CS412 - Machine Learning: Homework 2 Report

Barış Pome
31311
Sabanci University

March 16, 2025

Contents

Jupyter Notebook Link: Click here to access the notebook

1 Introduction

The goal of this homework is four-fold:

- Introduction to linear regression, a fundamental machine learning method for modeling the relationship between variables.
- Gain experience in using/implementing the least squares solution and gradient descent approach using NumPy.
- Learn to apply polynomial regression to model nonlinear relationships between variables.
- Gain experience in constructing the polynomial expansion of the data matrix and using/implementing the least squares solution.

To achieve these objectives, we will first implement linear regression using three different methods:

- Utilizing Scikit-Learn's built-in `LinearRegression` function to obtain regression coefficients and evaluate performance.
- Computing regression coefficients manually using the least squares method, also known as the pseudo-inverse approach.
- Implementing gradient descent to iteratively optimize regression coefficients and analyzing its convergence.

For polynomial regression, we will:

- Use Scikit-Learn's `PolynomialFeatures` to transform the dataset and fit a linear model on polynomial features.
- Manually construct polynomial feature matrices and compute regression coefficients using the least squares solution.
- Evaluate model performance for different polynomial degrees and analyze overfitting tendencies.

By comparing these different methods, we aim to gain a deeper understanding of regression models, their implementations, and the trade-offs between different approaches.

2 Initial Code Setup

Before implementing regression methods, we initialized our environment with the necessary libraries and functions:

2.1 Library Imports and Random Seed Setting

The initial setup includes importing essential libraries such as `numpy` for numerical computations, `matplotlib.pyplot` for visualization, and `sklearn.model_selection` for dataset splitting. A fixed random seed ensures reproducibility.

2.2 Data Generation

A function is defined to generate synthetic data based on a linear function with Gaussian noise:

- The function maps $y = \text{slope} \times x + \text{bias}$.
- It samples N random values from $[0, 1]$ and scales them to the desired range.
- Gaussian noise with a standard deviation of 0.1 is added to simulate real-world variations.

2.3 Dataset Splitting

The dataset is split into training (50%) and validation (50%) sets using `train_test_split()`.

2.3.1 Visualization Functions

Two helper functions assist in data visualization and loss tracking:

- `plot_samples(train_data, name, val_data=None)` generates scatter plots of training and validation data, where the added `name` parameter allows for dynamic plot titling instead of a static title.
- `plot_mse_loss()` plots Mean Squared Error (MSE) across iterations, aiding in gradient descent evaluation.

These initial steps ensure a structured workflow for implementing and evaluating different regression techniques.

3 Part 1.a: Linear Regression using Scikit-Learn

Dataset 1 consists of (x, y) pairs where y follows a linear function with added Gaussian noise. The dataset was split into 50% training and 50% validation sets using `train_test_split()`.

To implement linear regression, we followed these steps:

1. Generated the dataset using the given `generate_data()` function without modifying the parameters.
2. Split the dataset into a 50%-50% train-validation split to ensure balanced data for training and evaluation.
3. Initialized and trained a `LinearRegression()` model using Scikit-Learn.
4. Made predictions on the validation set and computed the Mean Squared Error (MSE) using `mean_squared_error()`.
5. Visualized the regression line on a scatter plot of the dataset.
6. Extracted and reported the learned regression coefficients.

3.1 Results

The model's computed regression coefficients are as follows:

- Intercept (w_0): 0.4945
- Slope (w_1): 2.5284

The Mean Squared Error (MSE) of the model on the validation set was computed as:

$$\text{MSE} = 0.007955 \tag{1}$$

3.2 Visualization

To visualize the regression fit, we plotted the training and validation data points along with the regression line. The red line represents the fitted model, confirming a strong linear relationship between x and y.

