Logistic Regression and Regularisation

Class	Machine Learning
Created	@May 22, 2020 6:41 PM
Materials	Lecture6.pdf Lecture7_reg.pdf
Reviewed	☐
Type	Lecture Notes

What is Logistic Regression ?

Logistic regression is a method for classifying data into discrete outcomes. For example, we might use logistic regression to classify an email as spam or not spam. In this, we will cover the cost function for logistic regression and the application of logistic regression which is multi-class classification.

▼ CLASSIFICATION

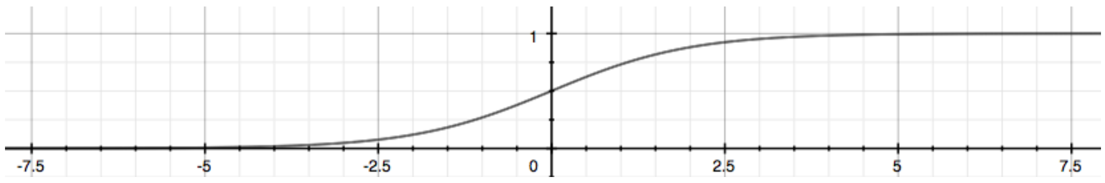
- To attempt classification, we can use Logistic Regression and map all predictions greater than 0.5 as 1 and all less than 0.5 as 0 so as to represent a binary system. However, this method doesn't work well in all cases of Machine Learning problems.
- Classification is just like a special case of regression, except that in this case we are interested in classifying or per se marking under discrete classes namely {0, 1, 2... n}.
- In this, we will focus on binary classification such as for example, classifying an image containing car **or not.** In this case, y, the output variable takes up only two values 0 or 1.

▼ HYPOTHESIS REPRESENTATION

- As here we are dealing with binary classification, we need to model the hypothesis function as to take values **strictly between 0 and 1.** It doesn't make sense to have predict greater than 1 or less than 0. To fix this, let us define the logistic function hypothesis as follows.

- We first need to understand the function Sigmoid. Let us represent the sigmoid function as to be

$$g(x) = 1/(1 + e^x)$$

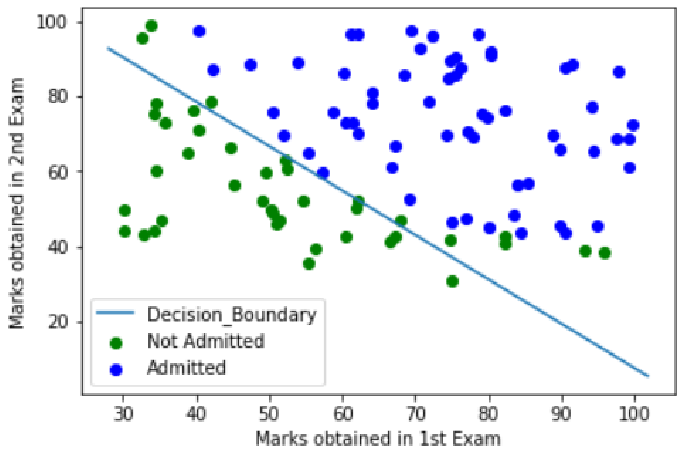


- Now, let the hypothesis function be as such:

$$h(x, \theta) = g(\theta^T * X)$$

- *h*(*x*) will give us the **probability** that our output is 1. For example, h(x) = 0.7 gives us the probability of 70% that our output is 1. Then, using the complement rule, we can figure out the probability of the output being 0 which is 30%.

▼ DECISION BOUNDARY



- In order to get our discrete 0 or 1 classification, we can translate the output of the hypothesis function as follows:
    - h(x) ≥ 0.5 → y = 1
    - h(x) < 0.5 → y = 0
- So, basically, if (θT*x) ≥ 0.5 → y = 1 or if (θT*x) < 0.5 → y = 0.

▼ EXAMPLE

$$
\begin{aligned}
&\theta = \begin{bmatrix} 5 \\ -1 \\ 0 \end{bmatrix} \\
&y = 1\ if\ 5 + (-1)x_1 + 0x_2 \geq 0 \\
&5 - x_1 \geq 0 \\
&-x_1 \geq -5 \\
&x_1 \leq 5
\end{aligned}
$$

- In this case, our decision boundary is a straight vertical line placed on the graph where x1 = 5 and everything to the left of that denotes y = 1, while everything to the right denotes y = 0.
- ⚠ The input to the sigmoid function g(z) doesn't need to be linear. We can use non-linear functions of any degree so as to fit to the data.

