Fall 2025
ECE 236A
Professor: Christina Fragouli
TAs: Merve Karakas and Xinlin Li

ECE236A: Linear Programming

Decentralized Classification under Feature Compression

Due: Monday, Dec 1st, 2025, before 11am.

To be completed by teams of 3-4 people.

Project Description:

In this project, you will design and analyze **linear-programming-based** methods for multi-class classification when *four sensors* each observe *one quarter of an image* and must *quantize/compress* what they send to a fusion center using a limited amount of bits. You will (i) formulate and implement LP/ILP models for classification and bit allocation, (ii) compare *decentralized* vs. *centralized* compression, and (iii) study the "inverse" problem: meeting a *target accuracy* with minimal bits (and how to allocate those bits across sensors). This is an open-ended project, which means there may be some questions with no "right" answers, and that we appreciate innovative design ideas. Your main constraint is that you need to use Linear Programming formulations.

You can work in teams of three (preferred) or four people. The project is due on **Monday, Dec 1st before 11 am** and needs to be uploaded on Gradescope.

1 Background

Many modern sensing systems increasingly operate at the *edge*: phones, cameras, and embedded devices collect rich signals but many times they need to transmit them to a decision center over communication links that have limited capacity. Data transmission heavily consumes the sensors battery, and thus, both for reasons of battery efficiency and communication constraints, sending full precision data can be highly impractical. A natural approach is to *compress* or *quantize* signals before sending them to a decision center so that only a small number of bits per sample are transmitted. This immediately raises core design questions: How many bits should we spend in total? How should we allocate them across different sensors or regions? And how does this choice affect the downstream task performance, e.g., *classification* accuracy for a classification task? In this project, we explore these questions in a setting where an image is observed either in a *decentralized manner* (four sensors, each seeing one quadrant) or *centrally* (a single encoder sees the whole image), and we study how compression/quantization and bit allocation affect multi-class classification through the lens of Linear Programming (LP/ILP).

1.1 Classification

Classification is a fundamental task in *supervised* machine learning where the goal is to map a given set of data points to predefined categories or classes. The input data can be represented as a dataset $X \in \mathbb{R}^{N \times M}$ containing N samples of dimension M, with rows $x^{(i)} \in \mathbb{R}^M$. Associated with each data point $x^{(i)}$ is a label $y^{(i)} \in \mathcal{Y}$ indicating its class or category; $\mathcal{Y}$ consists of discrete values representing the classes, e.g., it can be an enumeration $\{1, 2, \ldots, K\}$ for K-class classification. The objective of classification is to learn a predictive model or classifier, typically denoted as $f : \mathbb{R}^M \to \mathcal{Y}$, that maps the input data to its corresponding class labels, i.e., $\hat{y} = f(x)$ where $\hat{y}$ is an estimate of y. The classification model $f(\cdot)$ can be generally decomposed as $f(\cdot) = h\big(g(\cdot)\big)$, where $g(\cdot)$ is a transformation function that extracts information from the input vector x, and $h(\cdot)$ is a decision function that takes the output of $g(\cdot)$ and makes the final decision regarding the label estimate of y.

You will work with *linear* classifiers in this project. A linear classifier is a type of classifier for which the function $g(\cdot)$ is affine, i.e., for each class k,

$$g_k(x) = w_k^\top x + b_k, \qquad w_k \in \mathbb{R}^M,\ b_k \in \mathbb{R},$$

and the predicted label is $\hat{y} = \arg\max_{k \in \{1,\ldots,K\}} g_k(x)$.

Assessing Performance. A good classifier correctly classifies as many input vectors as possible. One common way to measure the quality of a classifier is the percentage of correctly classified points, namely *classification accuracy*. Specifically, let $\{(x^{(i)}, y^{(i)})\}_{i=1}^N$ denote a dataset. We define

$$P(X, Y) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\Big\{\hat{y}^{(i)} = y^{(i)}\Big\},$$

where $\mathbf{1}\{\cdot\}$ is the indicator function and $\hat{y}^{(i)} = f\big(x^{(i)}\big)$.

Training a Model. The classifier is trained on a labeled training set $(X_{\text{train}}, Y_{\text{train}})$ to determine parameters (such as w_k, b_k, and the decision rule h) that maximize performance on the training data (e.g., training accuracy $P(X_{\text{train}}, Y_{\text{train}})$). Once trained, it is used to predict labels of new, unlabeled data points. The premise is that if the training set is a good representative of each class, the classifier will generalize well on unseen data and achieve similar accuracy.

