Statistical Methods in AI - Monsoon 2025

Assignment 5 [100 Marks]

Deadline : 20th November 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 Kernel Density Estimation for Foreground Detection [25 Marks]

Figure 1: Foreground Detection Using KDE

1.1 Image Pre-Processing [2 Marks]

You are provided with two images:

- A **background image** (without any foreground object)
- A **test image** (with a foreground object present)

Your tasks are as follows:

1. **Image Alignment and Resizing:** Convert both images to the same dimensions, balancing computational efficiency and image quality.
2. **Feature Extraction:** Extract a relevant feature set (e.g., RGB channels, grayscale intensity, or color histograms) to be used as input for the KDE model.

Provide a brief justification for your chosen preprocessing approach and feature representation.

1.2 Custom KDE Class Implementation [15 Marks]

You are required to implement a custom **Kernel Density Estimation (KDE)** model from scratch, using only the `numpy` library (no external statistical or machine learning libraries are permitted).

Class Methods

1. ***__init__***
   Initialize the class with the following parameters:

   - `kernel`: a string specifying the kernel type (e.g., "gaussian", "triangular", "uniform").
   - `bandwidth`: a positive float controlling the smoothness of the density estimate.
   - `data`: the training data (e.g., background image features).

2. ***fit(data)***
   Implement the fitting method to store or preprocess the background data. Include a *smart sampling strategy* (e.g., random subset selection or grid-based sampling) to reduce computational cost while maintaining the representational quality of the model.

3. ***predict(samples)***
   Implement a prediction method that accepts a set of sample points (e.g., pixels from the test image) and returns their estimated probability densities under the fitted model. The prediction should use the chosen kernel function and bandwidth parameter to compute:

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

   where n is the number of training samples and d is the feature dimensionality.

1.3 Foreground Detection [8 Marks]

Use your KDE implementation to perform foreground detection as follows:

1. Fit the KDE model on the **background image features**.
2. Predict the density values for each pixel (or feature vector) in the **test image**.
3. Classify pixels as foreground or background based on an appropriate probability density threshold.

Experiment with different values of **bandwidth** and **kernel type** to obtain the best segmentation results. Provide a brief explanation for why your chosen parameter combination achieves superior performance.

For each class, briefly describe the architecture, hyperparameters, and optimization strategy used. Report the validation performance of all models on identical train/validation splits.

2 Data-Driven Discovery of a Discrete-Time Recurrence [20 Marks]

2.1 Dataset and Problem Setup [3 Marks]

You are provided with multiple univariate sequences $\{x_k\}_{k=0}^{T}$ generated by an **unknown, time-invariant, autonomous, discrete-time mechanism**. The data are noisily observed but follow a fixed rule within each sequence. Your goals are to (i) build predictors of x_k from its history, (ii) *identify* a compact analytical recurrence consistent with the data (including its effective order), and (iii) analyze how well it fits to the data. You are suggested to use an RNN for this task. Dataset is at Dataset link.

Tasks

1. **Data splits:** Create chronological train/validation/test splits. Justify lengths.
2. **Supervised pairs:** Form $(\mathbf{h}_k \mapsto x_k)$ with history vectors of length p and explain how p is selected.
3. **Normalization:** Apply normalization based on the training set only; describe inverse transform.

Suggested Implementation Skeleton

- `data_prep.py`: load data, create time splits, and normalize.
- `model.py`: implement baseline predictors (e.g., Linear AR, MLP, or small RNN).
- `train.py`: train chosen models, tune p, and log results.
- `identify.py`: derive analytical recurrence F_θ consistent with learned mapping.
- `analyze.py`: plot residuals, and report test performance.

2.2 Sequence Prediction [6 Marks]

Train one or more predictors mapping a chosen history to the next value:

$$\hat{x}_k = g_\phi\big(x_{k-1}, x_{k-2}, \ldots, x_{k-p}\big),$$

and later use this to recover a *recurrent* analytical expression. (Hint : *recurrent*)

Suggested Deliverables

- Trained models and hyperparameter table.
- Prediction plots and validation curves.
- Short discussion on history length and model complexity.

2.3 Analytical Recurrence Identification [6 Marks]

Identify a closed-form recurrence

$$\hat{x}_k = F_\theta\big(x_{k-1}, x_{k-2}, \ldots, x_{k-\hat{p}}\big),$$

and compare residuals with black-box predictors.

2.4 Evaluation Criterion

- With MAE, MSE as your metrics, evaluate on single-step prediction with your DL model.
- Evaluate autoregressive generation, both with your DL model, and the analytical expression that you've fit to your data.
- Show variation in error with these models, with increasing forecasting length.

2.5 Parsimony and Stability [3 Marks]

Explore parameter count vs. performance. Select the simplest accurate model and present as to how well it fits to the data in comparison to other sweeps. What does this tell you about the dataset ? Is there any temporal relation you can extract from the dataset.