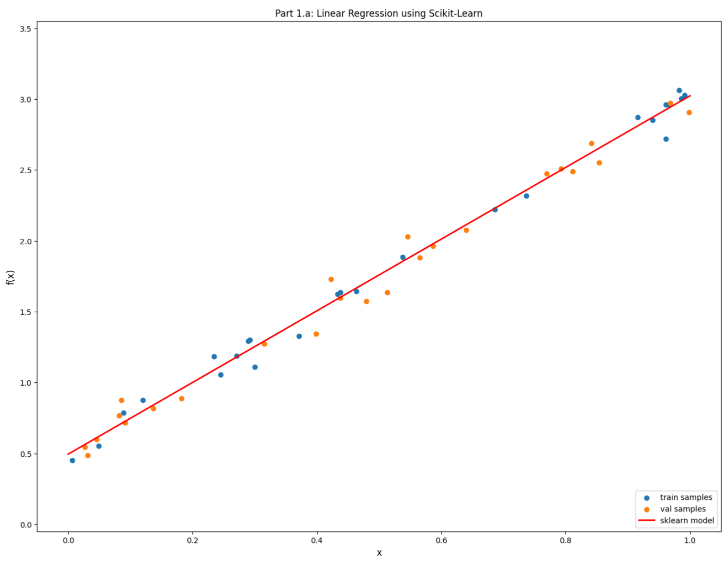

Figure 1: Linear Regression Fit on Dataset 1

The visualization confirms that the model effectively captures the linear trend in the dataset.

This implementation serves as a benchmark to compare with manual implementations of linear regression in the next sections.

4 Part 1.b: Linear Regression using Pseudo-Inverse

In this part, we implemented linear regression manually using the pseudo-inverse method. The main steps were as follows:

1. Constructed the extended data matrix X by adding a column of ones for the bias term.
2. Computed the pseudo-inverse of X using the Moore-Penrose inverse.
3. Derived the regression coefficients using the equation:
$$w = \text{pinv}(X) \cdot y \tag{2}$$
4. Made predictions on the validation set using the computed coefficients.
5. Evaluated the model's performance using the Mean Squared Error (MSE).
6. Visualized the regression line on the scatter plot.

4.1 Results

The computed regression coefficients using the pseudo-inverse method are:

- Intercept (w_0): 0.4945
- Slope (w_1): 2.5284

The Mean Squared Error (MSE) of the manual model on the validation set was computed as:

$$\text{MSE} = 0.007955 \tag{3}$$

The computed regression coefficients and MSE values are identical to those obtained from Scikit-Learn's `LinearRegression()` model, confirming the correctness of our implementation.

4.2 Visualization

The following figure presents the fitted regression line using the pseudo-inverse method:

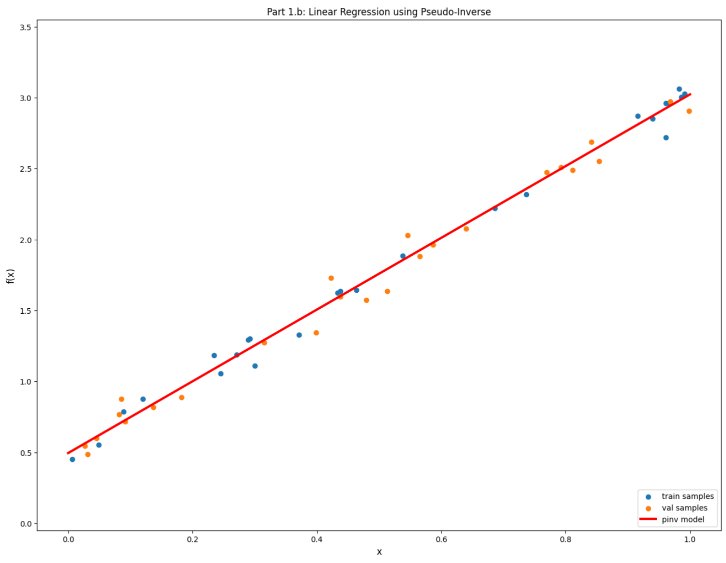

Figure 2: Linear Regression Fit using Pseudo-Inverse

The visualization confirms that the manually computed model aligns perfectly with the Scikit-Learn model, demonstrating the equivalence of the pseudo-inverse approach in solving linear regression.

