Statistical Methods in AI - Monsoon 2025

Assignment 4

Deadline : 3rd 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. 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**.
3. Keep visualization and computation logic in **separate functions**.
4. Add `docstrings` to methods explaining what they do.

Q1: Multi-Task CNN on Fashion-MNIST (75 Marks)

Objective

Convolutional Neural Networks (CNNs) are widely used for image-based tasks such as classification, regression, and object detection. In this assignment, you will implement a single **multi-task CNN** that jointly performs:

(a) **Classification** of Fashion-MNIST images into 10 clothing categories, and

(b) **Regression** of a continuous "ink" value corresponding to the normalized pixel intensity of each image.

The model will be trained using a joint loss:

$$\mathcal{L} = \lambda_1 \mathcal{L}_{\text{classification}} + \lambda_2 \mathcal{L}_{\text{regression}},$$

where $\mathcal{L}_{\text{classification}}$ is Cross-Entropy loss and $\mathcal{L}_{\text{regression}}$ is Mean Squared Error (MSE). You will explore the effect of λ_1 and λ_2 on the trade-off between classification and regression performance.

All experiments must be logged using Weights & Biases (wandb).

Dataset Description (Fashion-MNIST)

The Fashion-MNIST dataset (Zalando Research) contains 70,000 grayscale images (28×28) divided into 10 clothing classes:

```
0: T-shirt/top, 1: Trouser, 2: Pullover, 3: Dress, 4: Coat, 5: Sandal,
            6: Shirt, 7: Sneaker, 8: Bag, 9: Ankle boot.
```

Use the training split for model training and validation, and the provided test split for final evaluation.

Data Loading and Preprocessing (15 Marks)

1. **Dataset loading:** Implement a function `load_fashion_data()` that:
   - Loads the Fashion-MNIST dataset from Kaggle or torchvision,
   - Splits the original training set into `train` and `val` (e.g., 90/10 ratio),
   - Ensures no overlap or data leakage between splits.

2. **Custom Dataset class:** Implement `FashionMNISTDataset` that returns tuples `(image, class_label, ink_target)`:
   - **image:** normalized tensor using the mean and std of Fashion-MNIST,
   - **class_label:** the integer label $y \in \{0, \ldots, 9\}$,
   - **ink_target:** the normalized pixel intensity (average pixel value of the image), representing how much "ink" is used.

3. **Augmentations:** Use **light augmentations** (e.g., random crop, small rotation) for the training set only to encourage generalization. Keep validation and test sets unaltered.

Model Implementation: Multi-Task CNN (25 Marks)

1. **Architecture (7.5 Marks):** Implement a shared convolutional backbone with 2–3 Conv–BatchNorm–ReLU–Pool blocks and dropout. The model should then branch into two separate heads:
   - **Classification head:** outputs logits $[B, 10]$,
   - **Regression head:** outputs scalar values $[B, 1]$.
2. **Forward pass (7.5 Marks):** The `forward()` method should:
   - Pass inputs through shared convolutional layers,
   - Output both classification logits and regression predictions,
   - Optionally return intermediate feature maps for visualization.
3. **Joint loss (10 Marks):** Implement the total loss as:

$$\mathcal{L}_{\text{total}} = \lambda_1 \mathcal{L}_{\text{CE}} + \lambda_2 \mathcal{L}_{\text{MSE}}.$$

   Experiment with different values of λ_1 and λ_2 (e.g., 1:1, 2:1, 1:2, 0.5:1, etc.) to observe how they influence the model's performance on both tasks.

Hyperparameter Tuning and wandb Logging (20 Marks)

1. **Hyperparameters (7.5 Marks):** Explore key hyperparameters:
   - learning rate, dropout rate, optimizer choice (SGD, Adam, AdamW),
   - number of convolutional layers/filters, batch size, and
   - weighting factors λ_1 and λ_2.
2. **wandb Logging (Mandatory):** Each run must be logged on **Weights & Biases (wandb)**. Record:
   - Configuration (hyperparameters, λ_1, λ_2),
   - Training/validation loss curves (total, CE, MSE),
   - Validation metrics (accuracy, MAE, RMSE),
   - Final test performance and visualizations.
3. **Training runs (7.5 Marks):** Conduct at least **five distinct runs**, varying hyperparameters and λ_1, λ_2. Plot the training and validation losses for all runs (preferably via wandb line charts).

