CAP 4105 — Introduction to Machine Learning

Project 1: Linear Models

Topics: Linear Regression · Logistic Regression · Data Preprocessing

---

Project Overview

In this project, you will work with a student performance dataset to practice the complete machine learning pipeline: data preprocessing, model training, and evaluation. You will implement core algorithms **from scratch using NumPy** to solidify your understanding of the underlying mathematics, and then verify your results using **PyTorch**, the industry-standard deep learning framework.

This project uses **one unified dataset** that supports both **regression** (predicting exam scores) and **classification** (predicting pass/fail) tasks.

Logistics

Item	Detail
Team Size	1–3 students (individual or group, your choice)
Submission	Jupyter Notebook (`.ipynb`) uploaded to Canvas
Due Date	**February 15, 2026, 11:59 PM**

Environment Setup

You will need a Python environment with Jupyter Notebook support. Below are several recommended options — choose whichever you are most comfortable with. **We do not require a specific environment.**

Option 1: Google Colab (easiest to start)

- Go to colab.research.google.com, sign in with your Google account, and create a new notebook.
- Colab comes pre-installed with NumPy, Pandas, Matplotlib, and PyTorch — no setup needed.

- Upload `train.csv` and `test.csv` to your Colab session or mount Google Drive.
- When finished, download the `.ipynb` file via File → Download → Download .ipynb.

Option 2: VS Code + Jupyter Extension

- Install VS Code and the Jupyter extension.
- Create a Python virtual environment and install dependencies:

```
pip install numpy pandas matplotlib torch scikit-learn jupyter
```

- Create a new `.ipynb` file in VS Code and select your Python environment as the kernel.

Option 3: JupyterLab / Jupyter Notebook (classic)

- Install via pip:

```
pip install jupyterlab numpy pandas matplotlib torch scikit-learn
```

- Launch with `jupyter lab` and create a new notebook.

> **Tip:** Regardless of which environment you choose, make sure to **restart the kernel and run all cells from top to bottom** before your final submission. This ensures your notebook executes cleanly without hidden state issues.

Submission Instructions

1. Make sure your notebook has been **fully executed** — restart the kernel and run all cells from top to bottom before submitting. All outputs (tables, plots, printed values) must be visible.
2. Name your file: `Project1_LastName.ipynb` (for groups: `Project1_LastName1_LastName2.ipynb`).
3. Upload the `.ipynb` file to the **Canvas assignment** by the due date.
4. If your team has multiple members, **each member** should submit the same notebook on Canvas and include all team member names in the notebook's first Markdown cell.

Why NumPy + PyTorch?

This project requires you to implement algorithms at **two levels**. Each level serves a distinct educational purpose:

Level 1 — NumPy (from scratch): Implementing gradient descent manually forces you to translate mathematical formulas from the lecture slides into working code. You will compute predictions, loss values, and gradients step by step. This ensures you truly understand *what* the model is learning and *how* optimization works.

Level 2 — PyTorch (industry tool): PyTorch is the most widely used framework in industry and research for building ML/DL models. Its `autograd` engine computes gradients automatically, letting you focus on model design. By comparing your NumPy results with PyTorch, you verify correctness and learn the tool you will use throughout the rest of this course and in your career.

> **Note:** Your NumPy and PyTorch implementations should produce similar (not necessarily identical) results. Small numerical differences are expected due to floating-point precision and optimization details.

Dataset

You are provided with two CSV files:

- **`train.csv`** — Training data (~326 rows). This dataset may contain data quality issues that you need to identify and handle as part of the preprocessing step.
- **`test.csv`** — Test data (~80 rows). This is a clean held-out set for final evaluation only. **Do not use `test.csv` during training or preprocessing decisions.**

Refer to the separate **Dataset Description** document for column definitions. Note that the description provides column meanings but does **not** tell you which columns to use as features. Making that decision is part of your analysis.

Notebook Structure Requirement

Your Jupyter Notebook **must** be organized with the following Markdown headers, in this exact order. This structure is required for grading purposes. Each section should begin with a Markdown cell containing the section header.

```
# Project 1: Linear Models
# Author(s): [Name(s) and Student ID(s)]
# 0. Imports and Setup
# 1. Data Exploration & Preprocessing
## 1.1 Exploratory Data Analysis
## 1.2 Data Cleaning
## 1.3 Feature Selection
## 1.4 Data Normalization
# 2. Linear Regression
## 2.1 NumPy Implementation
## 2.2 PyTorch Verification
## 2.3 Polynomial Regression
# 3. Logistic Regression
## 3.1 NumPy Implementation
## 3.2 PyTorch Verification
## 3.3 Analysis & Visualization
# 4. Summary & Reflection
```

> **Important:** Use exactly these section headers. Do not rename, reorder, or merge sections. You may add sub-sections within each section if needed, but the top-level structure must remain intact. The first cell of your notebook must be a Markdown cell containing the project title and author information (name and Panther ID for each team member).

