Assignment4 - 머신러닝 이론과 실전

2021321148 조인식

1. MLE는 Likelihood function의 극대화를 통해 구할 수 있다. 이때, 단조변화는 극대점의 위치에 영향을 주지 않으므로, Log Likelihood Function의 극대화를 이용해도 된다.
따라서, 교재에 주어진 우도함수에 로그를 씌우면,

$$l(\beta_{0,}\beta_1) = \log L(\beta_0,\beta_1) = \sum_{i=1}^{n} y_i(\beta_0 + \beta_1 x_i) + \sum_{i=1}^{n} \left(1 + \exp(\beta_0 + \beta_1 x_i)\right)^{-1}$$

이를 모수 $\beta_{0,}\beta_1$에 대해 극대화 시 다음의 결과를 얻는다.

$[\beta_0]$

$$\sum y_i + \sum \exp(\beta_0 + \beta_1 x_i)(-1)(1 + \exp(\beta_0 + \beta_1 x_i))^{-2}(1 + \exp(\beta_0 + \beta_1 x_i)) = 0$$

$$\rightarrow \sum y_i = \sum \frac{\exp(\beta_0 + \beta_1 x_i)}{1 + \exp(\beta_0 + \beta_1 x_i)}$$

$$= \sum \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x_i))}$$

$[\beta_1]$

$$\sum x_i y_i + \sum x_i \exp(\beta_0 + \beta_1 x_i)(-1)(1 + \exp(\beta_0 + \beta_1 x_i))^{-2}(1 + \exp(\beta_0 + \beta_1 x_i)) = 0$$

$$\rightarrow \sum y_i = \sum \frac{x_i \exp(\beta_0 + \beta_1 x_i)}{1 + \exp(\beta_0 + \beta_1 x_i)}$$

$$= \sum \frac{x_i}{1 + \exp(-(\beta_0 + \beta_1 x_i))}$$

2.

시그모이드 함수를 $h_\theta = \dfrac{1}{1 + \exp(-\theta)}$라 할 시,

$$\frac{\partial h_\theta}{\partial \theta} = (-e^{\theta})(-1)(1 + \exp(-\theta))^{-2} = 1 = \frac{1}{1 + e^{-\theta}} \frac{e^{-\theta}}{1 + e^{-\theta}} = h_\theta(1 - h_\theta)$$

3.

$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_k \end{pmatrix}$라 하자. $\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_k x_k$라 할 시,

$$h_\theta = \frac{1}{1+\exp(\theta^T x)} = \frac{1}{1+\exp(\theta_0 + \theta_1 x_1 + ... + \theta_k x_k)}$$

위 식을 θ에 대해 미분하는 것은 $\theta_0, \theta_1, \cdots, \theta_k$에 대해 미분한 후 다시 정렬하는 것과 동일하므로, θ_i에 대해 미분 시,

이를 $J(\theta)$의 h_θ라 할 시,

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{n} \left[y^{(i)} \log(h_\theta(x^{(i)}) + (1-y^{(i)})(\log(1-h_\theta(x^{(i)})) \right]$$

θ_i에 대해 미분한다고 할 시,

$[\theta_i]$

$$-\frac{1}{m} \sum y^{(i)} x^{(i)} h_\theta^{-1} h_\theta (1-h_\theta) + \frac{1}{m} \sum (1-y^{(i)} x^{(i)}))(1-h_\theta)^{-1} h_\theta (1-h_\theta)$$

$$= \frac{1}{m} \sum x_i (h_\theta - y^{(i)})$$

4.

```
[31] #theta=회귀계수, alpha = 학습률, iteration = 반복수
def Gradient_descent( X ,  Y,  theta, alpha,iteration):
  def cost(X,Y,theta): #cost function(mse/2)
    m = len(Y)
    y = X @ theta
    logit = np.log(1/(1+np.exp(-y)))
    nlogit = np.log(np.exp(-y)/(1+np.exp(-y)))
    j = (-1) * (1 /m) * sum((Y*logit) +((1-Y)*nlogit))
    return j

  j=1000
  for c in range(iteration): # 반복
    y = X@theta
    for i in np.arange(X.shape[1]): #for문을 이용해 각각의 회귀계수에 대해 계산
      h = 1/ (1+np.exp(-y))
      theta[i] = theta[i] - alpha * ((sum((h-Y)*X[i])))
    j = cost(X,Y,theta)
    if c %10  ==0:
      print("진행상황: ",c)
      print("cost: ", j)
      print("계수: ", theta)

  print("최종계수: ", theta)

  return theta
```

```
[32] theta = np.zeros(shape = (X.shape[1]))
result = Gradient_descent(X,Y,theta,0.01, 1000)
```

```
[29] statlogit = sm.Logit(Y,X)
stat_fitted = statlogit.fit()
stats_results = stat_fitted.params
```

Gradient descent 방법과 statsmodels를 이용한 코드를 위와 같이 작성하여, 다음과 같은 결과를 얻었습니다.

```
result - Notepad
File Edit Format View Help

result check by gradient descent method
----------------
constant: 0.16829932254123778
beta1 : 2.8217765472736134
beta2 : 2.8192964496431374

result check by statsmodels
----------------
constant: 0.16829932254130062
beta1 : 2.821776547274214
beta2 : 2.819296449643787
```