.9000   2.0000

octave:10> v = 1:6
v =

   1   2   3   4   5   6

octave:11> zeros(1,3)
ans =

   0   0   0

octave:12> rand(1,3)
ans =

   0.43623   0.76554   0.23635

octave:13> randn(1,3)
ans =

   0.5602642   -0.0043628   0.1344922

octave:14> w = -6 + sqrt(10)*(randn(1,10000))

octave:15> hist(w)
```



```
octave:1> w = -6 + sqrt(10)*(rand(1,10000));

octave:2> hist(w,50)
```

Moving Data around

```
octave:1> A = [1,2;3,4;4,5]
A =

   1   2
```



```
   3   4
   4   5

octave:2> size(A)
ans =

   3   2

octave:3> sz = size(A)
sz =

   3   2

octave:4> size(sz)
ans =

   1   2

octave:5> size(A,1)
ans =  3
octave:6> size(A,2)
ans =  2
octave:7> length(A)
ans =  3
octave:8> length([1,2,3,4,5])
ans =  5
octave:9>
octave:9>
octave:9> pwd
ans = /home/sahasra
octave:10> cd /home/sahasra/
octave:11> pwd
ans = /home/sahasra
octave:12> ls
Android                  Documents         Music     Public     Videos
AndroidStudioProjects    Downloads         MyPaint   snap
Desktop                  examples.desktop  Pictures  Templates
octave:13> who
```

```
Variables in the current scope:

A    ans  sz

octave:14> whos
Variables in the current scope:

  Attr Name        Size                     Bytes  Class
  ==== ====        ====                     =====  =====
       A           3x2                         48  double
       ans         1x13                        13  char
       sz          1x2                         16  double

Total is 21 elements using 77 bytes

octave:15> clear
octave:16> whos
octave:17> A = [1,2;3,4;5,6]
A =

   1   2
   3   4
   5   6

octave:18> A(3,2)
ans =  6
octave:19> A(2,:)
ans =

   3   4

octave:20> A(:,2)
ans =

   2
   4
   6

octave:21> A([1,3], :)
ans =

   1   2
   5   6

octave:22> A([2,3], :)
```

```
ans =

   3   4
   5   6

octave:23> A(:,2) = [10;11;12]
A =

    1   10
    3   11
    5   12

octave:24> A = [A, [5;6;7]]
A =

    1   10    5
    3   11    6
    5   12    7

octave:25> A(:)
ans =

    1
    3
    5
   10
   11
   12
    5
    6
    7

octave:26> A
A =

    1   10    5
    3   11    6
    5   12    7

octave:27> B = [45;46;47]
B =

   45
   46
   47
```

```
octave:28> C = [A,B]
C =

   1   10    5   45
   3   11    6   46
   5   12    7   47
```

Computing on Data

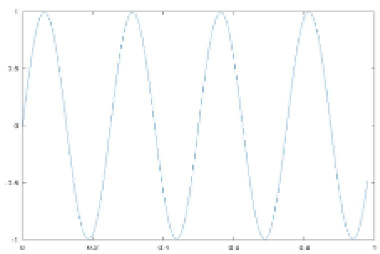
```
octave:1> A = [1 2;3 4;5 6]
A =

  1   2
  3   4
  5   6

octave:2> B = [11 12; 13 14; 15 16]
B =

  11   12
  13   14
  15   16

octave:3> C = [1  1; 2 2]
C =

  1   1
  2   2

octave:4> A*C
ans =

   5    5
  11   11
  17   17

octave:5> A .* B % A .* B gives element wise operation
ans =

  11   24
  39   56
  75   96

octave:6> 1 ./ A
ans =
```

```
    1.00000    0.50000
    0.33333    0.25000
    0.20000    0.16667

octave:7> v = [1;2;3]
v =

   1
   2
   3

octave:8> log(v)
ans =

   0.00000
   0.69315
   1.09861

octave:9> exp(v)
ans =

    2.7183
    7.3891
   20.0855

octave:10> abs([-1; 2; -3])
ans =

   1
   2
   3

octave:11> A
A =

   1   2
   3   4
   5   6

octave:12> A' % A' = A transpose
ans =

   1   3   5
   2   4   6
```

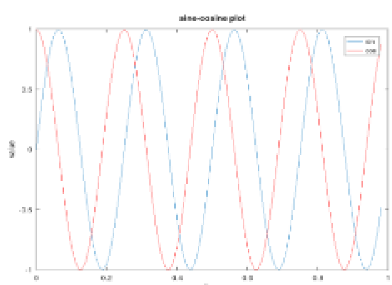
```
octave:13> val = max([1;2;3;6;7])
val =  7
octave:14> max(A)
ans =

   5   6

octave:15> A
A =

   1   2
   3   4
   5   6

octave:16> a = [1;4;6;7;9]
a =

   1
   4
   6
   7
   9

octave:17> a < 3
ans =

  1
  0
  0
  0
  0

octave:18> find(a<3)
ans =  1
octave:19> A = magic(3) % Magic Square
A =

   8   1   6
   3   5   7
   4   9   2

octave:20> [r,c] = find(a >= 7)
r =
```

```
    4
    5

c =

    1
    1

octave:21> a
a =

    1
    4
    6
    7
    9

octave:22> a = a'
a =

    1    4    6    7    9

octave:23> sum(a)
ans =  27
octave:24> rand(3)
ans =

   0.272471   0.059338   0.757392
   0.414497   0.174242   0.354694
   0.811891   0.935437   0.956667