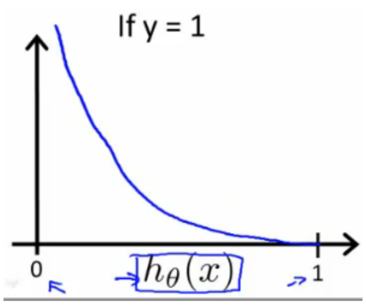
▼ COST FUNCTION

- We cant use the same cost function that we used for linear regression because the logistic function will case the output to be wavy, causing many local optima. In other words, it will not be a convex function.
- Instead, our cost function for logistic regression looks like:

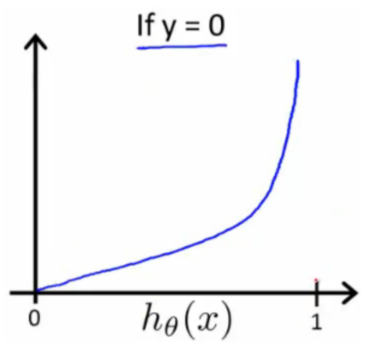
$$
\begin{aligned}
&J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)}) \\
&\mathrm{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if y = 1} \\
&\mathrm{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if y = 0}
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{Cost}(h_\theta(x), y) = 0 \text{ if } h_\theta(x) = y \\
&\mathrm{Cost}(h_\theta(x), y) \to \infty \text{ if } y = 0 \text{ and } h_\theta(x) \to 1 \\
&\mathrm{Cost}(h_\theta(x), y) \to \infty \text{ if } y = 1 \text{ and } h_\theta(x) \to 0
\end{aligned}
$$

- When y = 1, we get the following plot for J(theta) vs h(x):



- When y = 0, we get the following plot for the same:



▼ SIMPLIFIED COST FUNCTION AND GRADIENT DESCENT

- We can represent the cost function in one line by doing the following

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

- Vectorized implementation of the following is as such:

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

▼ GRADIENT DESCENT

- The general form of gradient descent is:

$$\begin{aligned}&Repeat\ \{\\ &\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)\\ &\}\end{aligned}$$

- Working out the calculus part, we get:

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

- A vectorized implementation is as follows:

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

▼ ADVANCED OPTIMIZATION

- "Conjugate gradient", "BFGS", and "L-BFGS" are more sophisticated, faster ways to optimize θ that can be used instead of gradient descent.
- We can use the functions available through the libraries, writing them from scratch is a painstakingly difficult task for a layman.
- They're already tested and highly optimized. Octave / MATLAB provides them.
- Before using those functions, we first need to provide a function that evaluates the following two functions for a given input value θ:

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

- Then we set the options to the optimizer and pass in the values as such.

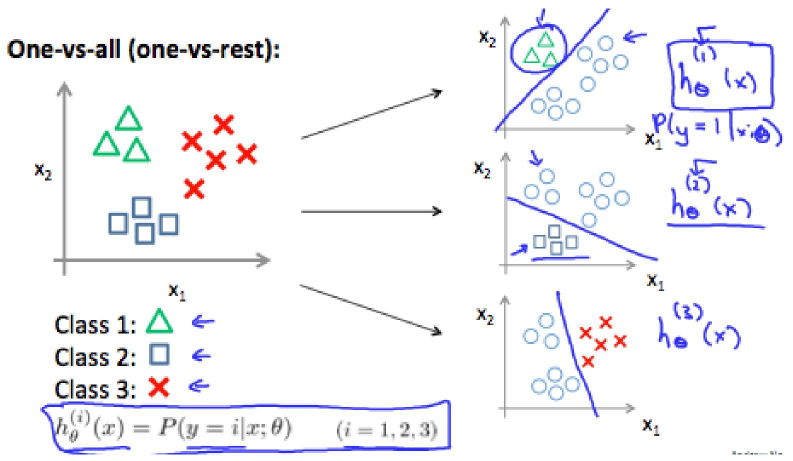
  ```
  options = optimset('GradObj', 'on', 'MaxIter', 100);
  initialTheta = zeros(2,1);
  [optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta, options);
  ```

▼ MULTICLASS CLASSIFICATION: ONE-VS-ALL

- Until now, we have dealt with binary classification. What if we have more than 2 classes? Such as classifying a car as **Lamborghini**, or **Bugatti** or **Mercedes** or **Benz**? This is the problem of multi-class classification.
- Instead of bounding the output variable y to be only two classes, we now can make it as such: y = {0, 1, 2, ... , n}
- Here, we are performing binary classification for each of the classes and form n + 1 hypothesis functions(where n is the number of classes) and pick the maximum one out of all those for the given input so as to classify the new input.
- Since y = {0,1...n}, we divide our problem into n+1 (+1 because the index starts at 0) binary classification problems; in each one, we predict the probability that 'y' is a member of one of our classes.

