Week 3 – Logistic Regression

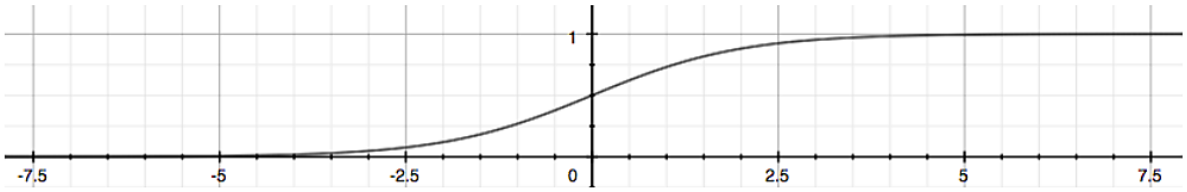
Q) Why linear regression can't be used for classification?

To attempt classification, one method is to use linear regression and map all predictions greater than 0.5 as a 1 and all less than 0.5 as a 0. However, this method doesn't work well because classification is not actually a linear function. One extra result can throw linear regression out of question.

We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given x. However, it is easy to construct examples where this method performs very poorly. Intuitively, it also doesn't make sense for $h_\theta(x)$ to take values larger than 1 or smaller than 0 when we know that y ∈ {0, 1}. To fix this, let's change the form for our hypotheses $h_\theta(x)$ to satisfy $0 \leq h_\theta(x) \leq 1$. This is accomplished by plugging $\theta^T x$ into the Logistic Function.

Our new form uses the "Sigmoid Function," also called the "Logistic Function":

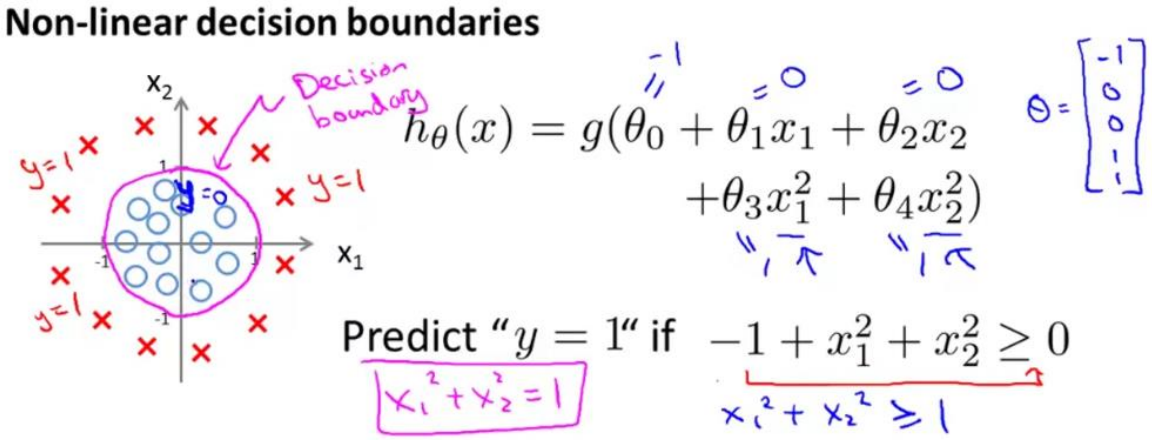
The following image shows us what the sigmoid function looks like:



