dplyr

함수들

1. select(df, col) = 원하는 col 뽑아오기
2. mutate(df, new col = operation(original col)) = 기존 col의 연산으로 새로운 col추가
3. filter(df, logic) = 원하는 row 뽑아오기
4. arrange(df, col, col, ...) = 첫 col의 값들을 오름차순으로 정렬(그 뒤 col부터는 값 동일시 우선순위)
5. summarise(df, new col = statistics(original col)) = 요약df 만들기

   *(statistics=sum, mean, n() - row갯수, n_distinct(col) - col의 종류 수)

   *group_by(df, col) = 해당 col을 범주별로 묶음
6. diff(range(vector)) = vector의 원소 값의 max-min값

추가 확인 사항

1. a %in% b

: b에 a가 있는지 없는지 T/F

2. NA를 제거해야할 필요가 있을 때
    1. !is.na(col)
    2. na.rm=T
3. pipe operation : df %>% function(not df,) %>% function(not df,) %>% ...

Data Integration

left_join : 왼쪽거만 예뻐해서 왼쪽거에 그냥 정보 추가하는것. (by로 설정한 key col말고도 col들 다 가지고 옴.)

right_join : 오른쪽거만 예뻐해서 오른쪽 거에 그냥 정보 추가하는 것. (위와 동일)

inner_join : by에 설정한 col 기준으로 왼쪽 오른쪽 모두 있어야 살아남음. (위와 동일)

full_join : by에 설정한 col 기준으로 어느 한쪽이라도 있으면 살아남음. (위와 동일)

semi_join : 교집합만 남기는데 왼쪽거의 col들로 구성함.

anti_join : 교집합의 여집합만 남기는데 왼쪽거의 col들로 구성함.

Linear Regression

: supervised learning + regression problem => 직선으로 데이터를 나타낼 수 있는 것.

우리가 세워야 할 것은 직선관계식, 즉 hypothesis이다.

input -> system(hypothesis) -> output

내가 어떤 input을 넣었을 때 나오는 output이 실제 값과 같을 수 있도록 최적의 세타(feature에 대한 weight) 값을 찾아야 할 것이다.

그러면 가장 직관적인 방법으로는 나오는 output과 실제 값의 오차가 적을수록 최적의 세타가 아닐까.

이 때 등장한 것이 Cost Function, 오차값 즉 cost값이 가장 작은 지점에서의 세타 값을 찾는다.

(여기서 cost 값은 MSE=mean(error^2) 이다.)

Cost function

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

우리가 찾아야 할 것은 ' min J(세타) ' 이므로 Gradient Descent, 즉 미분을 한다. 전략은 다음과 같다.

1. 찾아야 할 최적의 세타 세트를 처음에 0으로 초기화한다.
2. 세타 세트들을 바꿔가며 가장 낮은 비용의 지점에서 세타변화를 멈춘다.

   *이 때 세타 값들은 동시에 update해야한다. 순차적으로 바뀐다면 먼저 바뀐 세타0에 의해 h가 바뀌고 이 바뀐 h를 가지고 세타1을 구하는 꼴이니 말이다.

Gradient Descent algorithm

$$\text{repeat until convergence } \{$$

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

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

update θ_0 and θ_1 simultaneously

$$\}$$

m = size of training set, x(i), y(i) = data of training set // 여기서 x위에 있는 i는 i번째의 feature을 말한다.

그러면 세타(feature에 대한 weight)가 0과 1만 있는 것이 아니라 다른 feature들이 더 있는 경우는 h를 다음과 같이 표현한다.

$$h_\theta(x) = [\,\theta_0 \quad \theta_1 \quad \dots \quad \theta_n\,] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

그래서 변수의 갯수에 상관없이 Cost function과 Gradient descent를 정의하면 다음과 같다.

