

MACHINE LEARNING



LOGISTIC REGRESSION & REGULARIZATION

WEEK 3

Classification and Representation

Recall some classification problems

- Email: Spam / Not Spam?
- Online Transactions: Fraudulent (Yes / No)?
- Tumor: Malignant / Benign?

In all these problems, we try to predict a variable $y \in \{0, 1\}$

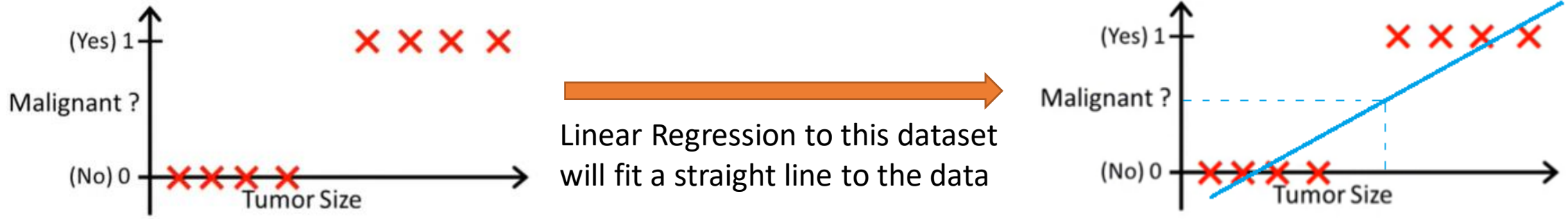
0 : “Negative Class” Eg: (Benign tumor)

1 : “Positive Class” Eg: (Malignant tumor)

This type of classification problem is called “Binary Classification”

For a multiclass classification problem, $y \in \{0, 1, 2, \dots, n\}$

Example:

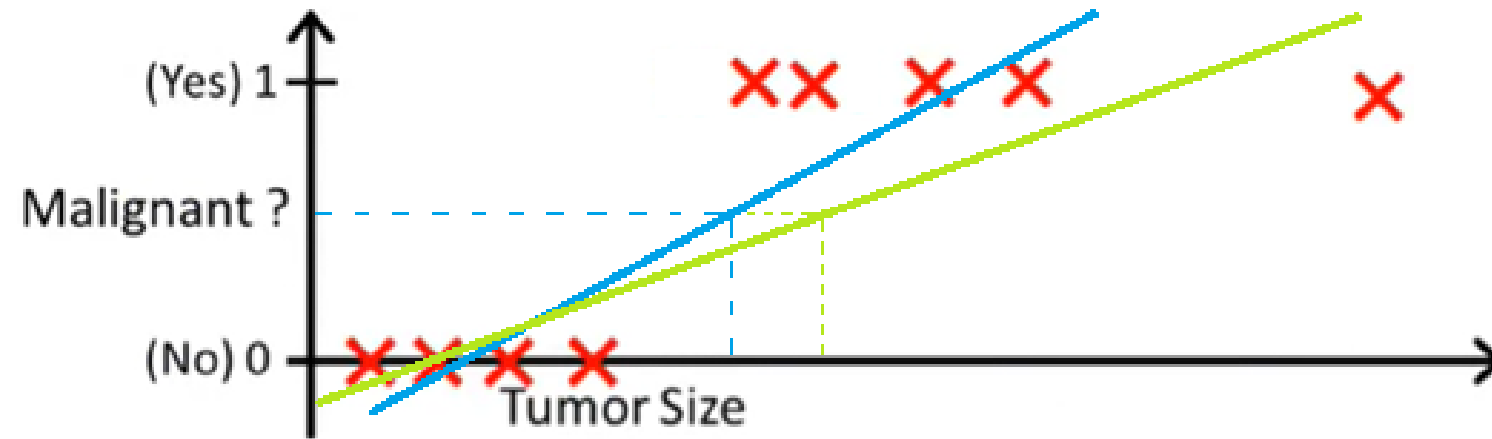


If we want to make predictions, we can set the threshold classifier output $h_{\theta}(x)$ at 0.5:

- If $h_{\theta}(x) \geq 0.5$, predict $y=1$
- If $h_{\theta}(x) < 0.5$, predict $y=0$

Linear Regression for this classification problem fits well

Now let's modify our dataset by adding an extra training example.



- The linear regression straight line does not fit well when an extra training example is added
- It is identifying Malignant tumor wrongly as Benign tumor
- Thus, linear regression to a classification problem is not a great idea always

What if $h_{\theta}(x) > 1$ or < 0 ?

- This is another problem caused when linear regression is used for a classification problem
- For the previous example, the labels belong to the range $y \in \{0, 1\}$
- Output cannot be greater than 1 or less than 0

Logistic Regression:

- Logistic Regression algorithm is used for classification problems instead of linear regression to overcome this problem
- $0 \leq h_{\theta}(x) \leq 1$

Which of the following statements is true?

- If linear regression doesn't work on a classification task as shown in previous example, then applying feature scaling may help
- If the training set satisfies $0 \leq y^{(i)} \leq 1$ for every training example $(x^{(i)}, y^{(i)})$, then linear regression's prediction will also satisfy $0 \leq h_{\theta}(x) \leq 1$ for all values of x
- If there is a feature x that perfectly predicts y i.e. if $y=1$ when $x \geq c$ and $y=0$ whenever $x < c$ (for some constant c), then linear regression will obtain zero classification error
- None of the above

Hypothesis Representation

Recap:

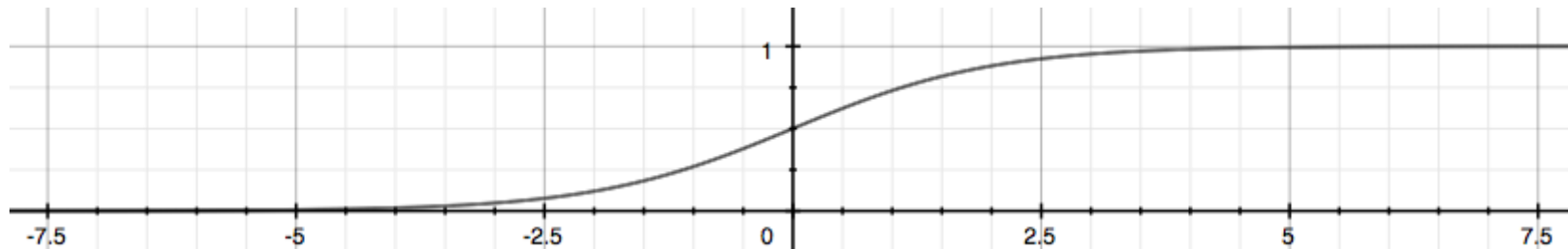
For linear regression, $h_{\theta}(x) = \theta^T x$

Logistic Function:

A logistic function, $g(z) = \frac{1}{1+e^{-z}}$

Therefore, hypothesis for logistic regression is given by,

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



Logistic Function is also called Sigmoid Function

Interpretation of Hypothesis Output

$h_{\theta}(x)$ = estimated probability that $y=1$ on input x

Example:

$$\text{If } x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$$

and $h_{\theta}(x) = 0.7$

Then this hypothesis is interpreted as **“there is a 70% probability that the cancer is malignant”**

