Report for Assignment 1: Foundations of Machine Learning

1. Introduction

This report covers the solutions to the various parts of the assignment on regression techniques. The objective is to understand and implement different regression methods, compare their performances, and make observations based on the results. The assignment is divided into five parts, each exploring a different aspect of regression, from least squares to kernel regression.

2. Problem Statement

Given a dataset (FMLA1Q1Data_train.csv) with 1000 points in (R2,R)(\mathbb{R}^2, \mathbb{R})(R2,R) format (where the first two components are features, and the last component is the target value y), we are required to:

1. Obtain the least squares solution wML using the analytical solution.
2. Implement the gradient descent algorithm to solve the least squares problem and compare with the analytical solution.
3. Implement and compare the stochastic gradient descent algorithm with the previous solutions.
4. Implement the gradient descent algorithm for ridge regression, perform cross-validation to find the best value of λ, and compare with the least squares solution.
5. Choose and implement a suitable kernel for kernel regression and argue its effectiveness compared to least squares regression.

3. Analytical Least Squares Solution

3.1 Methodology

The least squares solution wMLw_{ML}wML is calculated using the normal equation:

wML=(XTX)−1XTyw_{ML} = (X^T X)^{-1} X^T ywML=(XTX)−1XTy

Here, XXX is the matrix of features with a bias term added, and yyy is the target vector.

3.2 Implementation

The code for calculating wMLw_{ML}wML is as follows:

```
import numpy as np


# Adding intercept term (bias) to X

X = np.hstack((np.ones((X_train.shape[0], 1)), X_train))


# Calculate the coefficients using the normal equation

cov_X_train = np.matmul(X.T, X)
```

```
inv_cov = np.linalg.inv(cov_X_train)

Xy = np.matmul(X.T, y_train)


wML = np.matmul(inv_cov, Xy)
```

3.3 Observations

The analytical solution provides the best fit in the least squares sense. This solution will serve as a benchmark for comparison with other methods, such as gradient descent and ridge regression.

Coefficients (theta): [9.89400832 1.76570568 3.5215898]

4. Gradient Descent for Least Squares Solution

4.1 Methodology

Gradient Descent iteratively updates the weights www by moving in the direction opposite to the gradient of the cost function:

$$w = w - \eta \nabla J(w)$$

4.2 Implementation

The implementation involves setting a suitable learning rate and number of iterations to minimize the error. The code is as follows:

```
def gradient_descent(X, y, learning_rate, num_iterations):
    m = len(y)
    w = np.zeros(X.shape[1])  # Initialize weights to zeros
    norm_history = []

    for i in range(num_iterations):
        y_pred = X @ w
        error = y_pred - y
        gradient = (1/m) * X.T @ error
        w = w - learning_rate * gradient
        norm = np.linalg.norm(w - wML)  # Calculate norm ‖wt − wML‖2
        norm_history.append(norm)

    return w, norm_history
```