Cost function:

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Gradient descent:

Repeat $\{$

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

(simultaneously update θ_j for $j = 0, \dots, n$)

$\}$

LINEAR REGRESSION

COST : MSE

GD : 기존 - 알파 * ME * 해당

이제 조금 더 세세한 부분까지 커버를 해보자. 그 첫번째는 feature scaling이다. 이는 변수들의 range를 대략적으로 맞춰주는 것인데, 이는 gradient descent할 때 효과를 볼 수 있다.

(이 때, 추가해준 x0파트(모두 1로 채워진 부분)는 할 필요 없음.)

Feature Scaling

Mean normalization

: 대략적으로 평균을 0이 되게! 값을 ' (값 - 평균) / sd나 range ' 로 바꾼다.

*sd는 standard deviation, 표준편차로 산포도(흩어진 정도)를 나타낸다. 이 값이 낮을 수록 평균 값에 가까이 있다.

그리고 J(세타) 값은 iteration을 돌릴수록 작아져야 정상이다. 그런 그래프가 그려지지 않고 있다면 (앞 전의 x축이 세타였던 것과 헷갈리면 안됨. 지금은 x축이 iteration횟수임.) 알파 값을 3배 혹은 1/3배 해서 조정한다. 이 때, 알아둘 것은 알파 값을 줄이면 iteration이 증가할수록 반드시 J(세타) 값이 감소한다.(물론 너무 작으면 수렴하는데 오래 걸림) 이 점 참고하기 바란다.

왜 스케일링인가?

대부분의 경우 데이터 세트에는 크기, 단위 및 범위가 매우 다양한 기능이 포함됩니다. 그러나 대부분의 기계 학습 알고리즘은 연산에서 두 데이터 점 사이의 Eucledian 거리를 사용하므로 문제가됩니다.

혼자 남겨 두었다면,이 알고리즘은 단위를 무시한 피처의 크기를 취합니다. 결과는 5kg와 5000gms 단위에 따라 크게 다를 것입니다. 높은 크기의 기능은 낮은 크기의 기능보다 거리 계산에 훨씬 더 중요합니다. 이 효과를 억제하려면 모든 기능을 동일한 수준으로 가져와야합니다. 이것은 스케일링에 의해 달성 될 수 있습니다.

또 새로운 feature를 이용하는 방법이 있다.

Polynomial Regression

: 집 값을 결정하는데 feature가 마당의 가로길이와 세로길이, 이렇게 두개가 있다고 가정했을 때, 마당의 넓이 역시 유효한 feature가 될 수 있으므로 이럴때 새로운 feature를 추가하는 것이다.

하지만 2가지를 !! 주의 !! 해야하는데 추가한 feature의 특성에 맞는 함수를 생각해야 한다는 것이다. 가령 2차 함수는 최댓값을 찍으면 감소하는 그래프인데, 마당의 면적이 어느 면적 이상되면 집값이 떨어지는 것은 일반적으로 옳지 않다. 그리고 또 주의해야할 점은 scaling부분이다. 제곱 혹은 세제곱 된 feature을 추가하면 기존의 변수의 range에 제곱 혹은 세제곱 만큼 scaling된다.

Logistic Regression

: 이 친구는 classification을 위한 모델로서 Hypothesis의 값이 0과 1사이의 값을 가진다.

Logistic regression을 할 때는 sigmoid함수를 쓰는데 그 이유는 0과 1사이의 값을 가지고 미분이 가능하기 때문이다.

$$h_\theta(x) = P\left(y = 1 | x; \theta\right)$$

- x: feature
- θ: model parameter

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

Logistic regression

$$h_\theta(x) = g(\theta^T x)$$

시그모이드의 특성상 세타x가 0보다 크면 exp(-세타x)는 0으로 가게 되어 h(x)가 1이되고

세타x가 0보다 작으면 exp(-세타x) h(x)가 0이다.

(이거 이해 잘 안 되는 사람 많을거 같은데 계속해서!)

