

MATLAB Tutorial

http://www.mathworks.co.kr/help/pdf_doc/matlab/getstart.pdf

Introduction of Machine Learning

2015.10.13 Practice 2

Some tips

Entering Long Statements

If a statement does not fit on one line, use an ellipsis (three periods), . . . , followed by Return or Enter to indicate that the statement continues on the next line.

```
>> result = 1 + 2 + 3 + 4 + 5 +  
6 + 7 + 8 + 9 + 10  
??? result = 1 + 2 + 3 + 4 + 5 +  
|  
Error: Expression or statement is incomplete or incorrect.
```

Multiple Output from a Function

When there are multiple output arguments, enclose them in square brackets

$[maxA, location] = max(A)$

```
>> result = 1 + 2 + 3 + 4 + 5 + ...  
6 + 7 + 8 + 9 + 10  
  
result =  
  
55  
  
>> [maxA location] = max([3 1 5 2 4])  
  
maxA =  
  
5  
  
location =  
  
3
```

Matrix&Array Operators

Matrix Operators

Expressions use familiar arithmetic operators and precedence rules.

+	Addition
-	Subtraction
*	Multiplication
/	Division
\	Left division
^	Power
'	Complex conjugate transpose
()	Specify evaluation order

Array Operators

+	Addition
-	Subtraction
.*	Element-by-element multiplication
./	Element-by-element division
.\	Element-by-element left division
.^	Element-by-element power
.'	Unconjugated array transpose

Matrix Concatenation

Concatenation is the process of joining arrays to make larger ones. In fact, you made your first array by concatenating its individual elements. The pair of square brackets `[]` is the concatenation operator.

```
A =  
    1    4    7   10  
    2    5    8   11  
    3    6    9   12
```

Horizontal concatenation

```
>> [A A]  
  
ans =  
    1    4    7   10    1    4    7   10  
    2    5    8   11    2    5    8   11  
    3    6    9   12    3    6    9   12
```

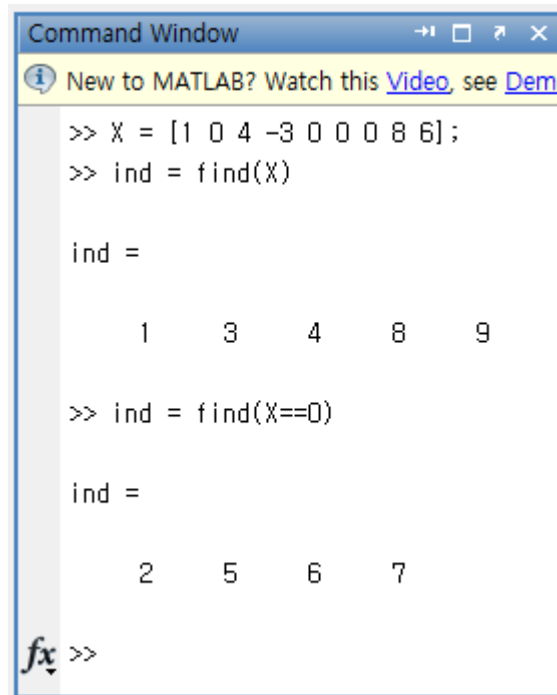
Vertical concatenation

```
>> [A; A]  
  
ans =  
    1    4    7   10  
    2    5    8   11  
    3    6    9   12  
    1    4    7   10  
    2    5    8   11  
    3    6    9   12
```

Matrix functions – find

Find Indices and Values of Nonzero Elements

$ind = find(X)$ locates all nonzero elements of array X , and returns the linear indices of those elements in vector ind .



```
Command Window
New to MATLAB? Watch this Video, see Demo

>> X = [1 0 4 -3 0 0 0 8 6];
>> ind = find(X)

ind =

     1     3     4     8     9

>> ind = find(X==0)

ind =

     2     5     6     7

fx >>
```

Matrix functions – size

Array dimensions

$d = \text{size}(X)$ returns the sizes of each dimension of array X in a vector, d

$X =$

```
0.7655    0.1869    0.4456    0.7094
0.7952    0.4898    0.6463    0.7547
```

$d = \text{size}(X)$

```
>> size(X)
```

```
ans =
```

```
2     4
```

$[m, n] = \text{size}(X)$

```
>> [m, n] = size(X)
```

```
m =
```

```
2
```

```
n =
```

```
4
```

