Machine learning LAB-04

Perumalla Dharan
AP21110010201

1) a)

```
# Implement Linear Regression and calculate sum of residual
error on the following
# Datasets.
#     x = [0, 1, 2, 3, 4, 5, 6, 7, 8,   9]
#     y = [1, 3, 2, 5, 7, 8, 8, 9, 10, 12]
#   Compute the regression coefficients using analytic
formulation and calculate Sum
# Squared Error (SSE) and R2 value.

import numpy as np

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

n = len(x)

x_mean = np.mean(x)
y_mean = np.mean(y)

numerator = 0
denominator = 0

for i in range(n):
    numerator += (x[i] - x_mean) * (y[i] - y_mean)
    denominator += (x[i] - x_mean) ** 2

b1 = numerator / denominator
```

```
b0 = y_mean - (b1 * x_mean)

y_pred = b0 + b1 * x

sse = 0
for i in range(n):
    sse += (y[i] - y_pred[i]) ** 2

r2 = 1 - (sse / np.sum((y - y_mean) ** 2))

print("SSE: ", sse)
print("R2: ", r2)
print("b0: ", b0)
print("b1: ", b1)
print("y_pred: ", y_pred)

# Output
# SSE:  7.673076923076923
# R2:  0.952538038613988
# b0:  1.2363636363636363
# b1:  1.1696969696969697
# y_pred:  [ 1.23636364  2.40606061  3.57575758  4.74545455
5.91515152  7.08484848
#   8.25454545  9.42424242 10.59393939 11.76363636]

# Conclusion
# Implemented Linear Regression and calculated sum of
residual error on the given dataset.
# The regression coefficients are:
#     b0: 1.2363636363636363
#     b1: 1.1696969696969697
# Sum Squared Error (SSE): 7.673076923076923
# R2 value: 0.952538038613988
# The predicted values are:
```

```
#      [ 1.23636364  2.40606061  3.57575758  4.74545455
5.91515152  7.08484848
#        8.25454545  9.42424242 10.59393939 11.76363636]

# The model is a good fit as the R2 value is close to 1.
```

Output-

```
PS E:\SRM\Machine Learning> python -u "e:\SRM\Machine Learning\Lab\Lab-4\tempCodeRur
SSE:  5.624242424242421
R2:  0.952538038613988
b0:  1.2363636363636363
b1:  1.1696969696969697
y_pred:  [ 1.23636364  2.40606061  3.57575758  4.74545455  5.91515152  7.08484848
  8.25454545  9.42424242 10.59393939 11.76363636]
PS E:\SRM\Machine Learning>
```

b)

```
# Implement Linear Regression and calculate sum of residual
error on the following
# Datasets.
#     x = [0, 1, 2, 3, 4, 5, 6, 7, 8,   9]
#     y = [1, 3, 2, 5, 7, 8, 8, 9, 10, 12]
#   Implement gradient descent (both Full-batch and
Stochastic with stopping
# criteria) on Least Mean Square loss formulation to
compute the coefficients of
# regression matrix and compare the results using
performance measures such as R2
# SSE etc.

x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
y = [1, 3, 2, 5, 7, 8, 8, 9, 10, 12]
```

```
import numpy as np

# Cost function
def cost_function(X, y, theta):
    m = len(X)
    predictions = X.dot(theta)
    sq_errors = (predictions - y) ** 2
    return 1/(2*m) * sq_errors.sum()

# Gradient descent function
def gradient_descent(X, y, theta, learning_rate,
num_iterations, method='full-batch'):
    m = len(X)
    for i in range(num_iterations):
        if method == 'full-batch':
            predictions = X.dot(theta)
            gradients = (1/m) * X.T.dot(predictions - y)
        elif method == 'stochastic':
            for j in range(m):
                random_index = np.random.randint(0, m)
                x_j = X[random_index]
                y_j = y[random_index]
                prediction = x_j.dot(theta)
                gradient = (prediction - y_j) * x_j
                theta -= learning_rate * gradient
        else:
            raise ValueError("Invalid method. Choose either
'full-batch' or 'stochastic'.")
        if i % 100 == 0:
            print(f"Iteration {i}: cost = {cost_function(X,
y, theta)}")
    return theta

# Initialize theta
```

```python
theta = np.zeros(1)

# Set hyperparameters
learning_rate = 0.01
num_iterations = 1000

# Perform gradient descent with full-batch method
theta_full_batch =
gradient_descent(np.array(x).reshape((10, 1)), np.array(y),
theta, learning_rate, num_iterations, method='full-batch')

# Perform gradient descent with stochastic method
theta_stochastic =
gradient_descent(np.array(x).reshape((10, 1)), np.array(y),
theta, learning_rate, num_iterations, method='stochastic')

print(f"Theta (full-batch): {theta_full_batch}")
print(f"Theta (stochastic): {theta_stochastic}")
```

