Homework 2: Linear Regression

Ariv Ahuja

DS4400 - Machine Learning

Code Link: The Jupyter notebook containing all code for computational problems is available at: `HW2_code.ipynb`

Problem 1: Linear Regression [A] (15 points)

Given:

- $N = 30$ samples
- Sample mean of X: $\bar{X} = 50$; sample standard deviation: $\sigma_X = 10$
- Sample mean of Y: $\bar{Y} = 100$; sample standard deviation: $\sigma_Y = 20$
- Correlation coefficient: $\rho = 0.8$

Part 1: Compute θ_0 and θ_1

For simple linear regression $h_\theta(x) = \theta_0 + \theta_1 x$, the least squares solution gives:

The slope θ_1 can be computed using the correlation coefficient:

$$\theta_1 = \rho \cdot \frac{\sigma_Y}{\sigma_X} \tag{1}$$

$$\theta_1 = 0.8 \cdot \frac{20}{10} \tag{2}$$

$$\theta_1 = 0.8 \cdot 2 \tag{3}$$

$$\theta_1 = \boxed{1.6} \tag{4}$$

The intercept θ_0 is found using the fact that the regression line passes through $(\bar{X}, \bar{Y})$:

$$\bar{Y} = \theta_0 + \theta_1 \bar{X} \tag{5}$$

$$\theta_0 = \bar{Y} - \theta_1 \bar{X} \tag{6}$$

$$\theta_0 = 100 - 1.6 \cdot 50 \tag{7}$$

$$\theta_0 = 100 - 80 \tag{8}$$

$$\theta_0 = \boxed{20} \tag{9}$$

Therefore, the least squares linear regression model is:

$$h_\theta(x) = 20 + 1.6x$$

Part 2: Correlation coefficient $\rho = -0.8$

When $\rho = -0.8$:

$$\theta_1 = \rho \cdot \frac{\sigma_Y}{\sigma_X} \tag{10}$$

$$\theta_1 = -0.8 \cdot \frac{20}{10} \tag{11}$$

$$\theta_1 = \boxed{-1.6} \tag{12}$$

$$\theta_0 = \bar{Y} - \theta_1 \bar{X} \tag{13}$$

$$\theta_0 = 100 - (-1.6) \cdot 50 \tag{14}$$

$$\theta_0 = 100 + 80 \tag{15}$$

$$\theta_0 = \boxed{180} \tag{16}$$

Interpretation: When the correlation changes from positive to negative, the slope θ_1 changes sign (from 1.6 to -1.6), indicating that instead of income increasing with age, it now decreases with age. The intercept increases significantly (from 20 to 180) to compensate for the negative slope while maintaining the same predicted value at the mean of X.

Problem 2: Linear Regression [C] (15 points)

Part 1: Training the Model with sklearn

Using sklearn's LinearRegression, the model was trained on all features (excluding id, date, and zipcode).

Model Coefficients:

Feature	Coefficient
Intercept (θ_0)	520.4148
bedrooms	-12.5220
bathrooms	18.5276
sqft_living	56.7488
sqft_lot	10.8819
floors	8.0437
waterfront	63.7429
view	48.2001
condition	12.9643
grade	92.2315
sqft_above	48.2901
sqft_basement	27.1370
yr_built	-67.6431
yr_renovated	17.2714
lat	78.3757
long	-1.0352
sqft_living15	45.5777
sqft_lot15	-12.9301

Training Metrics:

- MSE: 31486.17
- R^2: 0.7265

Part 2: Testing Metrics

Testing Metrics:

- MSE: 57628.15
- R^2: 0.6544

Part 3: Interpretation

Top 5 Contributing Features (by absolute coefficient magnitude):

1. **grade** (92.23): The quality grade of the house has the strongest positive impact on price
2. **lat** (78.38): Location (latitude) significantly affects price, likely indicating premium areas
3. **yr_built** (-67.64): Older houses tend to have lower prices (negative coefficient)

4. **waterfront** (63.74): Waterfront properties command significantly higher prices
5. **sqft_living** (56.75): Larger living space increases house price

Model Fit Analysis:

- The R^2 of 0.7265 on training data indicates the model explains about 72.65% of the variance in house prices, which is a reasonably good fit.
- The testing R^2 of 0.6544 shows some generalization loss, but the model still explains about 65% of price variance on unseen data.
- The higher testing MSE (57628 vs 31486) compared to training MSE indicates some overfitting, but the difference is not extreme.
- The model error (RMSE $\approx$ 240 on testing) means predictions are typically off by about \$240,000 (since prices are in thousands), which is substantial but reasonable given housing price variability.

