homework2

September 29, 2025

1 ECGR 4105-001, Homework 2

1.1 By Joshua Foster, 801268118

[52]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

2 Problem 1:

1. a) Develop a gradient decent training and evaluation code, from scratch, that predicts housing price based on the following input variables:

area, bedrooms, bathrooms, stories, parking

Identify the best parameters for your linear regression model, based on the above input variables.

Plot the training and validation losses (in a single graph, but two different lines). For the learning rate, explore different values between 0.1 and 0.01 (your choice). Initialize your parameters (thetas to zero). For the training iteration, choose what you believe fits the best.

[53]:
```
# Load data from CSV file
df = pd.read_csv('Housing.csv')

# Select inputs and output
features = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']
target = 'price'
X = df[features]
y = df[target]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Add bias term (intercept) to the input features
X_train_b = np.c_[np.ones((X_train.shape[0], 1)), X_train]
X_test_b = np.c_[np.ones((X_test.shape[0], 1)), X_test]
```

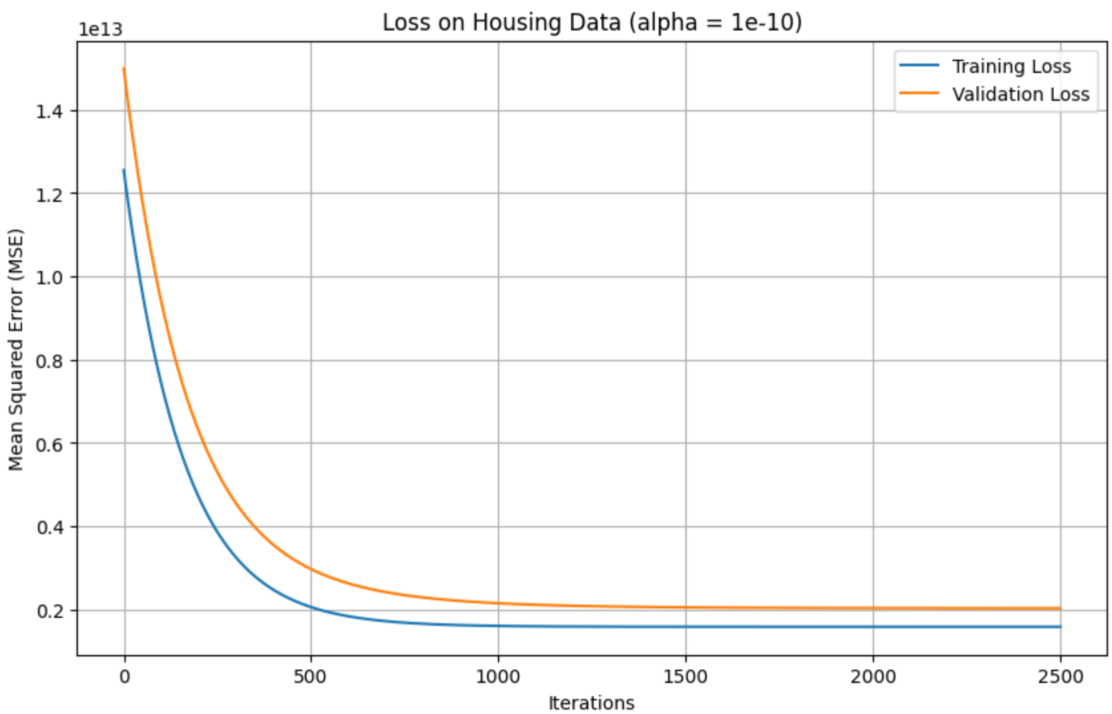
```python
# Convert outputs to numpy arrays
y_train = y_train.values
y_test = y_test.values

# Print shapes of the datasets
print("Training set shape:", X_train_b.shape, y_train.shape)
print("Testing set shape:", X_test_b.shape, y_test.shape)
print("First 5 rows of training inputs:\n", X_train_b[:5])
```

```
Training set shape: (436, 6) (436,)
Testing set shape: (109, 6) (109,)
First 5 rows of training inputs:
 [[1.000e+00 6.000e+03 3.000e+00 2.000e+00 4.000e+00 1.000e+00]
 [1.000e+00 7.200e+03 3.000e+00 2.000e+00 1.000e+00 3.000e+00]
 [1.000e+00 3.816e+03 2.000e+00 1.000e+00 1.000e+00 2.000e+00]
 [1.000e+00 2.610e+03 3.000e+00 1.000e+00 2.000e+00 0.000e+00]
 [1.000e+00 3.750e+03 3.000e+00 1.000e+00 2.000e+00 0.000e+00]]
```

[54]:
```python
def compute_cost(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    cost = (1 / (2 * m)) * np.sum(np.square(predictions - y))
    return cost

def gradient_descent(X_train, y_train, X_val, y_val, thetas, alpha, iterations):
    m = len(y_train)
    train_costs = []
    val_costs = []

    for i in range(iterations):
        predictions = X_train.dot(thetas)
        errors = predictions - y_train
        gradient = (1/m) * X_train.T.dot(errors)
        thetas = thetas - alpha * gradient

        train_cost = compute_cost(X_train, y_train, thetas)
        val_cost = compute_cost(X_val, y_val, thetas)
        train_costs.append(train_cost)
        val_costs.append(val_cost)

    return thetas, train_costs, val_costs
```