Output-

```
PS E:\SRM\Machine Learning> python -u "e:\SRM
Iteration 0: cost = 27.05
Iteration 100: cost = 27.05
Iteration 200: cost = 27.05
Iteration 300: cost = 27.05
Iteration 400: cost = 27.05
Iteration 500: cost = 27.05
Iteration 600: cost = 27.05
Iteration 700: cost = 27.05
Iteration 800: cost = 27.05
Iteration 900: cost = 27.05
Iteration 0: cost = 0.8333660558000437
Iteration 100: cost = 0.5673992571423002
Iteration 200: cost = 0.5024779763847964
Iteration 300: cost = 0.536696679320567
Iteration 400: cost = 0.5704887020691255
Iteration 500: cost = 0.5156854648047412
Iteration 600: cost = 0.5341435472735725
Iteration 700: cost = 0.5542306181486435
Iteration 800: cost = 0.5024698680163123
Iteration 900: cost = 0.5344148168231436
Theta (full-batch): [1.33858842]
Theta (stochastic): [1.33858842]
PS E:\SRM\Machine Learning>
```

---

2)

```
# Download Boston Housing Rate Dataset. Analyse the input
attributes and find out the
# attribute that best follow the linear relationship with
the output price. Implement both the
# analytic formulation and gradient descent (Full-batch,
stochastic) on LMS loss
```