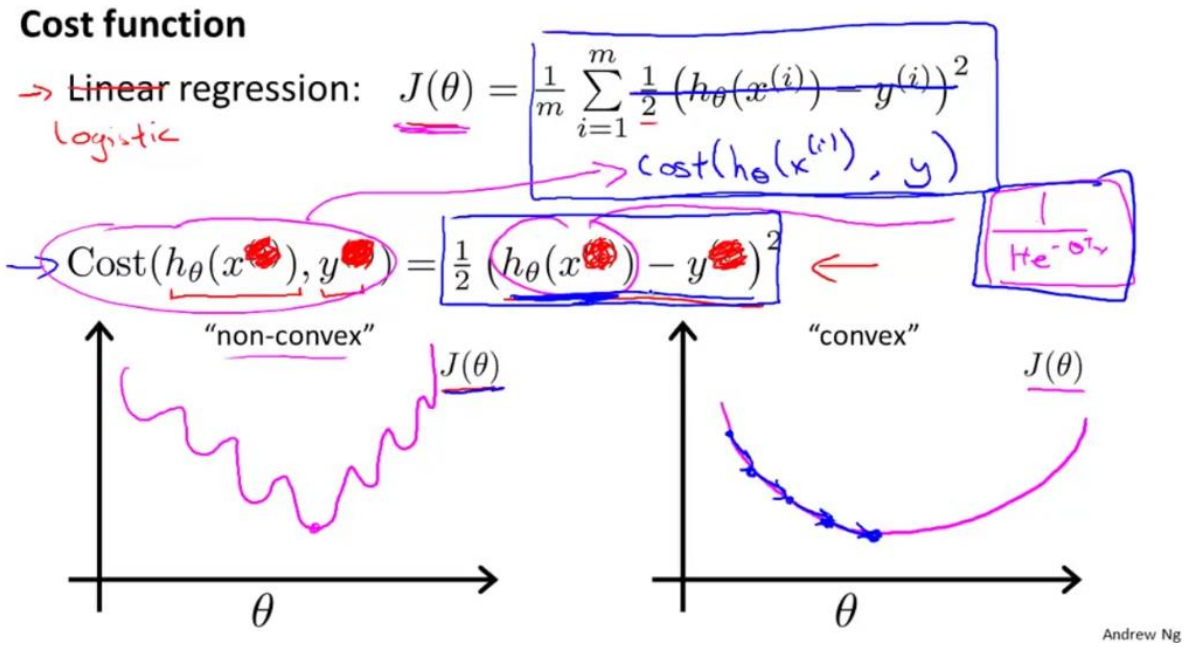
The function g(z), shown here, maps any real number to the (0, 1) interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification.

$h_\theta(x)$ will give us the **probability** that our output is 1. For example, $h_\theta(x) = 0.7$ gives us a probability of 70% that our output is 1. Our probability that our prediction is 0 is just the complement of our probability that it is 1 (e.g. if probability that it is 1 is 70%, then the probability that it is 0 is 30%).