4. **Model selection (5 Marks):** Identify:
   - The model with the highest validation classification accuracy,
   - The model with the lowest validation regression RMSE.

   Report the corresponding test metrics and discuss how varying λ_1, λ_2 affected the trade-off between the two objectives.

Feature Map Visualization (15 Marks)

1. **Intermediate feature maps (7.5 Marks):** Modify `forward()` (or use hooks) to extract intermediate outputs after each convolutional block. Visualize feature maps from each layer for any three test images.
2. **Interpretation (7.5 Marks):** Describe the kinds of features captured at different layers (edges, textures, shapes, clothing regions) and how they help both classification and ink regression.

Deliverables

- **wandb Dashboard:** include a shareable project link in your report.
- **Report (PDF):** Describe your approach, hyperparameters, wandb plots, best models, and feature map visualizations. Discuss the observed trade-off between classification and regression performance as λ_1, λ_2 vary.
- **Metrics:**
  - Classification — Accuracy (and optionally F1-score)
  - Regression — MAE and RMSE

Q2: Image Colourization using CNNs (75 Marks)

Overview

In this task, you will implement and train a convolutional encoder–decoder network for **image colourization** on the CIFAR-10 dataset (32×32 RGB images). All code must be written in **PyTorch**, and you must log training, validation, and hyperparameter sweeps using **Weights & Biases (wandb)**.

Given a grayscale input image, your goal is to predict the colour of each pixel, classifying it into one of 24 colour clusters.

Colour Centroids and Data Preparation

To simplify the image colourization task, colour prediction is treated as a **classification** problem rather than regression. Each RGB pixel in CIFAR-10 is assigned to one of 24 representative clusters (colour centroids).

Provided: `color_centroids.npy` (shape: [24,3]) — 24 representative RGB cluster centroids computed by applying k-means ($k = 24$) over CIFAR-10 training pixels.

- Each pixel (x, y) is mapped to the nearest centroid $\mathbf{c}_i$:

$$\text{label}(x, y) = \arg \min_{i \in \{1,\ldots,24\}} \|\text{RGB}(x, y) - \mathbf{c}_i\|_2$$

- This produces a class map of size [32,32] with values in $\{0, \ldots, 23\}$.
- The model input is a grayscale image [1,32,32].
- Model output is logits per pixel [24,32,32].
- The loss is per-pixel cross-entropy.

Model Architecture: (20 pts)

In this architecture, the encoder is designed to **progressively reduce the spatial dimensions** of the input image through multiple convolutional and pooling layers. This deeper compression enables the network to capture more abstract and spatially invariant features. The decoder, on the other hand, reconstructs the colourized output image by performing **learned upsampling** using `nn.ConvTranspose2d` (transpose convolution) layers instead of simple interpolation. This approach allows the model to learn the most effective way to recover fine spatial details during reconstruction, encouraging students to understand how receptive fields expand in the encoder and how transpose convolutions can learn spatial mappings in the decoder.

Recommended block sequence

Use this architecture as the baseline. Experiments with kernel, NIC, NF.

```
Input [B, NIC, 32, 32]
Conv2d -> BN -> ReLU -> MaxPool2d   -> [B, NF, 16, 16]
Conv2d -> BN -> ReLU -> MaxPool2d   -> [B, 2*NF, 8, 8]
Conv2d -> BN -> ReLU -> MaxPool2d   -> [B, 4*NF, 4, 4]     # extra downsample
ConvTranspose2d -> BN -> ReLU       -> [B, 2*NF, 8, 8]    # learned upsample
ConvTranspose2d -> BN -> ReLU       -> [B, NF, 16, 16]
ConvTranspose2d -> BN -> ReLU       -> [B, NC, 32, 32]
Conv2d (classifier)                  -> [B, NC, 32, 32]
```