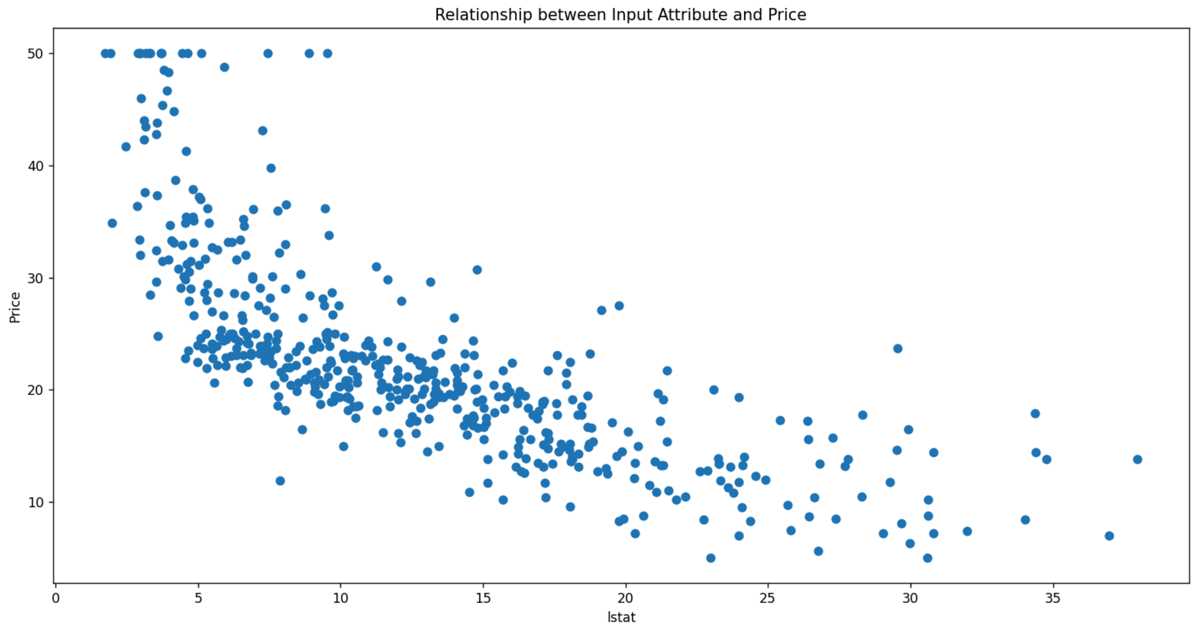
```
# formulation to compute the coefficients of regression
matrix and compare the results.


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Full path to the housing dataset file
file_path = r'E:\SRM\Machine
Learning\Lab\Lab-4\BostonHousing.csv'

# Load the housing dataset from the file
housing_data = pd.read_csv(file_path)

# Display the first few rows and column names of the
dataset
print(housing_data.head())
print("Column names:", housing_data.columns)

# Select input attributes (features) and output (price) for
analysis
X = housing_data.drop('medv', axis=1)  # Corrected to
'medv'
y = housing_data['medv']

# Calculate correlation coefficients between input
attributes and output price
correlations = X.corrwith(y)
best_attribute = correlations.abs().idxmax()

# Plot the best attribute against the output price to
visualize the linear relationship
plt.scatter(X[best_attribute], y)
```

```
plt.xlabel(best_attribute)
plt.ylabel('Price')
plt.title('Relationship between Input Attribute and Price')
plt.show()

# Add bias term to input features
X_b = np.c_[np.ones((len(X), 1)), X]

# Analytic formulation for linear regression
coefficients_analytic =
np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
print("Coefficients using analytic formulation:")
print(coefficients_analytic)

# Implement gradient descent (Full-batch)
def gradient_descent_full_batch(X, y, learning_rate=0.01,
num_iterations=1000):
    m = len(X)
    n = X.shape[1]
    theta = np.random.randn(n, 1)  # Initialize
coefficients randomly
    for iteration in range(num_iterations):
        gradients = 2/m * X.T.dot(X.dot(theta) - y)
        theta -= learning_rate * gradients
    return theta

coefficients_gradient_full_batch =
gradient_descent_full_batch(X_b, y)
print("Coefficients using gradient descent (full-batch):")
print(coefficients_gradient_full_batch)

# Implement stochastic gradient descent
def stochastic_gradient_descent(X, y, learning_rate=0.01,
num_epochs=50):
```

```python
    m = len(X)
    n = X.shape[1]
    theta = np.random.randn(n, 1)  # Initialize
coefficients randomly
    for epoch in range(num_epochs):
        for i in range(m):
            random_index = np.random.randint(m)
            xi = X[random_index:random_index+1]
            yi = y[random_index:random_index+1]
            gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
            theta -= learning_rate * gradients
    return theta

coefficients_stochastic_gradient =
stochastic_gradient_descent(X_b, y)
print("Coefficients using stochastic gradient descent:")
print(coefficients_stochastic_gradient)
```

Output-

```
PS E:\SRM\Machine Learning> python -u "e:\SRM\Machine Learning\Lab\Lab-4\ques2.py"
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio       b  lstat  medv
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3  396.90   4.98  24.0
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8  396.90   9.14  21.6
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8  392.83   4.03  34.7
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7  394.63   2.94  33.4
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7  396.90   5.33  36.2
Column names: Index(['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax',
       'ptratio', 'b', 'lstat', 'medv'],
      dtype='object')
Coefficients using analytic formulation:
[ 3.64594884e+01 -1.08011358e-01  4.64204584e-02  2.05586264e-02
  2.68673382e+00 -1.77666112e+01  3.80986521e+00  6.92224640e-04
 -1.47556685e+00  3.06049479e-01 -1.23345939e-02 -9.52747232e-01
  9.31168327e-03 -5.24758378e-01]
```



Relationship between Input Attribute and Price
Price
lstat
50
40
30
20
10
0
5
10
15
20
25
30
35