linear-regression-lab

January 23, 2023

1 Linear Regression Lab

1.1 Load Data

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

df = pd.read_csv("housing.csv")
#df['price'] /= 1000
#df['sqft'] /= 1000
df
```

[ ]:
```
# Set up our training data
x_train = df['sqft'].to_numpy()
y_train = df['price'].to_numpy()

# To use only a subset of the training data, use a slice of the arrays above.

print("Training data for x: ", x_train)
print("Training data for y: ", y_train)
```

[ ]:
```
# Make a nice plot

plt.scatter(x_train, y_train)
plt.show()
```

[ ]:
```
# Make a nice plot with a line based on the equation y = wx + b

plt.scatter(x_train, y_train)

# Set up dummy values for w and b
w = 0
b = 0

# Generate points on our line y = wx + b to graph
line_x_points = np.linspace(min(x_train), max(x_train), 100)
line_y_points = [w * x + b for x in line_x_points]
```

```python
plt.plot(line_x_points, line_y_points, c='red')
plt.show()
```

[ ]:
```python
# Let's turn it into a function:

def make_plot(x_train, y_train, w, b):
    plt.scatter(x_train, y_train)

    line_x_points = np.linspace(min(x_train), max(x_train), 100)
    line_y_points = [w * x + b for x in line_x_points]

    plt.plot(line_x_points, line_y_points, c='red')
    plt.show()

make_plot(x_train, y_train, 0, 0)
```

[ ]:
```python
# Let's define a function to predict y = f(x) = wx + b

def make_prediction(x, w, b):
    return w * x + b
```

[ ]:
```python
# Let's define a function to compute the cost J(w, b) on a set of data:

def compute_cost(x_data, y_data, w, b):
    """
    x_data and y_data are lists
    w and b are scalars
    return: gradient of w, gradient of b
    """
    m = len(x_data) # number of data points
    cost = 0

    for i in range(m):
        y_hat = make_prediction(x_data[i], w, b)
        y = y_data[i]
        cost += (y_hat - y)**2
    total_cost = (1 / (2 * m)) * cost

    return total_cost
```

[ ]:
```python
# Output total cost for our data set:

w = .3
b = 500

print(compute_cost(x_train, y_train, w, b))
```

[ ]:
```python
# Let's write a function to compute the gradient:

def compute_gradient(x_data, y_data, w, b):
    """
    x_data and y_data are lists
    w and b are scalars
    return: gradient of w, gradient of b
    """
    m = len(x_data) # number of data points
    gradient_w = 0
    gradient_b = 0

    for i in range(m):
        y_hat = make_prediction(x_data[i], w, b)
        y = y_data[i]

        gradient_w += (y_hat - y) * x_data[i]
        gradient_b += (y_hat - y)

    gradient_w /= m
    gradient_b /= m

    return gradient_w, gradient_b
```

[ ]:
```python
# Let's write code to run gradient descent:

w = 0
b = 0
ALPHA = .0000001

J_sequence = []

print("x train is", x_train)
print("y train is", y_train)

for ctr in range(0, 100):
    print(ctr)
    print(w, b)
    print("Cost is", compute_cost(x_train, y_train, w, b))
    (gradient_w, gradient_b) = compute_gradient(x_train, y_train, w, b)
    print("Gradients", gradient_w, gradient_b)
    w -= ALPHA * gradient_w
    b -= ALPHA * gradient_b
    J_sequence.append(compute_cost(x_train, y_train, w, b))

print("Final w and b:", w, b)
```

[ ]:
```python
# Let's plot the cost as a function of number of iterations of the
# gradient descent algorithm.

plt.scatter(range(0, len(J_sequence)), J_sequence)
plt.show()
```

[ ]:
```python
# Produce final plot of our data and our line:

make_plot(x_train, y_train, w, b)
```

[ ]:
```python
# Make a new prediction:

sqft = 2000
prediction = make_prediction(sqft, w, b)

print("A house with", sqft, "square feet would sell for around", prediction,␣
 ↪"dollars.")
```

[ ]:
```python

```