[55]:
```python
# Model parameters
alpha = 1e-10                   # Overflowing on anything larger
iterations = 2500              # Added more iterations due to smaller alpha
n_features = X_train_b.shape[1]

# Initialize theta (weights)
```

```python
thetas = np.zeros(n_features)

# Train the model using gradient descent
final_thetas, train_costs, val_costs = gradient_descent(X_train_b, y_train,␣
↪X_test_b, y_test, thetas, alpha, iterations)

# Plotting losses
plt.figure(figsize=(10, 6))
plt.plot(range(iterations), train_costs, label='Training Loss')
plt.plot(range(iterations), val_costs, label='Validation Loss')
plt.title(f'Loss on Housing Data (alpha = {alpha})')
plt.xlabel('Iterations')
plt.ylabel('Mean Squared Error (MSE)')
plt.legend()
plt.grid(True)
plt.show()

final_train_cost_1a = train_costs[-1]
final_val_cost_1a = val_costs[-1]

print(f"Final Training Loss: {final_train_cost_1a:,.2f}")
print(f"Final Validation Loss: {final_val_cost_1a:,.2f}\n")

print("Final Thetas (Parameters):")
for feature, theta in zip(['Intercept'] + features, final_thetas):
    print(f"  - {feature}: {theta:.6f}")
```



```
Final Training Loss: 1,589,377,249,371.06
Final Validation Loss: 2,034,967,268,653.01

Final Thetas (Parameters):
  - Intercept: 0.234359
  - area: 837.609534
  - bedrooms: 0.768682
  - bathrooms: 0.369945
  - stories: 0.545732
  - parking: 0.186700
```

1. b) Develop a gradient decent training and evaluation code, from scratch, that predicts housing price based on the following input variables:

Area, bedrooms, bathrooms, stories, mainroad, guestroom, basement, hotwaterheating, airconditioning, parking, prefarea

Identify the best parameters for your linear regression model, based on the above input variables.

Plot the training and validation losses (in a single graph, but two different lines) over your training iteration. Compare your linear regression model against problem 1 a. For the learning rate, explore different values between 0.1 and 0.01 (your choice). Initialize your parameters (thetas to zero). For the training iteration, choose what you believe fits the best.

[56]:
```python
df = pd.read_csv('Housing.csv')
# Mappinng categorical features to binary
categorical_features = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',␣
↪'airconditioning', 'prefarea']

for feature in categorical_features:
    df[feature] = df[feature].map({'yes': 1, 'no': 0})


# Selecting input and output
features_1b = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',␣
↪'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'parking',␣
↪'prefarea']
target_1b = 'price'
X_1b = df[features_1b]
y_1b = df[target_1b]

# Splitting the data into training and testing sets
X_1b_train, X_1b_test, y_1b_train, y_1b_test = train_test_split(X_1b, y_1b,␣
↪test_size=0.2, random_state=42)

# Add bias term (intercept) to the input features
X_1b_train_b = np.c_[np.ones((X_1b_train.shape[0], 1)), X_1b_train]
```