Part 1: Data Exploration & Preprocessing (25 points)

Task 1.1: Exploratory Data Analysis (EDA) — 5 pts

Before training any model, you must understand your data. Perform the following:

- Load `train.csv` and display basic statistics (shape, dtypes, `.describe()`, `.info()`).
- Identify and report any data quality issues you find. Describe each issue and its potential impact on model training.
- Visualize feature distributions (histograms or box plots) and examine feature–target relationships (scatter plots).

Expected outputs: Summary statistics, at least 3 visualizations, written description of data quality issues found.

Task 1.2: Data Cleaning — 8 pts

Based on your EDA findings, clean the training data. For each cleaning step, provide a brief justification for your approach:

- Handle any missing values.
- Handle any outliers.
- Handle any duplicate records.

> **Note:** There is no single correct answer for how to handle each issue. What matters is that your approach is **justified** — explain *why* you chose your method.

Expected outputs: Code for each cleaning step, before/after row counts, written justification for each decision.

Task 1.3: Feature Selection & Engineering — 5 pts

Not all columns in the dataset should be used as features for model training.

- Analyze which columns are appropriate input features and which are not. Provide reasoning for each inclusion/exclusion decision.
- Compute and visualize a correlation matrix to support your analysis.

Expected outputs: Correlation heatmap, explicit list of selected features with reasoning for each decision.

Task 1.4: Data Normalization — 7 pts

Apply normalization to your selected features:

- Apply **Z-score normalization** (standardization) to your features. Show the before/after statistics (mean, std).
- Apply **L2 (vector) normalization** to at least one feature set. Explain what this normalization achieves and when it is useful.
- Discuss: Why is normalization important for gradient descent? What could go wrong if you skip this step?

> **Important:** Fit normalization parameters (mean, std) on training data only, then apply the same transformation to test data.

Expected outputs: Before/after statistics, normalized data samples, written discussion of normalization importance.

Part 2: Linear Regression (30 points)

Target variable: `exam_score` (continuous). Use your cleaned, normalized features from Part 1.

Task 2.1: NumPy Implementation — 12 pts

Implement linear regression with gradient descent from scratch using **only NumPy**:

- Implement the **MSE cost function**: $J(\beta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$
- Implement the **gradient computation** for all parameters (refer to the gradient formulas from lecture).
- Implement the **gradient descent update rule** with a configurable learning rate.
- Train the model and **plot the loss curve** (MSE vs. iteration number).
- Report the final learned parameters and the final training/test MSE.

Expected outputs: Working gradient descent code, loss curve plot, printed final parameters and MSE values.

Task 2.2: PyTorch Verification — 8 pts

Implement the same linear regression model using PyTorch:

- Define the model using `torch.nn.Linear` or manual parameters with `requires_grad=True`.
- Use `torch.nn.MSELoss` and a PyTorch optimizer (e.g., `torch.optim.SGD`).
- Train and plot the loss curve.
- **Compare** the learned parameters and final loss with your NumPy implementation. Are they consistent? Discuss any differences.

Expected outputs: PyTorch training code, loss curve plot, side-by-side comparison table of NumPy vs. PyTorch parameters and MSE.

Task 2.3: Polynomial Regression — 10 pts

Extend your model to polynomial regression:

- Using **either NumPy or PyTorch**, train polynomial regression models with degree n = 2, 3, 5, and 10.
- For each degree, report the training MSE and test MSE.
- Plot the fitted curves for different polynomial degrees on the same figure.
- Discuss: Which degree gives the best test performance? At what point do you observe **overfitting** vs. **underfitting**? Relate your observations to the concepts from lecture.

Expected outputs: Table of train/test MSE per degree, overlay plot of fitted curves, written analysis of overfitting/underfitting.

Part 3: Logistic Regression (30 points)

Target variable: `passed` (binary: 0 or 1). Use the same cleaned, normalized features from Part 1.

Task 3.1: NumPy Implementation — 12 pts

Implement logistic regression with gradient descent from scratch using **only NumPy**:

- Implement the **sigmoid function**: $\sigma(z) = \frac{1}{1+e^{-z}}$
- Implement the **cross-entropy cost function**: $J(\beta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\sigma(z_i)) + (1-y_i)\log(1-\sigma(z_i))\right]$

- Implement the **gradient computation** and update rule.
- Train the model and **plot the loss curve** (cross-entropy vs. iteration number).
- Use threshold $T = 0.5$ to generate predictions. Report **accuracy** on both training and test sets.