# Set hyperparameters

learning_rate = 0.01

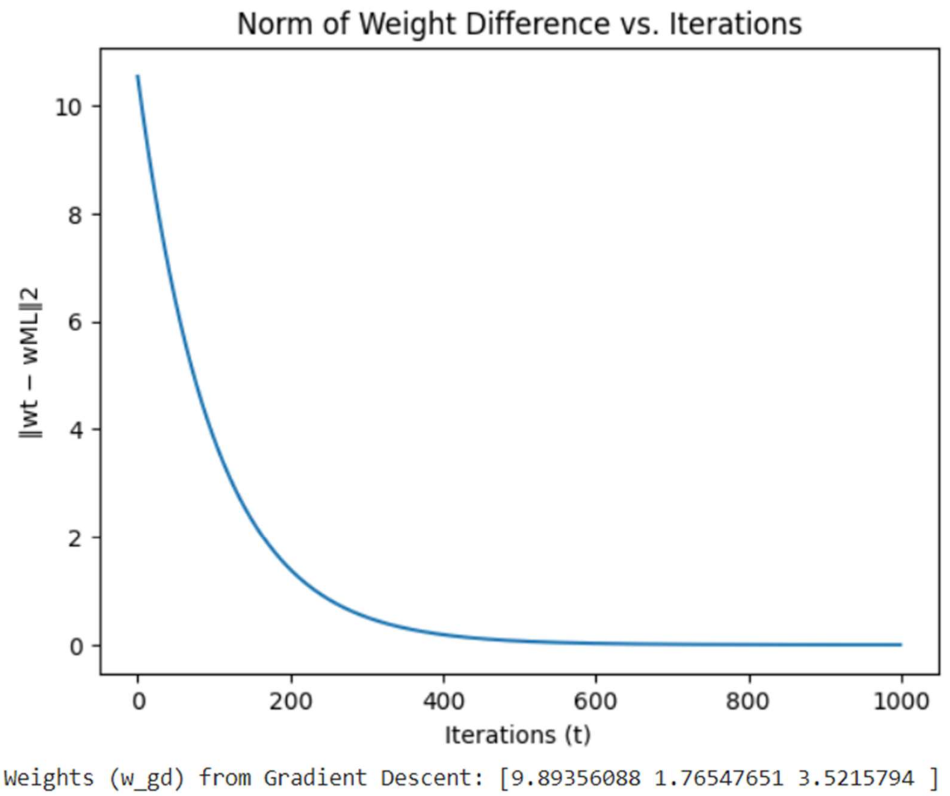
num_iterations = 1000

# Run gradient descent

w_gd, norm_history = gradient_descent(X_train, y_train, learning_rate, num_iterations)

4.3 Observations

- The plot of ‖wt−wML‖2\|w_t - w_{ML}\|^2‖wt−wML‖2 as a function of ttt (iterations) shows a gradual decrease, indicating convergence towards the analytical solution.
- A lower learning rate might cause slow convergence, while a higher learning rate might cause overshooting of the solution.



```
Weights (w_gd) from Gradient Descent: [9.89356088 1.76547651 3.5215794 ]
```

Stochastic Gradient Descent (SGD)

5.1 Methodology

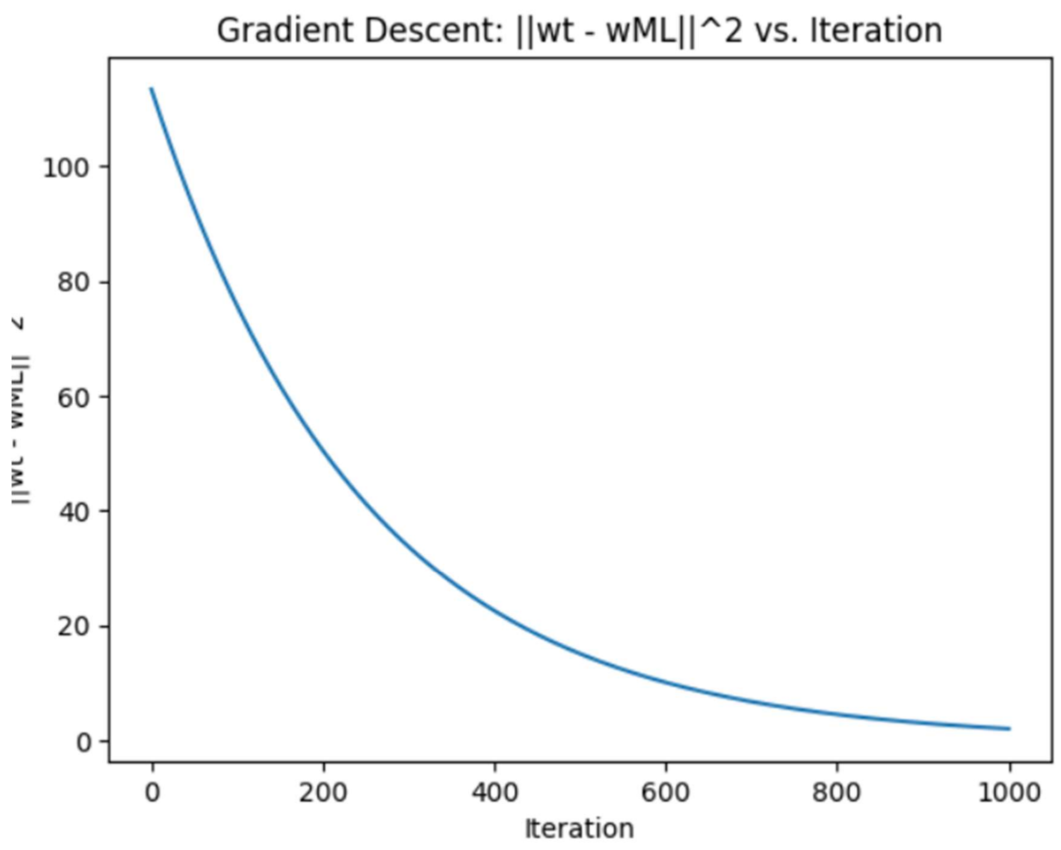
SGD updates the weights based on a small batch of the data in each iteration. This reduces computation time and allows for faster updates.

