University of Central Missouri

Department of Computer Science & Cybersecurity

CS5710 Machine Learning

Fall 2025

Home Assignment 1.

Student name:SANTHOSH REDDY KISTIPATI

Submission Requirements:

- Once finished your assignment push your source code to your repo (GitHub) and explain the work through the ReadMe file properly. Make sure you add your student info in the ReadMe file.
- Submit your GitHub link and video on the BB.
- Comment your code appropriately ***IMPORTANT.***
- Any submission after provided deadline is considered as a late submission.

1 — Function Approximation by Hand

Dataset: (x,y)={(1,1),(2,2),(3,2),(4,5)}

Essential Function (Prediction):

$$\hat{y} = \theta_1 x + \theta_2$$

Essential Function (MSE):

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2$$

Task:

1. Try model θ=(1,0).Fill in predictions, residuals, squared residuals, and compute MSE.
2. Try model θ=(0.5,1). Do the same.
3. Which model fits better?

ANS: Dataset: (x,y)={(1,1),(2,2),(3,2),(4,5)}

Model form: y^=θ1x+θ2

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2$$

MSE:

Model 1: h(x)=x (θ=(1,0))

1. **Predictions**

- x=1⇒y^=1
- x=2⇒y^=2
- x=3⇒y^=3
- x=4⇒y^=4

  Predictions: **[1, 2, 3, 4]**

2. **Residuals** ri=yi−y^i

- 1−1=0
- 2−2=0
- 2−3=−1
- 5−4=1
  r=[0,0,−1,1]

3. **Squared residuals** r2i

- • $0^2=0$
- • $0^2=0$
- • $(-1)^2=1$
- • $1^2=1$
- Squared Residuals: [0, 0, 1, 1], Σ=2

4. **MSE**

MSE J(θ) = (1/4)*2 = 0.5

2, Model 2: h(x)=0.5x+1 (θ=(0.5,1))

Predictions

• x=1 ⇒ 1.5

• x=2 ⇒ 2.0

• x=3 ⇒ 2.5

• x=4 ⇒ 3.0

Predictions: [1.5, 2.0, 2.5, 3.0]

2) Residuals

• $1-1.5=-0.5$

• $2-2=0$

• $2-2.5=-0.5$

• $5-3=2$

Residuals: $[-0.5, 0, -0.5, 2]$

3) Squared Residuals

• $(-0.5)^2=0.25$

• $0^2=0$

• $(-0.5)^2=0.25$

• $2^2=4$

Squared Residuals: $[0.25, 0, 0.25, 4]$, $\Sigma=4.5$

4, MSE

$J(\theta) = (1/4)*4.5 = 1.125$

3)ans : Model 1 (h(x)=x) fits better since it has lower MSE (0.5 < 1.125).-

2 — Random Guessing Practice

Cost function:

$J(\theta_1, \theta_2)=8(\theta_1 - 0.3)^2+4(\theta_2 - 0.7)^2$

Task:

1. Compute J(0.1,0.2) and J(0.5,0.9).
2. Which guess is closer to the minimum (0.3,0.7)?
3. In 2–3 sentences, explain why random guessing is inefficient.

1) Compute J(0.1, 0.2)

$(0.1 - 0.3) = -0.2 \Rightarrow (-0.2)^2 = 0.04$; $8 \times 0.04 = 0.32$

$(0.2 - 0.7) = -0.5 \Rightarrow (-0.5)^2 = 0.25$; $4 \times 0.25 = 1.00$

Sum: J(0.1, 0.2) = 0.32 + 1.00 = 1.32

Compute J(0.5, 0.9)

$(0.5 - 0.3) = 0.2 \Rightarrow (0.2)^2 = 0.04$; $8 \times 0.04 = 0.32$

$(0.9 - 0.7) = 0.2 \Rightarrow (0.2)^2 = 0.04$; $4 \times 0.04 = 0.16$

Sum: J(0.5, 0.9) = 0.32 + 0.16 = 0.48

2) Since $0.48 < 1.32$, the guess (0.5, 0.9) is closer to the minimum (0.3, 0.7).

3) Random guessing is inefficient because it wastes time testing many poor guesses; gradient-based methods systematically move toward the minimum much faster.