이 때, 변화하는 세타 값에 따라 세타(t) * x 를 0으로 만드는 지점을 기준으로 위와 아래 혹은 안과 밖으로 classification된다.

자! 시그모이드 함수를 잘 생각하고 0.5가 threshold이다. 그러니 z가 0보다 크면 1이고 z가 0보다 작으면 0인 것이다.

@ 그렇다면 threshold가 0.5인것 밖에 못 하는가.

=> 세타(t)x

안과 밖의 경우로 나뉘는 경우는 polynomial하게 feature demension을 높인 경우이다.

어찌 되었건간에 이제 cost function을 정의해서 적절한 세타 값을 찾고 모델을 세워야 할텐데, 잘 생각해보라.

cost(h(x), y)를 1/2 * error^2로 생각한다면

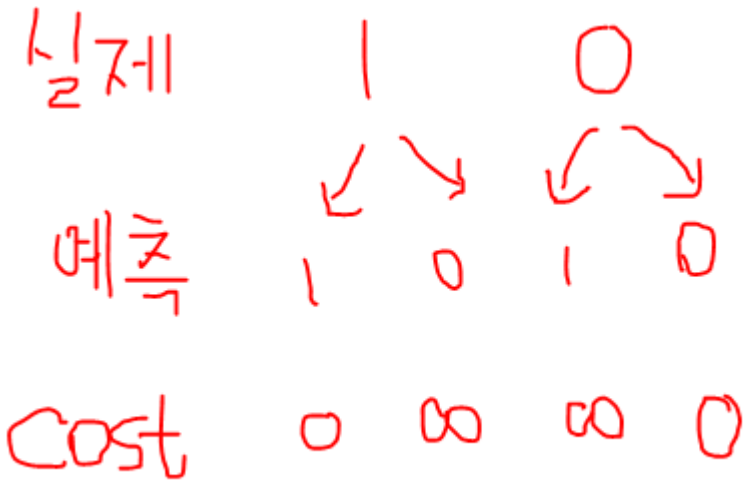
@ linear regression으로 cost function구하면 왜 non-convex인가. : 그건 직관으로..

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

$y = 1$일 때 cost function 의 특징은,

- $\text{cost} = 0$ when $h_\theta(x) = 1$
- $\text{cost} \to \infty$ as $h_\theta(x) \to 0$

자! 그냥 쉽게 생각하자.



더이상 쉽게 설명할 수는 없다.

최종적으로, logistic regression의 cost function은 다음과 같이 표현한다.

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

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

이 수식만 보면 linear regression의 gradient descent와 동일하다. 유일한 차이점은 hypothesis function이 $h_\theta(x) = \frac{1}{1+\exp(-\theta^T x)}$ 로 바뀐 것이다.

Want $\min_\theta J(\theta)$:

Repeat $\{$

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

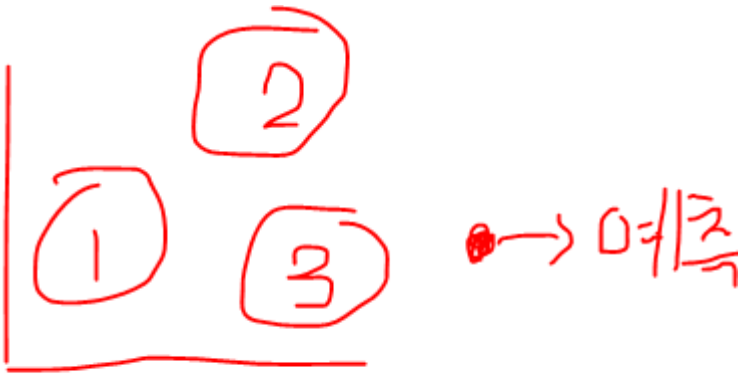
(simultaneously update all θ_j)

$\}$

Multi-class classification

one-vs-rest란?