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

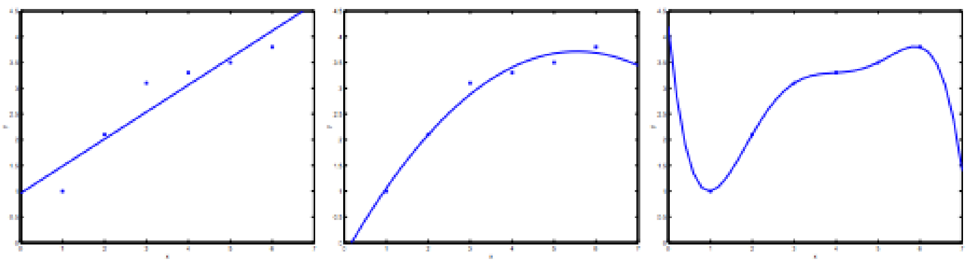
- We are basically choosing one class and then lumping all the others into a single second class. We do this repeatedly, applying binary logistic regression to each case, and then use the hypothesis that returned the highest value as our prediction.
- The following shows how this works for three classes.



SUMMARY: To make a prediction, pick the class that maximizes h(X).

PROBLEM OF OVERFITTING AND UNDERFITTING

- From the below picture, we can draw a few key insights.
  - **First Plot:** We are **underfitting** the data. In this case, we aren't using a linear function which doesn't give the optimal prediction for new data. This can be resolved by including some polynomial terms making it a non-linear hypothesis.
    - Naively, it might seem that the more features we add, better the fit to the data. However, this is not the case.
  - **Third Plot**: If we add too many features, we may **overfit** the training data which is not optimal because if we do so, for the testing data, the output wouldn't be as close to the true value.
  - **Second Plot:** And so, the second plot is the right hypothesis for the training data provided which doesn't overfit or underfit it.



- **Underfitting**, or high bias, is when the form of our hypothesis function h maps poorly to the trend of the data. It is usually caused by a function that is too simple or uses too few features.
- **Overfitting**, or high variance, is caused by a hypothesis function that fits the available data but does not generalize well to predict new data. It is usually caused by a complicated function that creates a lot of unnecessary curves and angles unrelated to the data.

There are two main options to address the issue of overfitting:

1. Reduce the number of features:

- Manually select which features to keep.
- Use a model selection algorithm (studied later in the course).

2. Regularization

- Keep all the features, but reduce the magnitude of parameters *θj*.
- Regularization works well when we have a lot of slightly useful features.

▼ REGULARIZATON: RESOLVING THE PROBLEM

▼ REGULARIZED LINEAR REGRESSION

▼ COST FUNCTION

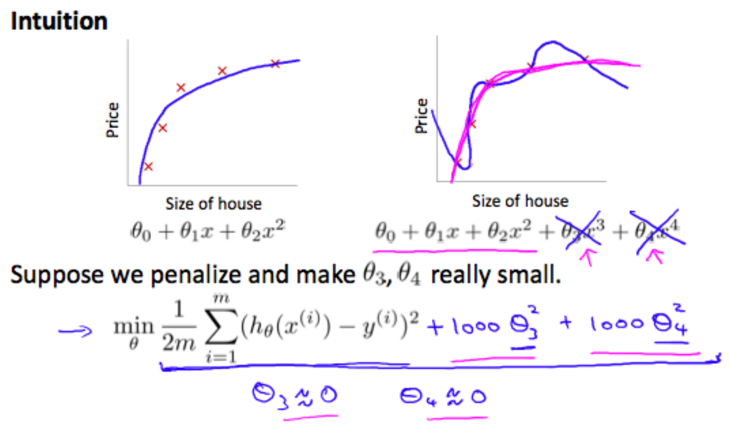
- If we detect overfitting from out hypothesis function, we can reduce the weight that some of the terms in our function carry by increasing their cost.
- Say we wanted to make the following function more quadratic:

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

- If we want to eliminate the influence of the last two terms without actually removing them, we can modify the cost function as follows

$$min_\theta \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000 \cdot \theta_3^2 + 1000 \cdot \theta_4^2$$

- We are adding those two extra terms at the end so as to penalize the last two features.



Intuition

$\theta_0 + \theta_1 x + \theta_2 x^2$

$\underline{\theta_0 + \theta_1 x + \theta_2 x^2} + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$

Suppose we penalize and make θ_3, θ_4 really small.

$$\longrightarrow \underline{\min_\theta \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\, \theta_3^2 + 1000\, \theta_4^2}$$

$\underline{\theta_3 \approx 0}$ $\quad \underline{\theta_4 \approx 0}$

- If we are interested in penalizing all the features, we can do it as follows.