Mathematically,

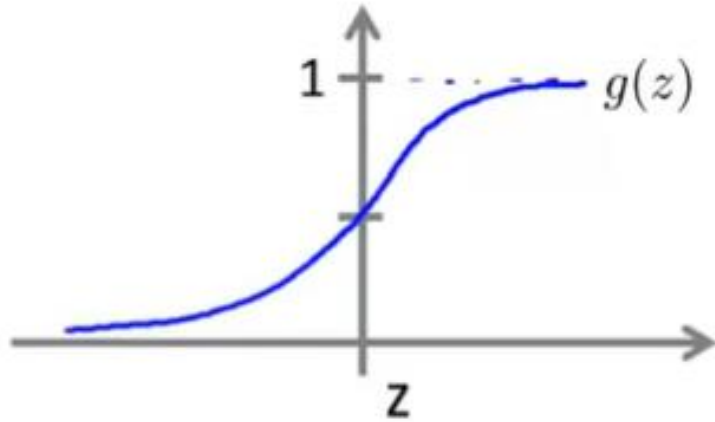
$$h_{\theta}(x) = P(y=1 \mid x; \theta)$$

(Probability that $y=1$, given x features, parameterized by θ)

Suppose we want to predict, from data x about a tumor, whether it is malignant ($y=1$) or benign ($y=0$). Our logistic regression classifier outputs, for a specific tumor, $h_{\theta}(x) = P(y=1 \mid x; \theta) = 0.7$, so we estimate that there is a 70% chance of this tumor being malignant. What should be our estimate for $P(y=0 \mid x; \theta)$, the probability the tumor is benign?

- $P(y=0 \mid x; \theta)=0.3$
- $P(y=0 \mid x; \theta)=0.7$
- $P(y=0 \mid x; \theta)=0.7^2$
- $P(y=0 \mid x; \theta)=0.3 \times 0.7$

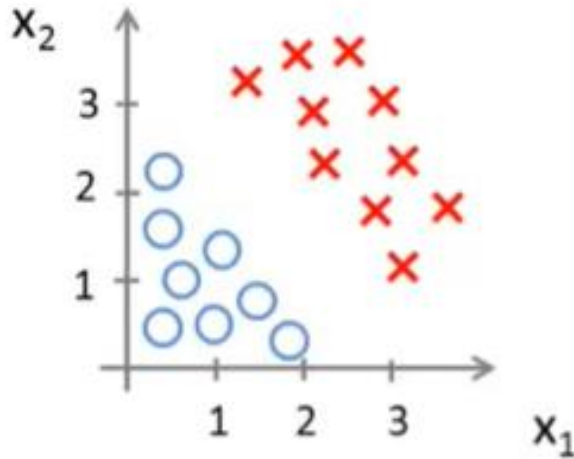
Logistic Regression



- $z = \theta^T x$
- $g(z) = g(\theta^T x) \geq 0.5$ when $z = \theta^T x \geq 0 \quad \Rightarrow \quad y = 1$
- $g(z) = g(\theta^T x) < 0.5$ when $z = \theta^T x < 0 \quad \Rightarrow \quad y = 0$

Decision Boundary

Consider a training set as shown below with following hypothesis



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

Suppose we choose $\theta_0 = -3$, $\theta_1 = 1$, $\theta_2 = 1$

The parameter vector θ is thus given by, $\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$

$$\therefore h_{\theta}(x) = g(-3 + x_1 + x_2)$$

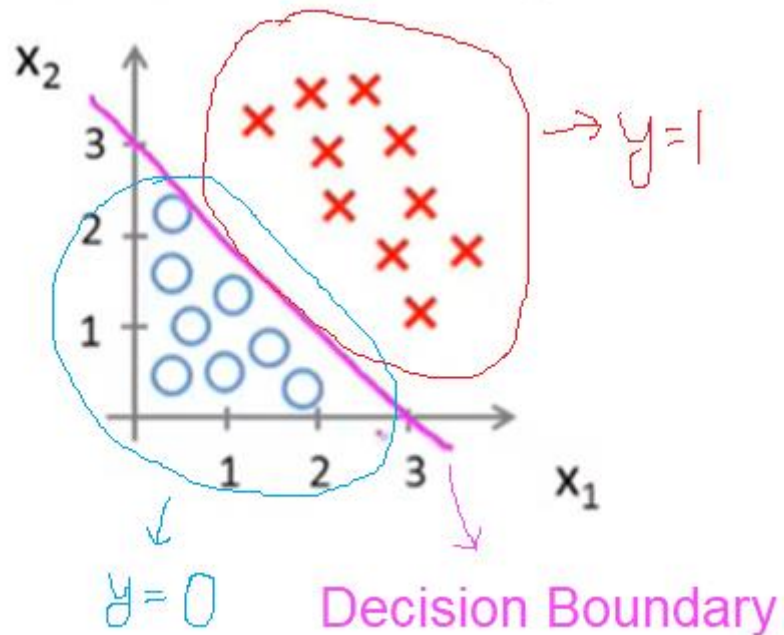
$$\underbrace{\hspace{1.5cm}}_{\theta^T x}$$

For $y = 1$,

$$z \geq 0 \Rightarrow -3 + x_1 + x_2 \geq 0$$

$$\therefore x_1 + x_2 \geq 3$$

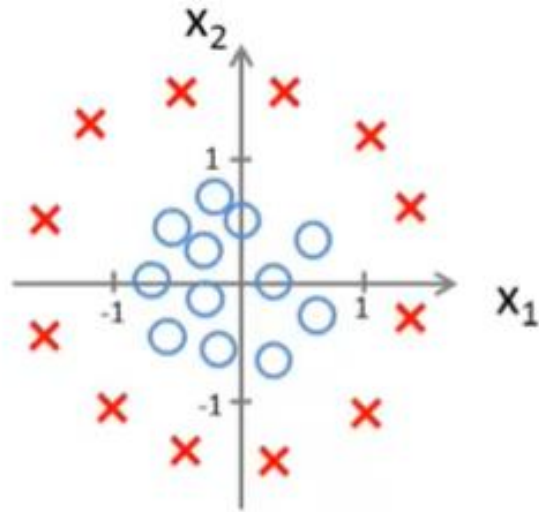
The plot will look something like this



The decision boundary for this example is linear

Non-Linear Decision Boundary

Consider a training set as shown below with following hypothesis



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Suppose we choose $\theta_0 = -1$, $\theta_1 = 0$, $\theta_2 = 0$, $\theta_3 = 1$, $\theta_4 = 1$

The parameter vector θ is thus given by, $\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