$m = \text{size}(X, \text{dim})$

```
>> size(X, 1)
```

```
ans =
```

```
2
```

```
>> size(X, 2)
```

```
ans =
```

```
4
```

Matrix functions – reshape

Reshape array

- $B = \text{reshape}(A, m, n)$ returns the m-by-n matrix B whose elements are taken column-wise from A.
- $B = \text{reshape}(A, m, [])$ calculates the length of the dimension represented by []

```
A =  
  
     1     4     7    10  
     2     5     8    11  
     3     6     9    12
```

$B = \text{reshape}(A, m, n)$

```
>> B = reshape(A, 2, 6)
```

```
B =
```

```
     1     3     5     7     9    11  
     2     4     6     8    10    12
```

$B = \text{reshape}(A, m, [])$

```
>> B = reshape(A, 2, [])
```

```
B =
```

```
     1     3     5     7     9    11  
     2     4     6     8    10    12
```

Other Matrix Functions

Identity Matrix

- $\text{eye}(m, n)$ returns an m -by- n rectangular identity matrix
- $\text{eye}(n)$ returns an n -by- n square identity matrix

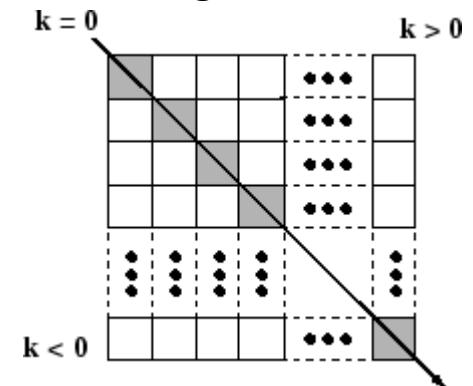
Normalization

- $\text{norm}(X, p)$ returns the p -norm of input X
- $\text{norm}(X)$ returns the 2-norm of input X . If X is a vector, this is equal to the Euclidean distance.

Diagonal Matrix

- $X = \text{diag}(v, k)$ when v is a vector of n components, returns a square matrix X of order $n + \text{abs}(k)$, with the elements of v on the k th diagonal. $k = 0$ represents the main diagonal, $k > 0$ above the main diagonal, and $k < 0$ below the main diagonal.

```
A =  
    1     2     3     4  
    5     6     7     8  
    9    10    11    12  
   13    14    15    16  
  
>> diag(A)      >> diag(A, 1)  >> diag(A, -1)  
  
ans =           ans =           ans =  
    1             2             5  
    6             7            10  
   11            12            15  
   16
```



Other Data Structures - cell

Cell arrays can contain data of varying types and sizes

- cell creation using $c = \text{cell}(m, n)$

```
>> C = cell(2, 2)
```

```
C =
```

```
    []    []  
    []    []
```

- cell creation using $\{\}$

```
>> C = {1 'cell array'; [] {1, 2; 3, 4; 5, 6}}
```

```
C =
```

```
    [1]    'cell array'  
    []    {3x2 cell}
```

- cell indexing c

```
>> C(1, 1)    >> C(1, 2)
```

```
ans =
```

```
    []
```

```
ans =
```

```
    'cell array'
```

- content indexing

```
>> C{1, 1}    >> C{1, 2}
```

```
ans =
```

```
    1
```

```
ans =
```

```
    cell array
```

Other Data Structures - **struct**

Structures are multi-dimensional MATLAB arrays with elements accessed by textual *field designators*.

- Structure creation $s = \text{struct}(\text{field1}, \text{value1}, \dots, \text{fieldN}, \text{valueN})$

```
>> S = struct('name', 'Imelda', 'score', 100, 'grade', 'A')
S =
    name: 'Imelda'
    score: 100
    grade: 'A'
```

```
>> S(2) = struct('name', 'Jane', 'score', 0, 'grade', 'F')
S =
1x2 struct array with fields:
    name
    score
    grade
```

- Access data in a struct array

```
>> s1name = S(1).name
s1name =
Imelda

>> names = {S.name}
names =
    'Imelda'    'Jane'
```

```
>> S(1).score
ans =
    100

>> scores = [S.score]
scores =
    100    0
```

