Statistical Methods in AI - Monsoon 2025

Assignment 2 [100 Marks]

Deadline : 27th September 2025 11:59 P.M.

Instructors: Prof Ravi Kiran Sarvadevabhatla, Prof Saikiran Bulusu

General Instructions

- Your assignment must be implemented in Python.
- Clearly label and organize your code, including comments that explain the purpose of each section and key steps in your implementation. Ensure that your files are well-structured, with headings, subheadings, and explanations as necessary.
- Make sure to test your code thoroughly before submission to avoid any runtime errors or unexpected behavior.
- Your assignment will be evaluated not only based on correctness but also on the quality of code, the clarity of explanations, and the extent to which you've understood and applied the concepts covered in the course.
- We are aware of the possibility of submitting the assignment late in GitHub Classroom using various hacks. Note that we will have measures in place accordingly and anyone caught attempting to do the same will be given a straight zero in the assignment.

AI Tools Usage Instructions (Mandatory if Applicable)

We are aware how easy it is to write code and solve questions with the help of LLM services, but we strongly encourage you to figure out the answers yourself. If you use any AI tools such as ChatGPT, Gemini, Claude, etc., to assist in solving any part of this assignment:

- **If you are unable to explain any part of the solution/code during evaluations, that solution/code will be considered plagiarized and you will be penalized.** You must also be able to briefly explain how you modified or verified the AI-generated content.
- You must include a **shareable link** to the AI Tool chat history or a **screenshot** of the relevant conversation. **Do NOT share any private or sensitive personal information** in the AI Tool conversations you include.

Submission Instructions

- Submit a self-contained **Jupyter notebook** containing all code, plots, and printed tables.
- Report all the analysis, comparison and any metrics in the notebook or a separate report that is part of the submission itself. No external links(Except for video for Q.4) to cloud storage files, wandb logs or any other alternate will be accepted as part of your submission. Only the values and visualizations as part of your commits will be graded.
- All plots must include your **email username** in the title or filename

```
plt.text(
    0.95, 0.95, "username",
    ha='right', va='top',
    transform=plt.gca().transAxes,
    fontsize=10, color='gray', alpha=0.7
)
```

Submission Policy (GitHub Classroom Assignments)

To encourage consistent progress and discourage last-minute submissions, the following policy applies:

- **Minimum Progress Requirement:** You must push at least **two meaningful commits on different days** prior to the deadline.
- **Commit Timing Check:** If over 80% of commits are made within the last 24 hours before the deadline, a **10% penalty** will be applied.
- **Commit Quality:** Commits must reflect actual progress. Non-informative or placeholder commits will not count.
- **Final Submission:** Your final grade is based on the latest commit before the deadline, but commit history will be reviewed.

Guidelines for Implementation and Code Design

1. Use object-oriented programming.
2. Dataset for the complete assignment can be found **here**
3. Use appropriate visualization libraries such as `matplotlib`, `seaborn`, or `plotly`. Each plot must include a **title, x-label, y-label, and legend (if applicable)**. Answers with missing labels or legends will receive **0 marks**.
4. Keep visualization and computation logic in **separate functions**.
5. Add `docstrings` to methods explaining what they do.

1 Linear Regression with Regularization [10 marks]

You will predict `GPA` (Refer Assignment 1 for dataset) using student features. Use a validation set to select the hyperparameters.
Start with a function of the following form:

```
def run_poly_regression(X_train, y_train,
                        X_val, y_val,
                        X_test, y_test,
                        degree=1,
                        regularizer=None,
                        reg_strength=0.0):
    """
    Fit a polynomial regression model with optional regularization.

    Parameters:
        degree (int): Degree of the polynomial to fit
        regularizer (str or None): 'l1', 'l2', or None
        reg_strength (float): Regularization coefficient (alpha)

    Returns:
        dict with train, val, and test MSEs, and learned
           coefficients
    """
```

Perform the following tasks.

