# IE 582 Statistical Learning for Data Mining

## Homework 2, due November 1$^{st}$, 2019

Instructions: Please solve the following exercises using R (http://www.r-project.org/) or Python (https://www.python.org/). You are expected to use GitHub Classroom and present your work as an html file (i.e. web page) on your progress journals. There are alternative ways to generate an html page for you work:

- A Jupyter Notebook including your codes and comments. This works for R and Python, to enable using R scripts in notebooks, please check:
  - https://docs.anaconda.com/anaconda/navigator/tutorials/r-lang/
  - https://medium.com/@kyleake/how-to-install-r-in-jupyter-with-irkernel-in-3-steps-917519326e41

  Things are little easier if you install Anaconda (https://www.anaconda.com/). Please export your work to an html file. Please provide your *. ipynb file in your repository and a link to this file in your html report will help us a lot.

- A Markdown html document. This can be created using RMarkdown for R and Python-Markdown for Python

Note that html pages are just to describe how you approach to the exercises in the homework. They should include your codes. You are also required to provide your R/Python codes separately in the repository so that anybody can run it with minimal change in the code. This can be presented as the script file itself or your notebook file (the one with *.ipynb file extension).

The last and the most important thing to mention is that academic integrity is expected! Do not share your code (except the one in your progress journals). You are always free to discuss about tasks but your work must be implemented by yourself. As a fundamental principle for any educational institution, academic integrity is highly valued and seriously regarded at Boğaziçi University.

### Tasks

#### ***Task 1: Multiple Instance Learning***

Wikipedia definition: In machine learning, multiple-instance learning (MIL) is a variation on supervised learning. Instead of receiving a set of instances which are individually labeled, the learner receives a set of labeled bags, each containing many instances. In the simple case of multiple-instance binary classification, a bag may be labeled negative if all the instances in it are negative. On the other hand, a bag is labeled positive if there is at least one instance in it which is positive. From a collection of labeled bags, the learner tries to either (i) induce a concept that will label individual instances correctly or (ii) learn how to label bags without inducing the concept.

In other words, multiple instance learning problems differ from regular learning problems. In traditional classification tasks, each object is represented with a feature vector and the aim is to predict the label of the object given some training data. However this modest approach becomes weak when the data has a certain structure. For example, in image classification, images are segmented into patches and instead of a single feature vector, each image is represented by a set of feature vectors derived from the patches. This type of applications fits well to Multiple Instance Learning (MIL) setting where each object is referred to as bag and each bag contains certain number of instances.

In this exercise, you are given a dataset, namely Musk1, dataset description is available on https://archive.ics.uci.edu/ml/datasets/Musk+(Version+1) and it is also uploaded to Moodle. Please use the uploaded version. The structure of the file is as follows:

| Bag class | Bag Id | Feature 1 | Feature 2 | … | Feature *p* |
|---|---|---|---|---|---|
| 1 | 1 | | | | |
| 1 | 1 | | | | |
| 1 | 1 | | | | |
| 1 | 1 | | | | |
| 0 | 2 | | | | |
| 0 | 2 | | | | |
| … | … | … | … | … | … |
| 1 | N | | | | |
| 1 | N | | | | |
| 1 | N | | | | |

For example, there are *N* bags and *p* features in the dataset illustrated above. First bag (The bag with id 1) is from first class and has 4 instances. Similarly second bag (the bag with id 2) has 2 instances and the bag is from class 0. The aim of multiple instance learning is to classify a bag given its instance characteristics. A typical example is smell classification.

From dataset definition: Musk1 describes a set of 92 molecules of which 47 are judged by human experts to be musks and the remaining 45 molecules are judged to be non-musks. The goal is to learn to predict whether new molecules will be musks or non-musks. However, the 166 features that describe these molecules depend upon the exact shape, or conformation, of the molecule. Because bonds can rotate, a single molecule can adopt many different shapes. To generate this data set, the low-energy conformations of the molecules were generated and then filtered to remove highly similar conformations. This left 476 conformations. Then, a feature vector was extracted that describes each conformation.

Normally, instance labels are not known in multiple instance learning. We only know the labels of bags. The structure of the dataset assumes that each instance has the same label as its bag (not a realistic assumption in the context of multiple instance learning). Suppose we are interested in understanding the data (since we have not covered any supervised learning approach for now). We can use dimensionality reduction techniques to do so.

**a)** Use PCA and MDS to understand the structure of the data. Report relevant measures and comment on how good is the assumption of the instance labels' being as the same as bag labels. You can plot reduced dimensions on a figure in which instances are color-coded with their bag labels.