Loops & Conditional Statements

Within a script, you can loop over sections of code and conditionally execute sections using the keywords `for`, `while`, `if`, and `switch`.

for & switch

```
for i=1:10
    switch i
        case 1
            disp('this is 1');
        case {2, 3}
            disp('this is 2~3');
        otherwise
            disp('this is 4~10');
    end
end
```

while & if

```
i = 1;
while i<=10
    if i==1
        disp('this is 1');
    elseif i==2 || i==3
        disp('this is 2~3');
    else
        disp('this is 4~10');
    end
    i = i+1;
end
```

Result:



```
this is 1
this is 2~3
this is 2~3
this is 4~10
this is 4~10
this is 4~10
this is 4~10
this is 4~10
this is 4~10
this is 4~10
```

Loading Data

Load Data

`load (dataFile)` or `load dataFile` Load variables from file into workspace

```
>> load('MNIST_Dataset.mat')
```

Name ▲	Value
 testingData	<1x1 struct>
 trainingData	<1x1 struct>

Missing Data

- *NaN* (Not a Number) value is normally used to represent missing data. *NaN* values allow variables with missing data to maintain their structure
- `isnan(c)` returns a logical vector the same size as `c`, with entries indicating the presence(1) or absence(0) of *NaN* values for each of the elements in the data.

```
>> isnan([1 1 NaN 2 NaN])
```

```
ans =
```

```
0 0 1 0 1
```

Read&Display Image

Read Image from Graphics File

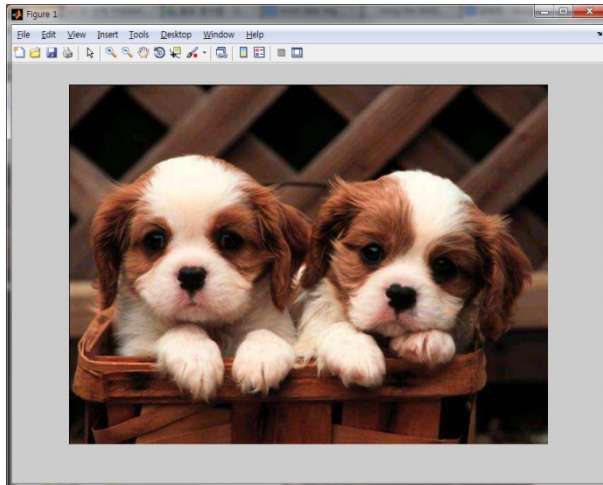
```
>> img = imread('img.jpg');
```

Name	Value	Min	Max
img	<513x684x3 uint8>	<Too many elements>	<Too many elements>

↑
HEIGHT x WIDTH x RGB

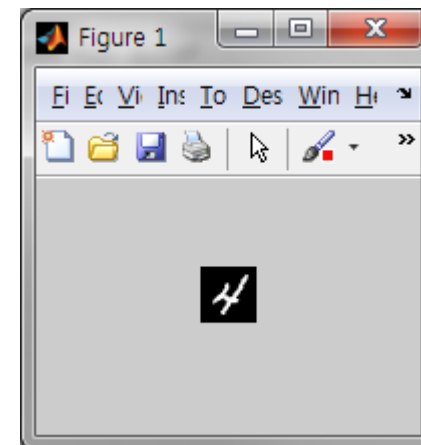
Display Image

```
>> imshow(img)
```



Exercise:

Display 10th MNIST data using load, reshape, imshow, accessing structure



Loops vs. Matrix Multiplication

- Don't use FOR or WHILE so much
- Think whether it can be converted to Matrix Multiplication

```
N = 1000;

A = rand(N,N);
B = rand(N,N);
C = zeros(N,N);
fprintf('C1: ');
tic;
for i = 1:N
    for j = 1:N
        for k = 1:N
            C(i,j) = C(i,j) + A(i,k) * B(k,j);
        end
    end
end
toc;
```

```
fprintf('C2: ');
tic;
C2 = A+B;
toc;
```

```
C1: Elapsed time is 79.169012 seconds.
C2: Elapsed time is 0.108808 seconds.
```

