Machine Learning

Lab 1

Data preprocessing and handling in Matlab

1. Objectives

This laboratory work introduces the basic concepts of handling data in Matlab. It first presents some functions for reading and writing either structured or unstructured data. It then presents a set of methods dealing with numerical data by finding some statistically relevant parameters. A set of data visualization techniques are then shown. Finally, some methods for data rescaling and missing data handling are discussed. Several assignments are then proposed.

2. Theoretical Background

You are given a set of data points corresponding to various features from a population. Each row represents a separate person. We call these samples. Each column presents the accounted values for a sample (eg. Gender, Age, Race, Education etc) – these are called attributes or features. Some of the features are categorical (Race, Education etc) while others are numerical (Income, PoveryLevel etc).

The set of points must be initially read, written to a file. Then, they can be properly visualized and significant statistics can be extracted.

2.1 Read, write functions in Matlab

The data is kept in the *'nhanes_matlab.xlsx'* excel file. There are many functions to import and read the data. The functions for the main 3 data reading options; The first can be used for simpler data structuring (single file), while the second can be used for more complex data. The 3rd option can be used for unstructured data. These are shown below:

```
%Option 2.1.1 read all data at once
data = readtable('nhanes_matlab.xlsx');

%Option 2.1.2 read multiple directories
ds = datastore('nhanes_matlab.xlsx'); % Create datastore
```

```
ds.ReadSize = 50; % Set ReadSize property in ds to 50 so we only read in 50 lines at a time
data50 = read(ds); % Read in first 50 lines
data100 = read(ds); % Read in next 50 lines
data_all = readall(ds); % Read in all data

%Option 3.1.3 Read unstructured data
fid = fopen(filename) %open file - first set a particular filename!
myLine = fgetl(fid) %read data - option 1
myData = textscan(fid, formatSpec) % read data - option 2
fclose(fid); % Close the file
```

Finally, we can write some results this can be done using:

```
write_data = data(1:5,1:5) %select the data you want
filename = 'new_data.xlsx'; %write to excel file
writetable(write_data,filename) %write the data
```

2.2 Statistical analysis of Matlab data

In this part, we highlight some methods to handle numerical data. First of all, we can extract the desired features (columns) from the dataset. This will form a numerical matrix. As the data is stored as a matrix, just selecting the proper column is enough.

```
v = {'id', 'Height', 'Weight'} %Create a vector containing the names of the desired columns
data_useful = data(:,v); %extract the required columns by specifying the names
data_useful = data(:,1:3); %alternative way: extract the required columns by specifying the index
```

A first set of statistically relevant data are the minimum, maximum and their position in the matrix or in the array containing the relevant data. For some of these function we need to convert the data into numerical format.

```
%Minimum
data_Height = min(data_useful); %2.1 minimum value for the numerical matrix
data_Height = data_useful(:,2); %2.1 just Height data
min_Height = min(data_Height) %2.1 minimum value for Height
max_Height = max(data_Height) %2.1 maximum value for Height

%ID of the minimum
data_ID_Height = data_useful(:,1:2); %select the ID and the Height
min_Height_arr = table2array(min(data_ID_Height)) %convert to numerical value – ID + Height
min_Height = min_ Height_arr(2) %extract minimum
id_min_Height = min_Height_arr(1) %extract ID of minimum

data_ID_Height_arr = table2array(data_ID_Height(:,2)) %convert to numerical value
id_Min = find(data_ID_Height_arr==min_Height)
id_found_Min = data_useful(id_Min,1) % ID on column 1
```

Some other statistically relevant information are provided by the mean, median or the mode.

- The mean (average) of a data set is found by adding all numbers in the data set and then dividing by the number of values in the set.
- The median is the middle value when a data set is ordered from least to greatest.
- The mode is the number that occurs most often in a data set.

