IMPERIAL

Department of Mathematics
MATH60026 - MATH70026
Methods for Data Science
Years 3/4/5

Coursework 2 Part 1 - Deadline: Wednesday, 19 March 2025 at 1pm.

General instructions - please read carefully

The goal of this coursework is to analyse a dataset using tools and algorithms introduced in the lectures, which you have also studied in detail through the weekly Python notebooks containing the computational tasks. You will solve the tasks in this coursework using Python.

You are allowed to use: Python code that you have developed in your coding tasks, including all the following Python packages that have been used there:

- `numpy, pandas, matplotlib, seaborn;`
- `collections, typing, copy, heapq, tqdm;`
- `Torch`, but only to build, train and evaluate the performance of CNNs (Task 1 below).

You are *not* allowed to use:

- `scikit-learn, statsmodels, JAX, Tensorflow/Keras, NetworkX`, or any other model-level, automatic differentiation or network analysis Python package (please ask if you are unsure about a package);
- ready-made code found anywhere online;
- conversational AI tools such as ChatGPT, Microsoft Bing, GitHub Copilot etc. to generate code and written answers.

Needless to say, your submission must be your own individual work: *You may discuss the analysis with your colleagues but code, written answers, figures and analysis must be written independently on your own. The Department uses code profiling and tools such as Turnitin to check for plagiarism of any sort.* ***Plagiarism is a major form of academic misconduct.***

Marks

Coursework 2 (CW2) is worth 60% of your total mark for the course.
The total marks for CW2 are 100: CW2 Part 1 contains tasks for 30 marks, CW2 Part 2 contains tasks for 70 marks. **Mastery component: the mastery component amounts to 20% of CW2. Task 1** has one version for 3rd year BSc students only and one version for MSc and 4th year MSci students only (Mastery component) - pay attention to this and choose the right option in this task! You should **only** answer the version that applies to your case.

For general guidance about writing your solutions and marking scheme, please read the document "Guidelines about writing your solutions" in the folder "Examples of Previous Coursework & Guidelines".

Submission

Files: For the submission of your coursework, you need to save **two documents**:

- a **Jupyter notebook (file format: ipynb)** with all your tasks clearly labelled. You should use the template notebook called *CID_Coursework2_Part1.ipynb* provided on Blackboard (folder 'Coursework/Coursework 2'). The notebook should have clear headings to indicate the answers to each question, *e.g.* 'Task 1.1'.

  The notebook should contain the cells with your code and their output, plus some brief text explaining your calculations, choices, mathematical reasoning, and discussion of results. (Important: Before submitting you

must restart the kernel and run the notebook, with all cells executed sequentially and the outputs of the cells must be printed). You can use Google Colab or create your Jupyter notebook in the local Python environment installed on your computer.

- Once you have executed all cells in your notebook and their outputs are printed, save the notebook as an **html file** (with name *CID_Coursework2_Part1.html*). Your ipynb file must produce all the output that appears in your html file, *i.e.*, make sure you have run all cells in the notebook before exporting the html.

  Trouble-shooting for conversion to HTML if you use Google Colab: as a default, Colab does not let you download your notebook as html. Please try one of the following: 1) press ctrl+S; 2) follow the easy instructions here; 3) simply download your notebook through `File > Download .ipynb` and use the standard tool for Jupyter Notebook conversion `nbconvert`.

The submission of your coursework must consist of **two items,** uploaded **separately**:

1) *A **single zip folder** containing your Jupyter notebook as an **ipynb file** and your notebook exported as an **html file**. Name your zip folder 'CID_Coursework2_Part1.zip', where CID is your student CID, e.g. 123456_Coursework2_Part1.zip.*
2) *The html file, named CID_Coursework2_Part1.html, which will be used for the plagiarism check.*

The submission should consist only of these 2 items - **Do not submit multiple files. Do not put your name on the files you submit** (only the CID), because the marking must be carried out preserving your anonymity.

Drop Boxes: The submission is done **online via Blackboard,** using the drop boxes inside the folder 'Coursework' on Blackboard.