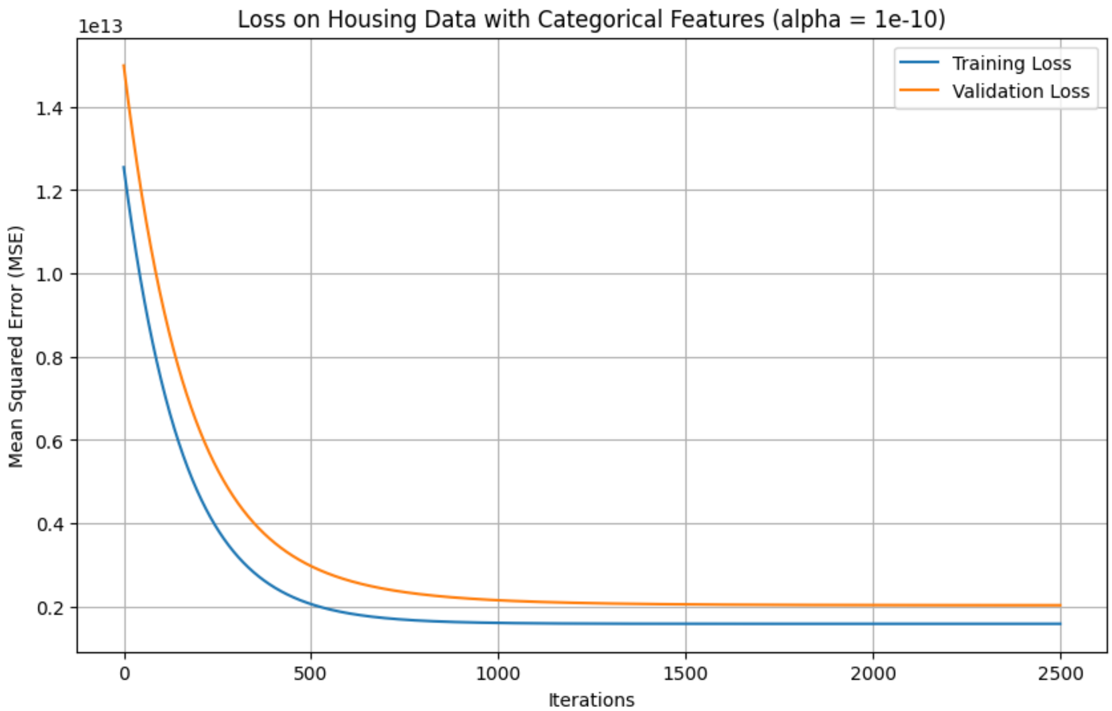
```python
X_1b_test_b = np.c_[np.ones((X_1b_test.shape[0], 1)), X_1b_test]

# Convert outputs to numpy arrays
y_1b_train = y_1b_train.values
y_1b_test = y_1b_test.values


# Print shapes of the datasets
print("Training set shape:", X_1b_train_b.shape, y_1b_train.shape)
print("Testing set shape:", X_1b_test_b.shape, y_1b_test.shape)
print("First 5 rows of training inputs:\n", X_1b_train_b[:5])
```

```
Training set shape: (436, 12) (436,)
Testing set shape: (109, 12) (109,)
First 5 rows of training inputs:
 [[1.000e+00 6.000e+03 3.000e+00 2.000e+00 4.000e+00 1.000e+00 0.000e+00
  0.000e+00 0.000e+00 1.000e+00 1.000e+00 0.000e+00]
 [1.000e+00 7.200e+03 3.000e+00 2.000e+00 1.000e+00 1.000e+00 0.000e+00
  1.000e+00 0.000e+00 1.000e+00 3.000e+00 0.000e+00]
 [1.000e+00 3.816e+03 2.000e+00 1.000e+00 1.000e+00 1.000e+00 0.000e+00
  1.000e+00 0.000e+00 1.000e+00 2.000e+00 0.000e+00]
 [1.000e+00 2.610e+03 3.000e+00 1.000e+00 2.000e+00 1.000e+00 0.000e+00
  1.000e+00 0.000e+00 0.000e+00 0.000e+00 1.000e+00]
 [1.000e+00 3.750e+03 3.000e+00 1.000e+00 2.000e+00 1.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00]]
```

[57]:

```python
# Model parameters
alpha = 1e-10                   # Overflowing on anything larger
iterations = 2500             # Added more iterations due to smaller alpha
n_features_1b = X_1b_train_b.shape[1]

# Initialize theta (weights)
thetas_1b = np.zeros(n_features_1b)

# Train the model using gradient descent
final_thetas_1b, train_costs_1b, val_costs_1b = gradient_descent(X_1b_train_b,
 ↪y_1b_train, X_1b_test_b, y_1b_test, thetas_1b, alpha, iterations)

# Plotting losses
plt.figure(figsize=(10, 6))
plt.plot(range(iterations), train_costs_1b, label='Training Loss')
plt.plot(range(iterations), val_costs_1b, label='Validation Loss')
plt.title(f'Loss on Housing Data with Categorical Features (alpha = {alpha})')
plt.xlabel('Iterations')
plt.ylabel('Mean Squared Error (MSE)')
plt.legend()
plt.grid(True)
```

```python
plt.show()


final_train_cost_1b = train_costs_1b[-1]
final_val_cost_1b = val_costs_1b[-1]

print(f"Final Training Loss: {final_train_cost_1b:,.2f}")
print(f"Final Validation Loss: {final_val_cost_1b:,.2f}\n")
print("Final Thetas (Parameters):")
for feature, theta in zip(['Intercept'] + features_1b, final_thetas_1b):
    print(f"  - {feature}: {theta:.6f}")
```




```
Final Training Loss: 1,589,377,087,734.80
Final Validation Loss: 2,034,967,119,793.32

Final Thetas (Parameters):
  - Intercept: 0.234359
  - area: 837.609487
  - bedrooms: 0.768682
  - bathrooms: 0.369945
  - stories: 0.545732
  - mainroad: 0.206691
  - guestroom: 0.059285
  - basement: 0.119515
```

```
- hotwaterheating: 0.019448
- airconditioning: 0.127361
- parking: 0.186700
- prefarea: 0.075154
```

3 Problem 2

2. a) Repeat problem 1 a, this time with input normalization and input standardization as part of your pre-processing logic. You need to perform two separate trainings for standardization and normalization. In both cases, you do not need to normalize the output!

Plot the training and validation losses for both training and validation set based on input standardization and input normalization. Compare your training accuracy between both scaling approaches as well as the baseline training in problem 1 a. Which input scaling achieves the best training? Explain your results.

