Lab Seminar: 2024. 01.19

Supervised Machine Learning

(Multiple Linear Regression & Classification)

IDEALAB
Improving lives through learning

Su-Gyeong Ban
School of Computer Science
Gyeongsang National University (GNU)

GNU 경상국립대학교 Gyeongsang National University

Contents

Feedback (1/3)

- **Loss function**
  - 딥러닝 모델을 학습할 때 정답값과 예측값의 오차를 계산해주는 함수
  - 점수가 높을수록 모델이 안좋은 것
  - 손실함수의 **함수값이 최소화** 되도록 하는 **weight**와 **bias**를 찾는 것이 딥러닝 학습의 목표
    - 손실함수의 종류 : **MSE, RMSE** 등

Feedback (2/3)

- **MSE(Mean Squared Error)**
  - 예측한 값과 실제 값 사이의 평균 제곱 오차를 정의
  - 오차가 커질수록 제곱 연산으로 인해 값이 더 뚜렷해짐
  - 제곱으로 인해 오차가 양수이든 음수이든 누적 값을 증가시킴

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

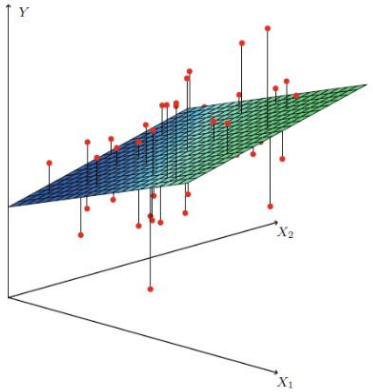
Feedback (3/3)

- **RMSE(Root Mean Squared Error)**
  - MSE에 루트를 씌운 것으로 MSE와 기본적인 것은 동일
  - MSE 값은 오류의 제곱을 구하기 때문에 실제 오류 평균보다 더 커지는 특성이 있음
  - MSE에 루트를 씌움으로써 **왜곡**을 줄여줌

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Multiple Linear Regression

- **What is Multiple Linear Regression?**
  - 독립변수 X의 변화에 따른 종속변수 y의 변화를 선으로 예측하는 기법 중 **독립변수 X가 여러 개인 분석 기법**
  - 구현 시, **벡터화** 사용
    - 도트 그래픽 표기법을 사용하면 더 적은 수의 문자를 사용하여 더 간결한 형태로 모델 작성



Vectorization (1/2)

- **What is Vector?**
  - 한 가지 타입의 여러 개의 원소를 변수에 저장한 배열
- **What is Vectorization?**
  - 벡터의 같은 인덱스에 위치한 원소들끼리 연산을 수행하는 기능
    - 학습 알고리즘 구현 시, 코드가 짧아지고 실행 효율성이 좋아짐

$$\vec{v} \times \vec{w} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{vmatrix} v_1 & w_2 \\ v_2 & w_2 \end{vmatrix} = v_1 w_2 - w_2 v_1$$

Vectorization (2/2)

- **Code of Vectorization?**
  - 벡터화 없이 계산하는 경우, 코드와 시간 모두 비효율적
  - For문을 활용한 경우, 벡터화 없이 계산하는 것 보단 낫지만 여전히 비효율적
  - **Numpy dot함수** 사용 시, n이 클 수록 훨씬 효율적
    - 대규모 데이터 셋에서 알고리즘을 실행할 때 사용

Parameters and features
b is a number

$$\vec{w} = [w_1\ w_2\ w_3]$$
$$\vec{x} = [x_1\ x_2\ x_3]$$

```
b = 4
w = np.array([1.0, 2.5, -3.3])
x = np.array([10, 20, 30])
```

```
# 벡터화 없이 계산하는 경우
f = w[0] * x[0] + w[1] * x[1] + w[2] * x[2] + b

# for문을 사용하여 계산하는 경우
f = 0
for j in range(0, 3):
    f = f + w[j] * x[j]
f = f + b

# dot함수를 사용하여 계산하는 경우
f = np.dot(w, x) + b
```

Gradient descent for MLR (1/5)