Report Checklist

- Model specifications, identified F_θ, parameter estimates.
- Complexity–accuracy trade-off figure.
- Stability plots and conclusions.

---

3 Time Series Forecasting: Cumulative GitHub Stars [20 Marks]

3.1 Dataset Description [3 Marks]

You are provided with M GitHub repositories, each represented by a time series of cumulative star counts:

$$y_t^{(i)} = \text{total stars for repository } i \text{ up to time } t.$$

The task is to forecast future growth trajectories and compare classical vs. deep learning methods. The dataset is present at Dataset Link.

Provided Files

- `stars_data.csv` — timestamps, repository IDs, and star counts.
- `repo_metadata.json` — optional meta-features (language, topic, etc.).

3.2 Single Repository Evaluation:

Choose 2 of the repositories present in the dataset, and perform all of the below tasks for the repositories you have chosen.

3.2.1 Preprocessing and Feature Engineering [4 Marks]

1. Clean and align time series; handle missing points or jumps.
2. Choose whether to work in cumulative or incremental domain:

$$\Delta y_t = y_t - y_{t-1}.$$

3. Look at scaling of your data.
4. Ensure temporal integrity—no leakage from future data.
5. Work in different time domains, visualize your data and look as to how the number of stars progress in the dataset. Consider the differences, creating your train-test split would make, based on how you split your forecasting data.

Suggested Implementation Skeleton

- `prep_stars.py`: data cleaning, feature creation, normalization.
- `split_repos.py`: chronological and repository-level splits.

3.2.2 Forecasting Models [7 Marks]

Fit and compare:

- **Classical:** ARMA.
- **Deep Learning:** small RNN and 1D CNN forecasters
- Note: Explain how youre preparing your data for each of these models. Justify your choice in loss function for this dataset.

Suggested Implementation Skeleton

- `classical.py`: wrapper around statsmodels ARMA/ARIMA.
- `dl_models.py`: simple PyTorch/TensorFlow RNN and CNN implementations.
- `train_models.py`:Training loop/calls for all models and hyperparameter search.

3.2.3 Evaluation Protocol [4 Marks]

Metrics and Evaluation

Evaluate model performance using standard error metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

- For **single-timestep forecasting**, compute these metrics at each prediction horizon to quantify immediate predictive accuracy.
- For **multi-step (increasing-window) forecasting**, autoregressively generate future predictions over extended horizons (e.g., $h = 1, 3, 7, 14, \dots$), and plot the mean prediction error as a function of forecast length to analyze how accuracy varies with longer horizons.

Suggested Deliverables

- `evaluate.py`: backtesting, metric computation, and plotting.
- Summary table of metrics across models and horizons.
- Representative forecast and calibration plots.

3.3 Report and Reproducibility [[2 Marks]

Submit a concise report including:

- Data preparation summary and transformations.
- Model architectures and tuning results.
- Quantitative metrics, forecast visuals, and generalization results.

Ensure full reproducibility with all your scripts.

Marking Rubric (Summary)

- Data preprocessing and handling: 4
- Model implementation and comparison: 7
- Evaluation protocol and metrics: 4
- Clarity and reproducibility: 3
- Report polish and visualization quality: 2

