1 Useful basics in Matlab

Matlab is case sensitive. Everything in Matlab is a matrix. % allows to comment a line. In Command Window, $help\$nameofthefunction$ returns help on that function.

- A scalar can be created in MATLAB as follows:

```
>> x = 23;
```

- Generate vectors

```
>> 1:10 % from 1 to 10

ans =

     1     2     3     4     5     6     7     8
9     10

>> 1:3:15 %from 1 to 15 with step 3

ans =

     1     4     7    10    13
```

- A matrix with only one row is called a row vector. A row vector can be created in MATLAB as follows (note the commas):

```
>> y = [12,10,-3]

y = 12   10  -3
```

- A matrix with only one column is called a column vector. A column vector can be created in MATLAB as follows:

```
>> z = [12;10;-3]

z = 12
    10
    -3
```

- A matrix can be created in MATLAB as follows (note the commas and semicolons)

```
>> X = [1,2,3;4,5,6;7,8,9]

X =
     1     2     3
     4     5     6
     7     8     9
```

- Acces an element from the matrix (indexes start from 1)

```
>> X(1,2) %first row, second column

ans =
     2
```

- Acces a row

```
>> X(1,:)
     1     2     3
```

- Acces a column

```
>> X(:,2)
     2
     5
     8
```

- Acces a portion of the matrix

```
X(1:2,:)

ans =
     1     2     3
     4     5     6
```

or

```
X(:,2:3)

ans =
     2     3
     5     6
     8     9
```

```
X(1:2,2:3) %from first to second row, take elements in column 2 t

ans =
     2     3
     5     6
```

- Access last row or last column

```
>> a(:,end)

ans =
     4
     9

>> a(end, :)
ans =
     8     9
```

- Different way of accessing elements/rows from a matrix

```
a =
     1     2     3
     1     3     4
     2     5     6


>> a([true,false,false],:)

ans =
     1     2     3

>> a([1,3], :)

ans =

     1     2     3
     2     5     6
```

- Access rows for which at a certain column, there is a certain value

```
>> a=[1,2,3;1,3,4;2,5,6]

a =

     1     2     3
     1     3     4
     2     5     6

>> idx = a(:,1)==1

idx =

  3  1  logical array

   1
   1
   0

>> a(idx,:)

ans =

     1     2     3
     1     3     4
```

- Extend matrix

```
X(4,2)=2

X =
     1     2     3
     4     5     6
     7     8     9
     0     2     0
```

Initially the matrix X had dimension $3x3$, and we added an element on the 4th row, therefore a new row was added, with 0 and the introduced value in the mentioned position

- Get the size of an array

  ```
  size(X)
  ```

  But, don't forget that each variable defined in Command Window or in any script, is accesible from Workspace Window.

- Add new column X(:,4)=[10,11,12,13]

  X =

  1 2 3 10 4 5 6 11 7 8 9 12 0 2 0 13

1.1 Operation on matrices

- Matrix multiplication

```
>> a=[1,2;3,4]
a =
     1     2
     3     4

>> b=[1;2]
b =
     1
     2

>> a*b

ans =
     5
    11
```

- Element wise multiplication

```
>> c=[1,2;3,4]

c =

     1     2
     3     4
```

```
>> a.*c

ans =

     1     4
     9    16
```

- Multiple/Divide/ a matrix with a scalar

  ```
  a/2

  ans =

      0.5000    1.0000
      1.5000    2.0000
  ```

  This is similar for all the operators: $+, -, *$

- $+, -$ between elements of two matrices

  ```
  a

  a =
       1     2
       3     4

  >> b

  b =
       1
       2

  >> a-b

  ans =
       0     1
       1     2
  ```

- generate matrices with random numbers Generate random numbers between 0 and 10 for a matrix of 2×3

```
>>randi(10,2,3)

ans =

     3     1     6
     4     8     6
```

Generate values from the uniform distribution on the interval (a, b).

$$r = 2 + (b - a). * rand(100, 1);$$

1.2 Functions

- Find non zero elemnts/elements satisfying a condition in a matrix

  $find(X)$ returns a vector containing the *linear* indices of each nonzero element in array X.

```
>>a=[1,4;8,9]
a =
     1     4
     8     9
>>find(a)
    1
    2
    3
    4
 >> find(a)

ans =

     1
     2
     3
     4

>> find(a==4)
```

```
ans =

     3

>> find(a<9)

ans =

     1
     2
     3
```

Find elements with respect more conditions

```
>> [row,col] = find(a>3 & a<9)

row =
     2
     1


col =
     1
     2
```

There are two elements greater than 3 and smaller than 9: one at (2,1) and the other one at (1,2).

