2 Programming (50 points) - Inquiry: 223040237@link.cuhk.edu.cn

2.1 Tree (15 points)

The goal of this exercise is to practice tree-based methods on Diabetes data-set[1] and predict if a patient has diabetes. You are required to conduct some analysis and submit a complete report as well as your executable code. Your report should contain :

- Preliminary data analysis. Some visualization of the data distribution/variability.
- Some data processing if you find it necessary. Data splitting with ratio 0.33.
- One implementation of uni-variate tree method for the classification problem.
- One implementation of an ensemble model for the classification problem. You can choose any ensemble model that you have heard of.
- Result visualizations, comments on the results, your personal thinking, etc.
- *Optional*: can you think of a way to show how your tree splits the whole space?

You are free to use sklearn and other published packages. Your work is graded according to the readability, the logic and the degree of completion presented in your report. The coherence of you code is also taken into account.

Submission (-15 points if you do not submit code)

1. A PDF report **HW3_Programming_Tree.pdf**. Your statements should be well supported by figures or code explanation.
2. Your executable code script or notebook.

2.2 Neural Networks (35 points)

In this exercise, you will build your own fully-connected Neural Network (MLP, multi-layer perceptron) as well as your own Convolutional Neural Network. You will apply them on MNIST data-set to perform hand-written digit classification.
You are provided with a notebook guide **HW3_Programming_NN_guide.ipynb** where PyTorch framework is adopted. You have 2 options:

1. Follow the guide and fill in the blanks. Then, export the notebook to pdf file as your report. You need to have either *pickle* or *torchvision* in your python environment to load the data-set.

[1] https://www.kaggle.com/datasets/piyushborhade/diabetes-dataset

2. If you are significantly more familiar with some other deep learning framework (e.g. TensorFlow). You can perform freely on this work. Please refer to **part 2: simple NN** and **part 4: CNN** of the guide for required procedures and outputs. You will be graded according to the standard given in the guide, and your grade for 2.2 will be re-scaled with $\frac{35}{25}$.

Note that:

- Do not modify the data-sets that the TA splitted for you in the guide (The train set is the first 5500 samples of the original MNIST train set; the validation set is the last 500 samples of the original MNIST train set; the test set is the MNIST test set with 10000 samples).
- If you obtain some bonus points, those points will ameliorate or saturate your grade of this exercise (2.2) exclusively (not the whole programming part nor the whole homework).
- If you really struggle with the **PyTorch Basics** part, you can ask the TA for solution directly:

  - subject: Demand for PyTorch Basics solution; content: your name + student ID

  You can still do the bonus part to obtain at most 32 points on this exercise.
- If any of your classmates asks to copy your code, please make sure that they have already asked the TA for the Basics solution. They are supposed to show you a screen shot of the TA's reply ("copy") to their demand.
- No model score (3) nor performance score (4.5) for **part 4: CNN** if you directly load any published CNN model to solve the problem.

Submission (-35 points if you do not submit code; no performance scores if you do not include your model parameters)

1. A PDF report **HW3_Programming_NN.pdf**.
2. Your executable code script or notebook, your best NN model, your best CNN model.

Discussion for computational cost:
If you have NVIDIA card on your PC, you can use **cuda** to accelerate the calculation. However, theoretically, 2 layers of CNN are enough to have at least 97% test accuracy for this exercise.
If your PC has no worse than Intel Core i5-8250U Processor (as had the TA's PC in 2019) and your code is well done, the program shall be guaranteed to run fast.
Conventionally, you are encouraged to construct not too complicated CNN to save computational cost, since MNIST is a relatively simple problem. However, as long as your code works out for you, no point will be removed if your CNN is huge.