[58]:
```
df = pd.read_csv('Housing.csv')
# select 1a features

features = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']
target = 'price'
X = df[features]
y = df[target]

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# convert to numpy arrays
y_train = y_train.values
y_test = y_test.values
```

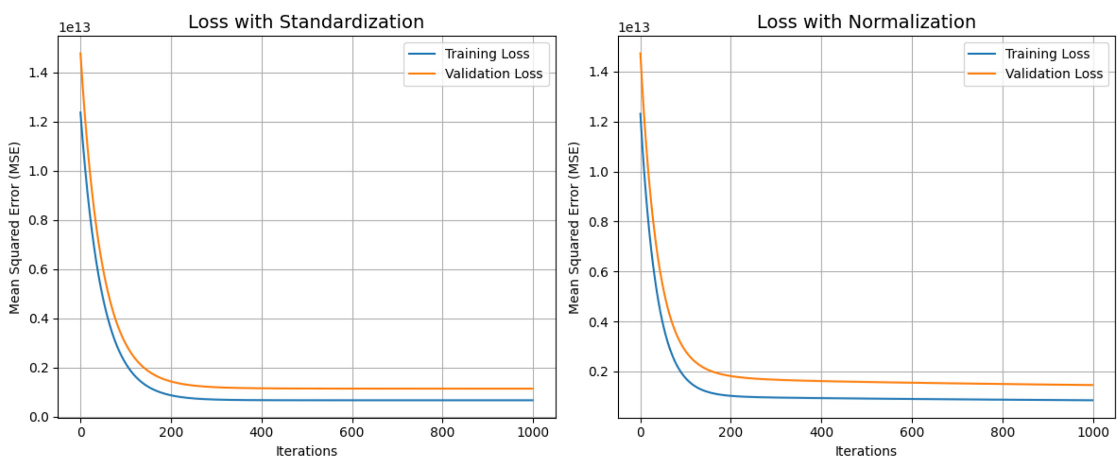
[59]:
```
# Input Standardization

X_train_std = X_train.copy()
X_test_std = X_test.copy()

# mean and dev from training set
train_mean = X_train_std.mean()
train_std = X_train_std.std()

# apply to both training and test data
X_train_std = (X_train_std - train_mean) / train_std
X_test_std = (X_test_std - train_mean) / train_std

# Add bias term (intercept) to the input features
X_train_std_b = np.c_[np.ones((X_train_std.shape[0], 1)), X_train_std]
X_test_std_b = np.c_[np.ones((X_test_std.shape[0], 1)), X_test_std]
```

```python
# Standardized Model Training
alpha = .01
iterations = 1000
n_features_std = X_train_std_b.shape[1]

# Initialize theta (weights)
thetas_std = np.zeros(n_features_std)

# Train the model using gradient descent
final_thetas_std, train_costs_std, val_costs_std =␣
↪gradient_descent(X_train_std_b, y_train, X_test_std_b, y_test, thetas_std,␣
↪alpha, iterations)
```

[60]:
```python
# Input Normalization
X_train_norm = X_train.copy()
X_test_norm = X_test.copy()

# Calculate min and max from training set
train_min = X_train_norm.min()
train_max = X_train_norm.max()

# Apply normalization to both training and test data
X_train_norm = (X_train_norm - train_min) / (train_max - train_min)
X_test_norm = (X_test_norm - train_min) / (train_max - train_min)

# Add bias term (intercept) to the input features
X_train_norm_b = np.c_[np.ones((X_train_norm.shape[0], 1)), X_train_norm]
X_test_norm_b = np.c_[np.ones((X_test_norm.shape[0], 1)), X_test_norm]

# Normalized Model Training
alpha = .01
iterations = 1000
n_features_norm = X_train_norm_b.shape[1]

# Initialize theta (weights)
thetas_norm = np.zeros(n_features_norm)

# Train the model using gradient descent
final_thetas_norm, train_costs_norm, val_costs_norm =␣
↪gradient_descent(X_train_norm_b, y_train, X_test_norm_b, y_test,␣
↪thetas_norm, alpha, iterations)
```

[61]:
```python
# Plot for Standardization
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(iterations), train_costs_std, label='Training Loss')
```

```
plt.plot(range(iterations), val_costs_std, label='Validation Loss')
plt.title('Loss with Standardization', fontsize=14)
plt.xlabel('Iterations')
plt.ylabel('Mean Squared Error (MSE)')
plt.legend()
plt.grid(True)

# Plot for Normalization
plt.subplot(1, 2, 2)
plt.plot(range(iterations), train_costs_norm, label='Training Loss')
plt.plot(range(iterations), val_costs_norm, label='Validation Loss')
plt.title('Loss with Normalization', fontsize=14)
plt.xlabel('Iterations')
plt.ylabel('Mean Squared Error (MSE)')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

# Compare final losses
final_train_cost_std = train_costs_std[-1]
final_val_cost_std = val_costs_std[-1]
final_train_cost_norm = train_costs_norm[-1]
final_val_cost_norm = val_costs_norm[-1]
print(f"Final Training Loss for 1a: {final_train_cost_1a:,.2f}")
print(f"Final Validation Loss for 1a: {final_val_cost_1a:,.2f}\n")
print(f"Final Training Loss with Standardization: {final_train_cost_std:,.2f}")
print(f"Final Validation Loss with Standardization: {final_val_cost_std:,.
↪2f}\n")
print(f"Final Training Loss with Normalization: {final_train_cost_norm:,.2f}")
print(f"Final Validation Loss with Normalization: {final_val_cost_norm:,.2f}")
```



```
Final Training Loss for 1a: 1,589,377,249,371.06
Final Validation Loss for 1a: 2,034,967,268,653.01

Final Training Loss with Standardization: 675,004,521,311.64
Final Validation Loss with Standardization: 1,146,408,517,350.48

Final Training Loss with Normalization: 849,748,793,747.95
Final Validation Loss with Normalization: 1,458,033,578,362.71
```

2. b) Repeat problem 1 b, this time with input normalization and input standardization as part of your pre-processing logic. You need to perform two separate trainings for standardization and normalization. In both cases, you do not need to normalize the output!

