Machine Learning

HW1 :: Linear Regression

By using Python & Google colab

모바일시스템공학과 손보경 (32212190)

ssg020910@naver.com

2023.10.14.

목 차

1. Introduction

선형 회귀란 알려진 데이터 값을 사용하여 알 수 없는 데이터 값을 예측하는 데이터 분석 기법이다. 알 수 없는 변수와 알려진 변수를 선형 방정식으로 수학적으로 모델링한다. 선형 회귀 모델은 비교적 간단하며 예측을 생성하기 위한 해석하기 쉬운 수학 공식을 제공한다. 기계 학습 및 인공지능은 선형 회귀를 사용하여 복잡한 문제를 해결한다.

수업에서 배웠던 여러 선형 회귀 방식 중 경사 하강법, 정규 방정식으로 주어진 데이터 셋에 대해 선형 회귀를 수행하는 프로그램을 작성해 보았다.

2. Requirements

Index	Requirement
1	Model : y = ax + b Approach: Gradient Descent
2	Model: y = ax2 + bx + c Approach: Normal Equation
3	Report a) Program description (comments on important code lines) b). How to run (so that I can test your program) c). A set of snapshots (of progress, final result, and etc.) d). Results · Table of parameters · Plots of the models

3. Program description

코드 설명에 들어가기에 앞서, 본 프로그램은 파이썬 및 구글 코랩을 통해 작성하였는데, 데이터 분석 및 머신 러닝 모델 개발을 위한 흔한 방법의 하나기 때문이다. 파이썬은 데이터 과학 및 머신 러닝 분야에서 널리 사용되는 프로그래밍 언어 중 하나로, 다양한 라이브러리와 도구를 지

원한다. 데이터 처리 및 시각화를 효과적으로 수행할 수 있으므로 사용하게 되었다. 구글 코랩은 무료로 사용할 수 있는 클라우드 기반의 주피터 노트북 환경을 제공한다. 이를 통해 브라우저에서 코드를 작성하고 실행할 수 있으며, 별도로 파이썬 환경을 설정하거나 설치할 필요가 없어서 편리하여 사용했다.

3-1. Import

```
[143] import matplotlib.pyplot as plt
      import numpy as np
      import csv
```

import matplotlib.pyplot as plt : 데이터 시각화를 위해 사용되었다. 데이터를 그래프로 그리고 결과를 시각적으로 표현하기 위해 pyplot 모듈을 사용하였다. 이를 통해 산점도를 나타내고 회귀 모델의 그래프를 그릴 수 있다.

import numpy as np : NumPy는 수치 계산을 수행하는 데 사용되는 파이썬 라이브러리이며, 위 코드에서 데이터를 배열 형식으로 다루는 데 사용되었다.

import csv : CSV 파일을 읽고 데이터를 파싱하기 위한 파이썬 표준 라이브러리 중 하나이며 주어진 CSV 파일의 데이터를 읽어오는 데 사용되었다.

3-2. Read file

```
[144] x, y = [], []

      with open('data_hw1.csv', 'r') as file:       # 데이터셋 입력
          next(file)  # Skip the header row
          reader = csv.reader(file, delimiter=',')
          for row in reader:
              x.append(float(row[0]))
              y.append(float(row[1]))

      x_data = np.array(x)                          # 넘파이 배열로 변환
      y_data = np.array(y)
```

with open('data_hw1.csv', 'r') as file : open 함수를 사용하여 파일을 연다.

next(file) : 첫 번째 행인 헤더 행을 건너뛴다. 헤더 행은 열 이름이 포함되기 때문에 건너뛰었다.

reader & for문 : csv.reader를 사용하여 파일을 CSV 형식으로 읽는다. 데이터를 읽어와서 각 데이터 포인트에서 첫 번째 열을 x 리스트에, 두 번째 열을 y 리스트에 추가한다. 데이터를 실수

로 변환하여 리스트에 저장한다.

x_data & y_data : 선형 회귀에 편리하게 NumPy 배열로 변환한다.

3-3. Set initial coefficients, hyper parameter, epoch

```
[3]  a = 0                                    # 초기화
     b = 0

     learning_rate = 0.01
     epochs = 1000
```

3-4. Gradient descent

```
[9]  for i in range(epochs) :
       y_pred = a*x_data + b
       error = y_data - y_pred

       a_grad = -(2/len(x_data))*sum(x_data*(error))
       b_grad = -(2/len(x_data))*sum(y_data - y_pred)

       a = a - learning_rate*a_grad
       b = b - learning_rate*b_grad

       if i%100 == 0 :
         print("epoch = %3.f ₩t| y = [%.04f]x + [%.04f] ₩t| error = %.04f" % (i, a, b, error.mean()))
```

```
epoch =   0     | y = [5.5105]x + [-0.8650]     | error = 9.6032
epoch = 100     | y = [0.5453]x + [3.2395]      | error = 0.3476
epoch = 200     | y = [0.7444]x + [3.6515]      | error = 0.1192
epoch = 300     | y = [0.8277]x + [3.8032]      | error = 0.0451
epoch = 400     | y = [0.8592]x + [3.8607]      | error = 0.0171
epoch = 500     | y = [0.8712]x + [3.8825]      | error = 0.0065
epoch = 600     | y = [0.8757]x + [3.8907]      | error = 0.0025
epoch = 700     | y = [0.8774]x + [3.8938]      | error = 0.0009
epoch = 800     | y = [0.8781]x + [3.8950]      | error = 0.0004
epoch = 900     | y = [0.8783]x + [3.8955]      | error = 0.0001
```

y_pred = a*x_data + b : 현재의 a와 b 값으로부터 예측된 종속 변수 y의 값들을 계산한다. 이것은 현재의 모델을 사용하여 예측한 값이다.

