Machine Learning and Applications
Fall 2025
Final Project Instructions

Deadline: Monday, 17th November 2025. If you choose Option 2, you must submit a proposal by **1st November 2025**. No proposal is needed for Option 1.

Submission: For Option 1, you need to submit the following files through the LMS:

- `final_report.pdf`: Your answers to Part A and B (or Option 2 report), as a PDF file. You must type your final report in LaTeX.
- For Option 1: the completed Python code used for Part A: `knn.py`, `item_response.py`, `matrix_factorization.py`, `neural_network.py`, and `ensemble.py`.
- For Part B: submit your code as a zip file `code.zip`. If you choose Option 2, include the relevant code in `code.zip` as well.
- `llm.pdf`: A document describing how (or if) you used any large language model during this assignment. Please specify:
  1. Which model you used (e.g., ChatGPT, GPT-4, Claude, Bard, etc.).
  2. What prompts you used with the model (most models keep a history of interactions).
  3. You do **not** need to submit the output of the model.

Late Submission: 10% of the marks will be deducted for each day late, up to a maximum of 3 days. After that, no submissions will be accepted.

Collaboration: Your final report should list the contributions of each team member. **Only one member** of each group should submit the final project.

1 Introduction

The main objective is to prepare you to apply machine learning algorithms to real-world tasks. This project involves:

- Applying algorithms to real data
- Modifying algorithms to improve performance
- Writing an analytical report

The final project is not intended to be a stressful experience. It is a good chance for you to experiment, think, play, and hopefully have fun. These tasks are similar to what you may be doing daily as a data analyst/scientist or machine learning engineer.

2 Background & Task

Online education platforms (e.g., Khan Academy, Coursera) enable wide access but make it hard to measure student understanding. Many platforms use diagnostic multiple-choice questions to reveal misconceptions and inform targeted support.

In this project, you will **build algorithms to predict whether a student can answer a specific diagnostic question correctly**, given their past answers and those of other students. This prediction supports adaptive learning.

You will:

- Apply machine learning algorithms from the course
- Compare, analyze, and modify algorithms for accuracy
- Experiment and report on modifications

Prediction accuracy will be the main metric:

$$\text{Prediction Accuracy} = \frac{\text{Number of correct predictions}}{\text{Number of total predictions}}$$

3 Data

You are provided a subset of answers by 542 students to 1,774 diagnostic questions from Eedi[1], an online educational platform.

3.1 Primary Data

- `train_data.csv`: Main training data
- `valid_data.csv`: For model selection
- `test_data.csv`: For final reporting

Each CSV has:

- `question id`: question answered (starts at 0)
- `user id`: student ID (starts at 0)
- `is correct`: 0 (incorrect), 1 (correct)

A `sparse_matrix.npz` is also provided: each row is a user, each column a question. Entries are 1 (correct), 0 (incorrect), or NaN (missing/held-out).

3.2 Question Metadata

- `question_meta.csv`:
  - `question id`, `subject id`; subject descriptions in `subject_meta.csv`

3.3 Student Metadata

- `student_meta.csv`:
  - `user_id`, `gender` (1=female, 2=male, 0=unspecified), `date_of_birth`, `premium pupil` (financially disadvantaged, if available)

[1] `https://eedi.com/`

4 Part A

Apply and analyze algorithms to predict student responses. Use only the primary data (`train_data.csv`, `sparse_matrix.npz`, `valid_data.csv`, `test_data.csv`). You may use `utils.py` helpers, and `NumPy, SciPy, Pandas, PyTorch`. Understand the code you use.

1. [5pts] k-Nearest Neighbors (`knn.py`)

In this question, you will experiment with the k-nearest neighbors (KNN) algorithm for collaborative filtering, using the starter code provided in `knn.py`. The KNN algorithm is a classical method for collaborative filtering and is based on the intuition that similar students will have similar response patterns.

The provided KNN code performs user-based collaborative filtering, which uses other students' answers to predict whether a specific student can correctly answer a given diagnostic question. The core underlying assumption is that if student A has the same correct and incorrect answers on other diagnostic questions as student B, then A's correctness on specific diagnostic questions is likely to match that of B.

(a) **User-based KNN:** Complete the function `user_knn_predict_hanu(matrix, valid_data, k, return_confusion=False)`. This function performs user-based KNN imputation and returns the validation accuracy. If `return_confusion=True`, it should also return the confusion matrix for the predictions (you may use `sklearn.metrics.confusion_matrix`). For each $k \in \{1, 6, 11, 16, 21, 26\}$, report the validation accuracy and show the confusion matrix. Identify and clearly state which k (`best_k`) gives the highest validation accuracy.