**b)** Most of the approaches in MIL literature aim at summarizing the instance level information to bag level information (i.e. there are 4 instances in first bag, how can I represent the first bag as a single feature vector). One easy way to represent the bag is to take the average of the instance features so that the bag is represented by the center of its instances. If I represent all bags in the same manner, the problem can be considered as a regular learning problem (i.e. I have *N* data points with *p* features). Figure 1 illustrates the idea. Use PCA and MDS to see if this is a good transformation approach for classifying bags.

**c)** (Bonus) Can you think of any other way to represent each bag with a vector? In other words, can you summarize a bag into a single feature vector (of size *r* where *r* can be any positive number) using the instance-based information (4 instance of size *p* as in Figure 1 below)?

| Bag Id | Feature 1 | Feature 2 | … | Feature *p* |
|---|---|---|---|---|
| 1 | | | | |
| 1 | | | | |
| 1 | | | | |
| 1 | | | | |

| Bag class | Bag Id | Feature 1 | Feature 2 | … | Feature *r* |
|---|---|---|---|---|---|
| 1 | 1 | | | | |

**Figure 1.** Summarizing instance level information to bag level information. Part (b) proposes to take the column means as the summary.

***Task 2***

In this task, you are requested to perform certain operations on images. The aim is to compress images using PCA.

Here is a background information about how a grayscale image is represented on our computers. A grayscale image is basically a matrix where each matrix entry shows the intensity (brightness) level. In other words, when you take a picture with a digital camera, the image is represented by a numerical matrix where the matrix size is defined by the resolution setting of your camera. If your resolution setting is 1280x720, then your image is represented by 1280x720= 921600 pixel values (Actually that is why higher resolution provides better quality pictures).

When you have a color image, the image stores the information of multiple channels depending on the image type. The most famous one is RGB type where R, G and B stand for “red”, “green” and “blue” respectively. Hence, you have a matrix as in greyscale images representing the intensity for each channel. Combining these matrices generates the color image.

Below is the steps you need to follow for this exercise:

- Take a picture of yours (or some object of your choice) and save it as **.jpg* or **.jpeg* file.
- Resize the image to size 256x256 px (pixel) using an image editor (i.e. *Paint* in Windows). This image is the one that you will use for this exercise.

**1-** Read the image and display it.

**2-** In order to create a noisy image, add a random noise from uniform distribution between min pixel value and 0.1 times the maximum pixel value to each pixel for each channel of original image.
  **a.** Display the new image.
  **b.** Display each channel separately using “image” function on a single plot.

**3-** Transform your noisy image to a greyscale one using either R or an image editor (both is fine). In signal processing, it is often desirable to be able to perform some kind of noise reduction on an image or signal (note that image can be considered as a 2D signal). Suppose we aim at reducing the size of the image with minimal loss (this reduction may or may not help in noise reduction). What we can do in order to perform such a reduction is to apply PCA to patches extracted from the images. How this is achieved is to visit each pixel by creating a window of certain size (which includes the neighboring entries). For 2D signals such as images, complex window patterns are possible (such as "box" or "cross" patterns) but think of a patch as a box around a certain pixel value as illustrated in Figure 1.

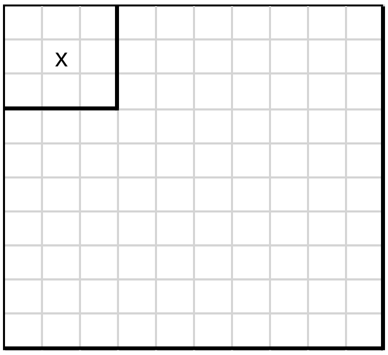

**Figure 2.** Sample patch from an image

Figure 2 is a sample image represented by 10 by 10 matrix. A 3 by 3 patch extracted for pixel at (2,2) is also shown. Suppose we extract all possible patches by sliding one pixel over the image and represent each patch as a vector of length 9. In other words, each patch becomes an instance. For this particular example you will have 8x8=64 patches. Suppose you decided to use 25 by 25 patch size and extract the patches from your original image. A patch will be represented by a feature vector of length 625. Generate all patches accordingly and perform the following tasks.

a) Apply PCA to this data matrix and comment on PCA results.
b) Let's say we decide to use the first component to reconstruct the image (i.e. use the mapping on the first component as the pixel value for the patch). Plot the scores (i.e. mapping) for the first component as an image (recall that each patch is extracted from a certain location so that you can obtain an image from the scores). Do the same for the second and third components.
c) Components (eigenvectors) themselves refers to a patch. Plot the first component as 25 by 25 image. Do the same for the second and third components and comment.