sp21-bcs-017-A-Muhammad_Haroon_Shahzad_assignment_1

October 21, 2023

1 SP21-BCS-017 SECTION A

1.1 MUHAMMAD HAROON SHAHZAD

1.2 INSTRUCTOR: DR ZEESHAN GILLANI

1.3 Assignmnet 1: Machine Learning

- apply multiple linear regression cost function, and gradient decen
- use 4 different learning rates to check iteration and accuracy
- u can use labs link share and modify code accordingly Github_Link

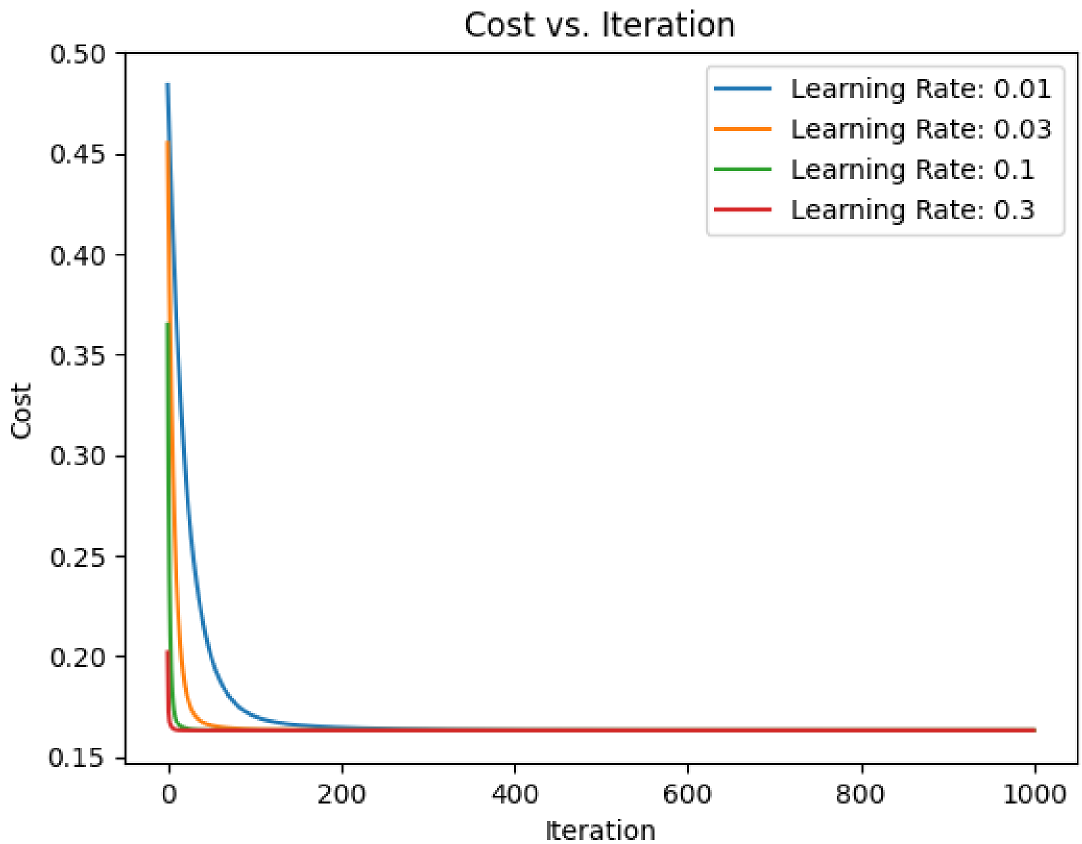
[ ]:
```
import copy
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('Housing.csv')

# Feature scaling
data = (data - data.mean()) / data.std()

# Extract the features and the target variable
X = data[['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom',␣
↪'basement', 'hotwaterheating', 'airconditioning', 'parking', 'prefarea']].
↪values
y = data['price'].values

# Add a column of ones to X to account for the intercept term
X = np.c_[np.ones(X.shape[0]), X]

# Apply multiple linear regression cost function and gradient descent

def compute_cost(X, y, w, b):
    m = X.shape[0]
    predictions = np.dot(X, w) + b
    cost = np.sum((predictions - y) ** 2) / (2 * m)
```

```python
    return cost

def compute_gradient(X, y, w, b):
    m = X.shape[0]
    predictions = np.dot(X, w) + b
    errors = predictions - y
    grad_w = np.dot(X.T, errors) / m
    grad_b = np.sum(errors) / m
    return grad_w, grad_b

def gradient_descent(X, y, w, b, alpha, iterations):
    J_history = []
    for i in range(iterations):
        grad_w, grad_b = compute_gradient(X, y, w, b)
        w -= alpha * grad_w
        b -= alpha * grad_b
        cost = compute_cost(X, y, w, b)
        J_history.append(cost)
        if i % 100 == 0:
            print(f'Iteration {i}: Cost {cost}')
    return w, b, J_history

# Different learning rates to test
learning_rates = [0.01, 0.03, 0.1, 0.3]

for alpha in learning_rates:
    # initialize parameters
    initial_w = np.zeros(X.shape[1])
    initial_b = 0.0

    # some gradient descent settings
    iterations = 1000

    # run gradient descent
    w_final, b_final, J_hist = gradient_descent(X, y, initial_w, initial_b, 
 ↪alpha, iterations)

    print(f"Learning Rate: {alpha}")
    print(f"Final parameters - b: {b_final:.2f}, w: {w_final}")

    # Calculate and print accuracy based on Mean Squared Error
    y_pred = np.dot(X, w_final) + b_final
    mse = np.mean((y_pred - y) ** 2)
    accuracy = 1 - mse / np.var(y)
    print(f"Accuracy: {accuracy}")

    # plot cost versus iteration
```

```python
    plt.plot(J_hist, label=f'Learning Rate: {alpha}')

plt.title("Cost vs. Iteration")
plt.xlabel("Iteration")
plt.ylabel("Cost")
plt.legend()
plt.show()
```

```
Iteration 0: Cost 0.4840623686158884
Iteration 100: Cost 0.17009374295868648