- Mean of an array: $mean(A)$ returns the mean of the elements of A along the first array dimension whose size does not equal 1.

```
>> a

a =
     1     4
     8     9

>> mean(a) %  mean per column
ans =
```

```
    4.5000    6.5000

>> mean(a,'all') % all elements

ans =

    5.5000
```

$mean(A, dim)$ returns the mean along dimension dim. For example, if A is a matrix, then mean(A,2) is a column vector containing the mean of each row.

```
>> mean(a,2)

ans =

    2.5000
    8.5000
```

- sum of elements - similar to mean: $sum(a)$, $sum(a,'all')$, $sum(a, dim)$
- other similar functions: prod, max, mix, median, std, var

1.3 Load/save functions

- *load* $filename$ loads all variables from the file filename
- *load* $filename\ x$ - loads only the variable x from the file
- *load* $filename\ a*$ - loads all variables starting with a
- *save* $filename$ - saves all workspace variables to a binary *.mat* file named filename.mat
- *save* $filename\ x, y$ - saves variables x and y in filename.mat

1.4 Flow control

- if

```
>> if (a<15)
    disp('smaller')
   else
    disp('bigger')
    end

smaller
```

– for

```
for i=1:3
    disp(i)
end
     1

     2

     3
```

Another example

```
>> a=zeros(1,4)

a =
     0     0     0     0
>> for i=1:4
      a(i)=i^3;
   end
>> a

a =

     1     8    27    64
```

1.5 Work with cells

When related pieces of data have different data types, you can keep them together in a **cell array**. Each cell contains a piece of data. To refer to elements of a cell array, use array indexing. You can index into

a cell array using smooth parentheses, (), and into the contents of cells using curly braces, .

```
>> C = {'2017-08-16',[56  67  78]}

  1  2  cell  array

    {'2017-08-16'}      {1  3  double}

>> C(2)

ans =

  1  1  cell  array

    {1  3  double}

>> C{2}

ans =

    56      67      78

>> C{2}(3)

ans =

    78
```

2 Functions for data preprocessing

2.1 Split the dataset into train and test

1. load the data $readTable("orginal_housing.csv")$
2. $N = size(house, 1)$ - number of samples
3. idx = randperm(N); - generate a vector of N distinct numbers - this are going to be the indexes of the selected instances

4. choose the percentage in train/test
5. create the train - $train = house(idx(1 : round(N * 0.8)), :);$
6. create the test $test = house(idx(round(N * 0.8) + 1 : end), :);$

Test the instructions one by one to understand their behaviour

3 Functions for evaluation of models

- Get confusion map $confusionmat(Actual, Predicted)$
- Compute accuracy $acc = sum(YPred == YTest)./numel(YTest)$ Numel(YTest) returns the number of elements in the dataset, while YPred ==YTest returns an array of 0 and 1.
- Compute recall - vezi in credit rating predict
- compute RMSE

$$rmse = sqrt(mean((YPred - YTest)^2)$$

Deep Learning Toolbox $[YPred, scores] = classify(net, X)$

3.1 Example on *housing.csv*

Use regressionLearner and other functions for spliting or evaluation in order to build and evaluate a model for Californian houses.

```
------ run in Command Window
house = readTable("orginal_housing.csv");
  Check in the Workspace window the variable house
N=size(house, 1)
idx = randperm(N)
train = house(idx(1:round(N*0.8)),:);
test = house(idx(round(N*0.8)+1:end),:)
  Start RegressionLearner and choose train in the New Session
  Choose and algorithm (or more) and train
  Export one model - name it houseModel
```

```
  Back in Command Window, apply the model on test:
y_test_predicted = houseModel.predictFcn(test);
  Get the actual y
y = test.median_house_value;
rmse = sqrt(mean((y_test_predicted - y).^2))
```

4 Sequence2Sequence Regression

```
openExample('nnet/SequencetoSequenceRegressionUsingDeepLearningExampl
```

Some further explanation are given here, but read all the comments in the live script.

1. Load the train data

```
>>dataTrain = dlmread("train_FD001.txt"); % similar to readTable,
>> size(dataTrain)

ans =
        20631          26
```

2. Get the number of observations

```
>> numObservations = max(dataTrain(:,1));
100
There are 100 engines for which different length series are known
```

Unit	time in cycles	operational setting1	os2	os3	sensor measurement 1	sm2	... sm 17
1	1	-0.0007	-0.0004	100	518.67	641.82	23.419
1	2	0.0019	-0.0003	100	518.67	642.15	23.4236
..							
1	191	0	-0.0004	100	518.67	643.34	23.1295
2	1	-0.0018	0.0006	100	518.67	641.89	23.4585
...							

3. Create sequences as they are expected by the deep network

   Since the sequences are of different length, the best representation is with cell arrays. For both Xtrain and Ytrain the array will have size 100 (the number of engines).

```
>> XTrain = cell(numObservations,1);
>> YTrain = cell(numObservations,1);
```

(a) Populate the empty cells with values from *dataTrain*. For each engine, get the rows.

Try for the first engine:

```
idx = dataTrain(:,1) == 1;
```

(b) Keep the right columns for X and y.

```
X = dataTrain(idx,3:end)'; %transpose
XTrain{i} = X;
```

```
timeSteps = dataTrain(idx,2)';
Y = fliplr(timeSteps);
YTrain{i} = Y;
```

X will be an array of size 24×192, meaning 24 features, 192 is the length of the sequence. Check in the file that indeed the first engine stops at time 192. The flip is used in order to state the number of cycles still to run.

In the end, XTrain and YTrain is a cell array with dimension 100. In general, each element in X for a sequence must be a matrix with n rows, where n is the number of features, and m columns, where m is the length of the sequence.

	time 0	time 1	...	time m
feature 1	$value_1^0$	$value_1^1$	...	$value - 1^m$
feature 2	$value_2^0$	$value_2^1$	...	$value_2^m$
...				
feature n	$value_n^0$	$value_n^1$	...	$value_n^m$