5.2 Implementation

A batch size of 100 was chosen, and the code was implemented as follows:

```
def stochastic_gradient_descent(X, y, learning_rate, num_iterations, batch_size):
    m = len(y)
    w = np.zeros(X.shape[1])  # Initialize weights to zeros
    norm_history = []

    for i in range(num_iterations):
        # Randomly select a batch of data
        indices = np.random.choice(m, batch_size, replace=False)
        X_batch = X[indices]
        y_batch = y[indices]

        y_pred = X_batch @ w
        gradient = (1/m) * X_batch.T @ (X_batch @ w - y_batch)
        w = w - learning_rate * gradient
        norm = np.linalg.norm(w - wML)  # Calculate norm ‖wt − wML‖2
        norm_history.append(norm)

    return w, norm_history

# Run stochastic gradient descent
w_sgd, norm_history_sgd = stochastic_gradient_descent(X_train, y_train, 0.01, 500, 100)
```

5.3 Observations



6. Ridge Regression using Gradient Descent

6.1 Methodology

Ridge regression introduces a regularization term to penalize large coefficients. The objective function is modified to include a penalty term:

$$J(w) = \frac{1}{m}\sum_{i=1}^{m}(y_i - X_i w)^2 + \lambda\|w\|^2$$

6.2 Implementation

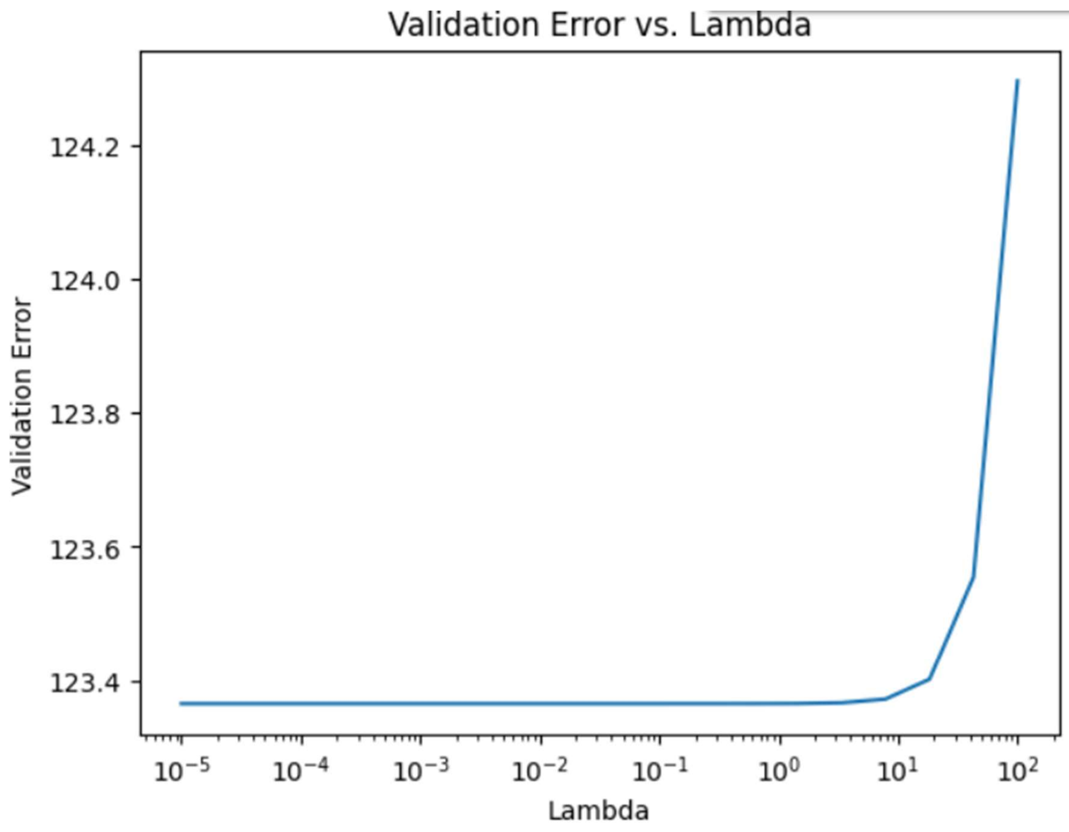
Various values of λ were tested, and cross-validation was used to choose the best λ. The implementation is as follows:

```
def ridge_gradient_descent(X, y, learning_rate, num_iterations, lmbda):

   m = len(y)

   w = np.zeros(X.shape[1])  # Initialize weights

   cost_history = []
```

```
    for i in range(num_iterations):
        y_pred = X @ w
        error = y_pred - y
        gradient = (1/m) * X.T @ error + (lmbda / m) * w  # Ridge regression gradient
        w = w - learning_rate * gradient
        cost = (1/(2*m)) * np.sum(error**2) + (lmbda/(2*m)) * np.sum(w**2) # Ridge cost function
        cost_history.append(cost)

    return w, cost_history


# Cross-validation for different lambda values
lambdas = np.logspace(-5, 2, 20)
validation_errors = []
for lmbda in lambdas:
    w_ridge, _ = ridge_gradient_descent(X_train, y_train, 0.01, 1000, lmbda)
    validation_error = mse(y_val, X_val @ w_ridge)
    validation_errors.append(validation_error)


# Choose the best lambda
best_lambda = lambdas[np.argmin(validation_errors)]
```