(b) **Item-based KNN:** Complete the function `item_knn_predict_hanu(matrix, valid_data, k, student_id="")`. This function performs item-based KNN imputation and returns the validation accuracy. Additionally, it must save the array of predicted values for the validation set as a file named {`student_id`}`_item_knn_preds.npy`.
For each $k \in \{1, 6, 11, 16, 21, 26\}$, report the validation accuracy, and save the predictions as instructed. Identify which k gives the best performance.

(c) **Evaluation and Metrics:** In addition to reporting accuracy, for at least one value of k (ideally the best k), compute and report the ROC-AUC score using `sklearn.metrics.roc_auc_score`. Include this in your report.

(d) **Summary and Reflection:** After you have completed your experiments, include in your report a brief summary statement (1–2 sentences) reflecting on KNN performance, and any patterns you observe about how accuracy or confusion matrix entries change as k varies. *This must be written in your own words and cannot be copied from any external source.*

2. [15pts] Item Response Theory (`item_response.py`)

In this problem, you will implement an Item-Response Theory (IRT) model to predict students' correctness on diagnostic questions.

The IRT model assigns each student an ability parameter θ_i and each question a difficulty

parameter β_j. The probability that student i correctly answers question j is given by:

$$p(c_{ij} = 1|\theta_i, \beta_j) = \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}$$

where c_{ij} is the binary indicator of correctness (1 for correct, 0 for incorrect).

Let C denote the sparse matrix of observed responses, where each entry C_{ij} represents whether student i answered question j correctly (1), incorrectly (0), or is missing (unobserved).

(a) **Model Architecture:** Complete the `AutoEncoder` class as defined in `neural_network.py`. The forward pass should use sigmoid activations for both layers.

(b) **Model Training:** Implement the `train(model, lr, lamb, train_data, zero_train_data, valid_data, num_epoch)` function to train your model, including an L2 regularization term in the cost function. Tune the hyperparameters (`lr`, `lamb`, `num_epoch`, and k (latent dimension)). For $k \in \{10, 50, 100, 200, 500\}$, report validation accuracy for each value and select the best k^*.

(c) **Reporting and Plots:** For your selected k^* and regularization parameter λ, plot the training cost and validation accuracy as a function of epoch. Also, report the final test accuracy. Save your main results plots as `autoencoder_results_{student_id}.png`.

(d) **Reflection:** Briefly discuss (1–2 sentences) how regularization affected your model performance and what you observed about the choice of k.

3. [15pts] Matrix Factorization OR Neural Networks

In this question, you are asked to read both options (i) and (ii), but you only need to complete **one** of the two for submission.

(i) **Option 1: Matrix Factorization (`matrix_factorization.py`)**

In this part, you will implement, train, and analyze matrix factorization techniques to predict missing student responses. Matrix factorization is a powerful method for collaborative filtering and is widely used in recommender systems and educational data mining.

Model Description:

- The student response matrix $C \in \mathbb{R}^{N_{\text{users}} \times N_{\text{questions}}}$ is assumed to be low-rank and can be approximated as the product of two low-dimensional matrices:

$$C_{nm} \approx u_n^\top z_m$$

  where $u_n \in \mathbb{R}^k$ is the latent factor vector for student n, and $z_m \in \mathbb{R}^k$ is the latent factor vector for question m. The parameter k is the chosen latent dimension.
- The goal is to learn $U \in \mathbb{R}^{N_{\text{users}} \times k}$ and $Z \in \mathbb{R}^{N_{\text{questions}} \times k}$ such that their product approximates the observed entries of C as closely as possible.

Objective Function:

- You will minimize the regularized squared error loss over the observed (non-missing) entries:

$$\mathcal{L}(U, Z) = \frac{1}{2} \sum_{(n,m) \in \mathcal{O}} \left(C_{nm} - u_n^\top z_m\right)^2 + \frac{\lambda}{2} \left(\|U\|_F^2 + \|Z\|_F^2\right)$$

where $\mathcal{O}$ is the set of observed (user, question) pairs, and λ is the regularization parameter.

Tasks:

(a) **SVD Initialization (Code):** Implement the function `svd_reconstruct_hanu(matrix, k, zero_mask=False)` to compute a rank-k SVD approximation of the observed matrix C. Use the validation set to select the best k. Report validation accuracy for each $k \in \{10, 50, 100, 200, 500\}$, and clearly state the k^* that achieves the highest validation accuracy. *Note: If missing values are present, consider zero-imputation or mean-imputation before SVD.*

(b) **ALS with Regularization (Code):** Implement the alternating least squares (ALS) method using stochastic gradient descent to minimize the loss above. You should alternate between updating all user vectors u_n and all item vectors z_m using gradient steps. Complete the functions `update_u_z(train_matrix, valid_data, k, lr, lambda_, num_epoch, student_id="")` and `als(train_matrix, valid_data, k, lr, lambda_, num_epoch, student_id="")`. Your implementation should include the **L2 regularization** term in the gradients.