- **Normal equation**
  - 고급 선형대수 라이브러리를 사용하여 **w**와 **b**를 반복 없이 하나의 목표로 풀 수 있는 것
    - 학습률을 정할 필요 없음
    - 입력 변수의 개수가 커지면 **커질수록 비효율적**
    - 역행렬이 존재하지 않을 수 있음
- **Gradient descent**
  - 최적의 가중치와 편향 값을 경사하강법을 통해 탐색
    - 적합한 학습률을 찾거나 정해야 함
    - 입력 변수의 개수가 커도 **효율적인 연산 가능**

Gradient descent for MLR (2/5)

- **Feature Scaling (1/3)**
  - 경사하강법이 더 적은 반복으로 전역 최소값에 도달하기 위해 피처 값의 범위를 조정하는 것
    - 기능 스케일링에 **2**가지 방법이 있음
      - **Min-Max Normalization**
      - **Z-score Normalization**

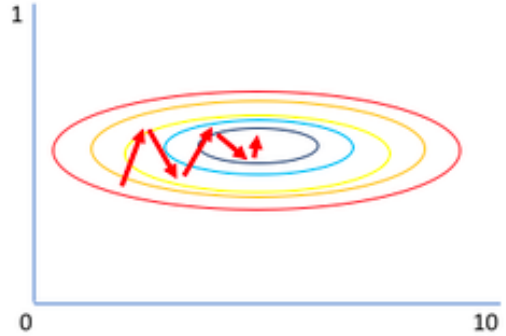
Gradient descent for MLR (3/5)

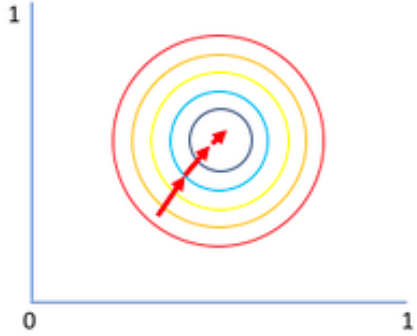
- **Feature Scaling (2/3)**
  - **Min-Max Normalization**
    - 원래의 범위를 범위의 최댓값으로 나눔
      - 장점 : 구현이 간단하고 연산이 빠름
      - 단점 : 이상치에 영향을 크게 받음

x_1 = size (0-2000 feet^2)
x_2 = number of bedrooms (1-5)

→

x_1 = size (feet^2) / 2000
x_2 = number of bedrooms / 5





Gradient descent for MLR (4/5)

- **Feature Scaling (3/3)**
  - **Z-score Normalization**
    - 값들의 평균을 0, 표준편차를 1이 되도록 스케일링 하는 것
      - 비교적 이상치의 영향을 적게 받음

$$Feature\ Scaling : \frac{x_i - \mu_i}{maximum\ value}$$

μ_i : 피처 x_i 의 값들의 평균

$maximum\ value$: 피처 x_i 의 최대값 또는 '최대값 – 최소값'

$$x_1 = \frac{size - 1000}{2000}$$

$$x_2 = \frac{\#bedrooms - 2}{5}$$

$$-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5$$

Gradient descent for MLR (5/5)

- **Polynomial regression**
  - 문제에 따라 기존의 피처를 사용하는 것보다 새로운 피처를 만드는 것이 좋음
  - 학습 데이터 셋의 모양에 따라 **다양한 그래프**가 나타남
  - 피처를 다양하게 변형할 때는 **스케일링**이 매우 중요함
    - 여러 피처가 비슷한 값의 범위를 가져야 비용함수의 최소값을 구하는 경사하강법이 제대로 작동하기 때문

Classification

- **What is Classification?**
  - 머신러닝의 지도학습에 대표적인 유형 중 하나
  - 주어진 데이터를 클래스 별로 구별해 내는 과정
  - 분류 알고리즘을 통해 데이터와 레이블 값을 학습시키고 모델 생성
  - 데이터가 주어졌을 때 학습된 모델을 통해 어느 범주에 속한 데이터인지 판단
    - 대표적인 분류 알고리즘 : **Logistic Regression, Naïve Bayes** 등

Logistic Regression (1/5)