3 — First Gradient Descent Iteration

Dataset: (1,3),(2,4),(3,6),(4,5)

Essential Function (Gradients):

$$\frac{\partial J}{\partial \theta_1} = -\frac{2}{N}\sum_{i=1}^{N} x^{(i)}(y^{(i)} - \hat{y}^{(i)}), \quad \frac{\partial J}{\partial \theta_2} = -\frac{2}{N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})$$

Update Rule:

$$\theta \leftarrow \theta - \alpha \nabla J$$

Start: $\theta^{(0)}$=(0,0), learning rate α=0.01.

Task:

1. Compute predictions at $\theta^{(0)}$.
2. Compute residuals and sums ∑r, ∑xr.
3. Compute gradient ∇J.
4. Update to $\theta^{(1)}$.
5. Compute J($\theta^{(0)}$) and J($\theta^{(1)}$).

Continue from Homework 3 with $\theta^{(1)}$

Update Rule (reminder):

$$\theta \leftarrow \theta - \alpha \nabla J$$

Task:

1. Compute predictions at $\theta^{(1)}$.
2. Compute residuals, ∑r, ∑xr .
3. Compute gradient.
4. Update to $\theta^{(2)}$.
5. Compare J($\theta^{(1)}$) and J($\theta^{(1)}$).

Ans:

Dataset: (1,3),(2,4),(3,6),(4,5)

Essential Function (Gradients):

$$\frac{\partial J}{\partial \theta_1} = -\frac{2}{N}\sum_{i=1}^{N} x^{(i)}(y^{(i)} - \hat{y}^{(i)}), \quad \frac{\partial J}{\partial \theta_2} = -\frac{2}{N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})$$

Update Rule:

$$\theta \leftarrow \theta - \alpha \nabla J$$

TASK 1

1) Predictions at $\theta^{(0)} = (0, 0)$:
$\hat{y} = [0, 0, 0, 0]$

2) Residuals $r_i = y_i - \hat{y}_i$ and sums:
$r = [3, 4, 6, 5]$
$\sum r = 18, \sum xr = 49$

3) Gradient ∇J:
$\frac{\partial J}{\partial \theta_1} = -\frac{2}{4} \cdot 49 = -24.5$
$\frac{\partial J}{\partial \theta_2} = -\frac{2}{4} \cdot 18 = -9$

4) Update to $\theta^{(1)}$:
$\theta^{(1)} = \theta^{(0)} - \alpha \nabla J = (0, 0) - 0.01\,(-24.5, -9) = (0.245,\ 0.09)$

5) Costs:
$J(\theta^{(0)}) = \frac{1}{4}(9 + 16 + 36 + 25) = 21.5$
$J(\theta^{(1)}) \approx 15.2560$

TASK 2

1) Predictions:
$\hat{y} = 0.245x + 0.09 \Rightarrow [0.335,\ 0.58,\ 0.825,\ 1.07]$

2) Residuals and sums:
$r = [2.665,\ 3.42,\ 5.175,\ 3.93]$
$\sum r = 15.19, \quad \sum xr = 40.75$

3) Gradient:
$\frac{\partial J}{\partial \theta_1} = -\frac{2}{4} \cdot 40.75 = -20.375$
$\frac{\partial J}{\partial \theta_2} = -\frac{2}{4} \cdot 15.19 = -7.595$

4) Update to $\theta^{(2)}$:
$\theta^{(2)} = (0.245, 0.09) - 0.01\,(-20.375, -7.595) = (0.44875,\ 0.16595)$

5) Compare costs:
$J(\theta^{(1)}) \approx 15.2560$
$J(\theta^{(2)}) \approx 10.9223$

4 — Compare Random Guessing vs Gradient Descent

Dataset: (1,2),(2,2),(3,4),(4,6)

Essential Function (MSE):

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - (\theta_1 x^{(i)} + \theta_2))^2$$

Task:

1. Try 2 random guesses: (θ_1, θ_2)=(0.2,0.5) and (0.9,0.1). Compute J.
2. Compare with J from the first gradient descent step (starting at θ=(0,0), α=0.01).
3. Which approach gave lower error? Why?