(c) **Hyperparameter Tuning (Experiment and Report):** For each $k \in \{10, 50, 100, 200, 500\}$, tune the learning rate (`lr`), regularization parameter (`lambda_`), and number of epochs (`num_epoch`). For each configuration, report both the validation and test accuracy. Clearly indicate which combination gives the best validation performance.

(d) **Plotting and Result Analysis:** For your selected k^* and λ, plot the training loss and validation accuracy as functions of epoch. Save your main result plot as `mf_results_{student_id}.png` (where `student_id` is your student ID, passed as an argument to your ALS function). Compare your ALS results to your SVD baseline and to the neural network/autoencoder results, discussing any significant differences in performance or convergence.

(e) **Reflection (Short Written Answer):** In 2–3 sentences, discuss:

- The effect of regularization on the learned factors and validation accuracy.
- Any trends observed as k increases (e.g., overfitting, underfitting, or interpretability).
- One potential limitation of using matrix factorization for this data.

Hints:

- When implementing ALS, make sure to only use the observed entries in the loss and gradient calculations.
- Plot your learning curves (loss/accuracy vs. epoch) to validate your implementation.
- Test your code on a small matrix to debug before scaling up.
- SVD can only be applied to matrices with no missing entries; use imputation if needed.
- Label all axes and include legends in your plots.

(ii) **Option 2: Neural Networks (`neural_network.py`)**

In this part, you will implement, train, and analyze an autoencoder neural network to predict missing entries in the student response matrix. An autoencoder is a type of neural network used for unsupervised representation learning, often for dimensionality

reduction or missing data imputation.

Model Description:

- The autoencoder consists of two linear layers with sigmoid activations.
- The input to the network is a student's response vector $v \in \mathbb{R}^{N_{\text{questions}}}$, where $N_{\text{questions}}$ is the number of questions and missing entries may be set to zero.
- The encoder maps v to a lower-dimensional representation $h \in \mathbb{R}^k$, where k is a chosen latent dimension:
$$h = \sigma(W^{(1)}v + b^{(1)})$$
where $W^{(1)} \in \mathbb{R}^{k \times N_{\text{questions}}}$, $b^{(1)} \in \mathbb{R}^k$, and σ is the sigmoid activation function.
- The decoder reconstructs the input:
$$\hat{v} = \sigma(W^{(2)}h + b^{(2)})$$
where $W^{(2)} \in \mathbb{R}^{N_{\text{questions}} \times k}$ and $b^{(2)} \in \mathbb{R}^{N_{\text{questions}}}$.
- The reconstructed vector $\hat{v}$ represents the predicted probabilities of correct responses for each question.

Objective Function:

- The loss function to minimize is the sum of squared reconstruction errors over all students, plus an L2 regularization term:
$$\mathcal{L}(\theta) = \sum_{v \in S} \|v - f(v;\theta)\|_2^2 + \frac{\lambda}{2}\left(\|W^{(1)}\|_F^2 + \|W^{(2)}\|_F^2\right)$$
where $f(v;\theta)$ denotes the output of the network, and $\|\cdot\|_F$ denotes the Frobenius norm. The regularization parameter λ helps prevent overfitting.

Tasks:

(a) **Model Implementation (Code):** Complete the `AutoEncoder` class, implementing both the encoder and decoder with sigmoid activations as described above. Your implementation should match the forward pass mathematically.

(b) **Training with Regularization (Code):** Implement the function `train(model, lr, lamb, train_data, zero_train_data, valid_data, num_epoch, student_id="")` to optimize the model parameters using stochastic gradient descent (or Adam optimizer). Your loss should include the L2 regularization term described above. *Tip: Use PyTorch's optimizers and make sure gradients are computed and applied correctly.*

(c) **Hyperparameter Tuning (Experiment and Report):** Train your model for different values of the latent dimension $k \in \{10, 50, 100, 200, 500\}$, and tune the learning rate (`lr`), regularization parameter (`lamb`), and number of epochs (`num_epoch`). For each value of k, record and report the validation accuracy. Clearly indicate which k^* yields the highest validation accuracy and is selected for further analysis.

(d) **Result Analysis (Plot and Interpretation):** For your selected k^* and best λ, plot both the training loss and validation accuracy as functions of the epoch number (on the same or separate plots as needed). Save your main results plot as `autoencoder_results_{student_id}.png`, where `student_id` is your student ID (this should be passed to your `train` function and used in the filename).

(e) **Test Accuracy and Model Selection:** Report the final test accuracy using your selected k^* and hyperparameters. Compare your best model's performance to other models you tried.