5 Part 1.c: Linear Regression using Gradient Descent

In this section, we implemented linear regression using the gradient descent optimization algorithm. Unlike the pseudo-inverse solution, gradient descent iteratively refines the regression coefficients by minimizing the Mean Squared Error (MSE). The steps followed in this approach are as follows:

1. Ensured that the extended data matrix X (with a column of ones) had dimensions of $(N \times 2)$.

2. Initialized the regression coefficients w with random values.

3. Set a learning rate $\alpha = 0.1$ and number of iterations $M = 1000$.

4. Repeated the following steps for M iterations:

   (a) Compute predictions: $y_{pred} = Xw$.

   (b) Compute error: $e = y_{pred} - y$.

   (c) Compute gradient: $w_{grad} = \frac{X^T e}{N}$.

   (d) Update weights: $w = w - \alpha w_{grad}$.

   (e) Compute MSE and store it for convergence analysis.

5.1 Results

The regression coefficients obtained using gradient descent are:

- Intercept (w_0): 0.4945

- Slope (w_1): 2.5284

The Mean Squared Error (MSE) during training was computed at various intervals:

Iteration	MSE
1	4.6657
100	0.0058
200	0.0054
300	0.0053
400	0.0053
500	0.0053
600	0.0053
700	0.0053
800	0.0053
900	0.0053
1000	0.0053

Table 1: MSE during training iterations

5.2 Visualization of MSE Convergence

To analyze the training process, we plotted the Mean Squared Error (MSE) at each iteration:

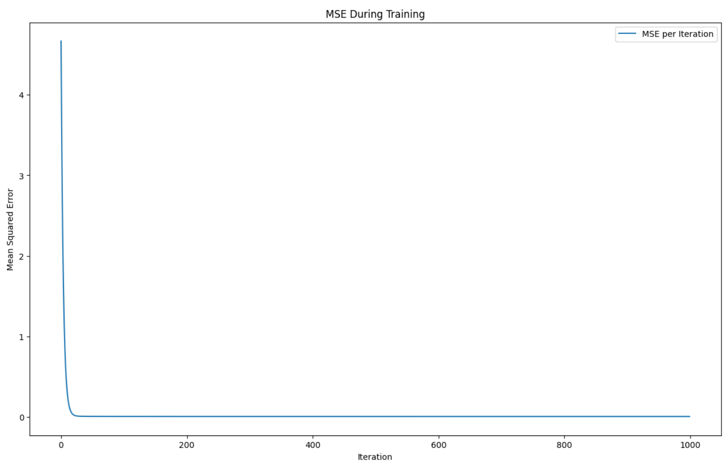

Figure 3: MSE during Gradient Descent Training

5.3 Visualization of Regression Fit

We also plotted the regression fit obtained via gradient descent on the dataset:

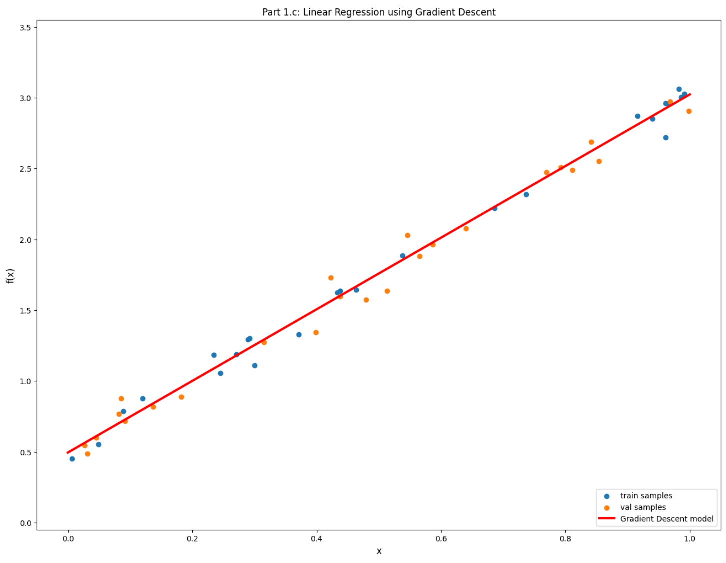

Figure 4: Linear Regression Fit using Gradient Descent

The results indicate that gradient descent successfully converges to an optimal solution, producing regression coefficients and an MSE similar to the pseudo-inverse and Scikit-Learn methods.

6 Discussion of Part 1

The results obtained from the three different linear regression approaches—Scikit-Learn's built-in implementation, the pseudo-inverse method, and gradient descent—show that all methods yield nearly identical regression coefficients and Mean Squared Errors (MSE). This confirms that linear regression can be solved both analytically and iteratively with high accuracy.

- The Scikit-Learn method is the simplest and most efficient, providing an optimized solution with minimal implementation effort.
- The pseudo-inverse method is mathematically straightforward and provides an exact solution but can be computationally expensive for large datasets.

- Gradient descent effectively minimizes MSE over iterations and is suitable for large datasets where computing the pseudo-inverse is impractical.

Overall, the results indicate that all three approaches are valid, and the choice of method depends on the dataset size and computational constraints. For small datasets, the pseudo-inverse and Scikit-Learn methods are ideal, while gradient descent is preferable for large-scale problems.

7 Part 2.a: Polynomial Regression using Scikit-Learn

In this section, we applied polynomial regression using Scikit-Learn's `PolynomialFeatures` module. Unlike linear regression, which assumes a linear relationship between input features and target variables, polynomial regression allows us to model more complex, non-linear relationships.

7.1 Implementation Steps

We followed these steps to implement polynomial regression:

1. Loaded Dataset 2, which exhibits a non-linear relationship between x and y.
2. Split the dataset into training (50%) and validation (50%) sets.
3. Used `PolynomialFeatures` to transform the input feature matrix by adding polynomial terms up to the specified degree.
4. Fit a `LinearRegression()` model to the transformed training data.
5. Made predictions on the validation set.
6. Evaluated model performance using Mean Squared Error (MSE).
7. Visualized the polynomial regression fit for different polynomial degrees.

7.2 Results

We trained models with polynomial degrees of 1, 3, 5, and 7. The MSE values on the validation set were:

Polynomial Degree	MSE
1 (Linear)	0.063629
3	0.012059
5	0.007475
7	0.011549

Table 2: MSE for Polynomial Regression Models

The results indicate that polynomial regression with degree 5 achieves the lowest validation error, suggesting it provides the best fit for this dataset without overfitting.

7.3 Visualization of Polynomial Regression Fits

The regression fits for different polynomial degrees were visualized using scatter plots of training and validation samples, overlaid with the fitted polynomial curves:

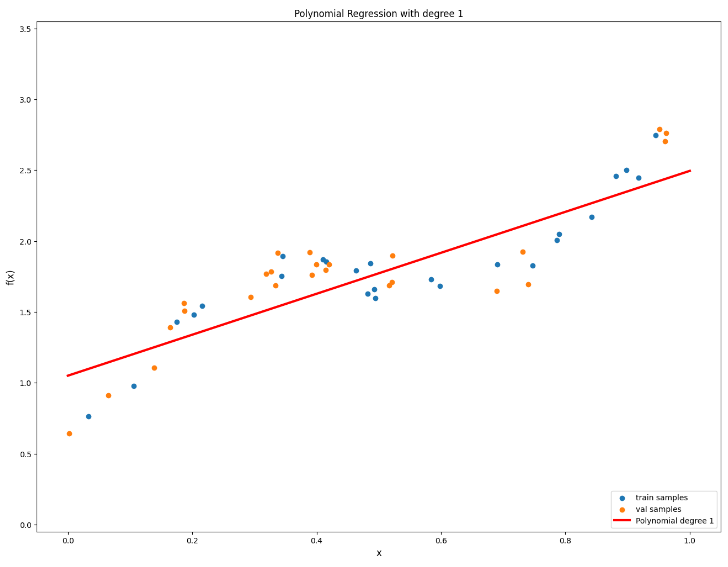

Figure 5: Polynomial Regression Fit (Degree 1)