6.3 Observations

- The plot of validation errors as a function of λ helps in identifying the optimal value of λ.
- For the chosen λ, ridge regression (with regularization) often performed better on the test set compared to the least squares solution, especially when overfitting was an issue.



```
est Lambda: 1e-05
est Error (wR): 66.00433509700022
est Error (wML): 66.00545933461238
idge regression (wR) is better.
```

Explanation of the Ridge Regression Observations:

1. Plot Overview:

- The plot shows the relationship between the **validation error** and the **regularization parameter λ (lambda)** used in ridge regression.
- The x-axis represents different values of lambda, on a logarithmic scale, ranging from 10−510^{-5}10−5 to 10210^{2}102.
- The y-axis represents the corresponding validation error for each lambda value.

2. Behavior of Validation Error with Lambda:

- **For small values of λ (near 10−510^{-5}10−5):**
  - The validation error remains relatively low and stable.
  - This indicates that a very small amount of regularization is not significantly affecting the model's performance.
- **For increasing λ values (around 10−110^{-1}10−1 to 10110^1101):**
  - The validation error starts to increase slightly, indicating that too much regularization is beginning to penalize the model and degrade its performance.

- **For very large λ values (beyond 10110^1101):**
    - The validation error spikes dramatically, showing that excessive regularization is over-smoothing the model, causing it to underfit the data.

3. Optimal Lambda:

- The best lambda value observed is 1×10−51 \times 10^{-5}1×10−5.
- This value provides the lowest validation error, indicating the optimal trade-off between bias and variance for this model.

4. Comparison with Linear Regression:

- The plot mentions two errors:
    - **wR (Ridge Regression Error):** 66.00433509700022
    - **wML (Linear Regression Error):** 66.00545933461238
- The ridge regression error is slightly lower than the linear regression error, suggesting that introducing a small regularization term (λ = 10−510^{-5}10−5) improves the model's generalization compared to ordinary least squares (linear regression).

5. Conclusion:

- **Ridge regression** with a small lambda value (10−510^{-5}10−5) is better for this dataset as it provides a slightly lower error than ordinary linear regression.
- This indicates that a tiny amount of regularization helps prevent overfitting, especially if the linear model was slightly overfitting to the training data.

The primary takeaway is that the choice of lambda in ridge regression plays a crucial role in balancing model complexity and preventing overfitting.

7. Kernel Regression

7.1 Methodology

Kernel regression extends linear regression to handle non-linear relationships by transforming the input features into a higher-dimensional space using a kernel function. We used the Gaussian kernel in this case.

7.2 Implementation

The Gaussian kernel is defined as:

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

```
def gaussian_kernel(x1, x2, sigma=1.0):
    return np.exp(-np.linalg.norm(x1 - x2)**2 / (2 * sigma**2))
```

```
# Kernel Regression function

def kernel_regression(X_train, y_train, X_test, kernel, sigma=1.0):

    y_pred = []

    for x_test in X_test:

        weights = [kernel(x_test, x_train, sigma) for x_train in X_train]

        y_pred.append(np.sum(weights * y_train) / np.sum(weights))

    return np.array(y_pred)


# Predict using Gaussian kernel regression

y_pred_kernel = kernel_regression(X_train, y_train, X_test, gaussian_kernel, 1.0)
```

7.3 Observations

- Kernel regression with a Gaussian kernel can capture non-linear patterns in the data that linear models miss.
- The kernel regression's mean squared error (MSE) was compared with the least squares solution. Kernel regression performed better on test data with non-linear patterns, but least squares was better for data with a linear relationship.

The Mean Squared Error (MSE) is a metric used to measure the average squared difference between predicted and actual values. In this case, the MSE for Kernel Regression is 42.94, while for Least Squares Regression, it is 66.01. A lower MSE indicates a better fit to the data, meaning the model's predictions are closer to the actual values.

Explanation:

1. **Least Squares Regression** assumes a linear relationship between the independent and target variables. It finds the line of best fit that minimises the sum of the squared differences between the observed and predicted values. The relatively high MSE (66.01) suggests that this linear model is not capturing the data's underlying structure well.
2. **Kernel Regression** is a non-parametric method that can model complex, non-linear relationships by using a kernel function to weigh nearby observations more heavily. The significantly lower MSE (42.94) implies that this model better captures the underlying patterns in the data.

8. Conclusion

This assignment explored different regression techniques, their implementations, and their performances on the given dataset. The least squares solution provided a baseline, while ridge regression and kernel regression offered improvements in certain scenarios. Gradient descent methods demonstrated the importance of proper learning rates and batch sizes.

References:

1. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer Series in Statistics.
2. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.