1) Two random guesses

- $(\theta_1, \theta_2) = (0.2,\ 0.5)$
  $\hat{y} = [0.7,\ 0.9,\ 1.1,\ 1.3]$
  $J = \frac{1}{4}\left((2-0.7)^2 + (2-0.9)^2 + (4-1.1)^2 + (6-1.3)^2\right) = 8.35$
- $(\theta_1, \theta_2) = (0.9,\ 0.1)$
  $\hat{y} = [1.0,\ 1.9,\ 2.8,\ 3.7]$
  $J = \frac{1}{4}\left(1.0^2 + 0.1^2 + 1.2^2 + 2.3^2\right) = 1.935$

2) First gradient-descent step from $(0, 0)$

At $(0,0)$: $\hat{y} = [0,0,0,0]$, residuals $r = y$.
$\sum r = 14, \quad \sum xr = 42$

Gradients:

$$\frac{\partial J}{\partial \theta_1} = -\frac{2}{4}\sum xr = -21, \qquad \frac{\partial J}{\partial \theta_2} = -\frac{2}{4}\sum r = -7$$

Update:

$$\theta^{(1)} = (0,0) - 0.01(-21,-7) = (0.21,\ 0.07)$$

Cost at $\theta^{(1)}$:

$$J(\theta^{(1)}) = \frac{1}{4}\left((2-0.28)^2 + (2-0.49)^2 + (4-0.70)^2 + (6-0.91)^2\right) \approx 10.50915$$

Which is lower and why?

- Best random guess above: J=1.935 for (0.9,0.1)
- First GD step: J≈10.509
  The random guess (0.9,0.1)(0.9,0.1)(0.9,0.1) gives a much lower error because it happens

to be closer to the true least-squares solution, while a single gradient-descent step from (0,0)(0,0)(0,0) only moves partway toward the optimum (it improves JJJ from 15 down to about 10.51, but not nearly as much as the lucky guess).

5 — Recognizing Underfitting and Overfitting

Imagine you train a model and notice the following results:

- Training error is very high.
- Test error is also very high.

Questions:

1. Is this an example of underfitting or overfitting?
2. Explain why this situation happens.
3. Suggest two possible fixes.

Ans:

1)underfitting.

2)Underfitting occurs when the model is too simple to capture the underlying patterns in the data. Because of this, it performs poorly not just on the test set but also on the training set

Typical causes: using a model with too few features, oversimplified assumptions (e.g., linear when the true relation is non-linear), or not enough training.

3)Two possible fixes

1. **Increase model complexity**:
   Use a richer hypothesis class (e.g., polynomial regression instead of linear, deeper neural network, more features).
2. **Train better**:
   Increase training time (more epochs, lower learning rate if needed).
   Add more informative features or engineer new ones to better represent the data.

6 — Comparing Models

You test two different machine learning models on the same dataset:

- Model A fits the training data almost perfectly but performs poorly on new unseen data.
- Model B does not fit the training data very well and also performs poorly on unseen data.

Questions:

1. Which model is overfitting? Which one is underfitting?
2. In each case, what is the tradeoff between bias and variance?
3. What would you recommend to improve each model?

Ans:

1)Model A: Overfitting

- Very low training error, but poor generalization on new data.

Model B: Underfitting

- High training error and poor performance on test data.

2) Model A (Overfitting):

- Low bias (fits training data almost perfectly).
- High variance (sensitive to noise and doesn't generalize).

Model B (Underfitting):

- High bias (too simple to capture patterns).
- Low variance (but consisten wrong).

3)**Recommendations**

- **For Model A :**
    - Simplify the model (reduce parameters, pruning, regularization like L1/L2, dropout).
    - Get more training data.
    - Use cross-validation or early stopping.
- **For Model B :**
    - Increase model complexity (use more features, deeper/larger models, polynomial terms).

- Train longer (more epochs, better hyperparameters).
- Feature engineering to better represent data.

7 —Programming Problem - Implement Gradient Descent for Linear Regression

Problem Statement

You are asked to implement linear regression using Gradient Descent *from scratch* (without using scikit-learn's LinearRegression). Your task is to compare the closed-form solution (Normal Equation) with Gradient Descent on the same dataset.