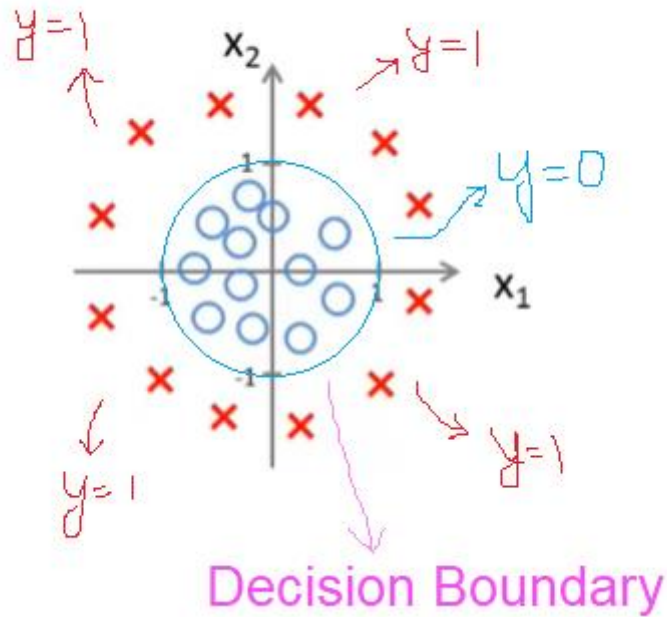
$$\therefore h_{\theta}(x) = g(-1 + x_1^2 + x_2^2)$$

For $y = 1$,

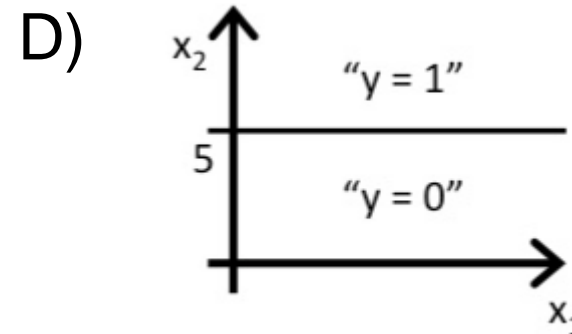
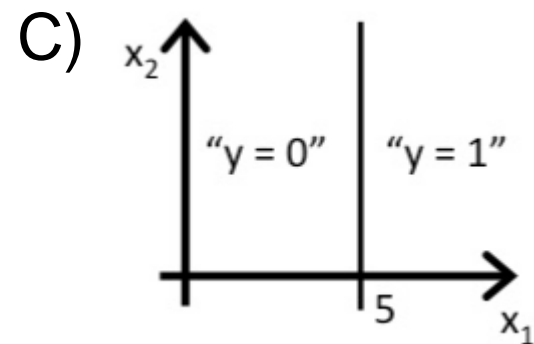
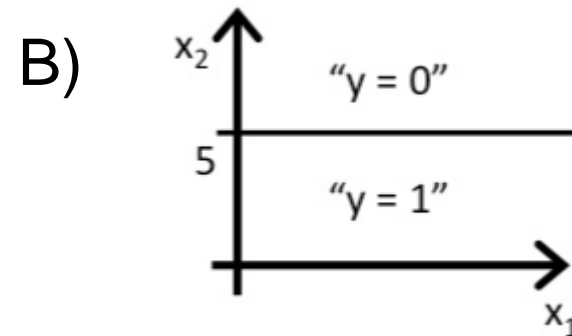
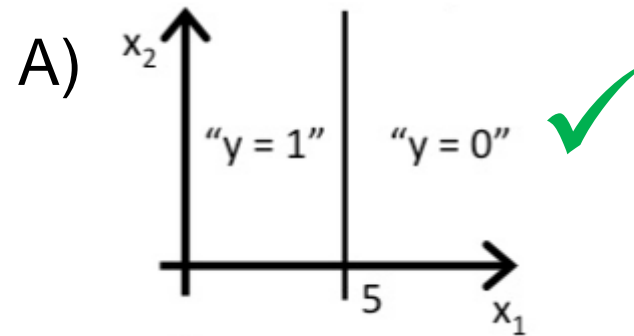
$$z \geq 0 \Rightarrow -1 + x_1^2 + x_2^2 \geq 0$$

$$\therefore x_1^2 + x_2^2 \geq 1$$

The plot will look something like this



Consider logistic regression with two features x_1 and x_2 .
Suppose $\theta_0 = 5$, $\theta_1 = -1$ and $\theta_2 = 0$, so that $h_{\theta}(x) = g(5 - x_1)$.
Which of these shows the decision boundary of $h_{\theta}(x)$?



Cost Function of Logistic Regression

Recall cost function for linear regression

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

We cannot use the same cost function for logistic regression because the logistic function will cause the output to be wavy, causing many local optima. In other words, it will not be a convex function.

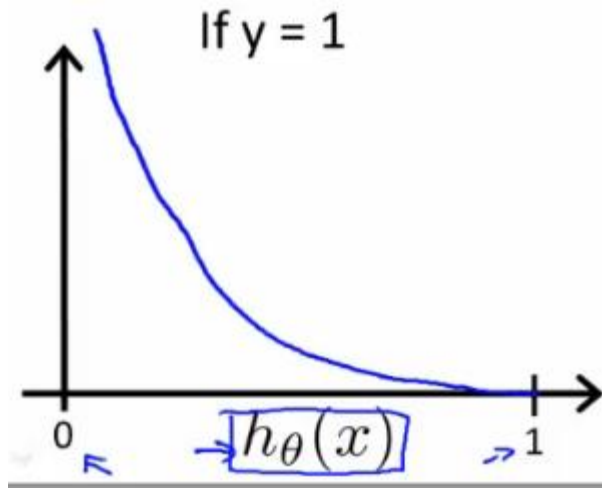
Instead, we define the cost function for logistic regression as follows:

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

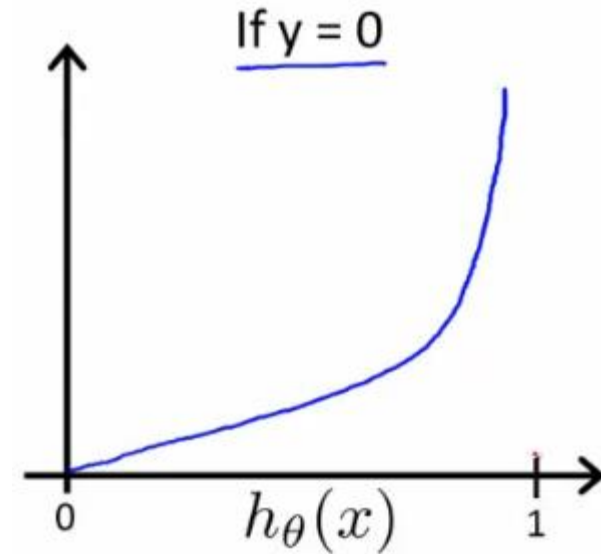
$$\text{Cost}(h_{\theta}(x), y) = -\log(h_{\theta}(x)) \quad \text{if } y = 1$$

$$\text{Cost}(h_{\theta}(x), y) = -\log(1 - h_{\theta}(x)) \quad \text{if } y = 0$$