Plot the training and validation losses for both training and validation sets based on input standardization and input normalization. Compare your training accuracy between both scaling approaches and the baseline training in problem 1 b. Which input scaling achieves the best training? Explain your results.

[62]:
```
df = pd.read_csv('Housing.csv')
categorical_features = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea']
for feature in categorical_features:
    df[feature] = df[feature].map({'yes': 1, 'no': 0})

# Selecting input and output
features_1b = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'parking', 'prefarea']
target_1b = 'price'
X_1b = df[features_1b]
y_1b = df[target_1b]

# Splitting the data into training and testing sets
X_1b_train, X_1b_test, y_1b_train, y_1b_test = train_test_split(X_1b, y_1b, test_size=0.2, random_state=42)

# convert to numpy arrays
y_1b_train = y_1b_train.values
y_1b_test = y_1b_test.values
```

[63]:
```
# Input Standardization
X_train_1b_std = X_1b_train.copy()
X_test_1b_std = X_1b_test.copy()

# mean and dev from training set
train_mean_1b = X_train_1b_std.mean()
train_std_1b = X_train_1b_std.std()
```

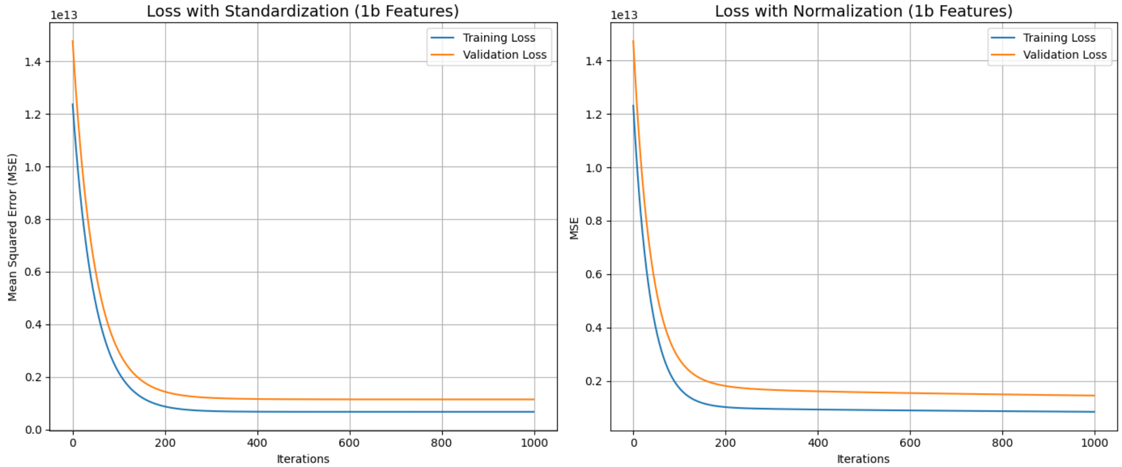
```python
# apply to both training and test data
X_train_1b_std = (X_train_1b_std - train_mean_1b) / train_std_1b
X_test_1b_std = (X_test_1b_std - train_mean_1b) / train_std_1b

# Add bias term (intercept) to the input features
X_train_1b_std_b = np.c_[np.ones((X_train_1b_std.shape[0], 1)), X_train_1b_std]
X_test_1b_std_b = np.c_[np.ones((X_test_1b_std.shape[0], 1)), X_test_1b_std]

# Standardized Model Training for 1b
alpha = .01
iterations = 1000
n_features_1b_std = X_train_1b_std_b.shape[1]

# Initialize theta (weights)
thetas_1b_std = np.zeros(n_features_1b_std)
final_thetas_1b_std, train_costs_1b_std, val_costs_1b_std =␣
↪gradient_descent(X_train_1b_std_b, y_1b_train, X_test_1b_std_b, y_1b_test,␣
↪thetas_1b_std, alpha, iterations)
```

[64]:

```python
# Input Normalization
X_train_1b_norm = X_1b_train.copy()
X_test_1b_norm = X_1b_test.copy()

# Calculate min and max from training set
train_min_1b = X_train_1b_norm.min()
train_max_1b = X_train_1b_norm.max()

# Apply normalization to both training and test data
X_train_1b_norm = (X_train_1b_norm - train_min_1b) / (train_max_1b -␣
↪train_min_1b)
X_test_1b_norm = (X_test_1b_norm - train_min_1b) / (train_max_1b - train_min_1b)

# Add bias term (intercept) to the input features
X_train_1b_norm_b = np.c_[np.ones((X_train_1b_norm.shape[0], 1)),␣
↪X_train_1b_norm]
X_test_1b_norm_b = np.c_[np.ones((X_test_1b_norm.shape[0], 1)), X_test_1b_norm]

# Normalized Model Training for 1b
alpha_1b_norm = .01
iterations = 1000

# Initialize theta (weights)
thetas_1b_norm = np.zeros(X_train_1b_norm_b.shape[1])
```

```
final_thetas_1b_norm, train_costs_1b_norm, val_costs_1b_norm =␣
↪gradient_descent(X_train_1b_norm_b, y_1b_train, X_test_1b_norm_b, y_1b_test,␣
↪thetas_1b_norm, alpha_1b_norm, iterations)
```