error = y_data - y_pred : 실제 값과 예측값의 오차를 계산한다.

a_grad & b_gard : 경사 하강법의 경사(gradient)를 계산한다. 경사는 모델의 파라미터(a와 b)를 업데이트하는 데 사용된다. 선형 회귀 모델의 평균 제곱 오차(Mean Squared Error, MSE)를 최소화하는 방향으로 파라미터를 조정하기 위해 사용하였다.

a = a - learning_rate*a_grad & b = b - learning_rate*b_grad : 파라미터를 업데이트 한다.

if문 : 매 100번째 에폭마다 현재의 에폭 번호, 업데이트된 a 및 b 값, 그리고 오차의 평균을 출력한다. 학습 중에 모델의 진행을 모니터링하기 위해서이다.

3-5. Print gradient descent

```
[5]  a = np.array(a)
     b = np.array(b)

     print("Result is")
     print("y = [%.04f]x + [%.04f]" % (a, b))
     print("₩n")

     plt.scatter(x, y, s = 0.1)
     plt.plot(x, a * x + b, color='red')
     plt.show()
```

a = np.array(a) & b = np.array(b) : 값을 계산하기 위해 형 변환을 했다.

print() : 파라미터를 소수점 4자리까지 출력한다.

pyplot : 산점도로 원본 데이터값을, 1차 함수로 학습된 선형 회귀 모델을 그린다.

3-6. Normal equation

```
[7]  # 새로운 x^2 특성 생성
     x_squared = x_data**2
     X = np.vstack((x_squared, x_data, np.ones(len(x_data)))).T

     # 계수를 찾기 위해 선형 회귀 수행
     coefficients = np.linalg.lstsq(X, y, rcond=None)[0]
     a, b, c = coefficients
```

x_squared = x_data**2 : 원본 변수를 제곱하여 새로운 변수 생성

X = np.vstack((x_squared, x_data, np.ones(len(x_data)))).T : x_squared, x_data, 그리고 상수항(1로 이루어진 배열)을 세로로 쌓아서 특성 행렬 X를 만든다.

coefficients = np.linalg.lstsq(X, y, rcond=None)[0] : 선형 회귀 모델의 계수(회귀 계수)를 찾기 위해 최소 제곱법을 사용한다. np.linalg.lstsq 함수는 최소 제곱법을 적용하여 최적의 계수를 찾는다.

3-7. Print normal equation

```
[8]  a = np.array(a)
     b = np.array(b)
     c = np.array(c)

     x_data.sort()
     y_new = a * (x_data**2) + b * x_data + c

     print("Result is")
     print("y = [%.04f]x^2 + [%.04f]x + [%.04f]" % (a, b, c))
     print("₩n")

     plt.scatter(x, y, s = 0.1)
     plt.plot(x_data, y_new, color='red')
     plt.show()
```

x_data.sort() : 그래프를 그릴 때 x_data의 값이 오름차순으로 정렬되어 시각화에 문제가 없게 만든다.

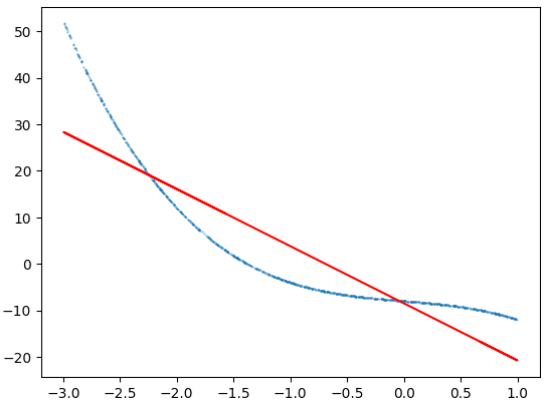
y_new = a * (x_data**2) + b * x_data + c : 학습된 2차 다항식 모델을 사용하여 새로운 예측값을 계산한다.

4. Result

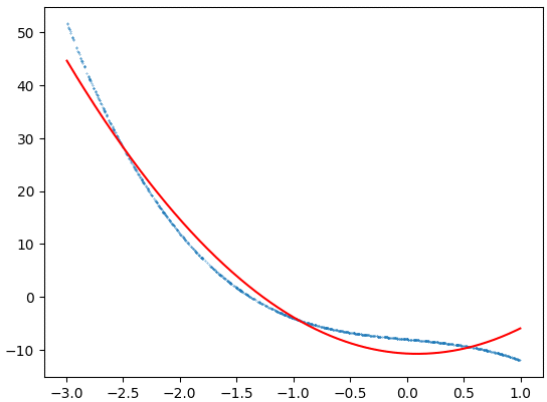
위에서 본 코드를 돌리면 어떻게 출력이 되는지 알아보도록 하겠다.

	a	b	c
Task1	-12.287677425739354	-8.49249639288247	
Task2	5.8207489315827345	-1.0570819679231782	-10.704822547454379

각각의 파라미터 값은 위의 표와 같다. 아래는 원래 데이터(파란색)와 학습된 데이터(빨간색)을 그래프로 출력한 모습이다.



▲ [그림 1] Gradient Descent



▲ [그림 2] Normal Equation

5. Build Environment

1) Google Colab(https://colab.research.google.com/)에 접속한다.

2) [노트열기] - [업로드]에서 해당 파일을 업로드한다.

3) 왼쪽 사이드바에 위치한 폴더를 들어간 후, data_hw1.csv를 세션 저장소에 업로드한다.

4) 상단바에 위치한 [런타임] - [모두 실행]을 이용해 코드를 실행한다.