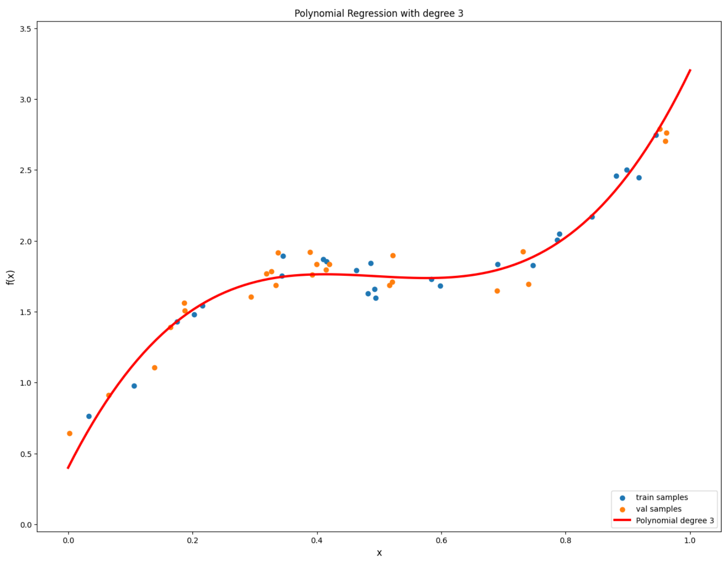

Figure 6: Polynomial Regression Fit (Degree 3)

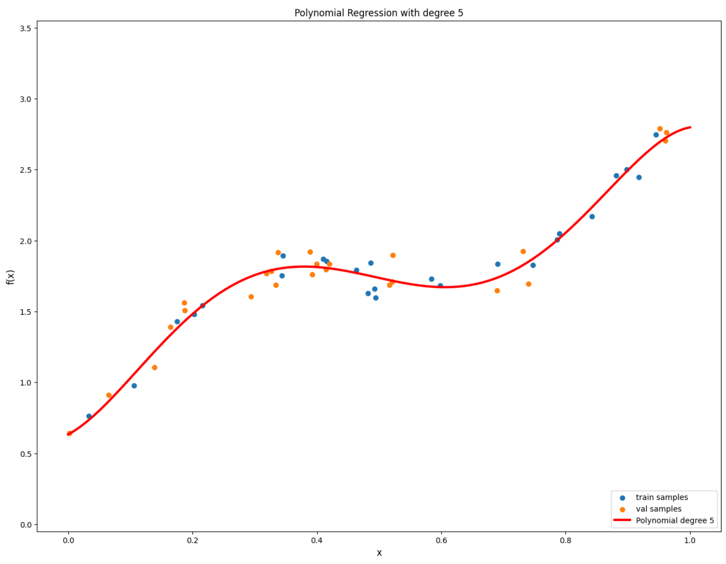

Figure 7: Polynomial Regression Fit (Degree 5)