Problem 3: Closed-Form Solution [C] (15 points)

Implementation

The closed-form solution for linear regression is derived from minimizing the MSE loss function:

$$\theta = (X^T X)^{-1} X^T y$$

I implemented this using NumPy's pseudo-inverse for numerical stability:

```
def fit(self, X, y):
    X_b = np.c_[np.ones((X.shape[0], 1)), X]  # Add bias column
    self.theta = np.linalg.pinv(X_b.T @ X_b) @ X_b.T @ y
```

Results Comparison

Method	**Train MSE**	**Train R^2**	**Test MSE**	**Test R^2**
sklearn	31486.17	0.7265	57628.15	0.6544
Closed-form	34144.20	0.7034	56177.34	0.6631

Discussion: The closed-form implementation produces similar but not identical results to sklearn. The small differences arise because:

1. sklearn uses SVD decomposition internally for better numerical stability
2. There may be multicollinearity among features causing numerical sensitivity
3. Both methods achieve similar predictive performance, validating the correctness of the implementation

Problem 4: Polynomial Regression [C] (15 points)

Using the `sqft_living` feature, polynomial regression models were trained for degrees 1 through 5.

Results

Degree	Train MSE	Train R^2	Test MSE	Test R^2
1	57947.53	0.4967	88575.98	0.4687
2	54822.67	0.5238	71791.68	0.5694
3	53785.19	0.5329	99833.48	0.4012
4	52795.77	0.5415	250979.27	-0.5053
5	52626.11	0.5429	570616.91	-2.4225

Observations

- **Training Performance:** As the polynomial degree increases, the training MSE consistently decreases and training R^2 increases. This is expected because higher-degree polynomials can fit the training data more closely.
- **Testing Performance:** The optimal test performance is achieved at degree 2 (Test $R^2 = 0.5694$). Beyond this, test performance degrades rapidly.
- **Overfitting:** For degrees 4 and 5, the test R^2 becomes negative, indicating severe overfitting. The model performs worse than simply predicting the mean.
- **Bias-Variance Tradeoff:**
  - Degree 1 (linear): High bias, low variance - underfitting
  - Degree 2: Good balance between bias and variance
  - Degrees 3-5: Low bias, high variance - overfitting
- **Recommendation:** A quadratic model (degree 2) provides the best generalization for this dataset, suggesting a non-linear but not overly complex relationship between square footage and price.

Problem 5: Gradient Descent [C] (20 points)

Part 1: Implementation

The gradient descent algorithm for linear regression:

```
for i in range(n_iterations):
    predictions = X_b @ theta
    errors = predictions - y
    gradients = (2 / n_samples) * (X_b.T @ errors)
    theta -= learning_rate * gradients
```

Part 2: Results with Different Learning Rates

LR	Iter	Train MSE	Train R^2	Test MSE	Test R^2
0.01	10	235727.77	-1.0474	280568.71	-0.6828
0.01	50	69720.50	0.3945	97049.54	0.4179
0.01	100	36820.35	0.6802	63333.04	0.6201
0.1	10	35105.10	0.6951	61630.43	0.6304
0.1	50	31497.26	0.7264	57722.48	0.6538
0.1	100	31486.43	0.7265	57638.96	0.6543
0.5	10	Diverged	-	Diverged	-
0.5	50	Diverged	-	Diverged	-
0.5	100	Diverged	-	Diverged	-