:



이 때 1일 확률 : __ , 2일 확률 : __ ,3일 확률 : __ 중에 가장 높은것으로 판단하는 것이다.

각 class 별로 해당 class vs 나머지 class로 binary decision을 내리도록 만들고 로지스틱 리그레션 hypothesis function 값이 가장 큰 것을 고르는 것이다.

Regularization

: 많은 feature들이 결과값을 예측하는데 도움을 줄 때, 세타 값을 작게 해주어 모두 사용할 수 있도록 하는 것. 결과적으로 overfitting의 문제를 해결한다.

배경]

Underfitting : 현 모델이 너무 단순해서 데이터를 적합하게 구분하지 못하는 것

Overfitting : 현 모델이 학습 데이터 셋에 너무 지나치게 맞춰져서 오차가 거의 없지만 새로운 데이터는 맞추짖 못하는 것. (일반화가 되지 않는 모델이다.)

2번째의 overfitting의 문제를 해결하기 위해서는 feature수를 줄이거나 세타 값들의 크기를 줄여 약간의 기여도를 얻어가는 방법이 있는데 후자의 경우가 regularization이다. 실제로 linear regression을 가지고 살펴보자. 자! 지금 하는 것은 세타의 값을 작게 만드는 것이다!

차근차근 linear regression의 경우부터 regularization해보자. Cost function을 보라.

Regularized linear regression

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

$$\min_\theta J(\theta)$$

원래의 linear regression의 J(세타)에서 ' 람다 시그마 ' 부분이 추가된 것을 볼 수 있다. 세타를 줄여야 하는데 오히려 +무언가를 해주고 있지 않은가. ?????????????? 정말 중요한 부분이다.

세타 changing은 cost가 줄어드는 방향으로 이루어진다. 그러면 세타의 크게 영향을 미치는 경우가 있을 때, cost만 크게 해준다면 그 방향으로 가지 않을 것이다. 가령, 세타가 cost가 어떤 세타 벡터의 조합으로 발생했을 때 그 순간의 penalty를 줌으로써 세타의 기여도를 줄이게 된다.

또 다르게 생각하면 ' J(세타) + 람다 ' 한 거에다가 미분을해서 기존 세타에 빼주는 것이다. 결국, 람다를 더하여 줌으로써 업데이트 되는 세타의 크기를 조절할 수 있는 것이다. 어느 쪽으로 이해하든 상관없다.

계속해서 Gradient Descent는 다음과 같다.

Gradient descent

Repeat $\{$

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

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

$$(j = 1, 2, 3, \ldots, n)$$

$\}$

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

이 시점에서 non-linear optimizer, R에서 제공하는 ucminf 패키지를 사용해보자. 이건 cost함수와 gradient함수를 지정하면 최적의 세타 값을 반환해준다.

ucminf(

par = 최초의 파라미터 값

fn = (우리가 찾아야 할 세타에 해당하는 par과 함께) cost함수 제공

gr = gradient 역시 세타와 함께 제공

)

logistic의 regularization

Cost Function

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

Repeat {

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

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

$j = 1, 2, \ldots, n$ ($\frac{\partial}{\partial \theta_j} J(\theta)$)

}

Note | $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

궁금 해결선]

1. 등고선

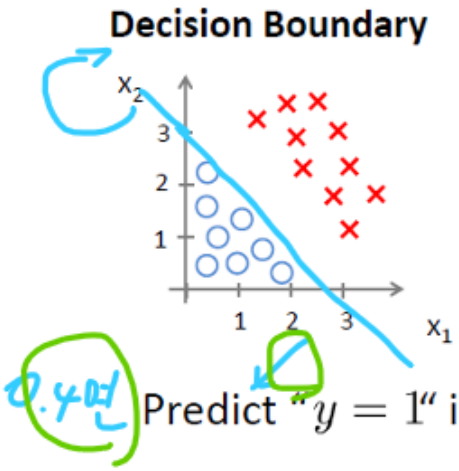
: 원이 되도록 해주는 것이 핵심이다. 그러면 gradient descent 방향이 일직선 중앙으로 잘 찾아간다. 타원이면 이리 저리해서 찾아간다. 등고선 전체의 크기는 scaling의 차이이다.