- **What is Logistic Regression?**
  - **0이나 1인 이산적 값**을 가지는 분류 알고리즘
  - 독립변수와 종속변수의 선형 관계성을 기반을 만들어짐
  - 참, 거짓과 같이 **이분법**으로 분류하는 알고리즘
    - 0으로 나타내는 클래스를 **Negative class** , 1로 나타내는 클래스를 **Positive class**
      - 로지스틱 함수 또는 시그모이드 함수
      - 예측값에서 y=0일 확률과 y=1일 확률의 합은 항상 1

$$h_\theta(x) = g(\theta^T x) = g(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-\theta^T x}} \quad (z = \theta^T x)$$

$$P(y=0 \mid x;\theta) + P(y=1 \mid x;\theta) = 1$$
$$P(y=0 \mid x;\theta) = 1 - P(y=1 \mid x;\theta)$$

Logistic Regression (2/5)

- **What is Decision Boundary?**
  - y=0의 영역과 y=1의 영역을 나누는 경계
    - h(x) = 0.5 인 영역
  - 학습 데이터셋의 속성이 아니라 가설 파라미터의 속성
    - 학습 데이터셋은 **결정 경계**를 정의하는 가설의 파라미터 θ 값을 찾기 위해 사용

$$h_\theta(x) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2) = g(\theta^T x) = g(z)$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

$$z = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2$$

Logistic Regression (3/5)

- **Cost function from Logistic Regression (1/2)**
  - 로지스틱 회귀의 비용함수는 선형회귀 비용함수에서 유도

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

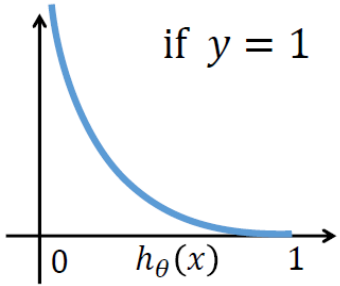
$$Cost(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

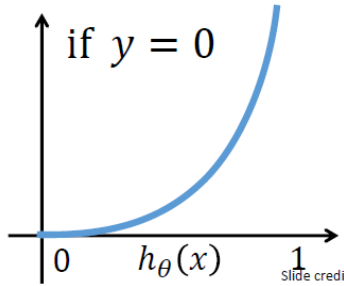
  - 알고리즘이 실제 레이블 y 를 가진 학습 예제에 대해 출력값 $h_\theta(x)$ 예측하기 위해 지불해야 할 비용

Logistic Regression (4/5)

- **Cost function for Logistic Regression (2/2)**
  - 로지스틱 회귀의 비용함수 특징
    - y=1일 때 h(x)=1, y=0일 때 h(x)=0
    - 만일, y=0일 때 h(x)=1을 예측하면 비용은 **무한대**로 증가

$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & if\ y = 1 \\ -\log(1 - h_\theta(x)) & if\ y = 0 \end{cases}$$





Slide credit: And

Logistic Regression (5/5)

- **Gradient descent for Logistic Regression**
  - 경사하강법이 적절하게 수렴하는지 관측하는 방법은 선형회귀와 동일
  - 로지스틱 회귀에서는 피처 값의 범위가 너무 다르다면 피처 스케일링 적용
    - **Gradient descent implementation**

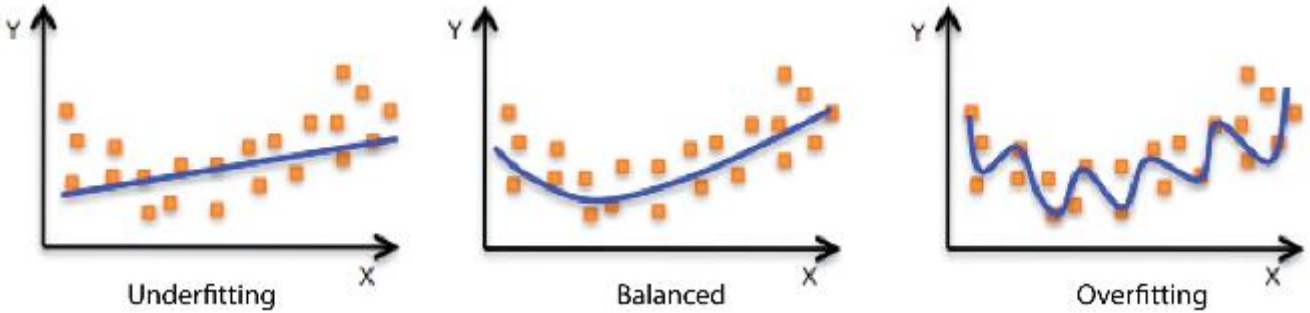
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad (\text{동시업데이트 } j = 0, 1, \dots, n)$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