Intuition



$J(\theta)$ vs $h_\theta(x)$ when $y=1$



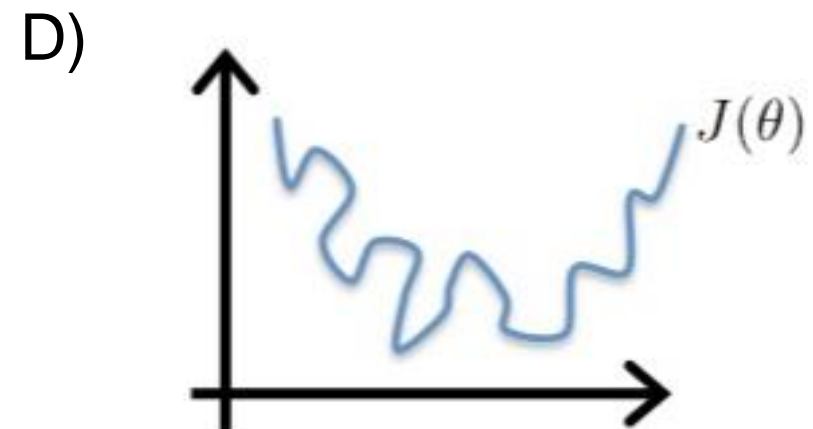
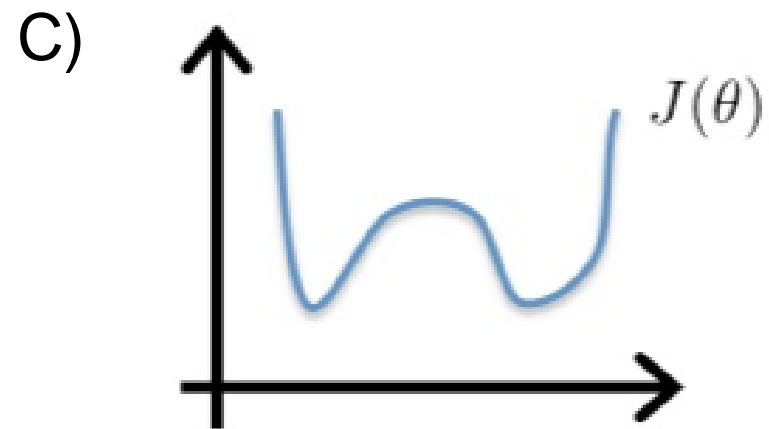
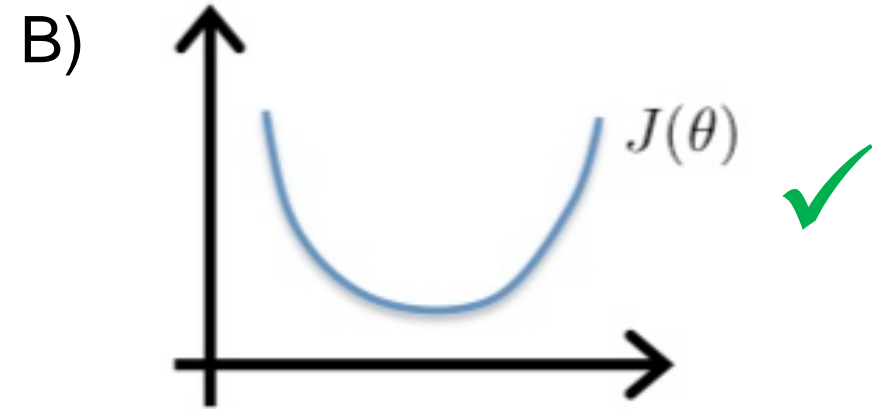
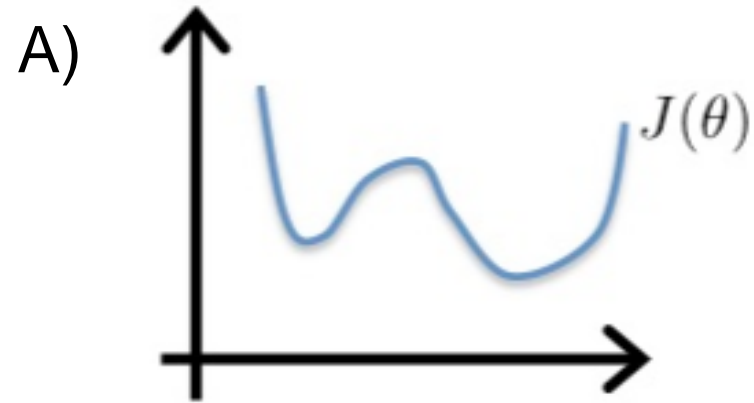
$J(\theta)$ vs $h_\theta(x)$ when $y=0$

From the plots, we deduce that –

- $\text{cost}(h_\theta(x), y) = 0$ if $h_\theta(x) = y$
- $\text{cost}(h_\theta(x), y) \rightarrow \infty$ if $y=0$ and $h_\theta(x) \rightarrow 1$
- $\text{cost}(h_\theta(x), y) \rightarrow \infty$ if $y=1$ and $h_\theta(x) \rightarrow 0$

Writing the cost function in this way guarantees that $J(\theta)$ is convex for logistic regression.

Consider minimizing a cost function $J(\theta)$. Which one of these functions is convex?



In logistic regression, the cost function for our hypothesis outputting (predicting) $h_{\theta}(x)$ on a training example that has label $y \in \{0,1\}$ is:

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

Which of the following are true? Check all that apply.

- If $h_{\theta}(x) = y$, then $\text{cost}(h_{\theta}(x), y) = 0$ (for $y=0$ and $y=1$)
- If $y=0$, then $\text{cost}(h_{\theta}(x), y) \rightarrow \infty$ as $h_{\theta}(x) \rightarrow 1$
- If $y=0$, then $\text{cost}(h_{\theta}(x), y) \rightarrow \infty$ as $h_{\theta}(x) \rightarrow 0$
- Regardless of whether $y=0$ or $y=1$ if $h_{\theta}(x)=0.5$, then $\text{cost}(h_{\theta}(x), y) > 0$

Simplified Cost Function

Recall cost function of logistic regression

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

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

$$\text{Cost}(h_{\theta}(x), y) = -\log(1 - h_{\theta}(x)) \quad \text{if } y = 0$$

This can be generalized as follows:

$$\text{cost}(h_{\theta}(x), y) = -y \log(h_{\theta}(x)) - (1-y) \log(1 - h_{\theta}(x))$$

Thus, the cost function for i^{th} training example is given by,

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

Gradient Descent

repeat until convergence

{

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

Surprisingly, the gradient descent algorithm looks identical to that in linear regression

Repeat {

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

}

(simultaneously update all θ_j)

Vectorized Implementation

Cost Function:

$$h = g(X\theta)$$

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

Gradient Descent:

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

Suppose you are running gradient descent to fit a logistic regression model with parameter $\theta \in \mathbb{R}^{n+1}$. Which of the following is a reasonable way to make sure the learning rate α is set properly and that gradient descent is running correctly?