Expected outputs: Working gradient descent code, loss curve plot, printed accuracy on train and test sets.

Task 3.2: PyTorch Verification — 8 pts

Implement the same logistic regression model using PyTorch:

- Use `torch.nn.BCEWithLogitsLoss` or `torch.nn.BCELoss` with sigmoid.
- Train using a PyTorch optimizer.
- **Compare** predictions, accuracy, and learned parameters with your NumPy results.

Expected outputs: PyTorch training code, side-by-side comparison of accuracy and parameters.

Task 3.3: Analysis & Visualization — 10 pts

- Plot the **sigmoid curve** with your learned parameters overlaid on the data points.
- If using 2 features, visualize the **decision boundary** separating the two classes.
- Experiment with a **different threshold** (e.g., $T = 0.3$ or $T = 0.7$). How does changing the threshold affect accuracy? When might you want to use a threshold other than 0.5?
- Discuss: Compare MSE (from Part 2) and cross-entropy loss (from Part 3). Why is cross-entropy more appropriate for classification tasks?

Expected outputs: Sigmoid/decision boundary plots, accuracy at different thresholds, written comparison of loss functions.

Part 4: Summary & Reflection (15 points)

Write a summary of **300–500 words** covering the following questions:

- What preprocessing steps had the biggest impact on model performance?
- How did your NumPy and PyTorch implementations compare? What did implementing from scratch teach you that using a library alone would not?
- What was the most challenging part of this project, and how did you resolve it?
- If you had more time, what would you try to improve your models?

> **Requirement:** The summary must be written **entirely by you** (or your team members) and must demonstrate your own understanding of the project. This section will be evaluated for originality and depth of understanding. Submissions where the summary does not reflect genuine engagement with the project will receive a zero for this section.

Grading Rubric

Each task is graded on two tiers:

- **Completion (C):** Did you attempt the task and produce reasonable outputs? If yes, you earn the completion points. Partial attempts with visible effort still earn most of the completion score.
- **Quality (Q):** Is your work well-executed — correct results, clear reasoning, insightful analysis, clean code? Strong work earns additional quality points.

Component	C pts	Q pts	Total	What earns Completion	What earns Quality
1.1 EDA	3	2	5	Loaded data, showed statistics, produced visualizations	Thorough exploration, identified all major issues
1.2 Data Cleaning	5	3	8	Addressed missing values, outliers, and duplicates	Well-justified decisions, before/after comparison
1.3 Feature Selection	3	2	5	Chose features with some reasoning	Correlation analysis, strong justification for each decision
1.4 Normalization	4	3	7	Applied Z-score normalization correctly	Also applied L2 normalization, discussed why normalization matters
2.1 Linear Reg (NumPy)	8	4	12	Implemented GD, produced loss curve	Correct gradients, converging loss, reported parameters
2.2 Linear Reg (PyTorch)	5	3	8	Trained model using PyTorch	Meaningful comparison with NumPy results

2.3 Polynomial Reg	6	4	10	Trained multiple polynomial degrees	Clear overfitting/underfitting analysis with visualizations
3.1 Logistic Reg (NumPy)	8	4	12	Implemented GD with sigmoid and cross-entropy	Correct gradients, converging loss, reported accuracy
3.2 Logistic Reg (PyTorch)	5	3	8	Trained model using PyTorch	Meaningful comparison with NumPy results
3.3 Analysis & Viz	6	4	10	Produced sigmoid/decision boundary plots	Threshold experiment, loss function comparison discussion
4 Summary & Reflection	8	7	15	Wrote 300–500 word summary addressing all prompts	Demonstrates genuine understanding, original and thoughtful
	61	**39**	**100**		

> **Grading philosophy:** If you make a genuine effort on every part of this project, you will earn a good grade. The quality tier rewards deeper analysis and cleaner execution, but completion of each task is the foundation of your score.

Code Quality Expectations

- Code must be well-organized and readable with meaningful variable names.
- Include comments explaining your logic, especially for gradient computations.
- All plots must have proper titles, axis labels, and legends.
- Use Markdown cells in your notebook to explain your approach before each code section.

Academic Integrity

All submitted code must reflect your own (or your team's) understanding of the material. You must be able to explain every line of your submitted code. All group members are expected to contribute meaningfully to the project.