- If you are a 3rd year BSc student: use the 'Coursework Drop Boxes' inside the folder 'Coursework'. The html file must be uploaded to 'Coursework 2/Part 1/Coursework 2 Part 1 Drop Box Spring 25 - HTML', the zip folder must be uploaded to 'Coursework 2/Part 1/Coursework 2 Part 1 Drop Box Spring 25 - ZIP File'.

- If you are a 4th year MSci or MSc student: use the 'Mastery Coursework Drop Boxes' inside the folder 'Coursework'. The html file must be uploaded to 'Mastery Coursework 2/Part 1 Mastery/Coursework 2 Part 1 Mastery Drop Box Spring 25 - HTML', the zip file to 'Mastery Coursework 2/Part 1 Mastery/Coursework 2 Part 1 Mastery Drop Box Spring 25 - ZIP File'.

Make sure you submit to the right drop box on Blackboard - any mistake in the submission folder will cause a delay in the release of your mark.

Recommendations about online submissions:

- *There are known issues with particular browsers (or settings with cookies or popup blockers) when submitting to Turnitin. If the submission 'hangs', please try another browser.*
- *You should **check that your files are not empty or corrupted after submission - check the HTML file!***
- ***To avoid last minute problems** with your online submission, we recommend that you **upload versions of your coursework early on**, before the deadline. You will be able to update your coursework until the deadline, but having this early version provides you with some safety backup. For the same reason, keep **backups of your work**, e.g. save regularly your notebook with its outputs as an .html file, which can be useful if something unpredicted happens just before the deadline.*
- *If you have any issue with the submission, or you realise you have submitted your work to the wrong drop box, please contact directly the UGMathsOffice at maths-student-office@imperial.ac.uk or your MSc programme administrator, in such a way that they can help you solve the issue.*
- *If you need an extension, or happen for any reason to submit your work late, please make a request for mitigating circumstances directly on ZINC.*

  *For these last two points, **do not contact us - we, as lecturers, are not able to grant extensions nor to make changes in the submission folders! We only get to see anonymised submissions.***

Coursework 2 - Part 1 (30 marks)

In this coursework, you will work with a dataset on **gene expression**, which is made available inside the folder 'Coursework/Coursework 2/Data' on Blackboard. You will perform learning tasks on this same dataset both in Part 1 and Part 2.

Background: Your dataset for CW2 is a single-cell transcriptomic dataset containing the level of expression for different genes for individual cells, i.e., the abundance of RNA that has been transcribed from the corresponding gene inside each cell. Gene expression data are crucial in biology to monitor how much certain sets of genes are used by a cell for protein production in specific conditions. This is one way you can think about it: genes here are the data features, and their expression levels give a high-dimensional representation of the cellular state in specific conditions, e.g. during development, or in response to a treatment.

Dataset: This dataset contains gene expression levels for 101 genes in different cells (our samples) from mouse tissues, where some of the cells were subject to a drug treatment while the others were not. Gene expression levels are reported in terms of RNA molecular concentration in $\log_2$ scale.

The dataset is available as `gene_expression_transcriptomic_data.csv` and contains $N = 2641$ samples (cells). The expression level of gene MAP7, a gene involved in intracellular transport whose abnormal expression is observed in several cancers, will be your *target variable* for regression (see column `Target: MAP7`). **You will use as data features the expression levels of the other 100 genes**, which potentially co-regulate MAP7, see the dataset columns marked by feature number and corresponding gene name, from `Feature 1: FOXO3` to `Feature 100: KLHL29`. The column `Treatment` contains binary classification labels $(0, 1)$ indicating whether the cell was subject to a drug treatment or not.

Important: Do **not** standardise the dataset in any of the tasks.