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

- Using the above cost function with the extra summation, we can smooth the output of our hypothesis function to reduce overfitting. If lambda is chosen to be too large, it may smooth out the function too much and cause underfitting. Hence, we are to pick and tune the value of λ carefully.

- We can apply regularization technique to both Linear and Logistic Regression.
- **GRADIENT DESCENT**
  - We will modify our gradient descent function to separate out *θ0* from the rest of the parameters because we do not want to penalize *θ0*.

$$\text{Repeat } \{$$
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$
$$\theta_j := \theta_j - \alpha \left[\left(\frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \right] \qquad j \in \{1, 2 \ldots n\}$$
$$\}$$

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

- The first term in the above equation (1 - α*(λ/m) will always be less than 1.

- **NORMAL EQUATION**
  - To add in regularization, the equation is the same as our original, except that we add another term inside the parentheses:

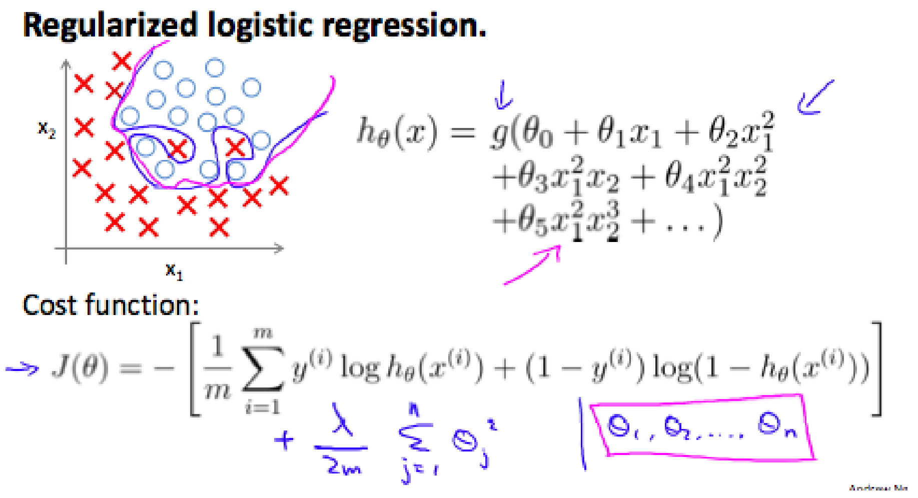
$$\theta = \left(X^TX + \lambda \cdot L\right)^{-1} X^Ty$$

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

▼ REGULARIZED LOGISTIC REGRESSION

- **COST FUNCTION**
  - We can regularize the equation by adding a term to the end of the cost function we defined earlier.

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



Regularized logistic regression.

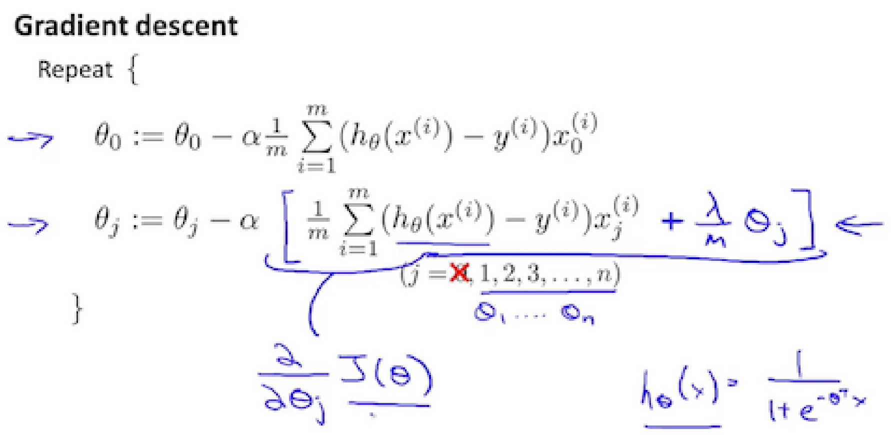
$$h_\theta(x) = g(\theta_0 + \theta_1x_1 + \theta_2x_1^2 + \theta_3x_1^2x_2 + \theta_4x_1^2x_2^2 + \theta_5x_1^2x_2^3 + \dots)$$

Cost function:

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

$$\boxed{\theta_1, \theta_2, \dots, \theta_n}$$

- **GRADIENT DESCENT**



Gradient descent

Repeat {

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

$$\rightarrow \quad \theta_j := \theta_j - \alpha \underbrace{\left[\frac{1}{m} \sum_{i=1}^{m} (\underline{h_\theta(x^{(i)})} - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right]}_{\frac{\partial}{\partial \theta_j} \underline{J(\theta)}} \leftarrow$$

$$(j = \cancel{0}, \underbrace{1, 2, 3, \ldots, n}_{\theta_1 \ldots \theta_n})$$

}

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