Architecture diagram

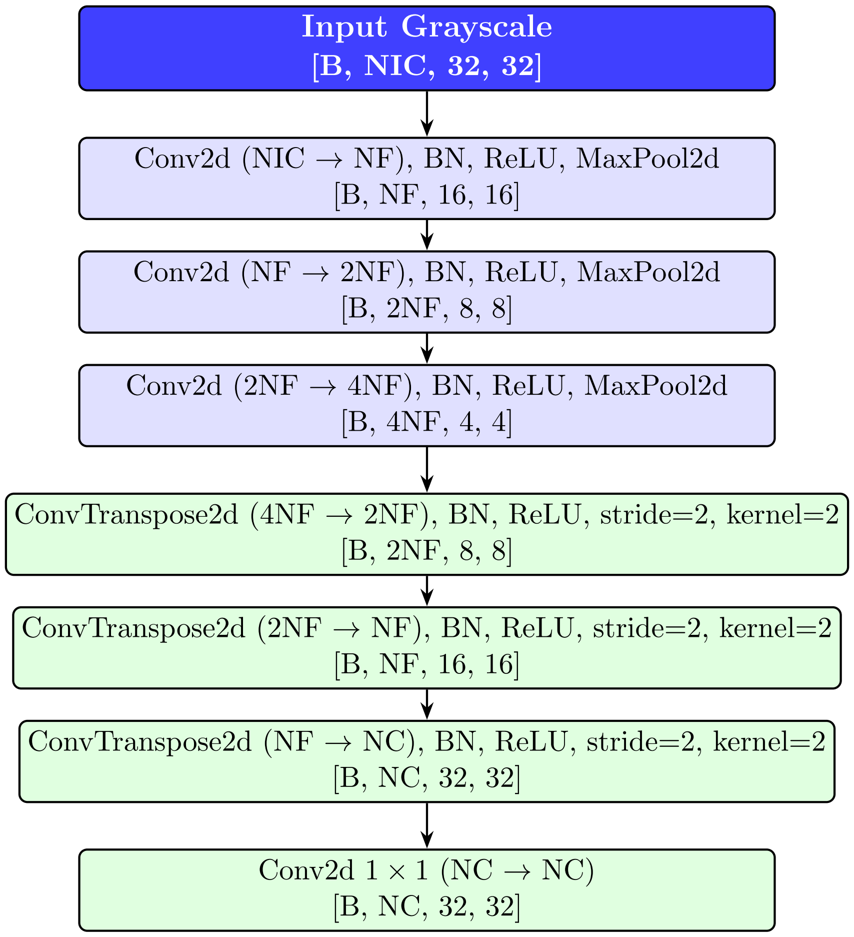
Input Grayscale
[B, NIC, 32, 32]
Conv2d (NIC → NF), BN, ReLU, MaxPool2d
[B, NF, 16, 16]
Conv2d (NF → 2NF), BN, ReLU, MaxPool2d
[B, 2NF, 8, 8]
Conv2d (2NF → 4NF), BN, ReLU, MaxPool2d
[B, 4NF, 4, 4]
ConvTranspose2d (4NF → 2NF), BN, ReLU, stride=2, kernel=2
[B, 2NF, 8, 8]
ConvTranspose2d (2NF → NF), BN, ReLU, stride=2, kernel=2
[B, NF, 16, 16]
ConvTranspose2d (NF → NC), BN, ReLU, stride=2, kernel=2
[B, NC, 32, 32]
Conv2d 1 × 1 (NC → NC)
[B, NC, 32, 32]

Figure 1: Model architecture

Training and Evaluation (20 pts)

- Dataset: CIFAR-10. Use supplied `color_centroids.npy` to map RGB images to 24-class labels; input is grayscale.

- Loss: per-pixel cross-entropy.
- Optimizer: SGD or Adam (log choice and hyperparameters in wandb).
- Train for at least 25 epochs; save and log the best checkpoint (lowest validation loss).
- At the end, produce:
    1. 10 example colourizations (input gray, predicted colourized, ground-truth colour).
    2. Training and validation loss curves.

Analytical Questions (10 pts)

For your `Model Architecture` compute (symbolically in terms of NIC, NF, NC):

1. Number of weights (ignore biases and BatchNorm parameters).
2. Number of outputs (total activation elements across all layers for a single input).
3. Number of connections (sum over layers of weight count times number of input activation elements each weight connects to).

Repeat the three quantities for input spatial size doubled (64×64). Show each step and state assumptions (padding/stride/kernels).

Hyperparameter Tuning (25 pts)

Perform a wandb sweep exploring at least 20 configurations. Tune at minimum:

- Learning rate: $\{$1e-4, 3e-4, 1e-3, 3e-3$\}$.
- Batch size: $\{32, 64, 128\}$.
- Number of filters (NF): $\{8, 16, 32\}$.
- Kernel size: $\{3, 5\}$ (adjust padding).
- Optimizer: SGD / Adam.

For each run log:

- Hyperparameters, train/val loss curves, best validation loss, example images, and best checkpoint.
- After the sweep, report the best run (hyperparameters and validation loss) and include the wandb run URL in your writeup.