Another factor that is useful to extract from the data set is the standard deviation. In statistics, the standard deviation is a measure of the amount of variation or dispersion of a set of values. A low standard deviation indicates that the values tend to be close to the mean of the set, while a high standard deviation indicates that the values are spread out over a wider range. Numerically, the standard deviation can be computed using the following formula:

$$\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}}$$

σ = population standard deviation
N = the size of the population
x_i = each value from the population
μ = the population mean

The mean, median, mode and standard deviation can be computed using the following code:

```
mean_Height1 = mean(data_Height) % Must create the height array first
median_Height1 = median(data_Height)
mode_Height1 = mode(data_Height)
stddev_Height1 = std(data_Height)
```

Data is not always perfect. Due to data acquisition process or to human interference, often part of the data can be corrupted, can simply be missing or various operations can lead to non-representable values (eg Infinity). Notice that in this case your data frames/matrices can contain “NaN” values. In such a case the above functions will not properly work. Thus, alternative options that properly deal with non-existing values have been proposed:

```
mean_Height2 = nanmean(data_ID_Height_arr) %mean that also deals with nan values
median_Height2 = nanmedian(data_ID_Height_arr)
stddev_Height2 = nanstd(data_ID_Height_arr)
```

An alternative way to the above is to specify “omit nan” as a parameter to the functions:

```
Mean_Height = mean(data.Height, 'omitnan'); % alternative!
```

2.3 Data visualization in Matlab

Clean and clear data visualization is one of the strong aspects of Matlab. Libraries from other languages such as python (eg. Matplotlib) or R (ggplot) either use or try to resemble the usability and the characteristics of Matlab visualization libraries. We will discuss here several visualization options that will be useful for your future labs.

2.3.1 Simple data plots

We will initially start with simple functions, that plot the values in the array either as an interpolated line or as simple points. We then present some useful plotting techniques.

```
%plot first 40 heights - as line
data_ID_Height_arr = table2array(data_ID_Height)
plot(data_ID_Height_arr(1:40,2))

%plot multiple lines
data_ID_Height_arr = table2array(data_ID_Height)
space = (1:40); % need to establish the number of points
l1 = data_ID_Height_arr(1:40,2)
l2 = data_ID_Height_arr(41:80,2)
plot(space, l1, space, l2)

% When plotting a matrix, each column is a line. Row index is the x-axis variable.
plot(space,l1,'--',space,l2,':')     % '--' will plot y2 as a dashed line, ':' will plot y3 as a dotted line

%More Line Specifications: Line Style, Color, and Marker Line specifications can be written in any
order. ('g--*' = '*g--')
plot(space,l1,'g',space,l2,'c*')     % 'g' plots y1 as a green line,
                                     % 'b--o' plots y2 as a blue dashed line with open circle markers,
                                     % 'c*' plots y3 as cyan asterisk markers with no line

%Subplots:
%create 2 other sets
l3 = data_ID_Height_arr(81:120,2)
l4 = data_ID_Height_arr(121:160,2)
% Create 2 rows and 2 columns of subplots (2x2 = 4 total plots)
```

```
subplot(2,2,1); plot(space,l1) % plots y1 in the first subplot
subplot(2,2,2); plot(space,l2) % plots y2 in the first subplot
subplot(2,2,3); plot(space,l3) % plots y3 in the first subplot
subplot(2,2,4); plot(space,l4) % plots y4 in the first subplot


%handles - change appearance
p = plot(space,l1);
p.Marker = '*';      % add asterisk markers
p.Color = 'black';    % change line color to black
```

2.3.2 More complex data plots

Other customizations - search online - text for axes, labels etc; font size; legend etc etc.

```
%Other visualization tools:
% stem plot - for discrete values!
stem(data_ID_Height_arr(1:200,2))

%stairs graph
stairs(data_ID_Height_arr(1:200,2))

%scatter plots - view relations between multiple variables - can color based on a variable
scatter(data_ID_Height_arr(1:200,2), data_ID_Height_arr(1:200,1))
```

2.3.3 Heatmaps

The next important technique deals with properly plotting values that belong to several classes and we want to see the relative “distance” between points either in-class or inter-class. This type of visualization technique is called **heatmap.** A heat map is a 2-dimensional data visualization technique that represents the magnitude of individual values within a dataset as a color. The variation in color will be correspod to the variation in intensity.