세로로 얇은 등고선은 x축은 100단위로 커지고, y축은 1단위로 커지는 것이다.

2. logistic regression의 threshold

: 세타(t)*x를 @라 두자.

h(x) = g(@) 인데, 시그모이드 함수를 보면 @의 값이 0보다 크면 1이고, 0보다 작으면 0이다. 그런데 이 이유는 @가 0인 지점일 때 h(x)값이 0.5지점이기 때문이다. 그런데 @의 값이 0보다 큰 수가 되는 지점을 잡게 되면 threshold를 0.5보다 큰 지점으로 잡을 수 있게된다.



$$g(z) = \frac{1}{1+e^{-z}}$$

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

0 → 0.5

>0 → >0.5

<0 → <0.5

0.4면 Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

이 그래프보다 밑으로 가야할테니.

차근차근 생각해보면 0.4가 되려면 @는 0보다 작은 값이여야한다. 그러면 -3 + x1 + x2 >=-1을 해보아라. 그래프가 떨어지는 것이 보일 것이다.

3. logistic은 cost함수를 그리면

: non-convex로 그려지는데 convex가 되게 하기 위해(local optimization에 빠지지 않기 위해) log함수를 사용한다.

4. 세타별로 기여도를 조절할 수 있는가

:

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

결국 빼질 때, 기존 - 기존*람다 이므로 세타별로 자연스럽게 기여도를 맞출수 있는 것이다

5. 알파 값을 조정하는 것은 어떤 의미인가.

: learning rate이 너무 클 때의 그림으로, overshooting 현상이 발생하는 것을 보여준다. 보폭이 너무 크면 최저점을 지나 반대편 경사면으로 이동할 수 있게 되고 심할 경우 경사면을 타고 내려가는 것이 아니라 올라가는 현상까지 발생할 수 있다. overshooting은 경사면을 타고 올라가다가 결국에는 밥그릇을 탈출하는 현상을 말한다.

오른쪽 그림은 너무 작을 때를 보여준다. 한참을 내려갔음에도 불구하고 최저점까지는 아직 멀었다. 일반적으로는 조금씩 이동하는 것이 단점이 되지 않는데, 머신러닝처럼 몇만 혹은 몇십만 번을 반복해야 하는 상황에서는 치명적인 단점이 된다. 최저점까지 갈려면 한 달 이상 걸릴 수도 있으니까.

learning rate을 너무 크지 않게, 그러면서도 작지 않게 조절하는 것은 무척 어렵고, 많은 경험이 필요한 영역이다. 그림에서는 overshooting이 더 안 좋은 것처럼 설명되지만, 실제로는 overshooting이 발생하면 바로 알아챌 수 있기 때문에 문제가 되지 않는다. 오히려 learning rate이 작은 경우에 늦게 알아챌 수가 있다. 마치 정상적으로 내려가는 것처럼 보이니까. 두 가지 모두에 있어 여러 번의 경험을 통해 적절한지 혹은 적절하지 않은지에 대한 안목을 키우는 것이 최선이다.

6. 람다를 조정하는 것은 어떤 의미인가.

: 람다 값이 너무 높으면 모델은 단순해지지만 데이터가 과소적합해질 위험이 있습니다. 그렇게 되면 모델은 유용한 예측을 수행할 만큼 학습 데이터에 대해 충분히 학습하지 못할 수 있습니다.

람다 값이 너무 낮으면 모델은 더 복잡해지고 데이터가 과적합해질 위험이 있습니다. 모델이 학습 데이터의 특수성을 너무 많이 학습하게 되고 새로운 데이터로 일반화하지 못하게 됩니다.