Centralized vs. Decentralized Sensing. In many sensing systems, an input vector $x \in \mathbb{R}^M$ may be acquired either by a single entity with access to all features (*centralized*) or by multiple entities that each observe only a subset of features (*decentralized*). In a decentralized system with S sensors, sensor $s \in \{1, \ldots, S\}$ observes only its local feature block $x^{(s)} \in \mathbb{R}^{d_s}$ such that $M = \sum_{s=1}^S d_s$. The local observations are typically sent to a *decision center*, which performs downstream tasks such as classification. In a centralized system, a single sensor observes the entire x.

In this project, we assume that our communication channel between the decision center and sensors (in both centralized and decentralized settings) is severely constrained. For each observation (an

image), only a few bits can be transmitted, therefore, the local image features need to be compressed before transmission. In the decentralized case, we consider $S = 4$ sensors, each observing one non-overlapping quadrant of an image. In the centralized case, a single sensor observes the full image, but still need to compress it before classification.

Centralized access can exploit all cross-feature correlations when forming a representation, while decentralized access may be constrained by sensor boundaries and communication limits. These architectural choices affect both achievable performance and how we model information flow.

Quantization. Quantization is a fundamental technique for representing real-valued features $x \in \mathbb{R}^M$ with a limited number of bits (i.e., one of 2^b number of discrete values). Scalar quantization treats each coordinate independently. At its simplest, given $x_m \in \mathbb{R}$ with training range $[x_m^{\min}, x_m^{\max}]$ and bit-depth $b \in \mathbb{Z}_{\geqslant 0}$, *uniform scalar quantization* uses the step Δ defined as

$$\Delta_m \;=\; \frac{x_m^{\max} - x_m^{\min}}{2^{\,b} - 1},$$

and maps x_m to the nearest grid level (with clipping at the end of the range), i.e.,

$$Q_b(x_m) = \text{Round}(\frac{x}{\Delta_m})\ \Delta_m.$$

The componentwise rounding error obeys $\left|x_m - Q_b(x_m)\right| \leqslant \Delta_m/2$.

Although uniform scalar quantization is simple and efficient, it ignores possible structure in the data. Vector quantization (VQ) instead groups the values into vectors and maps each vector to the nearest representative in a codebook. The codebook is typically learned from the data and captures common patterns or clusters. This allows each code index to represent a whole block of correlated values.

In both cases, the core idea is to replace high-precision numerical data with compact, discrete representations that require only a few bits to store or transmit.

2 Dataset

In this project, you will validate your algorithms and implementations on two datasets:

1. **Synthetic Dataset:** We generated $N = 1000$ samples with dimension $M = 2$. Samples are drawn from one of three multivariate Gaussian distributions. Labels $y^{(i)} \in \{0, 1, 2\}$ indicate which distribution $x^{(i)}$ belongs to.

2. **Fashion-MNIST[1] (reduced):** We will use a reduced version of the Fashion-MNIST dataset, which contains $N = 1000$ data samples from three classes. Each sample is a 28×28 image ($M = 784$) of a piece of clothing. The labels $y^{(i)} \in \{0, 3, 4\}$ represent:

[1] https://github.com/zalandoresearch/fashion-mnist

- 0: T-shirt/top
- 3: Dress
- 4: Coat

Note. This section only specifies which datasets you will use. You do not need to write data-loading code; you will call the provided utility functions to obtain the processed datasets.
Important. The synthetic 2-D dataset is provided only for visualization and intuition building (e.g., to display classifier decision regions for Task 1, and optionally to sanity-check centralized compression in Task 2). It is *not* suitable for the decentralized tasks (Tasks 3.1–3.3), which rely on multi-sensor image quadrants. All decentralized experiments should therefore be performed on the Fashion-MNIST dataset.

3 Project Goal and Details

For each task, you must design your own **Linear Program (LP) or Integer Linear Program (ILP)**. In your report, for every task: (i) provide a concise *Formulation* (decision variables, constraints, objective) without code; (ii) state *assumptions* (e.g., do you quantize features, any normalization); (iii) if you relax an ILP to an LP, *justify the relaxation* and explain your rounding. Keep train/validation/test strictly separate, fix random seeds, and report units (bits per image).

- **Task 1: Build Your Classifier**
  Formulation. Design an LP/ILP that trains a multi-class linear classifier on features received at the decision center. You may choose any linearizable margin/loss/regularization you can justify. Do not assume a specific template from us.

  Implementation & Evaluation. To evaluate the quality of your classifier, test it on the original (non-quantized) features. This also provides a baseline accuracy for Task 2.

  - *Metric:* test accuracy on the 3-class Fashion-MNIST subset (this is the metric used for grading in Task 1).
  - *Protocol:* use the provided fixed split. Tune hyperparameters on validation only.
  - *Plots:* one bar (or point) showing Task 1 test accuracy; optionally add a 2-D projection (e.g., PCA) with decision regions for intuition (not graded).
  - *Datasets:* report Task 1 results on **both** datasets (Synthetic and Fashion-MNIST). The 2-D synthetic set is encouraged for visual intuition (e.g., decision boundary).