[65]:
```
# Plot for Standardization and Normalization for 1b
plt.figure(figsize=(14, 6))

# Standardization Plot
plt.subplot(1, 2, 1)
plt.plot(range(iterations), train_costs_std, label='Training Loss')
plt.plot(range(iterations), val_costs_std, label='Validation Loss')
plt.title('Loss with Standardization (1b Features)', fontsize=14)
plt.xlabel('Iterations')
plt.ylabel('Mean Squared Error (MSE)')
plt.legend()
plt.grid(True)

# Normalization Plot
plt.subplot(1, 2, 2)
plt.plot(range(iterations), train_costs_norm, label='Training Loss')
plt.plot(range(iterations), val_costs_norm, label='Validation Loss')
plt.title('Loss with Normalization (1b Features)', fontsize=14)
plt.xlabel('Iterations')
plt.ylabel('MSE')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

# Compare final losses for 1b
final_train_cost_1b_std = train_costs_1b_std[-1]
final_val_cost_1b_std = val_costs_1b_std[-1]
final_train_cost_1b_norm = train_costs_1b_norm[-1]
final_val_cost_1b_norm = val_costs_1b_norm[-1]
print(f"Final Training Loss for 1b: {final_train_cost_1b:,.2f}")
print(f"Final Validation Loss for 1b: {final_val_cost_1b:,.2f}\n")
print(f"Final Training Loss with Standardization (1b): {final_train_cost_1b_std:
↪,.2f}")
print(f"Final Validation Loss with Standardization (1b): {final_val_cost_1b_std:
↪,.2f}\n")
print(f"Final Training Loss with Normalization (1b): {final_train_cost_1b_norm:
↪,.2f}")
print(f"Final Validation Loss with Normalization (1b): {final_val_cost_1b_norm:
↪,.2f}")
```




```
Final Training Loss for 1b: 1,589,377,087,734.80
Final Validation Loss for 1b: 2,034,967,119,793.32

Final Training Loss with Standardization (1b): 496,244,620,138.02
Final Validation Loss with Standardization (1b): 900,107,923,435.58

Final Training Loss with Normalization (1b): 640,942,055,907.61
Final Validation Loss with Normalization (1b): 1,077,541,865,435.66
```

4 Problem 3

3. a) Repeat problem 2 a, this time by adding a parameters penalty to your loss function. Note that in this case, you need to modify the gradient decent logic for your training set, but you don't need to change your loss for the evaluation set.

Plot your results (both training and evaluation losses) for the best input scaling approach (standardization or normalization). Explain your results and compare them against problem 2 a.

[66]:
```
def regularize_cost(X, y, theta, lambda_param):
    m = len(y)
    mse_cost = compute_cost(X, y, theta)
    reg_penalty = (lambda_param / (2 * m)) * np.sum(np.square(theta[1:]))
    return mse_cost + reg_penalty

def regularized_gradient_descent(X_train, y_train, X_val, y_val, theta, alpha,
    iterations, lambda_param):
    m = len(y_train)
    train_costs = []
    val_costs = []

    for i in range(iterations):
```

```
        predictions = X_train.dot(theta)
        errors = predictions - y_train

        # Calculate the gradient
        gradient = (1/m) * X_train.T.dot(errors)

        # Add the regularization term to the gradient (for all thetas except␣
↪the bias term)
        reg_gradient_term = (lambda_param / m) * theta
        reg_gradient_term[0] = 0

        # Update thetas
        theta = theta - alpha * (gradient + reg_gradient_term)

        # Calculate and store costs
        train_cost = regularize_cost(X_train, y_train, theta, lambda_param)
        val_cost = compute_cost(X_val, y_val, theta)

        train_costs.append(train_cost)
        val_costs.append(val_cost)

    return theta, train_costs, val_costs
```