Part 3: Observations

- **Learning Rate = 0.01:** The algorithm converges slowly. At 10 iterations, the model performs poorly (negative R^2). By 100 iterations, it achieves reasonable performance but hasn't fully converged to the optimal solution.
- **Learning Rate = 0.1:** This is the optimal learning rate for this problem. The algorithm converges quickly - even at 10 iterations, it achieves good performance. By 50-100 iterations, it matches the closed-form solution closely (Train MSE $\approx$ 31486, matching sklearn).
- **Learning Rate = 0.5:** The algorithm diverges completely, with MSE exploding to astronomical values. This demonstrates that too large a learning rate causes the gradient updates to overshoot the minimum, leading to divergence.
- **Convergence:** With $\alpha = 0.1$, approximately 50 iterations are needed to reach near-optimal performance. The algorithm successfully converges to the same solution as the closed-form method.
- **Key Insight:** The choice of learning rate is critical. Too small leads to slow convergence; too large leads to divergence. Feature scaling (standardization) helps by making the loss surface more uniform, allowing for a single learning rate to work well across all features.

Problem 6: Ridge Regularization [A/C] (20 points)

Part 1: Derivation of Closed-Form Solution [A]

The ridge regression loss function is:

$$J(\theta) = \sum_{i=1}^{N}(h_\theta(x_i) - y_i)^2 + \lambda \sum_{j=1}^{d} \theta_j^2$$

In matrix form:

$$J(\theta) = (X\theta - y)^T(X\theta - y) + \lambda\theta^T\theta$$

Expanding:

$$J(\theta) = \theta^T X^T X\theta - 2\theta^T X^T y + y^T y + \lambda\theta^T\theta$$

Taking the gradient with respect to θ:

$$\nabla_\theta J(\theta) = 2X^T X\theta - 2X^T y + 2\lambda\theta$$

Setting the gradient to zero:

$$2X^T X\theta - 2X^T y + 2\lambda\theta = 0$$

$$X^T X\theta + \lambda\theta = X^T y$$

$$(X^T X + \lambda I)\theta = X^T y$$

Solving for θ:

$$\boxed{\theta = (X^T X + \lambda I)^{-1} X^T y}$$

Note: In practice, we don't regularize the bias term θ_0, so we use a modified identity matrix with a 0 in the top-left corner.

Part 2: Gradient Descent Implementation [C]

The gradient for ridge regression is:

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

Update rule:

$$\theta := \theta - \alpha \left[\frac{2}{N} X^T(X\theta - y) + \frac{2\lambda}{N}\theta \right]$$

Part 3: Simulated Data Experiment [C]

Data generation: $N = 1000$, $X_i \sim \text{Uniform}(-2, 2)$, $Y_i = 1 + 2X_i + e_i$ where $e_i \sim N(0, 2)$.

True parameters: Intercept = 1.0, Slope = 2.0

λ	**Intercept**	**Slope**	**MSE**	R^2
0 (OLS)	1.1377	1.9453	1.9499	0.7258
1	1.1377	1.9439	1.9499	0.7258
10	1.1372	1.9311	1.9502	0.7258
100	1.1325	1.8124	1.9740	0.7224
1000	1.1057	1.1225	2.8735	0.5960
10000	1.0710	0.2335	5.9471	0.1638

Observations:

- **Small λ (1, 10):** The estimates remain close to OLS and the true values. Regularization has minimal effect.
- **Moderate λ (100):** The slope starts to shrink toward zero (1.81 vs true 2.0), but the model still performs reasonably well.
- **Large λ (1000, 10000):** The slope is heavily penalized and shrinks dramatically toward zero. At $\lambda = 10000$, the slope is only 0.23, severely underestimating the true relationship.
- **Intercept Stability:** The intercept remains relatively stable across all λ values because it's not regularized.
- **Bias-Variance Tradeoff:** As λ increases, we introduce more bias (slope shrinks toward 0) in exchange for reduced variance. For this simple problem with sufficient data, OLS already performs well, so regularization doesn't help.
- **When Ridge Helps:** Ridge regression is most beneficial when there's multicollinearity or when $p \approx N$ (high-dimensional settings), not in this simple univariate case.