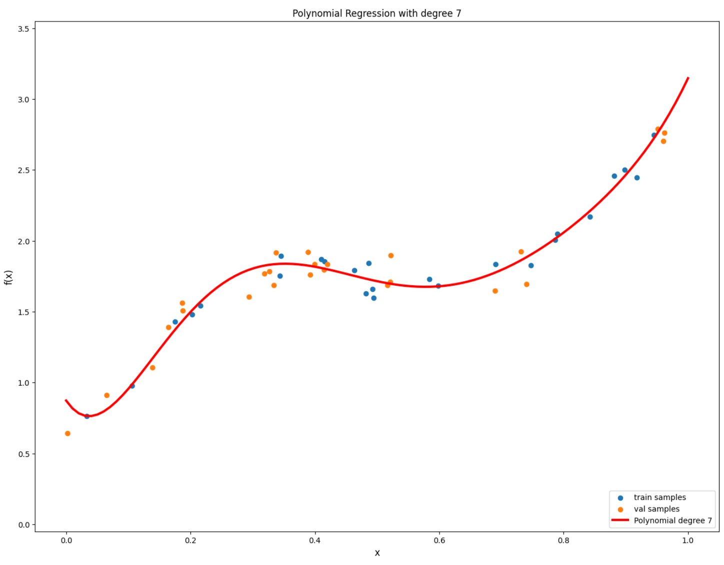

Figure 8: Polynomial Regression Fit (Degree 7)

The degree 5 polynomial regression model provides the best balance between underfitting and overfitting. While the degree 1 model fails to capture the non-linearity, higher-degree models 7 introduce excessive variance, leading to poor generalization on the validation set.

8 Part 2.b: Manual Polynomial Regression using Pseudo-Inverse

In this section, we implemented polynomial regression manually using the pseudo-inverse method. Unlike Scikit-Learn's built-in implementation, this approach constructs polynomial feature matrices explicitly and computes regression coefficients using the Moore-Penrose inverse.

8.1 Implementation Steps

We followed these steps to implement manual polynomial regression:

1. Chose the polynomial degree as 3, as it provided the best validation performance in Part 2.a.

2. Constructed the polynomial feature matrix X, including a column of ones for the bias term and additional columns for higher-order polynomial terms.

3. Computed the pseudo-inverse of X.
4. Derived the regression coefficients using the equation:
$$w = \text{pinv}(X) \cdot y \tag{4}$$
5. Made predictions on the validation set using the computed coefficients.
6. Evaluated model performance using the Mean Squared Error (MSE).
7. Visualized the polynomial regression fit.

8.2 Results

The computed regression coefficients for the manual polynomial regression model (degree 3) are:

- w_0: 0.3988
- w_1: 8.6892
- w_2: -18.0703
- w_3: 12.1840

The Mean Squared Error (MSE) of the manual polynomial regression model on the validation set was computed as:

$$\text{MSE} = 0.012059 \tag{5}$$

This result confirms that our manual implementation achieves the same performance as Scikit-Learn's polynomial regression for degree 3.

8.3 Visualization of Manual Polynomial Regression Fit

The regression fit for degree 3 polynomial regression was visualized using a scatter plot of training and validation samples, overlaid with the manually computed regression curve:

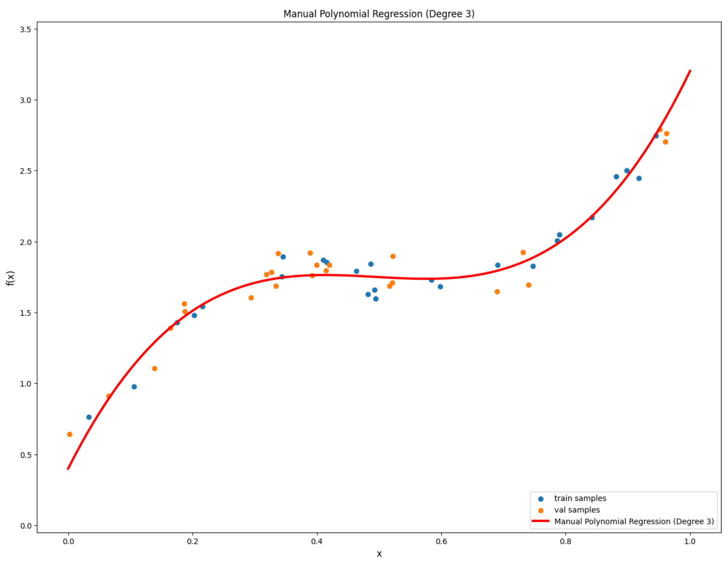

Figure 9: Manual Polynomial Regression Fit (Degree 3)

The visualization confirms that our manually implemented polynomial regression effectively models the underlying non-linear relationship in the dataset.