4 Variational Autoencoder (VAE) [35 Marks]

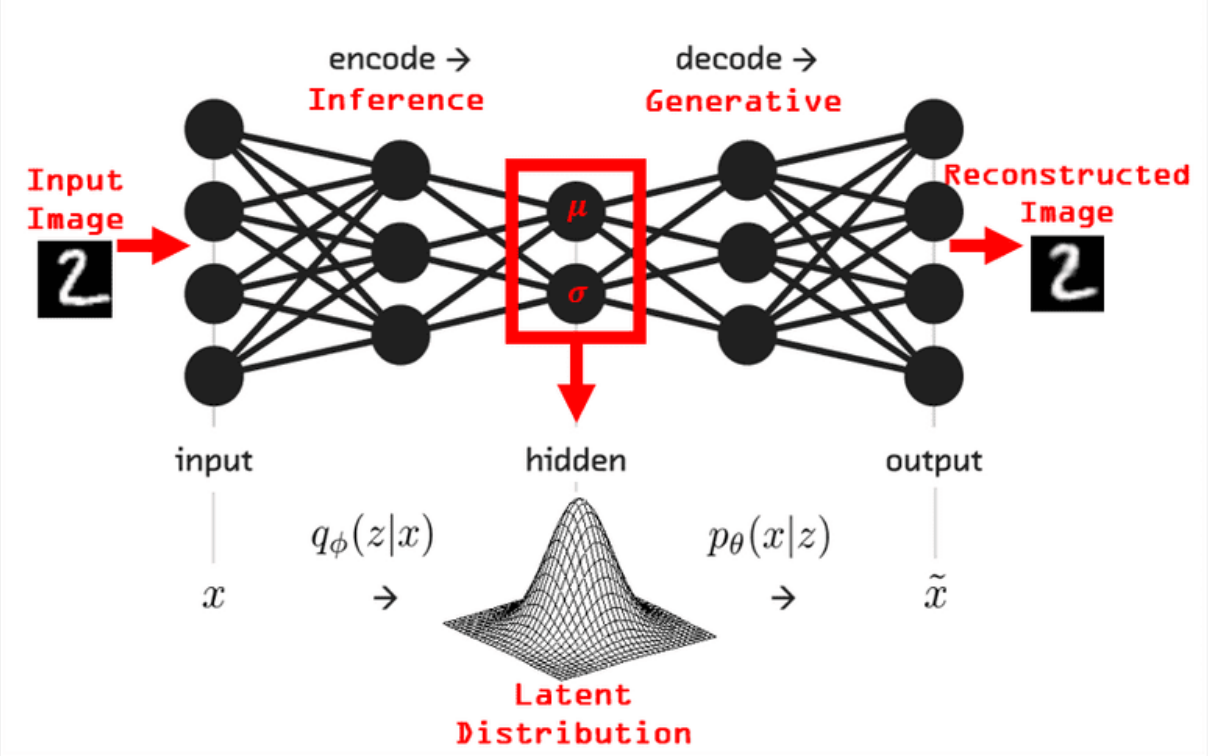

Figure 2: Variational Autoencoder Architecture

4.1 Introduction

A Variational Autoencoder (VAE) is a generative deep learning model that learns to encode data into a continuous latent space and decode it back to reconstruct the input. In this task, you will implement and analyze a VAE trained on the **Fashion-MNIST** dataset. You are allowed to use external libraries for this question.

4.2 Dataset Preparation [2 Marks]

Load and preprocess the **Fashion-MNIST** dataset using PyTorch utilities.

- Normalize the data and create DataLoader instances for both training and testing.
- Choose suitable batch size and normalization strategy.

4.3 Model Architecture [10 Marks]

Design and implement a Variational Autoencoder consisting of:

1. **Encoder:** A neural network mapping the input image to a latent representation defined by a mean vector μ and a log-variance vector $\log(\sigma^2)$.
2. **Reparameterization Trick:** Use $z = \mu + \epsilon \cdot \sigma$ with $\epsilon \sim \mathcal{N}(0, I)$ to enable stochastic sampling with gradient backpropagation.
3. **Decoder:** A neural network reconstructing images from latent samples.

Experiment with different latent dimensions and network depths, and discuss how these changes influence the smoothness of the latent space and reconstruction fidelity.

4.4 Loss Function [4 Marks]

The total loss for the VAE is a combination of:

- **Reconstruction Loss:** Measures pixel-wise similarity between input and reconstruction using Binary Cross-Entropy (BCE).
- **KL Divergence:** Encourages the latent distribution to approximate a standard normal prior.

Formally, the objective function is given by:

$$\mathcal{L} = \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{KL}(q(z|x) \,||\, p(z))$$

Explain the role of each term in shaping the learned latent representation.

4.5 Training Procedure [2 Marks]

Train your model using an optimizer of your choice (e.g., Adam).

- Plot the total loss and its components over epochs.

4.6 Experimental Analysis [9 Marks]

In this section, you will conduct systematic experiments to understand the influence of key parameters on VAE performance.

- Vary the weighting between reconstruction loss and KL divergence using a β-parameter (as in β-VAE). Create a gif/video showing how the latent space evolves for values of $\beta = [0.1, 0.5, 1]$ for 3 classes of your choosing, make sure to color-code the data points in the latent space according to their class to observe clustering in the latent space (An example video has been attached in the dataset folder). Discuss how increasing β affects image sharpness/quality, Diversity, Cluster separation etc.
- Summarize your findings in a short table comparing quantitative metrics (e.g., reconstruction loss) and qualitative observations.

4.7 Evaluation and Visualization [3 Marks]

- Display a set of original and reconstructed test images side-by-side.
- Generate new samples by sampling from $\mathcal{N}(0, I)$ in the latent space and decoding them.
- Interpret the quality, diversity, and realism of generated samples, measure the Fréchet Inception Distance (FID) to evaluate goodness of newly generated samples.

4.8 Effect of Frozen Latent Parameters [5 Marks]

Investigate the influence of fixed latent distribution parameters on generation quality.

- Freeze the mean μ as 0 and try $\sigma = [0.1, 0.5, 1]$
- Generate new samples by decoding these frozen latent vectors.
- Compare the generated samples to those obtained from the standard stochastic sampling ($z = \mu + \epsilon \cdot \sigma$).
- Evaluate and discuss the effect of freezing μ and/or σ on image diversity, sharpness, and representational smoothness in the latent space.