- Plot $J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)})^2$ as a function of the number of iterations (i.e. the horizontal axis is the iteration number) and make sure $J(\theta)$ is decreasing on every iteration
- Plot $J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log h_{\theta}(x^{(i)}) + (1-y^{(i)}) \log(1-h_{\theta}(x^{(i)}))]$ as a function of the number of iterations and make sure $J(\theta)$ is decreasing on every iteration ✓
- Plot $J(\theta)$ as a function of θ and make sure it is decreasing on every iteration
- Plot $J(\theta)$ as a function of θ and make sure it is convex

One iteration of gradient descent simultaneously performs these updates:

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

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}$$

\vdots

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

We would like a vectorized implementation of the form $\theta := \theta - \alpha \delta$ (for some vector $\delta \in \mathbb{R}^{n+1}$).

What should the vectorized implementation be?

- $\theta := \theta - \alpha \frac{1}{m} \sum_{i=1}^m [(h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}]$ ✓
- $\theta := \theta - \alpha \frac{1}{m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \right] \cdot x^{(i)}$
- $\theta := \theta - \alpha \frac{1}{m} x^{(i)} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \right]$
- All of the above are correct implementations

Advanced Optimization

Some advanced optimized algorithms can be used instead of Gradient Descent algorithm that provide faster ways to optimize θ .

- Conjugate Gradient
- BFGS
- L-BFGS

We first need to provide a function that evaluates the following two functions for a given input value θ :

- $J(\theta)$
- $\frac{\partial}{\partial \theta_j} J(\theta)$

Advantages of optimized algorithms:

- No need to manually pick α
- Often faster than gradient descent

Disadvantages of optimized algorithms:

- More complex

Example:

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

$$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$

$$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$$

$$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$$

To find the minimum of the cost function, $\frac{\partial}{\partial \theta_1} J(\theta) = 0$ and $\frac{\partial}{\partial \theta_2} J(\theta) = 0$

$$\Rightarrow \theta_1 = 5$$

$$\Rightarrow \theta_2 = 5$$

```
function [ jVal, gradient ] = costFunction(theta)
    jVal = (theta(1) - 5)^2 + (theta(2) - 5)^2;
    gradient = zeros(2,1);
    gradient(1) = 2 * (theta(1) - 5);
    gradient(2) = 2 * (theta(2) - 5);
end
```

Having implemented the cost function, you would then call the advanced optimization function called **'fminunc'**

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

Suppose you want to use an advanced optimization algorithm to minimize the cost function for logistic regression with parameters θ_0 and θ_1 . You write the following code:

```
function [ jVal, gradient ] = costFunction(theta)
    jVal = % code to compute J(theta)
    gradient(1) = CODE#1 % derivative for theta_0
    gradient(2) = CODE#2 % derivative for theta_1
end
```

What should CODE#1 and CODE#2 above compute?

- CODE#1 and CODE#2 should compute $J(\theta)$
- CODE#1 should be $\theta(1)$ and CODE#2 should be $\theta(2)$
- CODE#1 should compute $\frac{1}{m} \sum_{i=1}^m [(h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}]$ ($= \frac{\partial}{\partial \theta_0} J(\theta)$), and CODE#2 should compute $\frac{1}{m} \sum_{i=1}^m [(h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}]$ ($= \frac{\partial}{\partial \theta_1} J(\theta)$)
- None of the above

Multiclass Classification

Multiclass Classification consists of more than two classes.

Example:

Email tagging: Social, Updates, Work, Friends

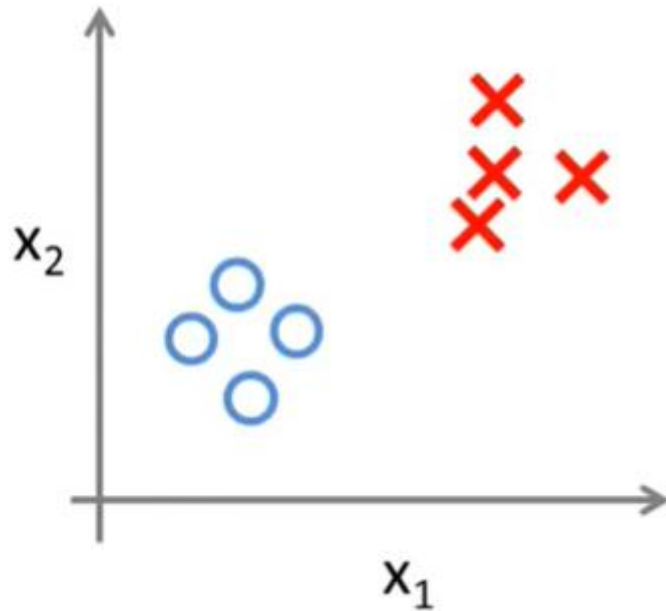
Medical conditions: Not ill, Cold, Flu

Weather: Sunny, cloudy, Rainy, Snow

Thus, $y = \{0, 1, \dots, n\}$

Representation

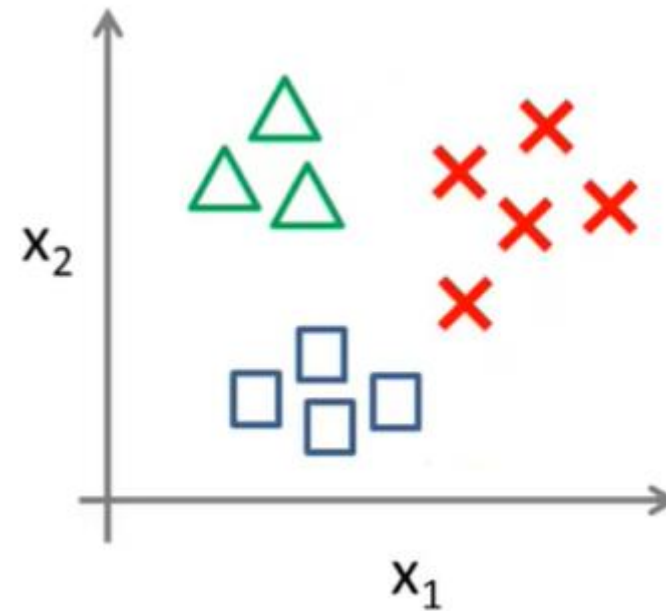
Binary Classification



Class 1: ○

Class 2: ✕

Multiclass Classification



Class 1: △

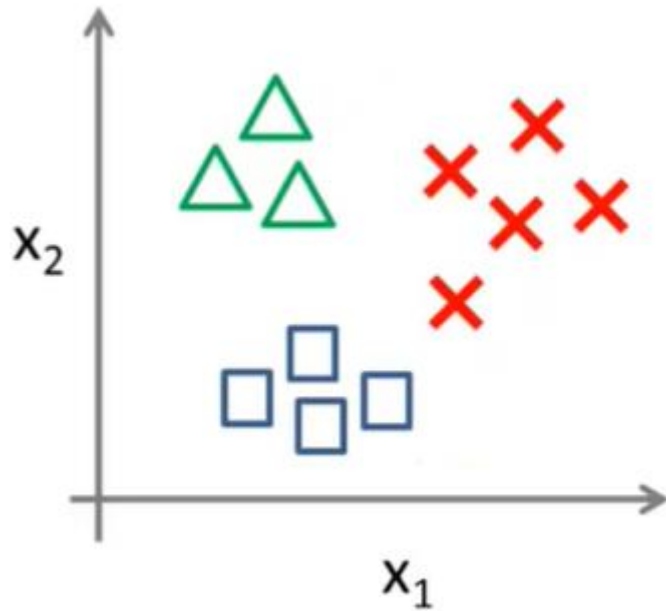
Class 2: ✕


Class 3: □


One-vs-All Classification


- Divide the multiclass classification problem into separate binary classification problems.
- We basically choose one class as a positive class and lump all the other classes as a single negative 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.