(f) **Reflection (Short Written Answer):** In 2–3 sentences, discuss:
- How regularization (λ) affected your model's performance.
- Any patterns or trends you observed as the latent dimension k increased (e.g., overfitting or underfitting).

Hints:
- Use appropriate weight initialization for neural networks.
- When plotting, label your axes and legend clearly.
- Validate your code by checking the reconstruction error decreases during training.
- Refer to the starter code comments for more guidance on usage.

4. [15pts] Ensemble (`ensemble.py`)

In this problem, you will implement a bagging ensemble to improve the stability and accuracy of your base models.
- Select and train 3 base models (these can be the same type or different types of models), using bootstrapped samples of the training set (i.e., each model is trained on a different random sample drawn with replacement).
- For each prediction, generate 3 predictions using the trained base models, and average the predicted correctness to produce the final prediction.
- Report the final validation and test accuracy of the ensemble.
- Clearly explain the ensemble process you implemented. Discuss whether the ensemble achieved better performance compared to individual base models, and provide a possible explanation for your result.

Part B

In the second part of the project, you are required to modify one of the algorithms you implemented in Part A in an effort to improve the accuracy of predicting students' answers to the diagnostic questions. Specifically, analyze the results obtained in Part A, reason about what factors are limiting the performance of one of the methods (e.g., overfitting, underfitting, optimization difficulties), and propose a modification to the algorithm that could help address this problem. Rigorously test the performance of your modified algorithm, and write up a report summarizing your results as described below.
- You will not be graded on how well the algorithm performs (i.e., its accuracy), but rather on the quality of your analysis.
- You may optionally use the provided metadata (`question_meta.csv` and `student_meta.csv`) to improve your model.
- You are free to use third-party ideas or code, provided it is publicly available and properly referenced in your write-up.

- The length of your report for Part B should be 3–4 pages. Use figures to illustrate your points where helpful.

The following guidelines and marking schemes apply:

1. [**15pts**] **Formal Description:** Provide a precise, formal definition of your proposed algorithmic extension. Clearly define any new equations and, if appropriate, include an algorithm box or pseudocode. Describe why your modification is expected to improve performance (e.g., improving optimization, reducing overfitting).
2. [**10pts**] **Figure or Diagram:** Include a clear figure or diagram that illustrates the overall model or idea. The aim is to make your report accessible, even to readers who are skimming.
3. [**15pts**] **Comparison or Demonstration:**
   - Compare the accuracy of your model to baseline models. Include a table or plot for illustrative comparison.
   - Based on your motivation, design and conduct an experiment to test your hypothesis. For example, test whether improvements are due to better optimization or regularization.
4. [**15pts**] **Limitations:**
   - Describe situations in which your approach is expected to perform poorly, or where all existing models fail.
   - Offer explanations for these limitations.
   - Suggest possible extensions or open problems that could address these limitations.

Option 2: Open Project (Custom Topic)

Timeline:

- **Proposal due:** 1st November 2025 (submit a half-page description of your proposed topic via LMS). No late proposals accepted.
- **Final project due:** 24th November 2025.

Your first task is to pick a project topic. If you need ideas, please attend office hours.

You may choose:

- **Application project:** Choose an application that interests you and explore how best to apply learning algorithms to solve it.
- **Algorithmic project:** Choose a family of problems and develop a new learning algorithm or a novel variant of an existing algorithm to solve it.

Your topic may combine applications, algorithms, and theory. Choose something you find motivating. Your grade will focus on care and rigor, not only on performance.

Report length: 3–5 pages. Suggested sections:

- Introduction (what is the context and motivation for the problem?)
- Literature Review (has anyone else solved a similar problem? How is your approach different? Cite at least three papers.)
- Problem Formulation (formally describe your problem and approach)
- Results (describe results, show figures/tables as needed, and discuss limitations and future work)

- Conclusions (main takeaways)

Rubric:

1. **Proposal [10%]:** Due 1st November 2025; describe problem and dataset(s).
2. **Quality [50%]:** Are claims well-supported by analysis or experiments?
3. **Clarity [20%]:** Is the report well-written, organized, and are figures/tables labeled?
4. **Originality [20%]:** Are the problem/approach or techniques novel? Is related work referenced?

General Advice

- **Read carefully:** Follow all guidelines. Use office hours or the course forum if uncertain.
- **Be honest:** You are graded on clarity and analysis, not just results.
- **Be careful:** Avoid testing on training data, setting hyperparameters by test accuracy, unfair comparisons, unlabeled axes, undefined symbols, etc. Check your work (e.g., multiple runs, gradient checking).

References

1. Wang, Zichao, et al. "Diagnostic Questions: The NeurIPS 2020 Education Challenge." *arXiv preprint arXiv:2007.12061* (2020).