Iteration 200: Cost 0.16440533313908606
Iteration 300: Cost 0.16351626112193934
Iteration 400: Cost 0.16324619793358733
Iteration 500: Cost 0.1631554504276159
Iteration 600: Cost 0.16312288813029743
Iteration 700: Cost 0.1631105539775403
Iteration 800: Cost 0.1631056790058378
Iteration 900: Cost 0.16310369061871588
Learning Rate: 0.01
Final parameters - b: 0.00, w: [1.46362025e-16 2.87418870e-01 5.15306090e-02
2.70248426e-01
 2.11017323e-01 8.73246716e-02 6.53554374e-02 9.67589853e-02
 1.00547882e-01 2.19227312e-01 1.37533562e-01 1.49351742e-01]
Accuracy: 0.6731946238161817
Iteration 0: Cost 0.45507918848176016
Iteration 100: Cost 0.16349956550118083
Iteration 200: Cost 0.16312196769925522
Iteration 300: Cost 0.1631036185444531
Iteration 400: Cost 0.1631023535826239
Iteration 500: Cost 0.16310225483060986
Iteration 600: Cost 0.1631022468432009
Iteration 700: Cost 0.1631022461909305
Iteration 800: Cost 0.163102246137526
Iteration 900: Cost 0.1631022461331504
Learning Rate: 0.03
Final parameters - b: 0.00, w: [1.41183701e-16 2.87933326e-01 5.06976970e-02
2.70222552e-01
 2.12013759e-01 8.69553205e-02 6.49451613e-02 9.75219277e-02
 1.00517067e-01 2.18932048e-01 1.37505997e-01 1.49201025e-01]
Accuracy: 0.6731958671236324
Iteration 0: Cost 0.36473887062443394
Iteration 100: Cost 0.16310272752870453
Iteration 200: Cost 0.16310224623109496
Iteration 300: Cost 0.16310224613278057
Iteration 400: Cost 0.16310224613275984
Iteration 500: Cost 0.16310224613275978
Iteration 600: Cost 0.16310224613275984
Iteration 700: Cost 0.1631022461327598
```

```
Iteration 800: Cost 0.16310224613275984
Iteration 900: Cost 0.1631022461327598
Learning Rate: 0.1
Final parameters - b: 0.00, w: [7.05449970e-17 2.87933431e-01 5.06975316e-02
2.70222505e-01
 2.12014007e-01 8.69552383e-02 6.49450785e-02 9.75221083e-02
 1.00517049e-01 2.18931963e-01 1.37506018e-01 1.49200984e-01]
Accuracy: 0.6731958671236982
Iteration 0: Cost 0.20177349913598103
Iteration 100: Cost 0.16310224613276528
Iteration 200: Cost 0.1631022461327598
Iteration 300: Cost 0.16310224613275978
Iteration 400: Cost 0.16310224613275978
Iteration 500: Cost 0.16310224613275978
Iteration 600: Cost 0.16310224613275978
Iteration 700: Cost 0.16310224613275978
Iteration 800: Cost 0.16310224613275978
Iteration 900: Cost 0.16310224613275978
Learning Rate: 0.3
Final parameters - b: 0.00, w: [9.20364702e-17 2.87933431e-01 5.06975316e-02
2.70222505e-01
 2.12014007e-01 8.69552383e-02 6.49450785e-02 9.75221083e-02
 1.00517049e-01 2.18931963e-01 1.37506018e-01 1.49200984e-01]
Accuracy: 0.6731958671236982
```


Cost vs. Iteration
Learning Rate: 0.01
Learning Rate: 0.03
Learning Rate: 0.1
Learning Rate: 0.3
Cost
Iteration
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0
200
400
600
800
1000