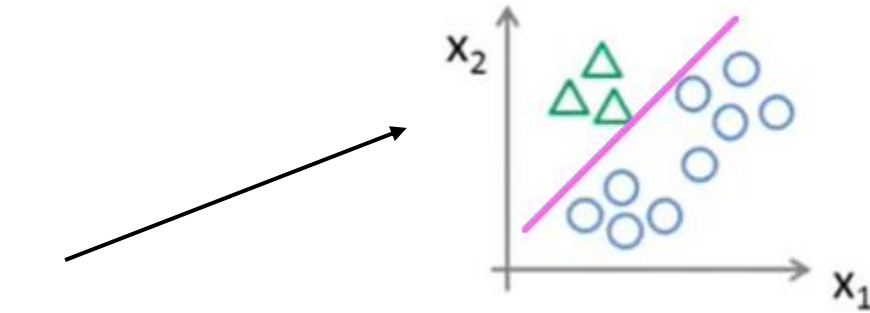
Example:



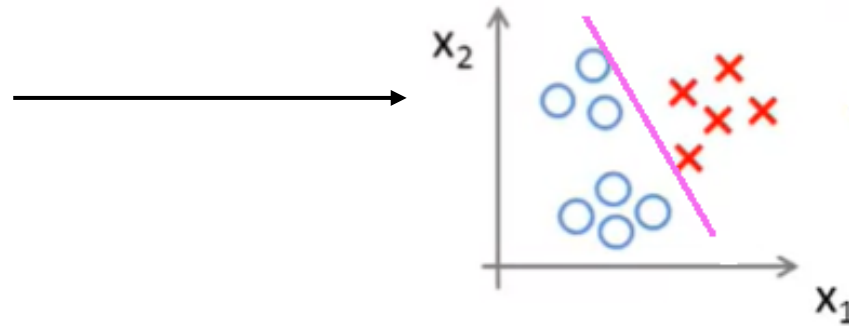
Class 1: 

Class 2: 

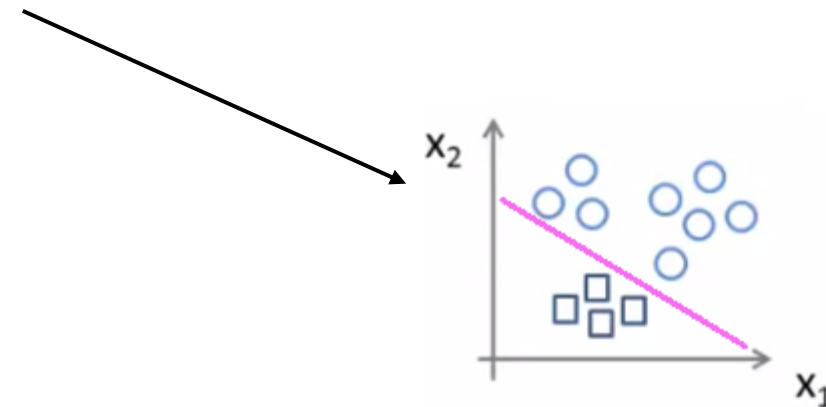
Class 3: 



$$h_{\theta}^{(1)}(x) = P(y=1 \mid x; \theta)$$



$$h_{\theta}^{(2)}(x) = P(y=2 \mid x; \theta)$$



$$h_{\theta}^{(3)}(x) = P(y=3 \mid x; \theta)$$

Summary

Train a logistic regression classifier $h_{\theta}^{(i)}(x)$ for each class 'i', to predict the probability that $y=i$

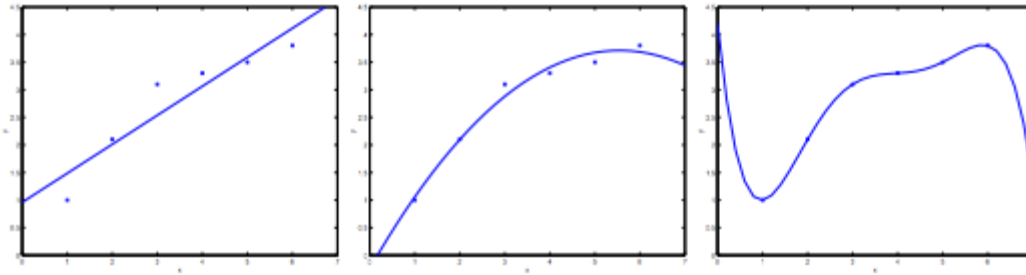
On a new input 'x', to make a prediction, pick the class 'i' that maximizes $h_{\theta}^{(i)}(x)$

Suppose you have a multi-class classification problem with k classes (so $y \in \{1, 2, 3, \dots, k\}$). Using the One-vs-All method, how many different logistic regression classifiers will you end up training?

- $k-1$
- k
- $k+1$
- Approximately $\log_2(k)$

Overfitting

Consider we have to predict y from $x \in \mathbb{R}$.



- Leftmost figure shows result of fitting $y = \theta_0 + \theta_1 x$ to a dataset. The fit is not good because the data points don't lie on the straight line. This problem is called **Underfitting**.
- We add an extra feature x_2 and fit $y = \theta_0 + \theta_1 x + \theta_2 x^2$. Middle figure shows that fit obtained is slightly better. It can be naively inferred that, **more features we add, better is the fit**.
- The rightmost figure is the result of fitting a 5th order polynomial $y = \sum_{j=0}^5 \theta_j x^j$. Even though the fitted curve passes through the data perfectly, it will not predict the output for new examples. This problem is called **Overfitting**.

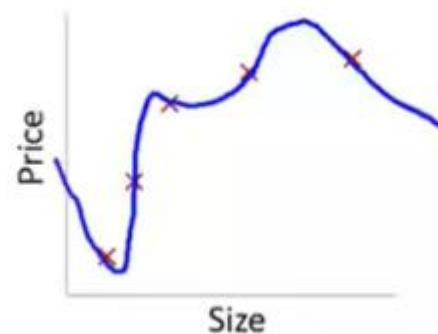
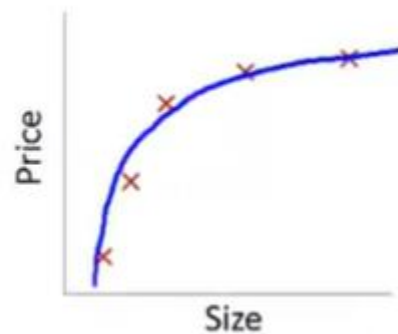
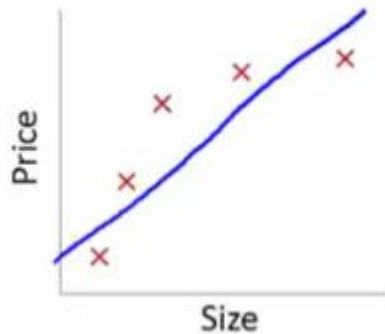
Underfitting:

- **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.

