점계수평균법을 이용한 선형회귀 연구

(PCM Linear Regression)

김동환

2021년 1월

차례

요약

- cost function을 사용하여 최적의 모델을 찾는 일반적인 회귀 문제의 접근법과는 다르게, cost function을 사용하지 않고, 선형 회귀문제를 해결하는 알고리즘이다. 점계수평균법(Pcm Linear Regression)의 핵심기술은 구역을 나누어 샘플을 뽑고, 함수를 예측하고, 예측된 함수들의 평균으로 최종 모델을 예측하는 것이다. 학습과정에는 많은 데이터를 압축하기 위하여 데이터 전처리 과정을 거치고, 압축한 데이터로 골고루 샘플을 뽑아 함수를 예측할 수 있도록 구역을 나누고, 연립방정식을 이용하여 여러 함수를 예측해낸다. 그 후 예측된 함수들의 평균을 구하여 최종 모델을 예측할 수 있다. 학습이 완료가 되면 선형 회귀 모델이 만들어진다. Cost function을 이용하지 않아 회귀 모델의 본질과는 멀어질 수 있지만, 이 기술이 새로운 본질자체를 만들어내는 계기가 될 수 있기 때문에 cost function을 이용하지 않는 새로운 접근법을 만들어냄으로서, 회귀 문제를 해결하는 새로운 방법을 만들고 싶었고, 문제를 해결할 때 시도할 수 있는 방법의 범위를 넓혀 더 많은 성과와 연구가 이루어졌으면 하는 마음에 만들게 되었다. 비록 지금은 짧디 짧은 줄기일 수 있겠지만, 시간이 지나고 많은 시도를 거듭함으로서 그 짧은 줄기에 갈래가 갈라지고 혹은 조금씩 변해가고, 계속 성장하며 여러 성과들이 나올 것이다.

1. 서론

1.1. 연구 목적

- 현재 머신러닝의 회귀 부분에 많은 역할을 하고 있는 선형회귀를 cost function 이라는 알고리즘을 기반하여 만들 수 있다. 저자는 cost function을 공부하고 직접 cost function을 이용한 머신러닝을 개발하던 도중 회귀 분야에서 좀 더 새로운 접근하여 회귀 문제를 해결하는 알고리즘이 없을까 하는 생각에 연구하게 되었다. 또한 점계수평균법을 연구하고, 개발함으로서 머신러닝의 회귀를 더 재밌게 배우고, 많은 사람들이 점계수평균법을 조금씩 변형해가며, 재밌는 기술들이 나오고 성과를 만들어내고, 4차산업혁명에 더 많은 기여를 했으면 하기 때문에 연구하게 되었다.

2. 선행연구

2.1 용어와 정의

- 점계수평균법: 데이터들의 구역을 나누고, 구역에서 데이터를 하나씩 뽑아 연립방정식을 이용하여 함수들을 예측하고, 함수들의 평균을 내어 최종 모델을 예측하는 기술이다. (자세한 내용은 기술설명을 참고)

- PCM Linear Regression: PCM는 Point Coefficient Mean의 약자이고, PCM Linear Regression 또한 점계수평균법과 같이 데이터들의 구역을 나누고, 구역에서 데이터를 하나씩 뽑고 연립방정식으로 함수들을 예측하여, 함수들의 평균을 구하여 최종 모델을 예측하는 기술이다. (자세한 내용은 기술설명을 참고)

3. 기술설명

3.1 점계수평균법

- 점계수평균법은 데이터 전처리, 함수 예측. 의 순서로 설계되어있다. 파이썬으로 작성되었다.