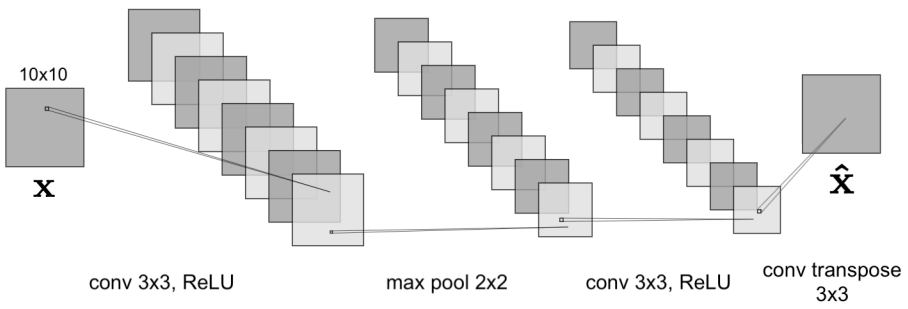
Task 1: Dimensionality Reduction and Regression with Convolutional Neural Networks - CNNs (20 marks)

In this task you will train a Convolutional Neural Network (CNN) to perform dimensionality reduction and to predict the regression target variable contained in the column `Target: MAP7`. You need to use PyTorch as in your coding task of Week 6 notebook. First, use `numpy.reshape` to reshape each sample from an array of 100 features to a 2-dimensional matrix of 10×10 pixels. Second, split the $N = 2641$ samples in three parts: the first $N^{\text{train}} = 1848$ samples to use as training set, the next $N^{\text{val}} = 264$ to use as validation set, the final $N^{\text{test}} = 529$ samples to use as test set. Do **not** reshuffle the samples in the dataset before splitting it.

For this entire Task 1 there is one version for BSc/3rd-year students and one version for MSc/4th-year students: make sure you complete the task version that applies to you.

Task 1, BSc/3rd-year students only.

1.1 (10 marks) - CNN autoencoder. Using PyTorch, build a CNN autoencoder architecture that takes as input a sample $\mathbf{x}$ (reshaped into a 10×10 matrix), encodes it in a lower-dimensional representation and from this reconstructs an approximation of $\mathbf{x}$, denoted $\hat{\mathbf{x}}$. Follow the following scheme to build the CNN autoencoder:



In words, the CNN autoencoder should be composed of:

1. An *encoder*, which takes $\mathbf{x}$ as input and contains:
   - A 2-dimensional convolutional layer with 8 filters with kernel shape (3, 3), a ReLU activation function, and padding $= 1$;
   - A max-pooling layer with window shape (2, 2) and stride $= 2$;
   - Another 2-dimensional convolutional layer, identical to the first one.
2. A *decoder*, which takes the encoder's output as input and contains:
   - A 2-dimensional transposed convolutional layer (see the relevant PyTorch documentation) with 1 filter with kernel shape (3, 3), a ReLU activation function, padding $= 1$ and stride $= 2$. Add the `output_padding` option, adjusting its value in such a way that the decoder's output has the dimensions expected for $\hat{\mathbf{x}}$.

Remark: Leave all the optional arguments of PyTorch functions that we have not specified to their default value.

Define the loss function to train the CNN autoencoder as the Mean Squared Error of data reconstruction $\text{MSE}_{\text{reconstruct}}$:

$$\text{MSE}_{\text{reconstruct}} = \frac{1}{N^{\text{train}}} \sum_{i=1}^{N^{\text{train}}} ||\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}||^2 \quad \text{where} \quad ||\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}||^2 = \sum_{p=1}^{100} \left(x_p^{(i)} - \hat{x}_p^{(i)} \right)^2$$

where $\mathbf{x}^{(i)}$ is a sample in the training set, $\hat{\mathbf{x}}^{(i)}$ is its reconstruction by the autoencoder and $p = 1, .., 100$ is the data feature index. As your optimisation algorithm for training, use `Adam` with learning rate equal to 10^{-3}. Run the training for a maximum of 100 epochs with batches of size 32. Include early stopping during the training, using the $\text{MSE}_{\text{reconstruct}}$ loss as your metric of performance and setting the `max_patience` parameter to 5 epochs.

Monitor the performance during training by computing and printing $\text{MSE}_{\text{reconstruct}}$ on the training and validation sets at regular intervals. Plot $\text{MSE}_{\text{reconstruct}}$ as a function of the epochs both for the training and validation set, marking the epoch at which early stopping was triggered.