- For three setups - no regularization, L1 and L2 regularization, repeat the below steps:
  - Fit polynomial regression models across degrees 1 to 6.
  - Plot polynomial degree vs MSE (on train and validation sets). Describe the trend you observe as degree increases.
  - For each degree, use val MSE to choose the best regularization strength.
  - Plot regularization strength (log scale) vs val MSE for best degree.
- Comment on performance improvement (if any) from regularization. Which overall experimental setup (degree, regularizer) yielded the best test performance?
- For the best setup using L1 regularization, which features had non-zero weights? List the most important predictors for GPA. Repeat the same with L2 regularization. Comment on the differences.
- You are allowed to use `sklearn.linear_model.LinearRegression`

2.0 K-Means Clusteing [10 marks]

2.1 Implement a K-Means Class [6 marks]

Implement the K-Means Clustering Class that replicates the core functionalities of the built-in K-Means library, including methods such as fit(), predict(), and getCost().

- The `fit()` function should: Train the K-Means model by using K clusters on the dataset.
- The `predict()` method should: Assign a cluster number to each data-point based on the centroids obtained from the fit() function.
- The `getCost()` method should: Return the cost of K-Means, i.e., the Within-Cluster Sum of Squares (WCSS).
- The number of clusters (k) must be specified when instantiating the class.
- Ensure it works with the given **dataset** after the required preprocessing is applied.

2.2 Determine the Optimal Number of Clusters [4 marks]

Use the following methods to determine the optimal number of clusters for the given dataset (you are allowed to use inbuilt-library functions):

- **Elbow Method (References)**
  - Vary the value of k and plot the Within-Cluster Sum of Squares (WCSS) against k.
  - Comment on the optimal value of k according to this method.
- **Silhouette Method (References)**
  - Plot the average silhouette score for each k.
  - Choose the value of k that maximizes the silhouette score.
  - What can you say about the clusters obtained at the optimal value?

3. Gaussian Mixture Models [10 marks]

3.1 Implement the GMM Class [6 marks]

- Write your own GMM class.
- Your GMM class should include methods like `fit()`, `getMembership()` and `getLikelihood()`.
- The `fit()` method implements the Expectation-Maximization (EM) algorithm on the dataset to determine the optimal parameters for the model.

- The `getMembership()` method returns the membership values Y_{ic} for each sample in the dataset. The membership value Y_{ic} for a sample x_i is the probability that the sample belongs to cluster c in the given GMM.
- The `getLikelihood()` method returns the overall likelihood of the entire dataset under the current model parameters.
- The `drawLikelihood()` draws the plot of likelihood vs iterations for the number of clusters used.
- Ensure it works with the given **dataset** after the required preprocessing is applied. The number of clusters (k) must be specified when instantiating the class. Plot the graph of overall likelihood vs iteration also (using `drawLikelihood()` function).

3.2 Determine the Optimal Number of Clusters [4 marks]

Use the following methods to determine the optimal number of clusters for the given dataset (you are allowed to use inbuilt-library functions):

- **BIC (Bayesian Information Criterion) (References)**
  - Vary the value of k and plot the BIC against k.
  - Comment on the optimal value of k according to this method.
- **Silhouette Method (References)**
  - Plot the average silhouette score for each k.
  - Comment on the optimal value of k according to this method.

4 Image Segmentation [20 marks]

In this question, you will use your custom-implemented Gaussian Mixture Model (GMM) from the previous question to perform color-based image segmentation on two satellite images **(satellite_1.png and satellite_2.png)**. The goal is to visualize the convergence of the Expectation-Maximization (EM) algorithm by creating a video that shows the segmentation process and the model's log-likelihood at each iteration.

4.1 Image Segmentation [8 marks]

- Load each of the two provided satellite images.
- Fit your GMM to the flattened dataset. Keep number of Gaussian components (k) as 3 to segment the image into distinct regions **(Land, Water and Vegetation cover)**.
- Choose colors that make the 3 distinct segments **clearly visible**. Ambiguous submissions will receive a score of zero.

4.2 Dynamic Visualization Video [12 marks]

- For each of the two satellite images, generate a video that visualizes the step-by-step fitting process of your GMM. You are free to use any library of your preference for video generation.
- Each frame in the video must correspond to a single iteration of the EM algorithm.
- The video frame must be composed of three panels displayed side-by-side:
  - Panel 1 (Left): The original, unchanged satellite image.
  - Panel 2 (Center): The segmented image at the current iteration. To generate this, replace each pixel's original color with the mean color of the Gaussian component it has been assigned to at that specific iteration.
  - Panel 3 (Right): A real-time graph of the log-likelihood versus the iteration number. The plot should update at each frame, showing the curve extend as the algorithm progresses towards convergence.
- Upload the videos to your OneDrive account and share a view-only link in the markdown.