Dataset

- Generate synthetic data following: $y=3+4x+\epsilon$ where ϵ is Gaussian noise.
- Create 200 samples with $x \in [0,5]$.

Requirements

1. Generate the dataset and plot the raw data.
2. Closed-form solution (Normal Equation):
   - Compute $\theta = (X^T X)^{-1} X^T y.$
   - Print the estimated intercept and slope.
   - Plot the fitted line.
3. Gradient Descent implementation:
   - Initialize $\theta=[0,0]$.
   - Use learning rate $\eta=0.05$.
   - Run for 1000 iterations.
   - Plot the loss curve (MSE vs iterations).
   - Print the final intercept and slope.
4. Comparison:
   - Report both solutions (Closed-form vs Gradient Descent).
   - Comment: Did Gradient Descent converge to the same solution as the closed form?

Expected Deliverables

- Python code file (.py or Jupyter notebook).

- A plot showing:
  - Raw data points.
  - Closed-form fitted line.
  - Gradient Descent fitted line.
- A plot of loss vs iterations (for Gradient Descent).
- A short explanation (2–3 sentences) of the results.

Hints

- Don’t forget to add a bias column of 1’s to your X.
- The gradient for MSE is:

$$\nabla_\theta J(\theta) = \frac{2}{m} X^T (X\theta - y)$$

- Use np.dot() for matrix multiplication.
- To plot multiple lines on the same figure, use plt.plot() several times.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages

rng = np.random.default_rng(42)
m = 200
X_raw = rng.uniform(0, 5, size=m)
noise = rng.normal(0, 1.0, size=m)
y = 3 + 4 * X_raw + noise
X = np.c_[np.ones(m), X_raw]

XtX_inv = np.linalg.inv(X.T @ X)
theta_closed = XtX_inv @ X.T @ y
intercept_closed, slope_closed = theta_closed

eta = 0.05
n_iters = 1000
theta = np.zeros(2)
loss_history = []

def mse_loss(theta, X, y):
    preds = X @ theta
    return np.mean((preds - y) ** 2)

for i in range(n_iters):
    preds = X @ theta
    grad = (2/len(X)) * (X.T @ (preds - y))
    theta = theta - eta * grad
    loss_history.append(mse_loss(theta, X, y))

intercept_gd, slope_gd = theta

xs = np.linspace(0,5,200)
ys_closed = intercept_closed + slope_closed * xs
ys_gd = intercept_gd + slope_gd * xs

fig1 = plt.figure(figsize=(7,5))
plt.scatter(X_raw, y, s=15, alpha=0.7, label="Raw data")
plt.plot(xs, ys_closed, linewidth=2, label="Closed-form line")
plt.plot(xs, ys_gd, linewidth=2, linestyle="--", label="Gradient Descent line")
plt.xlabel("x"); plt.ylabel("y"); plt.title("Linear Regression: Data and Fits"); plt.legend(loc="best")
fig1.tight_layout()

fig2 = plt.figure(figsize=(7,5))
plt.plot(np.arange(1, n_iters+1), loss_history)
plt.xlabel("Iteration"); plt.ylabel("MSE Loss"); plt.title("Gradient Descent: Loss vs Iterations")
fig2.tight_layout()

img1_path = "linear_regression_fits.png"
img2_path = "gd_loss_curve.png"
fig1.savefig(img1_path, dpi=180, bbox_inches="tight")
fig2.savefig(img2_path, dpi=180, bbox_inches="tight")

with PdfPages("linear_regression_gd_report.pdf") as pdf:
    pdf.savefig(fig1, bbox_inches="tight")
    pdf.savefig(fig2, bbox_inches="tight")

print("Closed-form  intercept, slope:", intercept_closed, slope_closed)
print("GradientDesc intercept, slope:", intercept_gd, slope_gd)
print("Final GD loss:", loss_history[-1])
```

```
Closed-form  intercept, slope: 2.6908413834642375 4.131841832861142
GradientDesc intercept, slope: 2.69084138327786 4.131841832919722
Final GD loss: 0.9958085506981229
```