octave:25> A
A =

   8   1   6
   3   5   7
   4   9   2

octave:26> max(A,[],1)
ans =

   8   9   7

octave:27> max(A,[],2)
ans =
```

```
   8
   7
   9

octave:28> max(max(A))
ans =  9
octave:29> A
A =

   8   1   6
   3   5   7
   4   9   2

octave:30> pinv(A)
ans =

   0.147222  -0.144444   0.063889
  -0.061111   0.022222   0.105556
  -0.019444   0.188889  -0.102778

octave:31> temp = pinv(A)
temp =

   0.147222  -0.144444   0.063889
  -0.061111   0.022222   0.105556
  -0.019444   0.188889  -0.102778


octave:32> temp * A
ans =

   1.0000e+00   2.0817e-16  -3.1641e-15
  -6.1062e-15   1.0000e+00   6.2450e-15
   3.0531e-15   4.1633e-17   1.0000e+00

octave:33> % this is the 3x3 Identity matrix,
           % not having exact values beacuse of variable overflow

octave:1> t = [0:0.01:0.98];
octave:2> y1 = sin(2*pi*4*t);
octave:3> plot(t,y1)


octave:4> y2 = cos(2*pi*4*t);
octave:5> plot(t,y1);
```



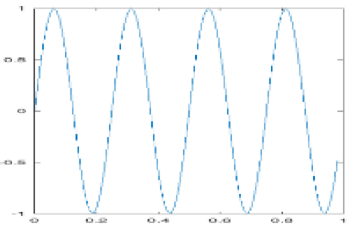
```
octave:6> hold on;
octave:7> plot(t, y2, 'r');
octave:8> xlabel('time')
octave:9> ylabel('value')
octave:10> legend('sin', 'cos')
octave:11> title('sine-cosine plot')
octave:12> print -dpng 'myPlot.png'
```

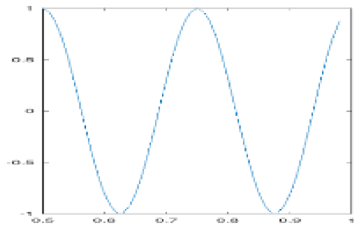


```
octave:13> close
octave:14> figure(1); plot(t, y1);
octave:15> figure(2); plot(t, y2);
octave:16> subplot(1,2,1);   % Divides plot a 1x2 grid
octave:17> plot(t,y1);
octave:18> subplot(1,2,2)
octave:19> plot(t,y2);
octave:20> axis([0.5 1 -1 1])
```

Control Statements: for, while, if, else-if ...

```
octave:1> v = zeros(10,1)
v =

   0
```





```
    0
    0
    0
    0
    0
    0
    0
    0
    0

octave:2> for i=1:10,
> v(i) = 2^i;
> end;

octave:3> v
v =

       2
       4
       8
      16
      32
      64
     128
     256
     512
    1024

octave:4> i=1;
octave:5> while i <= 5,
> v(i) = 100;
> i = i+1;
> end;
octave:6> v
v =
```

```
     100
     100
     100
     100
     100
      64
     128
     256
     512
    1024

octave:7> i = 1;
octave:8> while true,
> v(i) = 999;
> i = i+1;
> if i == 6,
>       break;
> end;
> end;

octave:9> v
v =

     999
     999
     999
     999
     999
      64
     128
     256
     512
    1024
```

Chapter 3

Week 3

3.1 Classification and Represention

3.1.1 Classification

The classification problem is just like the regression problem, except that the values we now want to predict take on onle a small number of discrete values. For now, we'll discuss binary classification problem.

3.1.2 Hypothesis Representation

We may use out old regression algorithm by classifying data on the basis of a threshold. But it will have very poor performance.

We will introduce "Sigmoid Function", also called the "Logistic Function":

$$h_\theta(x) = g(\theta^T x) \tag{3.1}$$

$$z = \theta^T x \tag{3.2}$$

$$g(z) = \frac{1}{1 + e^{-z}} \tag{3.3}$$

This is how the Sigmoid Function looks like:

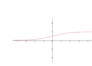


Figure 3.1: Sigmoid Funtion 3.3

3.1.3 Decesion Boundary

The decesion boundary is the line that separates the area where y=0 and where y=1. It is similar to the decesion boundary for linear regression, the only difference is distibution of values (linear and sigmoid)

3.2 Logistic Regression Model

3.2.1 Cost Function

Cost funtion for logistic regression looks like:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) \tag{3.4}$$

$$Cost(h_\theta(x), y) = -\log\left(h_\theta(x)\right) \qquad \text{if } y = 1$$

$$Cost(h_\theta(x), y) = -\log\left(1 - h_\theta(x)\right) \qquad \text{if } y = 0$$

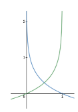


Figure 3.2: Cost Funtion

Siplified Cost Funtion

This cost funtion can be compressed into a single funtion:

$$Cost(h_\theta(x), y) = -y\log\left(h_\theta(x)\right) - (1-y)\log\left(1 - h_\theta(x)\right) \tag{3.5}$$

A vectorised implementation is:

$h = g(X\theta)$
$J(\theta) = \frac{1}{m}.(-y^T \log h - (1-y)^T \log 1 - h)$

Vectorised implementation for Gradient Descent:

$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - y)$

3.3 Multiclass Classification

3.3.1 One-vs-all

This approach is when data has more than two categories.We divide our problem into n[1] binary classification problems, in each one, we predict the probability considering one of the category to be +ve and all other to be −ve. Repeating this for all other categories will finally give us all the decesion boundaries.

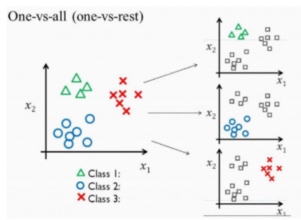


Figure 3.3: One vs all classifiaction method

[1] n = no of categories in dataset