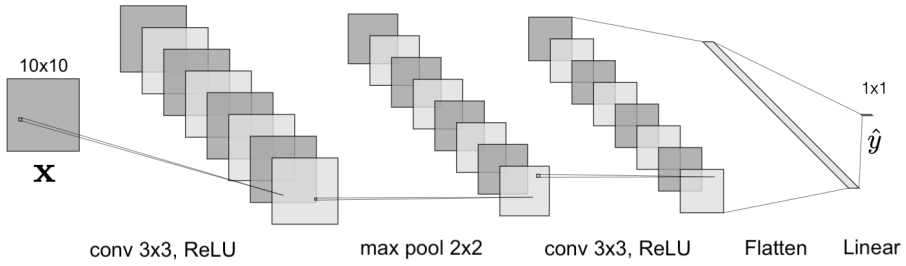
Compute the per-feature reconstruction error *on the test set* :

$$\text{MSE}^p_{\text{reconstruct}} = \frac{1}{N^{\text{test}}} \sum_{i=1}^{N^{\text{test}}} \left(x_p^{(i)} - \hat{x}_p^{(i)} \right)^2$$

Visualise the test-set per-feature reconstruction error $\text{MSE}^p_{\text{reconstruct}}$ as a 10×10 heatmap. Discuss and justify the pattern that you observe.

1.2 (10 marks) - Re-purposing the encoder's output for regression. The output of the encoder from **1.1** provides a lower-dimensional representation of the original dataset. First, compute its dimensions, explaining the steps of your reasoning. Visualise these lower-dimensional representations for the data points in the test set, by plotting the elements with indices $[1, 0, 1]$ against the ones with indices $[0, 0, 0]$ (following Python's indexing) and colouring each point with the corresponding value of the target variable MAP7. Discuss your result.

Next, use the encoder's lower-dimensional representation to build a regressor, which takes the encoder's output as input and predicts the target regression variable MAP7 according to the following scheme:



In words, the regressor consists of:
- A layer flattening the encoder's output [hint: you can use `numpy.reshape` here];
- A linear layer with one output unit (the regression variable).

Train the regressor's layers (i.e., keep the encoder frozen) optimising the MSE of the regression task given by:

$$\text{MSE}_{\text{regress}} = \frac{1}{N^{\text{train}}} \sum_{i=1}^{N^{\text{train}}} \left(y^{(i)} - \hat{y}^{(i)} \right)^2$$

where $y^{(i)}$ is the value of target variable `MAP7` for sample $\mathbf{x}^{(i)}$ in the training dataset, while $\hat{y}^{(i)}$ is the corresponding value predicted by the regressor.

Like in **1.1**, use `Adam` with learning rate equal to 10^{-3} as your optimisation algorithm for training. Run the training for a maximum of 1000 epochs with batches of size 32. Include early stopping during the training, monitoring the $\text{MSE}_{\text{regress}}$ loss as your metric of performance and setting the `max_patience` parameter to 10 epochs. Evaluate the performance of the regressor on the test set calculating the R^2 score and discuss your results.

Task 1, MSc/4th-year students only.

1.1 (10 marks) - U-Net CNN architecture. In this subtask, you will train a CNN architecture called U-Net, first introduced in this article, and you will use it for dimensionality reduction. This type of architecture is composed of a *contracting path* and a symmetric *expanding path*, leading to a U-shaped architecture, see Figure 1: follow the scheme in this figure to build your U-Net CNN architecture using PyTorch.