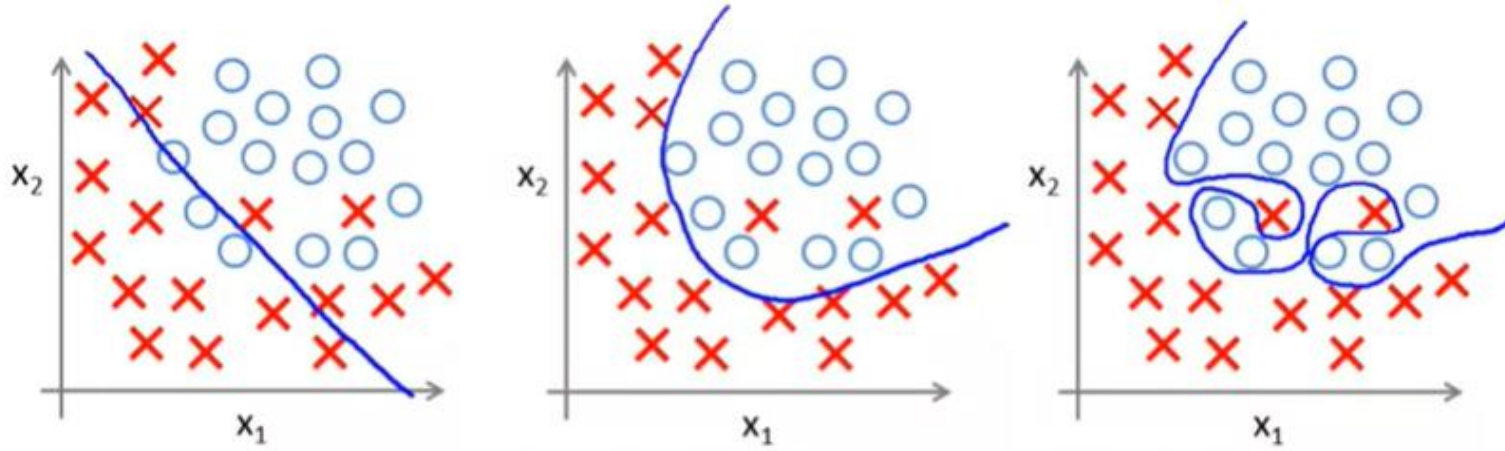
Overfitting:

- **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.

This terminology is applied to both –
1) Linear Regression



2) Logistic Regression



Problem of overfitting can be dealt in two ways –

1) Reduce the number of features:

- Manually select which features to keep
- Using [Model Selection](#) algorithm

2) Regularization:

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

Example:

House price prediction

x_1 = size of house

x_2 = no. of bedrooms

x_3 = no. of floors

x_4 = kitchen size

x_5 = age of house

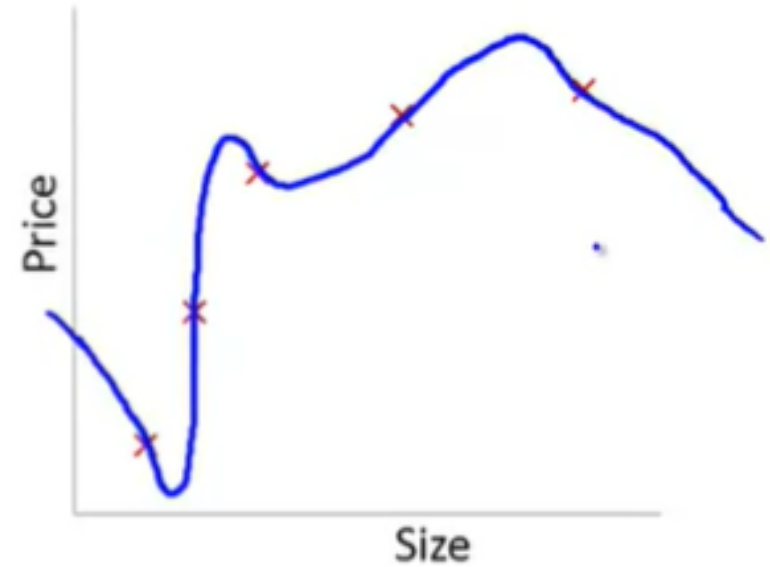
x_6 = average income in neighborhood

.

.

.

x_{100}



A lot of features and a very little training data can cause the problem of overfitting

Consider the medical diagnosis problem of classifying tumors as malignant or benign. If a hypothesis $h_{\theta}(x)$ has overfit the training set, it means that:

- It makes accurate predictions for examples in the training set and generalizes well to make accurate predictions on new, previously unseen examples
- It does not make accurate predictions for examples in the training set, but it does generalize well to make accurate predictions on new, previously unseen examples
- It makes accurate predictions for examples in the training set, but it does not generalize well to make accurate predictions on new, previously unseen examples
- It does not make accurate predictions for examples in the training set and does not generalize well to make accurate predictions on new, previously unseen examples

Regularization – Cost Function

If we have overfitting from our hypothesis function, we can reduce the weight of some terms in our function by increasing their cost.

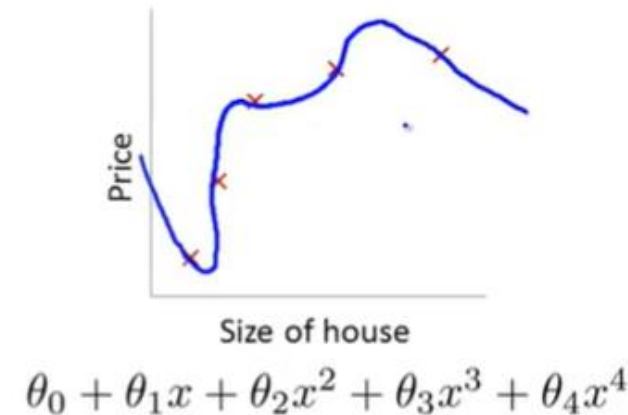
Example:

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

This function overfits the training set because of more number of features.