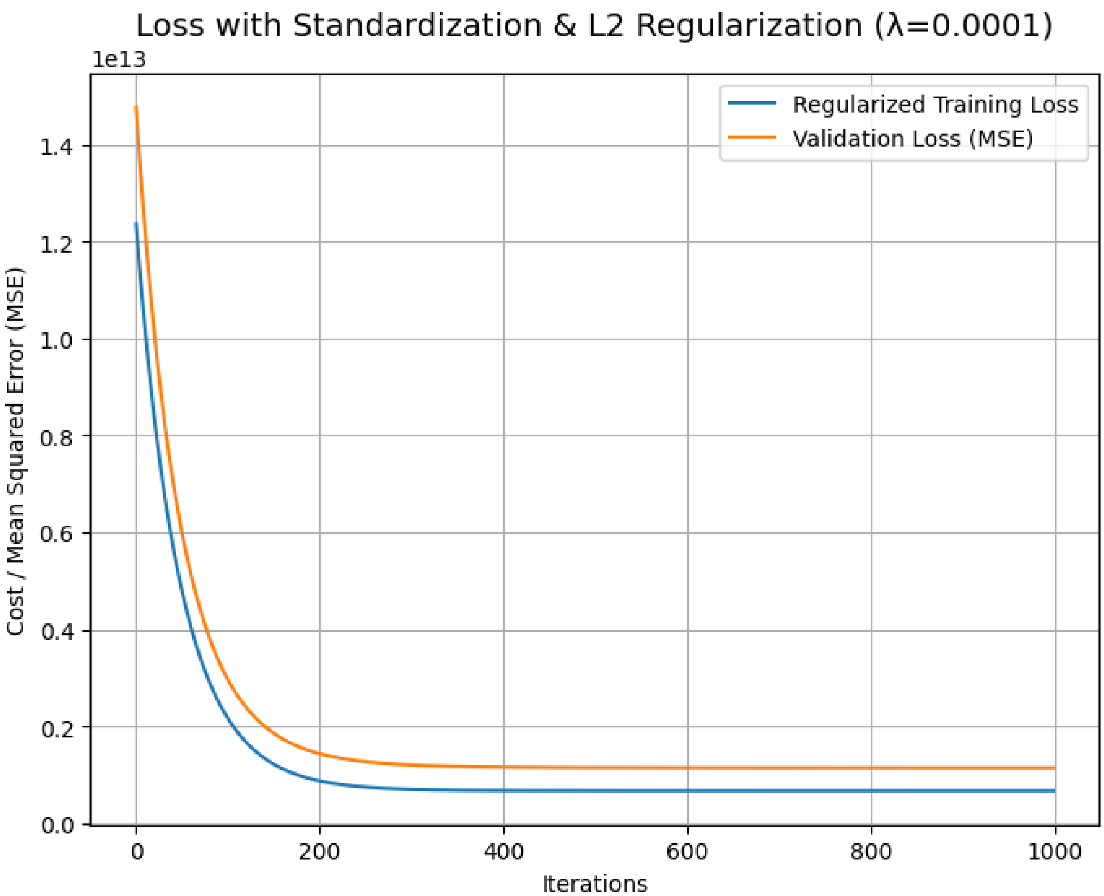
[67]:

```
df = pd.read_csv('Housing.csv')
features = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']
X = df[features]
y = df[target]

# Split and preprocess data using Standardization
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
y_train = y_train.values
y_test = y_test.values

train_mean = X_train.mean()
train_std = X_train.std()
X_train_std = (X_train - train_mean) / train_std
X_test_std = (X_test - train_mean) / train_std

X_train_b = np.c_[np.ones((X_train_std.shape[0], 1)), X_train_std]
X_test_b = np.c_[np.ones((X_test_std.shape[0], 1)), X_test_std]
```

[68]:

```
# Model training with Regularization
alpha = .01
iterations = 1000
lambda_param = .0001
```

```
initial_thetas = np.zeros(X_train_b.shape[1])

final_thetas_reg, train_costs_reg, val_costs_reg = regularized_gradient_descent(
    X_train_b, y_train, X_test_b, y_test, initial_thetas, alpha, iterations,␣
  ↪lambda_param
)

# Plotting
plt.figure(figsize=(8, 6))
plt.plot(range(iterations), train_costs_reg, label='Regularized Training Loss')
plt.plot(range(iterations), val_costs_reg, label='Validation Loss (MSE)')
plt.title(f'Loss with Standardization & L2 Regularization ( ={lambda_param})',␣
  ↪fontsize=14)
plt.xlabel('Iterations')
plt.ylabel('Cost / Mean Squared Error (MSE)')
plt.legend()
plt.grid(True)
plt.show()

# Comparison of final losses
final_train_cost_reg = train_costs_reg[-1]
final_val_cost_reg = val_costs_reg[-1]
print(f"Final Training Loss with Standardization: {final_train_cost_std:,.2f}")
print(f"Final Validation Loss with Standardization: {final_val_cost_std:,.
  ↪2f}\n")
print(f"Final Training Loss with Regularization: {final_train_cost_reg:,.2f}")
print(f"Final Validation Loss with Regularization: {final_val_cost_reg:,.2f}")
```




```
Final Training Loss with Standardization: 675,004,521,311.64
Final Validation Loss with Standardization: 1,146,408,517,350.48

Final Training Loss with Regularization: 675,004,642,936.23
Final Validation Loss with Regularization: 1,146,408,574,691.62
```

3. b) Repeat problem 2 b, this time by adding a parameters penalty to your loss function. Note that in this case, you need to modify the gradient decent logic for your training set, but you don't need to change your loss for the evaluation set.

Plot your results (both training and evaluation losses) for the best input scaling approach (standardization or normalization). Explain your results and compare them against problem 2 b.

