3주차 - 로지스틱 회귀

- 선형회귀로 분류문제를 풀 수 없는 이유.

y in {0, 1}인 분류문제에 대하여 선형회귀는 h(x)의 범위가 넓고 이상치 데이터로 인하여 잘못 분류할 확률이 크다. 반면 0 <= h(x) <= 1의 범위를 가지는 로지스틱 회귀는 그런면에서 분류문제에 적합하다.

- 로지스틱 회귀의 의미

로지스틱 회귀는 따라서 hypothesis를 x와 theta에 대한 선형함수로 만들지 않고 시그모이드 함수를 이용한다. (로지스틱 = 시그모이드) 그리고 이 hypothesis의 의미는 P(y = 1 | x ; theta)이다.

- decision boundary

한편 h(x)가 0.5 확률을 가지는 것을 기준으로 y = 1 과 y = 0을 나눌 수 있는데 이 나누는 경계선을 decision boundary라고 한다.
decision boundary => h(x) = 0.5 => theta * X = 0 (그래프를 그려보면 자명히 알 수 있다.)
따라서 특징값이 다항식으로 구성되어있으면 다양한 모양으로 boundary를 설정할 수 있다.

- 로지스틱 회귀에서 cost function

선형회귀에서 사용하는 cost function은 h(x) = theta * x하에서 볼록함수였다. 즉, local optimum으로 global optimum을 찾을 수 있다는 말이다. 하지만 그 cost function을 로지스틱 회귀의 h(x)을 적용하면 h(x)가 선형식이 아니고 비선형식이기 때문에 non-convex(비볼록함수)가 되고 경사하강법으로 global optimum을 보장받을 수 없다. 따라서 로지스틱 회귀에서는 아래와 같이 다른 비용함수를 사용한다.
y = 1 -> J(theta) = -log(h(x)), y = 0 -> J(theta) = -log(1-h(x)).
그래프를 그려보면 위 비용함수가 어떤 의미인지 직관적으로 알 수 있다.
결과에 대해서 다른 판단(y =1 인데 h(x) = 0인 경우 혹은 반대상황)을 할 경우에는 비용이 크게 증가하고 반대로 결과와 같은 판단을 하게되면 비용이 0이되는 모양이다. 이 비용함수는 볼록함수이며 그 이유도 그래프를 보면 직관적으로 알 수 있다.

Simplified Cost Function and Gradient Descent

위 붉은 글씨의 cost function을 y가 0과 1의 값만 가진다는 것을 이용하여 보다 간단하게 만들 수 있다.

$$\mathrm{Cost}(h_\theta(x), y) = -y\ \log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$$

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

그리고 벡터화 시키면… (벡터화는 효율적인 연산에 도움이 된다)

$$h = g(X\theta)$$
$$J(\theta) = \frac{1}{m} \cdot \left(-y^T \log(h) - (1-y)^T \log(1-h)\right)$$

gradient descent는 linear regression과 형식이 동일하며 h(x)만 다르다.

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

증명 : 로지스틱 회귀(증명 : diff cost function)
gradient descent도 벡터화 시키면...

$$\theta := \theta - \frac{\alpha}{m} X^T(g(X\theta) - \vec{y})$$

- Advanced Optimization

경사하강법(gradient descent)의 목적은 J(theta)를 최소화시키는 theta를 구하는 것이다.
그런데 경사하강법은 learning rate를 설정해 주어야하고 앞으로 설명할 다른 알고리즘 보다 상대적으로 느리다는 단점이 있다.
Conjugate gradient, BFGS, L-BFGS << 이 세가지 알고리즘은 다소 복잡하지만 learning rate를 자동으로 설정해주고 경사하강법보다 빠른 장점을 가지고 있다.
하지만 말그대로 정말 복잡한 개념이기 때문에, 이해하기는 힘들고 그냥 그 결과를 라이브러리 형태로 사용하는 것이 좋다.(물론 이해한다면 더 좋겠지만..)

octave에서는 위의 복잡한 알고리즘을 제공한다. 따라서 우리 입장에서는 J(theta)와 그의 미분계수만 구하면 되는데 이는 다음과 같은 비용함수의 양식대로 만든다.

```
function [jVal, gradient] = costFunction(theta)
  jVal = [...code to compute J(theta)...];
  gradient = [...code to compute derivative of J(theta)...];
end
```

그 뒤에 아래와 같이 octave에서 제공하는 라이브러리를 사용하여 학습시키면 된다.

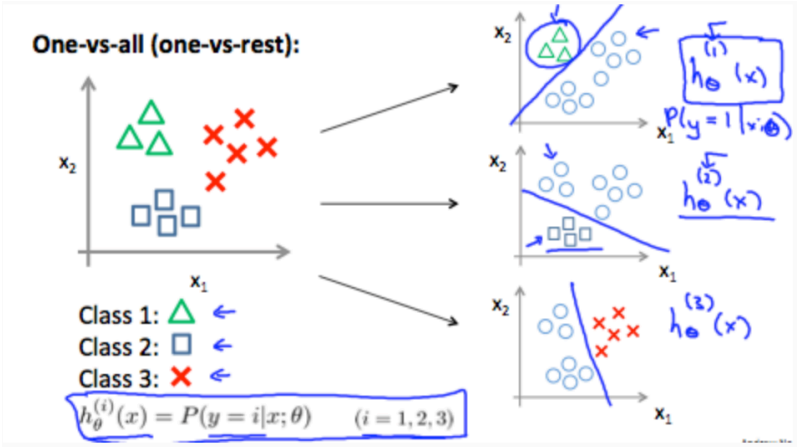
```
options = optimset('GradObj', 'on', 'MaxIter', 100);
initialTheta = zeros(2,1);
   [optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta,
options);
```

GradObj(= gradient objective parameter)이 ‘on’는
“our function returns both the cost and the gradient. This allows fminunc to use the gradient when minimizing the function.”라는 의미이고.

‘MaxIter’, ‘100’은 iteration이 100이라는 의미이다.
fminunc가 octave에서 제공하는 최소 theta를 구하는 함수이며, costFunction앞에 @는 포인터를 의미한다.

- Multiclass Classification : One-vs-all

다범주 분류의 경우 y(target variable)의 범위가 {0, 1}이 아닌 {0,1,2… k}임을 말한다.
이 경우 각각의 i에 대해서(i = 0, 1, 2, …, k) P(y=i | x; theta)을 구하고 그 중에서 확률 값이 가장 큰 i 로 분류한다. 이 방법을 one-vs-all이라고 한다.
즉 hypotheses를 i개의 벡터로 만든 뒤 그 중의 최대값으로 y를 예측하는 방법이라고 볼 수 있다.
벡터의 각 요소를 구하는 것은 y = i일 경우와 아닌 경우를 분류하는 것을 의미한다.



$$y \in \{0, 1 \ldots n\}$$
$$h_\theta^{(0)}(x) = P(y = 0|x;\theta)$$
$$h_\theta^{(1)}(x) = P(y = 1|x;\theta)$$
$$\cdots$$
$$h_\theta^{(n)}(x) = P(y = n|x;\theta)$$
$$\text{prediction} = \max_i(h_\theta^{(i)}(x))$$