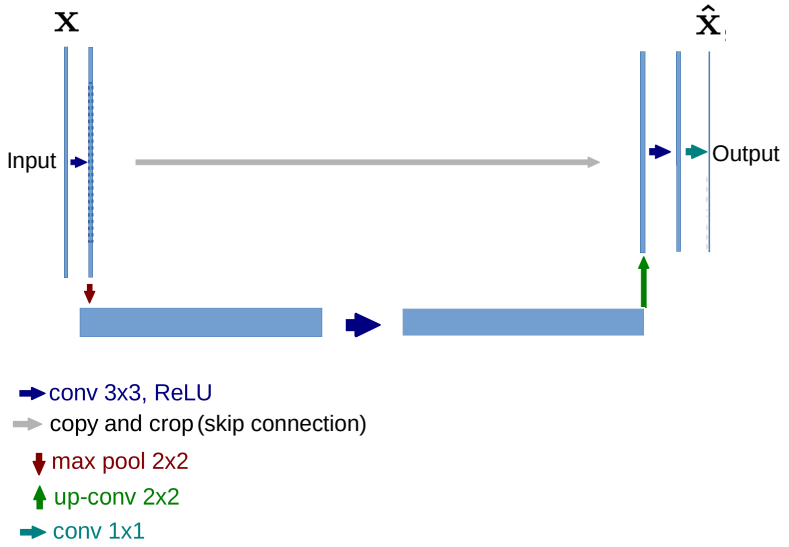


Figure 1: The U-Net CNN architecture for this task: the figure is appropriately adapted from Figure 1 of the original U-Net article to illustrate the architecture that you need to implement here.

In words, your U-Net CNN architecture takes as input $\mathbf{x}$ (reshaped into a 10×10 matrix), encodes it in a lower-dimensional representation and from this reconstructs an approximation of $\mathbf{x}$, denoted $\hat{\mathbf{x}}$. Specifically, the U-Net should be constructed as follows:

1. The contracting path (left side of Figure 1) takes $\mathbf{x}$ as input and contains typical elements of a convolutional layer, i.e.:
   - A 2-dimensional convolutional layer with 8 filters with kernel shape (3, 3), a ReLU activation function, padding $= 1$;
   - A layer implementing max-pooling with a pooling window shape of (2, 2) and stride $= 2$;
   - Another 2-dimensional convolutional layer with 16 filters with kernel shape (3, 3), a ReLU activation function, padding $= 1$ (the output of this layer is the lower-dimensional representation).

2. The expanding path (right side of Figure 1) takes the contracting path's output as input, and it performs upsampling by replacing pooling operators by upsampling operators, followed by convolutions ('up-convolutions'). Specifically, it contains:
   - For upsampling, a 2-dimensional transposed convolutional layer (see the relevant PyTorch documentation) with 8 filters of kernel shape (3, 3), a ReLU activation function, padding $= 1$ and stride $= 2$ (this is

the 'up-convolution');
- A skip connection: the 'copy and crop' arrow in Figure 1 denotes what is now called 'skip connection', whereby the input to the 2×2 max-pooling layer (red arrow) and the output of the up-convolution (green arrow) are concatenated along `dim=1`;
- A 2-dimensional convolutional layer with 8 filters with kernel shape (3, 3), a ReLU activation function, padding $= 1$, acting on the output of the skip connection;
- A final 2-dimensional convolutional layer with 1 filter with kernel shape (1,1), giving the output $\hat{\mathbf{x}}$.

Add the `output_padding` option to the up-convolution above, adjusting its value in such a way that the output of your U-Net architecture has the dimensions expected for the data reconstruction $\hat{\mathbf{x}}$.

Remarks: 1) Leave all the optional arguments of PyTorch functions that we have not specified to their default value. 2) You are expected to use the original U-Net article to fill in the details on the general construction of a U-Net architecture.

Define the loss function to train the U-Net CNN architecture as the Mean Squared Error of the data reconstruction $\mathrm{MSE}_{\mathrm{reconstruct}}$:

$$\mathrm{MSE}_{\mathrm{reconstruct}} = \frac{1}{N^{\mathsf{train}}} \sum_{i=1}^{N^{\mathsf{train}}} ||\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}||^2 \quad \text{where} \quad ||\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}||^2 = \sum_{p=1}^{100} \left(x_p^{(i)} - \hat{x}_p^{(i)} \right)^2$$

where $\mathbf{x}^{(i)}$ is a sample in the training set, $\hat{\mathbf{x}}^{(i)}$ is its reconstruction by the U-Net and $p = 1, .., 100$ is the data feature index.