- **Task 2: Build Your Quantizer**
  Formulation. Assume the centralized setting, where a single sensor observes the whole image. Design LP/ILP(s) to quantize each image to a B_{tot}-bit representation. You can either use the given naive uniform scalar quantization scheme and optimized bit-allocation across features or design your own vector quantization scheme.

Implementation & Evaluation.

- *Metric:* test accuracy vs. quantization budget B_{tot} (centralized).
- *Comparison:* compare the test accuracies using quantized features with Task 1 accuracy.
- *Plots:* a single figure overlaying both settings across B_{tot}.
- *Dataset:* perform Task 2 on the **Fashion-MNIST** dataset. (Optionally, you may also demonstrate centralized compression on the Synthetic dataset for illustration.)

- **Task 3: Feature Compression**

  In this task, we move to the decentralized setting, where each sensor observes a quadrant of a image, which will be quantized independently and transmitted to the decision center.

  All experiments in Task 3 (3.1–3.3) should be conducted on the Fashion-MNIST dataset; the 2-D synthetic set is not applicable to the quadrant/sensor setting.

  Task 3.1 Fixed Per-sensor Budget (k bits per image per sensor)

  [Optional if there are < 3 members in your group.]

  Formulation. For each sensor, solve an LP/ILP to quantize the observed image features into k-bit representation. You can either reuse the LP/ILP from Task 2 or develop new programs if needed, this choice would not affect your grade.

  Implementation & Evaluation.

  - *Metric:* test accuracy vs. per-sensor budget k (several k values spanning practical b_s).
  - *Comparison:* overlay centralized and decentralized curves at matched B_{tot}; discuss the centralized upper-bound intuition.
  - *Plots:* accuracy (y-axis) vs. k (x-axis), showing your method and baseline; include the chosen b_s values per point in the caption or a small table.

  Task 3.2 Fixed total budget (B_{tot} across sensors)

  Formulation. Given a total budget B_{tot}, determine a bit-allocation (b_1, b_2, b_3, b_4) over sensors with $\sum_s b_s \leqslant B_{\text{tot}}$. You may use an outer search over budgets/allocations with inner LP/ILP solves, or encode the choice directly in an ILP/MILP; justify your approach.

  Implementation & Evaluation.

  - *Metric:* test accuracy vs. B_{tot}.
  - *Reporting:* for each B_{tot}, report the selected allocation $(b_1, \ldots, b_4)$.
  - *Comparison:* overlay centralized and decentralized curves at matched B_{tot}; discuss the centralized upper-bound intuition.
  - *Plots:* accuracy (y-axis) vs. B_{tot} (x-axis) with labels/legend indicating the chosen allocations.

Task 3.3: Feature Compression with a Target Accuracy

[Optional if there are < 4 members in your group.]

Formulation. Given a target accuracy, e.g., $\alpha \in \{70\%, 80\%, 90\%\}$, determine a bit allocation that achieves it. Do this (i) in the decentralized setting (minimize B_{tot}, choose $(b_1, \ldots, b_4)$ with $\sum_s b_s \leqslant B_{\text{tot}}$), and (ii) in the centralized setting (minimize B_{tot}). You may use an outer search over budgets/allocations with inner LP/ILP solves, or encode the choice directly in an ILP/MILP; justify your approach.

Note: You are allowed to change the target accuracy range to get more meaningful and comparable plots; however, your algorithm should work for any target accuracy.

Implementation & Evaluation.

- *Metric:* minimal total bits to reach each α, and the corresponding allocation: decentralized $(b_1, \ldots, b_4)$, centralized b (or blockwise).
- *Plots:* minimal bits (y-axis) vs. target accuracy α (x-axis) for decentralized and centralized. You may plot both training accuracy and test accuracy, since the test accuracy might not reach the target ones.
- *Discussion:* briefly interpret the gap between decentralized and centralized results.

What to include for each item.

