# CS4045 - Deep Learning

**Assignment #02** **Instructor: Dr.Usman Haider**

**Deadline: 20th October 2024**

---

## Instructions:

The assignment deadline is 20th October 2024. There is no extension in the deadline. Please submit on or before the deadline. You must verify that your submissions are correct.

Please ensure that you follow the proper file naming convention when submitting your assignments. The required format is as follows:

**i21xxxx_A2.ipynb**

Failure to adhere to this naming convention may result in your submission being marked incorrectly.

## Question #01: Rating-based Clustering of IMDB Movie Reviews and Genre Analysis Using K-means Clustering

**Objective:**

The goal of this assignment is to cluster movie reviews based on rating and analyze whether movies within the same rating clusters belong to the same genre. This assignment requires you to implement K-means clustering using Numpy and evaluate how rating-based clusters align with predefined genres.

**Dataset - IMDB-Movie-Data:**

Use the IMDB Movie Reviews dataset, which contains the year, runtime(minutes), rating, votes, revenue (Millions), and meta scores. This dataset will help explore the relationship between rating and movie genres. The **IMDB-Movie-Data.csv** is attached along with the Assignment instructions.

Part 1: Data Preprocessing and Feature Extraction

**Note:** Use the following rating scale to derive the following clusters:

- **Top Rated**: **≥ 8**
- **Average Rated: 6 to 7**
- **Low Rated: < 6**

Part 2: Implement K-means Clustering from Scratch

1. Cluster Reviews Based on Rating Scores: Implement K-means clustering using Numpy to group the reviews into clusters. You must check for convergence when centroids no longer change.
2. Cluster Label Assignment

Part 3: Genre Analysis and Model Evaluation

1. Compare Genres Within Clusters: Compare whether movies within the same cluster belong to the same genre.
2. Evaluation Metrics: Generate a confusion matrix comparing the cluster ratings with the original ratings.

Part 4: Visualize the Clusters.

## Question #02: Handwritten Digit Classification Using SVM (from Scratch)

**Objective:**

Build an SVM classifier from scratch to recognize handwritten digits from the MNIST dataset, using no built-in SVM libraries. You will optimize the kernel function (linear, polynomial, and RBF), tune hyperparameters, and compare their performance. Analyze classification accuracy and visualize misclassified digits.

**Dataset: MNIST Handwritten Digit Dataset**

- **Source**: MNIST dataset contains 70,000 grayscale images of size 28x28 pixels, representing digits from 0 to 9.
- **Task**: Build an SVM-based model to classify these images into their corresponding digit categories (0 to 9).

Link: https://www.kaggle.com/datasets/hojjatk/mnist-dataset

Part 1: Data Loading and Preprocessing

Part 2: SVM Classifier Implementation from Scratch

You are required to implement the SVM classifier using Python without using any SVM-related libraries like scikit-learn. The following components should be built manually:

1. Define the SVM Model:
    - Implement the primal form of SVM for binary classification.
    - For multi-class classification, use the one-vs-rest (OvR) approach to extend the model to classify digits (0-9).
2. Kernel Functions:
    - Implement three types of kernels:
        1. Linear Kernel
        2. Polynomial Kernel
        3. Radial Basis Function (RBF) Kernel

3. Optimization:

- Use techniques like gradient descent to optimize the margin and solve for the support vectors.

- Implement regularization using a parameter to control the trade-off between maximizing the margin and minimizing classification errors.

Part 3: Model Training

1. Hyperparameter Tuning:
    - Tune key hyperparameters such as the kernel type, degree for the polynomial kernel, gamma for the RBF kernel, and regularization parameter.
    - Use cross-validation to find the best combination of hyperparameters.
2. Train the Model:
    - Train the SVM on the training set using the manually implemented SVM classifier.
    - Use the one-vs-rest strategy for multi-class classification.

Part 4: Model Evaluation

1. Test the Model:
    - Evaluate the model on the test set.
    - Calculate the following evaluation metrics:
        - Accuracy: Overall classification accuracy.
        - Confusion Matrix: Visualize the confusion matrix to identify which digits are most often misclassified.
        - Precision, Recall, and F1-Score: Calculate these for each digit class (0-9).
2. Visualize Misclassified Digits:
    - Display the first 10 misclassified images along with the predicted label and the true label.

Part 5: Performance Comparison

1. Compare Kernels:
    - Compare the performance of the linear, polynomial, and RBF kernels.
2. Impact of Regularization:
    - Compare models with different values of the regularization parameters.