Q3: Trees and Random Forests (75 Marks)

Overview

In this question, you will implement and analyze decision trees and random forests for a sentiment classification task. You will complete coding tasks to build a decision tree from scratch, use Scikit-Learn to perform hyperparameter tuning, and interpret the learned models.

Dataset: Bag-of-words for Amazon Product Reviews

We will use a text sentiment classification task where each example is a plain-text online review from amazon.com. Our goal is to predict whether the sentiment is positive or negative from the text. The data has been preprocessed into a bag-of-words representation using a fixed vocabulary of 7729 terms.

- **Feature vector** $x_n \in \mathbb{R}^F$: A binary vector of size $F = 7729$, indicating which vocabulary terms are present in the review.
- **Label** $y_n \in \{0, 1\}$: A binary label indicating negative ($y_n = 0$) or positive ($y_n = 1$) sentiment.

The starter code repository includes a training set (6346 documents), a validation set (792 documents), and a test set (793 documents).

Part 1: Decision Tree Implementation from Scratch (20 Marks)

In this problem, you'll complete a from-scratch implementation of a Decision Tree. The starter code for this assignment can be found in the repository provided.

- **Code Task A:** Implement `predict` for a `LeafNode`. See the `LeafNode` class within starter code file: `tree_utils.py`.
- **Code Task B:** Implement `predict` for an `InternalDecisionNode`. See the `InternalDecisionNode` class within starter code file: `tree_utils.py`.
- **Code Task C:** Implement function to select the best binary split. See the starter code file: `select_best_binary_split.py`.
- **Code Task D:** Implement function to train the tree. See the starter code file: `train_tree.py`.

Part 2: Decision Trees for Review Classification (25 Marks)

We'll now examine decision trees for our sentiment classification problem using Scikit-Learn.

2.1 Train a Simple Tree and Visualize (10 Marks)

- Train a `DecisionTreeClassifier` with `criterion='gini'`, `max_depth=3`, `min_samples_leaf=1`, `min_samples_split=2` and `random_state=101`.
- Show the ASCII-text representation of your trained tree by calling the helper function `pretty_print_sklearn`.
- Is there any internal node that has two child leaf nodes corresponding to the same sentiment class? Why would having two children predict the same class make sense?

2.2 Hyperparameter Tuning (15 Marks)

- Perform a grid search for the hyperparameters of `DecisionTreeClassifier` over the following settings. Use `sklearn.model_selection.GridSearchCV` with `scoring = 'balanced_accuracy'`, the provided validation split (`cv=my_splitter`), `return_train_score=True`, and `refit=False`.
  - `max_depth` in `[2, 8, 32, 128]`
  - `min_samples_leaf` in `[1, 3, 9]`
  - `random_state` in `[101]`
- What are the values of `max_depth` and `min_samples_leaf` for the best tree found by the grid search?

Part 3: Random Forests for Review Classification (30 Marks)

We'll now see if an ensemble of decision trees can improve performance.

3.1 Train a Simple Random Forest and Analyze Features (10 Marks)

- Train a `RandomForestClassifier` with `max_depth=3` and `n_estimators=100` (and other hyperparameters from the starter code).
- Access the `feature_importances_` attribute of your trained forest.
- Create a two-panel table. The left panel should list the top 10 vocabulary words with the highest feature importance. The right panel should list 10 randomly chosen terms with near-zero importance (importance ¡ 0.00001), and state how many such terms were eligible.

3.2 Hyperparameter Tuning for Random Forest (10 Marks)

- Perform a grid search for `RandomForestClassifier` over the following settings:
  - `max_features` in `[3, 10, 33, 100, 333]`
  - `max_depth` in `[16, 32]`

  - `min_samples_leaf` in [1]
  - `n_estimators` in [100]
  - `random_state` in [101]

- What is the value of `max_features` of your best forest?
- What is the maximum possible value for `max_features` for this dataset? Why is it beneficial to tune this hyperparameter?
- When fitting random forests, what is the primary tradeoff controlled by the `n_estimators` hyperparameter? Can you overfit by setting it to be too large? Why or why not?

3.3 Final Model Comparison (10 Marks)

- Create a table summarizing the balanced accuracy for all four models developed (Simple DT, Best DT, Simple RF, Best RF). The table should have columns for model type, number of trees, max depth, and balanced accuracy on the train, validation, and test sets.
- Summarize your conclusions.

Good luck with the assignment