- **Formulation:** decision variables (e.g., u, v, b, ξ; and t if you use a robust LP), constraints, objective.
- **Assumptions:** feature scaling; how your quantizer is defined (e.g., scalar, companded, vector/codebook) and how you count bits per image; validation protocol used to pick (b_s) or b or other hyperparameters; strict train/val/test separation (no test leakage).
- **Relaxations:** any ILP→LP relaxation; rounding scheme to obtain integral decisions if used.
- **Plots:**
  1. **Task 1 (both datasets):** one bar (or point) per dataset showing Task 1 test accuracy (Synthetic and Fashion-MNIST).
  2. **Task 2 (Fashion-MNIST):** accuracy (y) vs. total budget B_{tot} (x); we recommend overlaying the Task 1 baseline as a dashed line.
  3. **Task 3.1 (Fashion-MNIST):** accuracy (y) vs. per-sensor budget k (x); also report the chosen $(b_1, \ldots, b_4)$ or equivalent per-sensor allocation for each point.
  4. **Task 3.2 (Fashion-MNIST):** accuracy (y) vs. B_{tot} (x), and the selected allocation per point; provide a centralized vs. decentralized overlay at matched B_{tot}.
  5. **Task 3.3 (Fashion-MNIST):** minimal total bits (y) vs. target accuracy α (x) for centralized and decentralized.

4 Implementation Requirements

Please download `project_code.zip` from Bruin Learn. It contains:

- `utils.py` (do not modify): utility functions for data loading, preprocessing, sensor splits, and plotting.
  - `prepare_synthetic_data()` – returns the small 2-D, 3-class synthetic dataset with fixed train/val/test splits (for optional visualization and Task 1 intuition).
  - `prepare_mnist_data()` – returns the reduced 3-class Fashion-MNIST subset (flattened 28×28 images) with fixed train/val/test splits.
  - `split_into_quadrants(X)` – splits each Fashion-MNIST image into four non-overlapping quadrant blocks (sensors); returns index sets $\{\mathcal{I}_s\}_{s=1}^{4}$ and the corresponding block views.
  - `plot_result_per_sensor(result_dict)` – Task 3.1: plots accuracy vs. per-sensor budget k.
  - `plot_result_total(result_dict)` – Task 2 / Task 3.2: plots accuracy vs. total budget B_{tot} and can annotate chosen $(b_1, \ldots, b_4)$.
  - `plot_result_centralized_vs_decentralized(result_dict)` – overlays centralized and decentralized accuracy curves at matched B_{tot}.
  - `plot_result_target(result_dict)` – Task 3.3: plots minimal bits vs. target accuracy α for both settings.

  You do not need to write data-loading or plotting code. Each plotting function expects a small dictionary; the required keys are documented in the comments inside `utils.py`. **Do not** change function signatures or internal code in `utils.py`.

- `Data.zip`: contains the synthetic 2-D dataset and the reduced 3-class Fashion-MNIST dataset. Unzip before calling the loaders.

- `MySolution.py`: a skeleton you will complete. It contains three classes you must implement:
  - `MyDecentralized` – for **Decentralized Classification** (no compression).
  - `MyFeatureCompression` – for **Feature Compression**: **(Task 3.1)** fixed per-sensor budget k, **(Task 3.2)** fixed total budget B_{tot}, and the **Task 2** centralized baseline.
  - `MyTargetAllocator` – for **Target-accuracy bit allocation** (minimal bits to reach α).

  Submit your file as `MySolution_{groupnumber}.py`. You may add helper methods or auxiliary files, but keep the provided class/method names and signatures intact so we can run the autograder.

Notes.

1. **Language & libraries.** Implement in Python. You may use CVXPY, PuLP, OR-Tools, GLPK/CBC, Gurobi, Mosek, etc. If you use a commercial solver, your code must also run with a free alternative.

2. **Do not modify utilities.** Do not change `utils.py` or the dataset split. You may add auxiliary helper functions or classes in your submission.

3. **No test leakage.** Do not use `X_test` or `Y_test` during training, model selection, or bit-allocation search. Test labels are used only by the provided evaluation helpers.

4. **Reproducibility.** Fix random seeds. Keep train/val/test strict. Log solver tolerances/time limits, search grids for (b_s) or b, and any preprocessing flags (e.g., normalization)—and ensure all budget comparisons are fair.

5 Report

Beyond the code, we also expect a report of up to 3 pages (excluding figures, appendix and references), that describes in a complete way what is the rationale you used to to formulate the LPs/ILPs as well as the specific algorithm descriptions. Be concise and clear about your algorithm definitions and rationale behind. Put the required figures and discuss your observations. You are welcome to add extra plots in the appendix if they help with your illustration.

6 Grading

- 15 points: This project will be graded mainly based on LP formulations, and completeness of results. We will grade based on if your formulation is actually a linear program, how you justified the algorithms you came up with, and your ability to get results by implementing those algorithms.

- 5 bonus points: Bonus points up to 5 points will be given to at most 5 groups based on the creativity and performance of your proposed algorithms.

- You need to submit a folder named **'group_{groupnumber}'** (group_5 for the group with name group 5) on Gradescope. Inside this folder there should be a file named **MySolution_{groupnumber}.py** as well as your report and experiment files (where you run the simulations).