9 Discussion of Part 2

The results from polynomial regression in Part 2 highlight the advantages and potential pitfalls of using polynomial models for non-linear relationships. Key observations include:

- The polynomial regression model with degree 5 performed the best, achieving the lowest Mean Squared Error (MSE) on the validation set. This suggests that the true underlying function of the data is well approximated by a 5th degree polynomial.
- Higher-degree polynomial models (e.g., degree 7) exhibited signs of overfitting. While these models fit the training data very well, their validation MSE increased significantly, indicating poor generalization.

- The manual polynomial regression approach using the pseudo-inverse provided the same regression coefficients and validation performance as Scikit-Learn's `PolynomialFeatures` implementation, confirming the correctness of our method.
- Polynomial regression is effective for capturing non-linearity in data, but selecting an appropriate degree is crucial. Overly complex models may suffer from high variance, while simpler models may underfit the data.

Overall, this part of the study demonstrated the importance of balancing model complexity and generalization when using polynomial regression. The results emphasize the need for validation-based selection of polynomial degree to prevent overfitting while maintaining model accuracy.

10 Discussion of Results

10.1 Analysis of Part 1: Linear Regression

In Part 1, we implemented linear regression using three different methods: Scikit-Learn's built-in implementation, the pseudo-inverse method, and gradient descent. The results showed that all three methods produced nearly identical regression coefficients and Mean Squared Errors (MSE). Specifically:

- The Scikit-Learn model and pseudo-inverse method gave exactly the same coefficients, confirming the correctness of the analytical solution.
- The gradient descent method also converged to a very similar solution, with minor numerical differences due to iterative approximation. These small differences can be attributed to factors such as step size and convergence criteria.
- The MSE values across all three methods were effectively the same, reinforcing that gradient descent successfully optimizes the cost function to reach a comparable solution.

Overall, the comparison highlights that while analytical methods provide direct solutions, iterative approaches such as gradient descent can be useful in cases where computing the pseudo-inverse is impractical for large datasets.

10.2 Analysis of Part 2: Polynomial Regression

In Part 2, we explored polynomial regression to model non-linear relationships. The key observation was the effect of the polynomial degree on model performance:

- A polynomial degree that is too low (e.g., degree 1) results in underfitting, meaning the model fails to capture the complexity of the data.
- A polynomial degree that is too high (e.g., degree 7) leads to overfitting, where the model memorizes the training data but generalizes poorly to new data, as indicated by a higher validation MSE.
- The optimal polynomial degree in this case was 5, which achieved the lowest validation MSE, balancing model complexity and generalization ability.

This highlights the importance of selecting the right model complexity. Higher-degree polynomials may seem to fit the training data well, but they can introduce excessive variance and reduce generalization. Validation-based model selection is crucial for avoiding underfitting and overfitting.

11 Conclusion

This study explored different approaches to linear and polynomial regression, analyzing their strengths, limitations, and impact on model performance. Key conclusions include:

- Linear regression methods (Scikit-Learn, pseudo-inverse, and gradient descent) produced equivalent solutions, demonstrating that both analytical and iterative approaches can be effective.
- Gradient descent is a viable alternative when computing the pseudo-inverse is computationally expensive, especially for large datasets.
- Polynomial regression demonstrated the trade-off between underfitting and overfitting, emphasizing the importance of selecting an optimal polynomial degree.
- The validation process plays a crucial role in model selection, ensuring the chosen model generalizes well to unseen data.

These findings reinforce fundamental concepts in regression modeling and provide insights into the practical application of machine learning techniques for both linear and non-linear relationships.