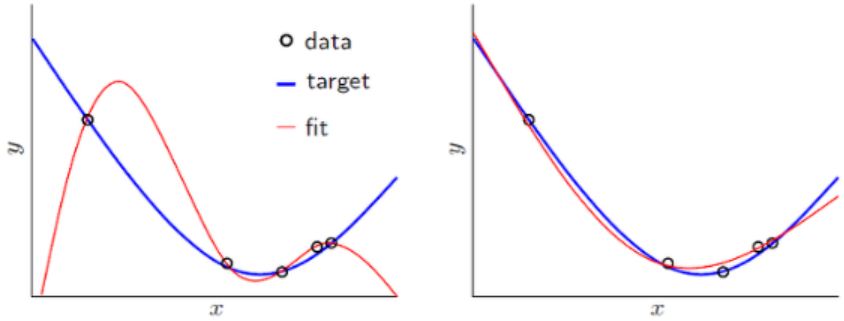
Regularization (1/5)

- **The problem of overfitting**
  - Underfit(과소 적합) : 함수가 학습 데이터 셋에 적합하지 않은 것
  - Balanced : 함수가 데이터에 잘 맞는 것
  - **Overfit**(과대 적합) : 데이터에 완전히 적합한 불규칙 곡선



Regularization (2/5)

- **What is Regularization?**
  - **모델의 복잡도를 낮추기 위한 것**
  - 과 적합 문제를 개선하여 학습 알고리즘의 성능을 향상함
    - 비용함수를 작아지는 쪽으로 학습하면 특정 가중치 값들이 커지면서 결과를 나쁘게 만들기 때문에 비용함수를 바꿈



(a) without regularization (b) with regularization

Regularization (3/5)

- **Cost function with Regularization**
  - 기존 비용함수에 정규화 항을 추가하여 모든 파라미터에 페널티 부여
  - 정규화 파라미터 λ는 두 가지 목표 사이에서 트레이드오프 제어
    - **목표 1**. 첫 번째 항으로 학습 데이터 셋에 적합한 파라미터 θ를 찾아 비용함수 최소화
    - **목표 2**. 두 번째 항으로 파라미터 θ를 작은 값으로 유지

비용함수 :

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

정규화를 적용한 비용함수 :

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + \lambda \sum_{i=1}^{n} \theta_j^2 \right]$$

Regularization (4/5)

- **Regularized Linear Regression**
  - 정형화된 선형 회귀는 θ_0과 $\theta_1\theta_2\theta_3, \ldots \theta_n$ 로 분리 후 미분항의 비용함수를 미분
    - θ_0 은 0으로 페널티를 부과하지 않기 때문

선형 회귀 경사 하강법 :

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}\right] - \alpha \frac{\lambda}{m} \theta_j$$

$$\theta_j := \theta_j \left(1 - \alpha \frac{\lambda}{m}\right) - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}$$

$$\left(\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}\right)$$

정규방정식 :

역행렬 X

$$\theta = (X^T X)^{-1} X^T y = pinv(X' * X) * X' * y$$

역행렬O

$$\theta = \left(X^T X + \lambda \cdot L\right)^{-1} X^T y$$

$$L = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix}$$

Regularization (5/5)

- **Regularized Logistic Regression**
  - 일반적으로 로지스틱 회귀에 너무 많은 피처가 있다면 다항식이 아니더라도 과적합일 확률이 높음
  - 정규화하는 방법은 선형회귀와 같음

정규화된 로지스틱 회귀 경사 하강법 :

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)} - \frac{\lambda}{m} \theta_j\right]$$

경상국립대학교

Gyeongsang National University

Improving lives through learning

IDEALAB