As your optimisation algorithm for training, use `Adam` with learning rate equal to 10^{-3}. Run the training for a maximum of 100 epochs with batches of size 32. Include early stopping during the training, using $\mathrm{MSE}_{\mathrm{reconstruct}}$ as your metric of performance and setting the `max_patience` parameter to 5 epochs.

Monitor the performance during training by computing and printing $\mathrm{MSE}_{\mathrm{reconstruct}}$ on the training and validation sets at regular intervals. Plot $\mathrm{MSE}_{\mathrm{reconstruct}}$ as a function of the epochs both for the training and validation set, marking the epoch at which early stopping was triggered.

Compute on the test set the per-feature reconstruction error:

$$\mathrm{MSE}^p_{\mathrm{reconstruct}} = \frac{1}{N^{\mathsf{test}}} \sum_{i=1}^{N^{\mathsf{test}}} \left(x_p^{(i)} - \hat{x}_p^{(i)} \right)^2$$

Visualise the test-set per-feature reconstruction error $\mathrm{MSE}^p_{\mathrm{reconstruct}}$ as a 10×10 heatmap. Discuss and justify the pattern that you observe.

1.2 (10 marks) - Multi-task learning. In this subtask you will implement multi-task learning within your U-Net CNN architecture. The basic idea of multi-task learning is to train an architecture in such a way as to achieve multiple prediction tasks at the same time; you can read more about it in Section 7.7 of Goodfellow, Bengio, Courville *Deep learning* (2016). Here, you will implement multi-task learning to perform dimensionality reduction (as in **1.1**) *and* regression against the target variable MAP7.

To this end, add a flatten layer to the last layer of the contracting path of your U-Net architecture from **1.1**, followed by a linear layer connecting the flatten layer to one output unit standing for the predicted value of MAP7, as schematically represented in Figure 2.

Train this U-Net CNN architecture by minimising the multi-task loss function $\mathrm{L}_{\mathrm{multi}}$ given by:

$$\mathrm{L}_{\mathrm{multi}} = \mathrm{MSE}_{\mathrm{regress}} + \alpha \, \mathrm{MSE}_{\mathrm{reconstruct}}$$

Here, α is a hyperparameter, $\mathrm{MSE}_{\mathrm{reconstruct}}$ is the same as defined in **1.1**, while $\mathrm{MSE}_{\mathrm{regress}}$ is defined as:

$$\mathrm{MSE}_{\mathrm{regress}} = \frac{1}{N^{\mathsf{train}}} \sum_{i=1}^{N^{\mathsf{train}}} \left(y^{(i)} - \hat{y}^{(i)} \right)^2$$

where $y^{(i)}$ is the value of target variable MAP7 for sample $\mathbf{x}^{(i)}$ in the training dataset and $\hat{y}^{(i)}$ is the corresponding predicted value. Provide an interpretation of the multi-task loss function $\mathrm{L}_{\mathrm{multi}}$.

Like in **1.1**, use `Adam` with learning rate equal to 10^{-3} as your optimisation algorithm for training. Run the training for a maximum of 1000 epochs with batches of size 32. Include early stopping during the training, monitoring the loss $\mathrm{L}_{\mathrm{multi}}$ as your metric of performance and setting the `max_patience` parameter to 10 epochs.

Search for an optimal value for the hyperparameter α among the set of values $\{0.1, 1, 10\}$ by cross-validation, using the validation set of size N^{val} you have extracted at the start and using $\mathrm{MSE}_{\mathrm{regress}}$ as your

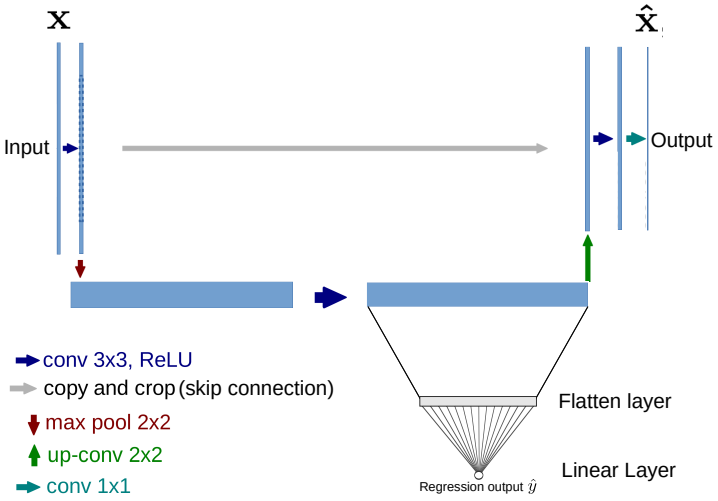


Figure 2: The U-Net CNN architecture for this task.