For instance, lets assume we want to see the height values for all types of races. This might be useful when for finding whether correlations between the values in a class appear.

```
%heatmap - can view categories
v2 = {'id', 'Race', 'Height'} %select the “categorical” attribute that separates the data into clases
as well as the “numerical” attribute that provides the color for each class
data_useful2 = data100(:,v2); %select just 100 rows and the desired columns
h = heatmap(data_useful2,'Race','id','ColorVariable','Height');
```

Heatmap loads data in alphabetical order by default. This rearranges it to something more intuitive.

2.3.4 Histogram

Another important concept in data visualization is the **histogram.** A histogram is a graph that shows the frequency of numerical data using rectangles. The height of a rectangle (the vertical axis) represents the distribution frequency of a variable (the amount, or how often that variable appears).

This type of visualization is extremely important for understanding data distribution as well as is for understanding if the data follows a particular pattern or is modelled in a specific way.

To construct a histogram, the first step is to "bin" (or "bucket") the range of values—that is, divide the entire range of values into a series of intervals—and then count how many values fall into each interval. The bins are usually specified as consecutive, non-overlapping intervals of a variable. The bins (intervals) must be adjacent and are often (but not required to be) of equal size.

```
%histogram function
histogram(data_ID_Height_arr(1:2000,2)) %histogram of the heights in the first 2000 samples
histogram(data_ID_Height_arr(1:2000,2),10) % same histogram, using exactly 10 bins
histogram(data_ID_Height_arr(1:2000,2),[100,130,160,190]) %histogram, specifying intervals
```

2.4 Data scaling

Scaling your data in machine learning (ML)is important because many algorithms use distance-based computations/derivations, which are sensitive to the scale of the variables. Data scaling is the process of transforming the values of the features of a dataset untill they are within a specific range, e.g. 0 to 1 or -1 to 1. This is to ensure that no single feature dominates the calculations in an algorithm. This is one of the main tricks that can help to improve the performance of the algorithm.

Some of the scaling methods have been discussed at the lectures. A summary can be seen below:

- Scale in a specific interval: Change the values such that they fit into that particular interval
- Standardization: The mean of each feature becomes 0 and the standard deviation becomes 1.
- Normalization: The values of each feature are between 0 and 1.
- Min-Max Scaling: The minimum value of each feature becomes 0 and the maximum value becomes 1.

In Matlab, data scaling can be done using the function *rescale.*

```
data_ID_Height_scaled = rescale(data_Height) % simple scale

% scale in interval
lower = 0 %set lower boundary
upper = 10 %set upper boundary
```

R = rescale(data_ Height,lower,upper) % rescale into the established margins

2.5 Missing data handling

Missing data occurs when certain observations in a dataset lack values for one or more attributes/features. Addressing this issue is critical because most machine learning algorithms cannot handle missing values directly.

There are various techniques to handle missing data – also called imputation – where missing values are filled in based on statistical methods or inferred from other available data points. However, the choice of imputation method must be made judiciously, as it can influence the outcomes of the analysis.

First of all, in Matlab we need to find the positions of the missing values. This can be done using the code below:

TF = ismissing(data_Height)

You need to then insert the desired value on the missing positions.

3. Activities and tasks

3.1 Run all the Matlab code presented in section 2 for reading/writing and visualizing the data.

3.2 Compute the standard deviation using the formula presented on page 3.

3.3 Generate a heatmap containing samples with/without Diabetes for which you highlight the BMI index.

3.4 Perform a min-max scaling of the data using the formula shown in lecture 2.

3.5 Plot the histogram before and after the scaling in 2 different windows of the same plot (use subplot).

3.5 Perform a data imputation using the following options:

- Mean of the all values
- Median of the first 20 values
- Minimum of the normalized data

3.6 Separate the data into 3 sub-frames called “train”, “test” and “validate”. The first set should contain around 60% of the entire dataset (first 60% of the rows) while the second and the third must each contain around 20% of the data. Check if the difference in **mean** between the train and the validate datasets is smaller than the standard deviation of the test dataset. You can use any numerical feature for computing the means (eg. Height). Plot the histograms for the 3 dataset distributions.

3.7 For the previous problem – find other ways to partition the data. Try to obtain a smaller mean difference.