This can be made more quadratic by eliminating the influence of $\theta_3 x^3$ and $\theta_4 x^4$

Without actually getting rid of these features or changing the form of our hypothesis, we can instead modify our **cost function**

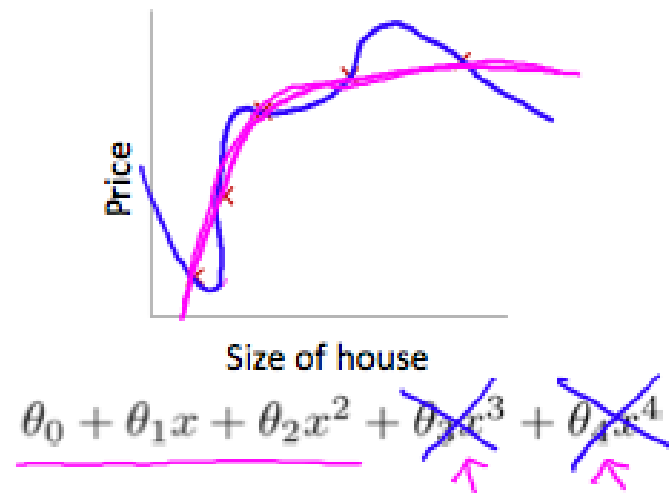


$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \underbrace{1000 \cdot \theta_3^2 + 1000 \cdot \theta_4^2}_{\text{very small value}}$$

We've added two extra terms at the end to inflate the cost of θ_3 and θ_4 . Now, in order for the cost function to get close to zero, we will have to reduce the values of θ_3 and θ_4 to near zero.

This will in turn greatly reduce the values of $\theta_3 x^3$ and $\theta_4 x^4$ in our hypothesis function.

The new hypothesis looks like a **quadratic function** but fits the data better due to extra small terms $\theta_3 x^3$ and $\theta_4 x^4$



We could also regularize all of our theta parameters in a single summation as:

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

The λ , or lambda, is the **regularization parameter**. It determines how much the costs of our theta parameters are inflated.

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**.

What if $\lambda=0$ or λ is very small?

In regularized linear regression, we choose θ to minimize:

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

What if λ is set to an extremely large value (perhaps too large for our problem, say $\lambda = 10^{10}$)?

- Algorithm works fine; setting λ to be very large can't hurt it
- Algorithm fails to eliminate overfitting
- Algorithm results in underfitting (fails to fit even the training set)
- Gradient descent will fail to converge

Consider an example of housing price prediction where,

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

very small value

If λ is very large, then values of $\theta_1 x$, $\theta_2 x^2$, $\theta_3 x^3$ and $\theta_4 x^4$ are reduced greatly.

Thus, $h_{\theta}(x) = \theta_0$

The new hypothesis function therefore **underfits** the data



Regularized Linear Regression

The cost function for regularized linear regression is given by,

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

Gradient descent for regularized linear regression is thus given by,

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)} \quad \text{for } j=0$$

$\frac{\partial}{\partial \theta_0} J(\theta)$ (red arrow pointing to the sum)

θ_0 is not penalized by the regularization parameter (orange arrow pointing to the equation)

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \quad \text{for } j=1, 2, \dots, n$$

$\frac{\partial}{\partial \theta_j} J(\theta)$ (blue arrow pointing to the sum and the regularization term)

}

Suppose you are doing gradient descent on a training set of $m > 0$ examples, using a fairly small learning rate $\alpha > 0$ and some regularization parameter $\lambda > 0$. Consider the update rule:

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

Which of the following statements about the term $1 - \alpha \frac{\lambda}{m}$

- $1 - \alpha \frac{\lambda}{m} > 1$
- $1 - \alpha \frac{\lambda}{m} = 1$
- $1 - \alpha \frac{\lambda}{m} < 1$
- None of the above

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

Less than 1 \Rightarrow shrinks θ_0 towards zero

Original gradient descent equation

- The first term in the equation is always less than 1. Hence, it reduces the value of θ_j by some amount on every update (approaching towards 0).
- The second term is exactly the same as the original gradient descent equation

Regularized Normal Equation

Regularized normal equation is given by,

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

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

- L is a matrix with 0 at the top left and 1's down the diagonal, with 0's everywhere else.
- It should have dimension $(n+1) \times (n+1)$. Intuitively, this is the identity matrix (though we are not including x_0), multiplied with a single real number λ .

Example: For $n=2$, $L = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ is a 3x3 matrix

Non-invertibility


Suppose m (#examples) $\leq n$ (#features)

The Normal Equation is given by,
 $\theta = (X^T X)^{-1} X^T y$

X is non-invertible if $m < n$ and may be non-invertible if $m = n$

If $\lambda > 0$,

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \right)^{-1} X^T y$$


Never singular

Hence, regularization takes care of non-invertibility issue of the matrix.

Regularized Logistic Regression

We can regularize logistic regression in a similar way that we regularize linear regression. As a result, we can avoid overfitting.

Consider a hypothesis,

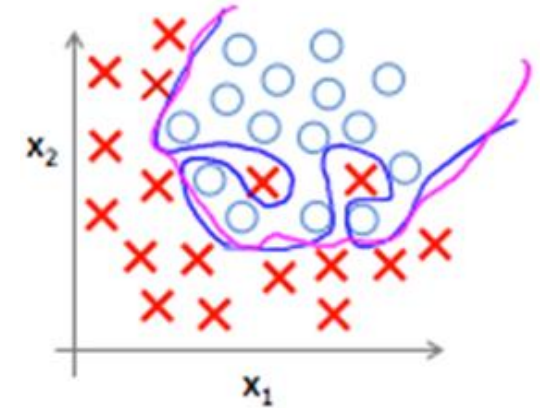
$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \dots)$$

The cost function for regularized logistic regression is given by,

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

Before regularization: Overfitting


After regularization: No overfitting




Gradient descent for regularized logistic regression is given by,

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)} \quad \text{for } j=0$$

$\frac{\partial}{\partial \theta_0} J(\theta)$  θ_0 is not penalized by the regularization parameter

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \quad \text{for } j=1, 2, \dots, n$$


$\frac{\partial}{\partial \theta_j} J(\theta)$ 

It is observed that gradient descent for regularized linear regression and regularized logistic regression is **exactly the same** but the definition of $h_{\theta}(x)$ is different.

For linear regression, $h_{\theta}(x) = \theta^T X$

For logistic regression, $h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$

When using regularized logistic regression, which of these is the best way to monitor whether gradient descent is working correctly?

- Plot $-\left[\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1-y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] \right]$ as a function of the number of iterations and make sure it's decreasing.
- Plot $-\left[\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1-y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] - \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2 \right]$ as a function of the number of iterations and make sure it's decreasing.
- Plot $-\left[\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1-y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2 \right]$ as a function of the number of iterations and make sure it's decreasing. 
- Plot $\sum_{j=1}^n \theta_j^2$ as a function of the number of iterations and make sure it's decreasing.

Advanced Optimization

Regularized linear regression can be worked using more advanced optimization methods.

function [jVal, gradient] = costFunction(theta)

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$

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

gradient(1) = [code to compute $\frac{\partial}{\partial \theta_0}$]; $\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$

gradient(2) = [code to compute $\frac{\partial}{\partial \theta_1}$]; $\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)} + \frac{\lambda}{m} \theta_1$

gradient(3) = [code to compute $\frac{\partial}{\partial \theta_2}$]; $\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)} + \frac{\lambda}{m} \theta_2$

•

•

gradient(n+1) = [code to compute $\frac{\partial}{\partial \theta_n}$];

end

If you implement this cost function and pass this into fminunc or to one of those advanced optimization techniques, that will minimize the regularized cost function J(θ)