3.1.1 데이터 전처리

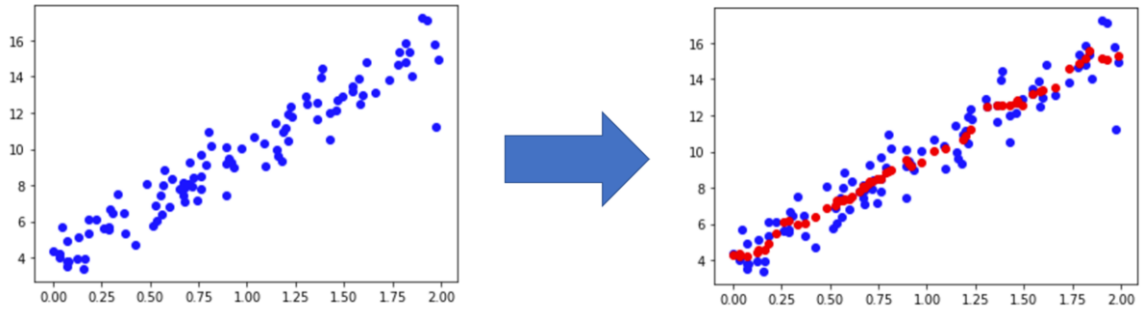
3.1.1.1 데이터 압축

- 먼저 **데이터 압축** 과정에서는 x 값이 비슷한 y 값들끼리의 평균값을 구하고, 실제 데이터에 반영한다. 데이터 전처리를 하는 이유는 노이즈가 있는 데이터들을 최대한 줄여 불필요한 함수가 예측되지 않도록 하기 위해서이다. 해당 코드는 다음과 같다.

```
data = list(zip(X, y))
data = pd.DataFrame(data, columns=["X", "y"])

def duplicate(x):
    duplicate_data = data[(data["X"] > (x - dp)) & (data["X"] < (x + dp))]["y"]
    return sum(duplicate_data) / len(duplicate_data)
data["y"] = data["X"].apply(lambda x: duplicate(x))
data = data.drop_duplicates(["y"], keep="first")
data = data.sort_values(by=["X"], axis=0)
data = data.reset_index(drop=True)
```

파란색을 원래 데이터, 빨간색을 전처리한 데이터로 표시하면 다음과 같은 그래프를 그릴 수 있다.



3.1.1.2 데이터 구역 나누기

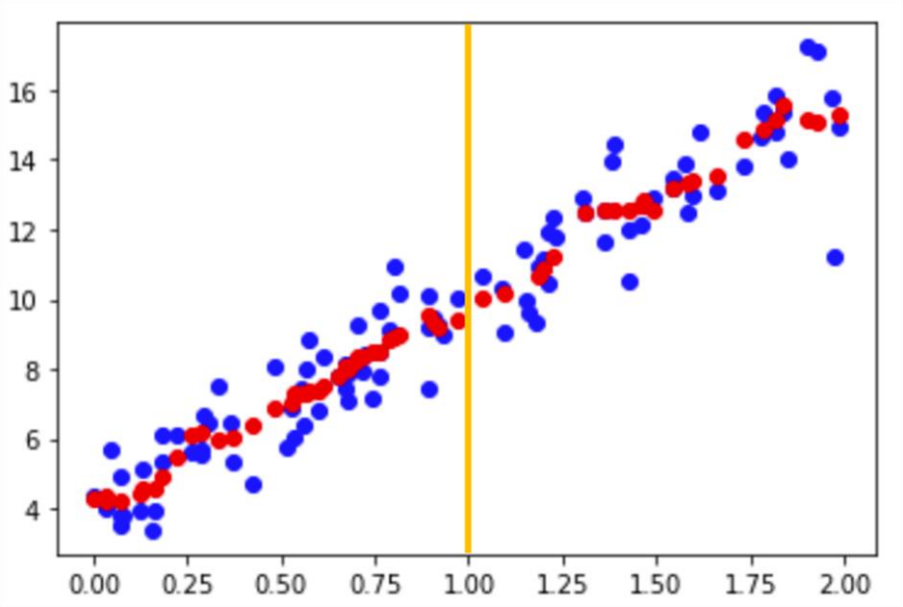
- 비슷한 위치의 데이터를 샘플로 잡은 후에 함수를 예측하면 불필요한 함수가 예측이 되므로, 예측하려는 함수의 차수에 따라 구역을 나눈다. 해당 코드는 다음과 같다.

```
zone_unit = (data.iloc[-1, 0] - data.iloc[0, 0]) / (degree+1)  # 구역 단위
zones = [data[data["X"] < (data.iloc[0, 0] + zone_unit)]]

ifdegree >= 2:
    for i in range(2, degree+1):
        zones.append(data[(data["X"] >= (data.iloc[0, 0] + zone_unit * (i-1))) & (data["X"]
< (data.iloc[0, 0] + zone_unit * i))])

zones.append(data[data["X"] >= (data.iloc[0, 0] + zone_unit * degree)])
```

구역을 나눈 후에 시각화를 하면 다음과 같다. (degree = 1)