metric of performance. Implement early-stopping for each value of α scanned. Re-train the final architecture with the optimal α (and number of epochs found using early stopping).

Evaluate the regression performance of the final architecture on the test set, calculating the R^2 score of regression. Discuss your results.

Task 2: Graph-based learning (10 marks)

In this task, you will construct a graph between samples to gain insight into the data structure and, in particular, the effect of treatment on gene expression via graph centrality.

Important: For this task:

- use **only the first 400 samples** in `gene_expression_transcriptomic_data.csv` (so, from now on, $N = 400$ in this Task 2);
- use **only the first 10 columns of the dataset**, which contain the expression levels of the subset of 10 genes known to be most correlated to MAP7: `Feature 1: FOXO3`, `Feature 2: TCF7L2`, `Feature 3: GSK3B`, `Feature 4: NRG1`, `Feature 5: PIK3CB`, `Feature 6: PRKCE`, `Feature 7: RHOQ`, `Feature 8: PLXNA2`, `Feature 9: TACC2`, `Feature 10: DYRK2`.

The template notebook `CID_Coursework2_Part1.ipynb` contains already lines to extract only the first 10 columns and 400 samples.

2.1 (6 marks) - Sparse graph construction. Compute the $N \times N$ matrix S of cosine similarities S_{ij} between each pair of samples $\mathbf{x}^{(i)}, \mathbf{x}^{(j)}$, $i = 1, \ldots, N$. We can interpret S as the adjacency matrix of a dense, weighted and undirected graph G_{dense} with nodes $\{1, ..., N\}$. To sparsify G_{dense}, we define the thresholded adjacency matrix for threshold $0 \leq \delta \leq 1$ given by:

$$A_{ij}^{(\delta)} = \begin{cases} S_{ij} & \text{if } S_{ij} \geq \delta \text{ and } i \neq j, \\ 0 & \text{else,} \end{cases}$$

which corresponds to a sparsified graph $G^{(\delta)}$ that only retains edges with weights larger than δ. Let $n(\delta)$ denote the number of connected components of $G^{(\delta)}$ and let $s(\delta)$ denote its sparsity level defined as:

$$s(\delta) := \frac{\text{Number of edges in } G^{(\delta)}}{\text{Maximal number of edges}}.$$

Plot $n(\delta)$ and $s(\delta)$ as a function of the similarity threshold δ for 200 values ranging equidistantly from $\min(S)$ (i.e., the minimum of S taken element-wise) to $\max(S)$ (i.e., the maximum of S taken element-wise). Determine the largest threshold $\delta^* \in [\min(S), \max(S)]$ that keeps the graph entirely connected and mark δ^* in the plots you have just produced. Discuss your results.

Define the final sparsified graph $G := G^{(\delta^*)}$. Report the sparsity level of G and visualise the graph using the spectral layout introduced in the notebook from Week 9, where the x- and y-coordinates of the nodes are given

by the entries of the normalised second and third eigenvectors of the symmetric normalised graph Laplacian. Colour the nodes of the graph according to treatment labels (`Treatment`=1 and `Treatment`=0), and discuss the layout of the graph in relation to the treatment label. Explain your results.

2.2 (4 marks) - Degree centrality. Compute the degree centrality for each node in the graph G from **2.1**. Produce a plot containing histograms of the centralities for the two different treatment labels (using one colour for each treatment label). Report the mean degree for the two groups of samples, i.e., the samples with label `Treatment`=1 and the ones with label `Treatment`=0. Discuss your results, explaining the possible reasons for any differences in the degree centrality that you might observe for the different treatment labels.