Loops vs. Matrix Multiplication

- Don't use FOR or WHILE so much
- Think whether it can be converted to Matrix Multiplication

```
train_x = trainingData.Images(:,1:1000)';
train_c = trainingData.Labels(1:1000,:);
test_x = testingData.Images(:,1:100)';
test_c = testingData.Labels(1:100,:);
classifier_c = -ones(100,1);

tic;
nearest_idx = zeros(size(test_x,1),1);
for j = 1:size(test_x,1)
    distance = zeros(N,1);
    for i = 1:size(train_x,1)
        for k = 1:size(train_x,2)
            distance(i) = distance(i) + (train_x(i,k)-test_x(j,k))*(train_x(i,k)-test_x(j,k));
        end
    end
    [min_val nearest_idx(j)] = min(distance);
    classifier_c(j) = train_c(nearest_idx(j));
end
toc;
accuracy == sum((test_c - classifier_c) == 0) / size(test_c,1)
```

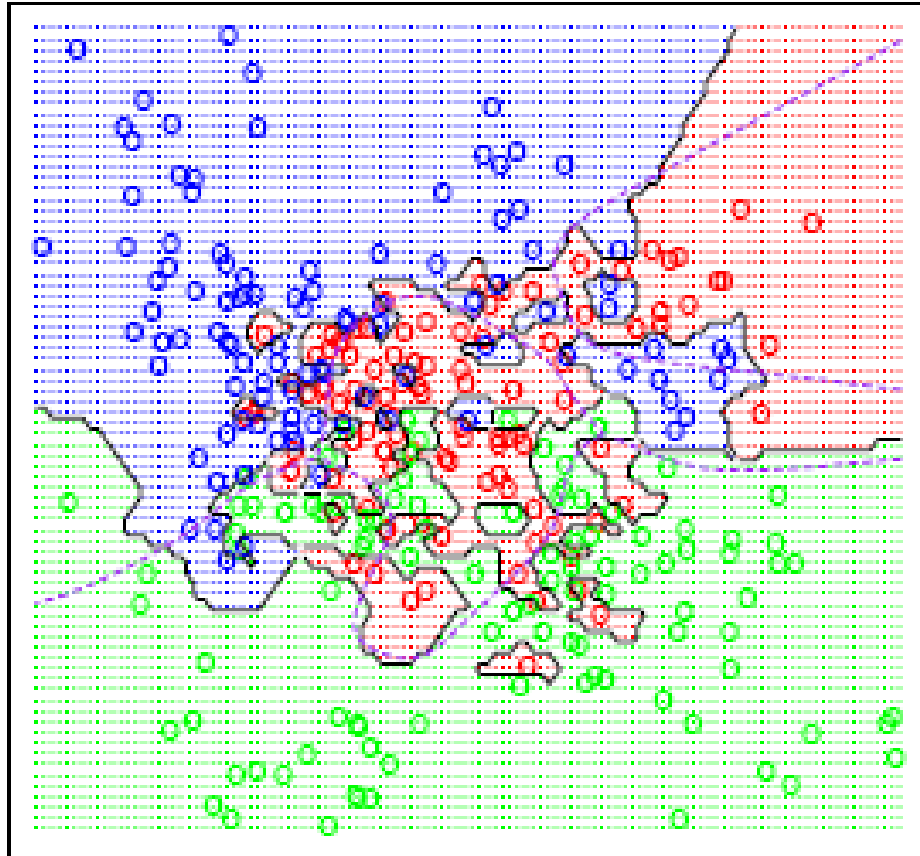
Loops vs. Matrix Multiplication

- **Don't use FOR or WHILE so much**
- **Think whether it can be converted to Matrix Multiplication**

```
train_x = trainingData.Images(:,1:1000)';
train_c = trainingData.Labels(1:1000,:);
test_x = testingData.Images(:,1:100)';
test_c = testingData.Labels(1:100,:);
classifier_c = -ones(100,1);

tic;
nearest_idx = zeros(size(test_x,1),1);
for j = 1:size(test_x,1)
    distance = zeros(size(train_x,1),1);
    for i = 1:size(train_x,1)
        for k = 1:size(train_x,2)
            distance(i) = distance(i) + (train_x(i,k)-test_x(j,k))*(train_x(i,k)-test_x(j,k));
        end
    end
    [min_val nearest_idx(j)] = min(distance);
    classifier_c(j) = train_c(nearest_idx(j));
end
toc;
accuracy = sum((test_c - classifier_c) == 0) / size(test_c,1)
```