3.1.2 함수 예측하기

- **함수 예측** n차 함수식을 예측하기 위해서는 n+1 개의 연립방정식이 필요하다. 그렇기 때문에 연립방정식을 해결하는 Functions 라는 자체 모듈을 만든 후에 epoch 만큼 함수들을 예측한다. 예를 들어 degree 가 1 인 일차함수 그래프를 epoch = 2 만큼 학습한다면, f(x) = ax + b 꼴의 함수 두개를 예측하는 것이다.

```
for i in range(epoch):
    functions = Functions()
    funcs = []

    for idx in range(len(zones)):
        funcs.append(zones[idx].sample(n=1).iloc[0, :].tolist())

    for x, y in funcs:
        functions.add_func((x, y))

    all_coefficients.append(functions.predict_func())
```

example

$$f(x) = ax + b$$

$$degree = 1$$
$$epoch = 2$$

X	Y
1	5.4
2	8.2
3	11.3
4	13.5
5	17.4

$$Epoch = 1$$
$$\begin{cases} a + b = 5.4 \\ 5a + b = 17.4 \end{cases}$$
$$a = 4$$
$$b = 1.4$$
$$f(x) = 4x + 1.4$$

$$Epoch = 2$$
$$\begin{cases} 2a + b = 8.2 \\ 4a + b = 13.5 \end{cases}$$
$$a = 2.65$$
$$b = 2.9$$
$$f(x) = 2.65x + 2.9$$

3.1.3 최종 모델 예측하기

- **최종 모델 예측**은 3.1.2에서 예측했던 함수들의 각각의 계수들의 평균을 구하여 예측해낸다. 예를 들면 f(x) = 4x + 1.4 와 f(x) = 2.65x +2.9를 이용하여 최종 모델을 예측한다면 x의 계수인 4 와 2.65의 평균인 3.325, 그리고 상수인 1.4 와 2.9의 평균인 2.15를 이용하여 f(x) = 3.325x + 2.15 라는 최종모델을 예측할 수 있다. 해당 코드는 다음과 같다.

```
for i in all_coefficients:
    for j in range(len(i)):
        self.coefficients[j].append(i[j])

for i in range(len(self.coefficients)):
    self.coefficients[i] = sum(self.coefficients[i]) / len(self.coefficients[i])
```

$$f(x) = ax + b$$

$$Epoch = 1$$

$$\begin{cases} a + b = 5.4 \\ 5a + b = 17.4 \end{cases}$$

$$a = 4$$

$$b = 1.4$$

$$f(x) = 4x + 1.4$$

$$Epoch = 2$$

$$\begin{cases} 2a + b = 8.2 \\ 4a + b = 13.5 \end{cases}$$

$$a = 2.65$$

$$b = 2.9$$

$$f(x) = 2.65x + 2.9$$

$$a = \{4, 2.65\}$$

$$b = \{1.4, 2.9\}$$

$$a_2 = \frac{\sum_{i=1}^{epoch} a_i}{epoch} = 3.325$$

$$b_2 = \frac{\sum_{i=1}^{epoch} b_i}{epoch} = 2.15$$

$$f(x) = 3.325x + 2.15$$

4. 결론

- 점계수평균법(PCM Linear Regression)은 보다 어렵지 않게 머신러닝의 회귀 분야에 대해 재미있게 공부할 수 있게 해주고, 많은 것들을 예측할 수 있는 머신러닝 알고리즘이다. 이 점계수평균법의 핵심 내용은 이름에서 알 수 있듯이, 데이터들을 샘플로 뽑아 여러 함수를 예측하고 그 함수들의 계수의 평균으로 최종적인 회귀를 예측해내는 것이라고 할 수 있다. 점계수평균법 알고리즘이 많이 알려져서, 보다 많은 사람들이 점계수평균법을 이용한 머신러닝을 개발하고, 인공지능에 관심을 갖고 기여를 했으면 하는 생각이고, 많은 사람들이 회귀문제의 새로운 접근법인 점계수평균법이라는 새로운 알고리즘에 관심을 가져 인공지능 분야가 더욱 성장했으면 하는 마음에 만들게 되었다.

5. 이용한 패키지

- numpy: Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 0.1038/s41586-020-2649-2. (Publisher link).

(https://www.nature.com/articles/s41586-020-2649-2)

- pandas: Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010) (publisher link)

(http://conference.scipy.org/proceedings/scipy2010/mckinney.html)

- matplotlib: John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007), DOI:10.1109/MCSE.2007.55 (publisher link)

(https://ieeexplore.ieee.org/document/4160265/)

6. 점계수평균법 오픈소스

- 현재 점계수평균법은 github에 MIT License 로 등록하여, 오픈소스로 공개되어 있는 상태이다. 아래 링크를 통해 연구개발이 완료된 점계수평균법을 확인할 수 있다. (https://github.com/SimplePro/PcmLinearRegression/)