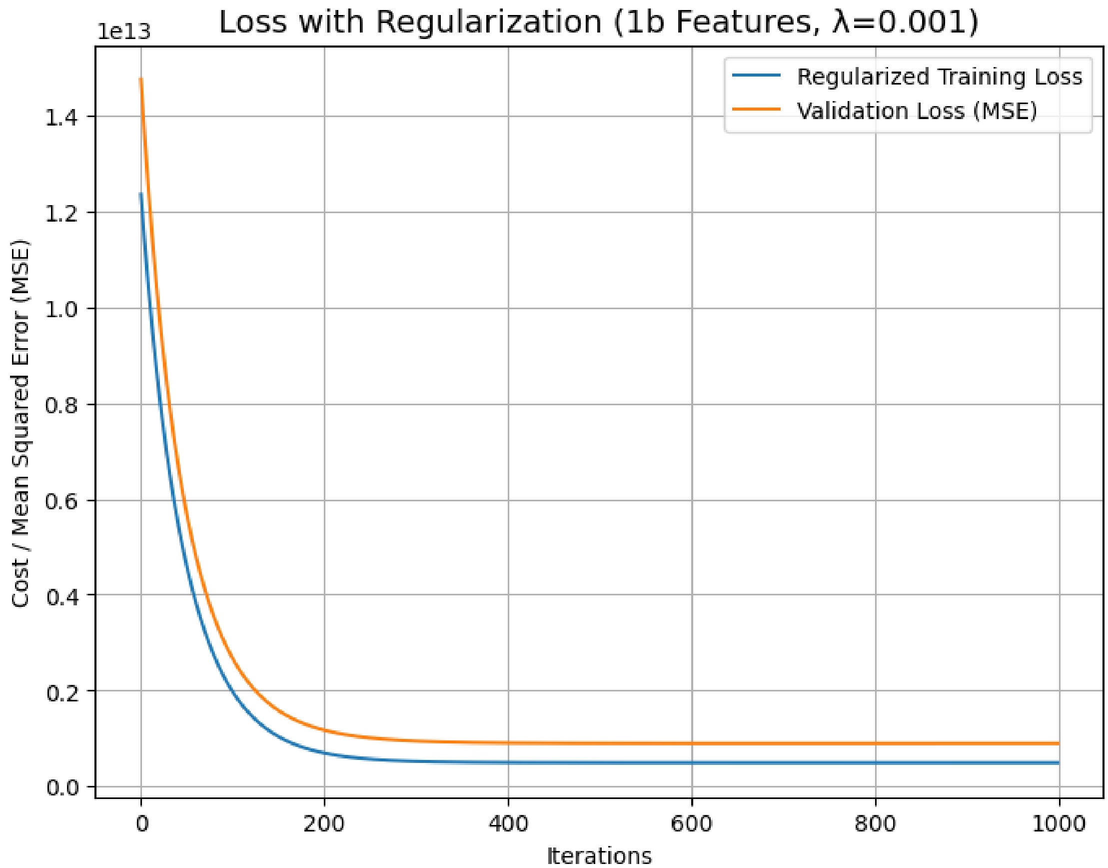
[69]:
```
df = pd.read_csv('Housing.csv')
# Convert categorical features
categorical_features = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',␣
 ↪'airconditioning', 'prefarea']
for feature in categorical_features:
    df[feature] = df[feature].apply(lambda x: 1 if x == 'yes' else 0)
```

```python
# Select input and output
features = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'parking', 'prefarea']
target = 'price'
X = df[features]
y = df[target]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
y_train = y_train.values
y_test = y_test.values

# Standardize based on the training set
train_mean = X_train.mean()
train_std = X_train.std()
X_train_std = (X_train - train_mean) / train_std
X_test_std = (X_test - train_mean) / train_std

# Add bias term
X_train_b = np.c_[np.ones((X_train_std.shape[0], 1)), X_train_std]
X_test_b = np.c_[np.ones((X_test_std.shape[0], 1)), X_test_std]
```

[70]:

```python
# Model parameters
alpha = 0.01
iterations = 1000
lambda_param = 0.001

initial_thetas = np.zeros(X_train_b.shape[1])

final_thetas_reg_1b, train_costs_reg_1b, val_costs_reg_1b = regularized_gradient_descent(
    X_train_b, y_train, X_test_b, y_test, initial_thetas, alpha, iterations, lambda_param
)

# Plotting
plt.figure(figsize=(8, 6))
plt.plot(range(iterations), train_costs_reg_1b, label='Regularized Training Loss')
plt.plot(range(iterations), val_costs_reg_1b, label='Validation Loss (MSE)')
plt.title(f'Loss with Regularization (1b Features,  ={lambda_param})', fontsize=14)
plt.xlabel('Iterations')
plt.ylabel('Cost / Mean Squared Error (MSE)')
plt.legend()
```

```
plt.grid(True)
plt.show()

# Comparison of final losses
final_train_cost_reg_1b = train_costs_reg_1b[-1]
final_val_cost_reg_1b = val_costs_reg_1b[-1]
print(f"Final Training Loss with Standardization (1b): {final_train_cost_1b_std:
↪,.2f}")
print(f"Final Validation Loss with Standardization (1b): {final_val_cost_1b_std:
↪,.2f}\n")
print(f"Final Training Loss with Regularization (1b): {final_train_cost_reg_1b:
↪,.2f}")
print(f"Final Validation Loss with Regularization (1b): {final_val_cost_reg_1b:
↪,.2f}")
```




```
Final Training Loss with Standardization (1b): 496,244,620,138.02
Final Validation Loss with Standardization (1b): 900,107,923,435.58

Final Training Loss with Regularization (1b): 496,245,820,843.93
```

```
Final Validation Loss with Regularization (1b): 900,108,228,731.53
```