5. PCA [10 marks]

In this question, you will be using the MNIST dataset (28x28 pixels), on which you will apply PCA from scratch.

5.1 Custom Implementation [4 marks]

- Write your own PCA class. Your PCA class should include methods like `fit()`, `transform()` and `checkPCA()`. The `fit()` method obtains the principal components of our dataset.
- The `transform()` method transforms the data using the principal components after we fit the data.
- The `checkPCA()` method should help you check if your class reduces the dimensions appropriately. It should return True if the class works and False otherwise.
- Specify the number of components (n_components) when instantiating the class to define how many dimensions the data should be reduced to.

5.2 PCA on MNIST [6 marks]

- Load the **MNIST dataset (28x28 pixels)** using any library of your preference and construct a **balanced subset of 1000 samples**, ensuring there are exactly 100 examples for each digit (0-9).

- Flatten each 28x28 pixel image in your subset into a single 784-dimensional vector.
- Using the PCA functions you implemented in section 4.1, project the 1000 sample dataset into the following target dimensions: **500, 300, 150, and 30**.
- Plot the graph of explained variance vs. the number of principal components.
- Select any 5 images from these samples. Plot them before dimensionality reduction, and after projecting them back to the original space (do this for each of the final dimension values). Write your observations.

6. PCA + Classification [20 marks]

In this question you will be investigating the effect of dimensionality reduction using Principal Component Analysis (PCA) on the performance of a K-Nearest Neighbors (KNN) classifier. You will use the **8x8 pixel images** MNIST digits dataset, apply your custom PCA implementation, classify the data using sklearn's KNN, and visualize the results.

6.1 Data Loading and Preparation [2 marks]

- Load the MNIST digits dataset, which consists of **8x8 pixel images** of handwritten digits. You can import it using `sklearn.datasets.load_digits()`.
- Split the dataset into a training set and a testing set. Use an 80-20 split and set random_state=42 to ensure your results are reproducible.

6.2 Dimensionality Reduction with PCA [5 marks]

- You will use the custom PCA class you implemented in previous questions (Q4).
- Define a list of dimensions to reduce the data to: **n_components_list = [2, 5, 10, 20, 30, 40, 50, 64].**
- For each value in n_components_list, fit your PCA model on the training data and then use it to transform both the training and testing data.

6.3 Classification with KNN [8 marks]

- For each of the transformed datasets from the previous step, you will **train and evaluate a KNN classifier**. You can use the KNeighborsClassifier from sklearn.neighbors.
- You need to test the classifier's performance for different values of k (the number of neighbors). Define a list of **k_values = [5, 25, 50, 100]**.
- For each combination of n_components and k, perform the following:
  - Initialize and train a KNeighborsClassifier model with the current k value on the transformed training data.

- Use the trained model to predict the labels for the transformed test data.
- **Calculate the accuracy** of the predictions and store it.

6.4 Visualization and Analysis [5 marks]

- Create a single plot to visualize the relationship between the **number of principal components and the classification accuracy** for each value of k.
- Based on your graph, briefly discuss your findings. How does the number of principal components affect the accuracy of the KNN classifier? How does the choice of k influence this relationship? Is there a point of diminishing returns where adding more components does not significantly improve accuracy?

7. Enhancing Clustering Accuracy with Data Transformation [20 marks]

You are provided with a modified version of the MNIST dataset containing 1,000 digit images and their labels **here**. Unlike the standard MNIST dataset, these images have been altered, making it challenging for clustering algorithms to correctly group them. Your task is to investigate this dataset, analyze the nature of the alterations, and design an appropriate transformation strategy. Use your custom implementation from previous questions for any algorithm that you use in this question.

7.1 Input

- `X_modified.npy`: Array of 1,000 altered MNIST digit images of shape (1000, 28, 28).
- `y_true.npy`: Ground truth digit labels of shape (1000,).

7.2 Expected Output

- A clear description of the observed differences between the modified dataset and standard MNIST. [4]
- A suitable transformation technique that addresses these differences, with justification. [6]
- A comparison of clustering performance before and after applying your transformation, using an evaluation metric of your choice. [5]
- A concise analysis explaining why the transformation improved clustering outcomes. [5]

Good luck with the assignment