K-nearest Neighbor



Multivariate Normal Distribution

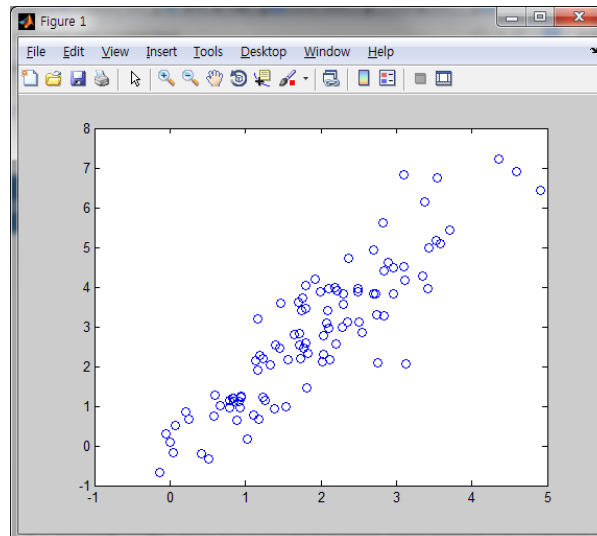
`mvnrnd` - Multivariate Normal Random Numbers

- $R = \text{mvnrnd}(MU, SIGMA)$ returns an n -by- d matrix R of random vectors chosen from the multivariate normal distribution with mean MU , and covariance $SIGMA$. (MU is an n -by- d matrix, $SIGMA$ is a d -by- d symmetric positive semi-definite matrix, or a d -by- d -by- n array.)
- $R = \text{mvnrnd}(MU, SIGMA, cases)$ returns a $cases$ -by- d matrix R of random vectors chosen from the multivariate normal distribution with a common 1-by- d mean vector MU , and a common d -by- d covariance matrix $SIGMA$.

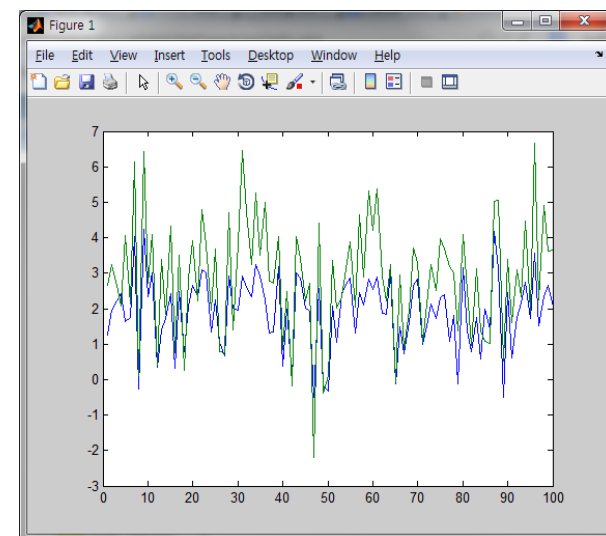
```
>> mu = [2 3];  
>> SIGMA = [1 1.5; 1.5 3];  
>> r = mvnrnd(mu, SIGMA, 100);  
>> plot(r(:, 1), r(:, 2), 'o')
```

↑ ↑ ↙
X축 X축 □ □ □

```
>> plot(r(:, 1), r(:, 2), 'o')
```



```
>> plot(r)
```



Multivariate Normal Distribution

MVNPDF - Multivariate Normal Probability Density Function

- $y = mvnpdf(X)$ returns the n-by-1 vector y , containing probability density of the multivariate normal distribution with zero mean and identity covariance matrix, evaluated at each row of the n-by-d matrix X
- $y = mvnpdf(X, MU)$ returns the density of the multivariate normal distribution with mean μ and identity covariance matrix, evaluated at each row of X .
- $y = mvnpdf(X, MU, SIGMA)$ returns the density of the multivariate normal distribution with mean MU and covariance $SIGMA$, evaluated at each row of X .

```
>> mu = [1 -1];  
>> SIGMA = [.9 .4; .4 .3];  
>> X = mvnrnd(mu,SIGMA,100);  
>> p = mvnpdf(X,mu,SIGMA);  
>> plot(p)|
```