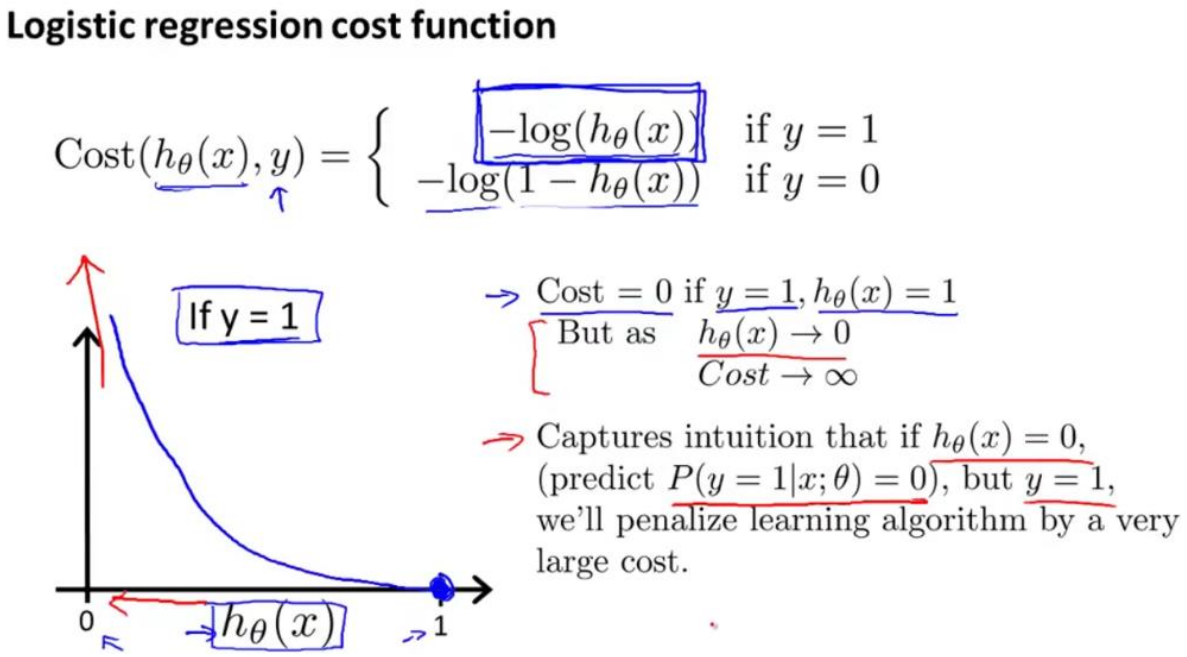
Complex/non-linear decision boundaries



Convex vs Non convex- How to apply gradient descent



Since here due to the exponentiation(sigmoid fcn) term, the function we get is non-convex, so gradient descent is not possible due to the multiple local minimums, so we need to convert the non-convex fcn to convex to be able to apply the gradient descent algorithm.



So if **h(x) predicts 1** and **y is 1**, then then cost fcn is **-log (h(x)),** which is –log (1) = 0

So if **h(x) = 0** and **y=1**, the cost fcn is **-log (h(x)),** which is –log (0) = Inf

So if **h(x) = 1** and **y=1**, the cost fcn is **-log (1-h(x)),** which is –log (1-1) = Inf

So if **h(x) = 0**, and **y=1**, the cost fcn is **-log (1-h(x)),** which is –log (0) = 0

In other words,

If our correct answer 'y' is 0, then the cost function will be 0 if our hypothesis function also outputs 0. If our hypothesis approaches 1, then the cost function will approach infinity.

If our correct answer 'y' is 1, then the cost function will be 0 if our hypothesis function outputs 1. If our hypothesis approaches 0, then the cost function will approach infinity.

Single cost function

Logistic regression cost function

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

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

Note: $y = 0$ or 1 always

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

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

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

To sum up

Logistic regression cost function

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

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

To fit parameters θ:

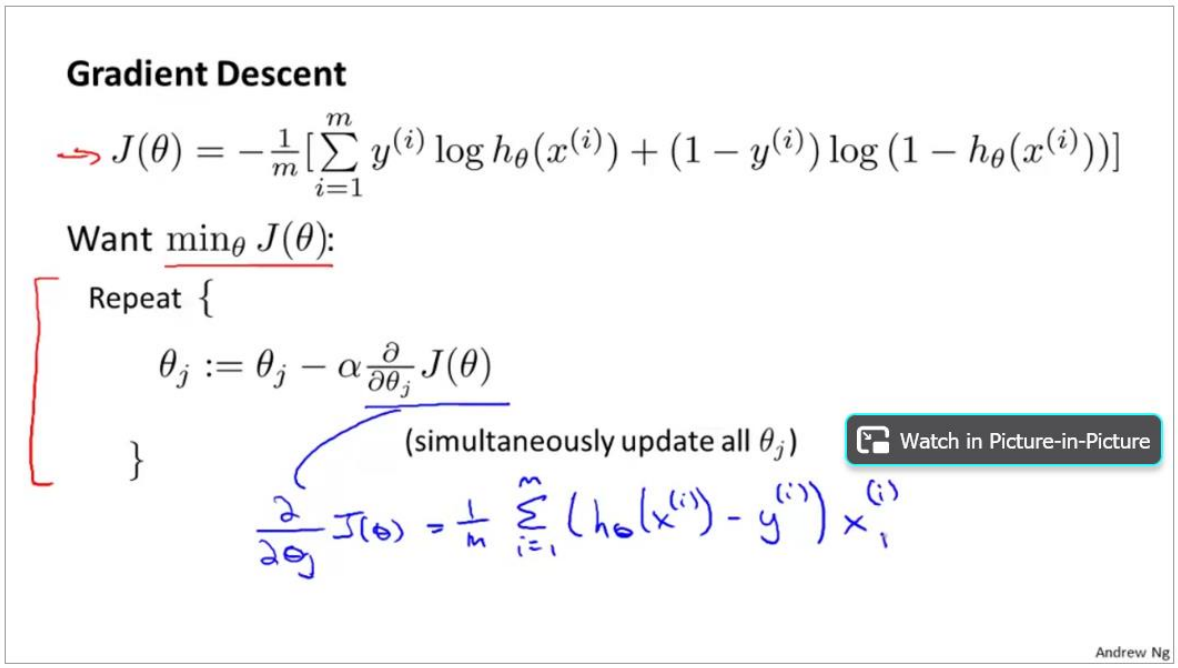
$$\min_\theta J(\theta)$$ Get θ

To make a prediction given new x:

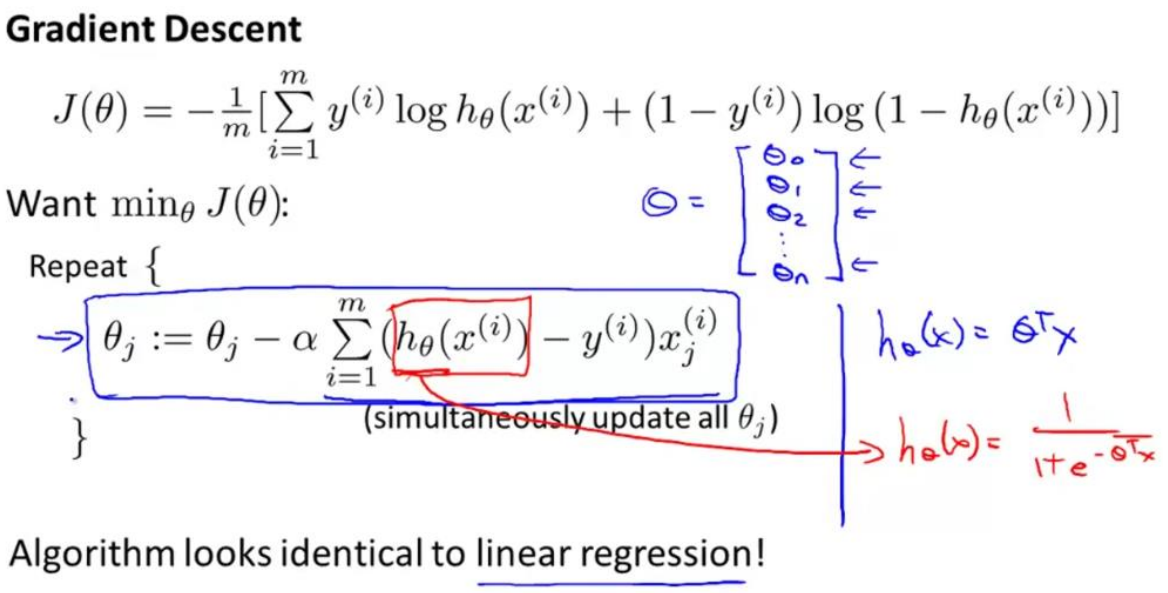
Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ $p(y=1 \mid x;\theta)$

Gradient descent for logistic regression

We have to minimize the cost function



Take partial derivative of cost fcn and we get the sample update rule as in linear regression, so we get same update rule as linear regression, but the definition of hypothesis is different.



Vectorised implementation

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

Advanced optimization

Given θ, we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial \theta_j} J(\theta)$ (for $j = 0, 1, \ldots, n$)

Optimization algorithms:

- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS

Advantages:

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

Disadvantages:

- More complex

Implementing advanced optimization on octave

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

$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$ $\theta_1 = 5, \theta_2 = 5.$

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

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

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

```
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);
```

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

General form

$\text{theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$ theta(1), theta(2), ..., theta(n+1)

```
function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute J(theta)];
    gradient(1) = [code to compute ∂/∂θ0 J(θ)];
    gradient(2) = [code to compute ∂/∂θ1 J(θ)];
    ⋮
    gradient(n+1) = [code to compute ∂/∂θn J(θ)];
```

"Conjugate gradient", "BFGS", and "L-BFGS" are more sophisticated, faster ways to optimize θ that can be used instead of gradient descent. We suggest that you should not write these more sophisticated algorithms yourself (unless you are an expert in numerical computing) but use the libraries instead, as they're already tested and highly optimized. Octave provides them.

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

\begin{align*} & J(\theta) \newline & \dfrac{\partial}{\partial \theta_j}J(\theta)\end{align*}

We can write a single function that returns both of these:

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

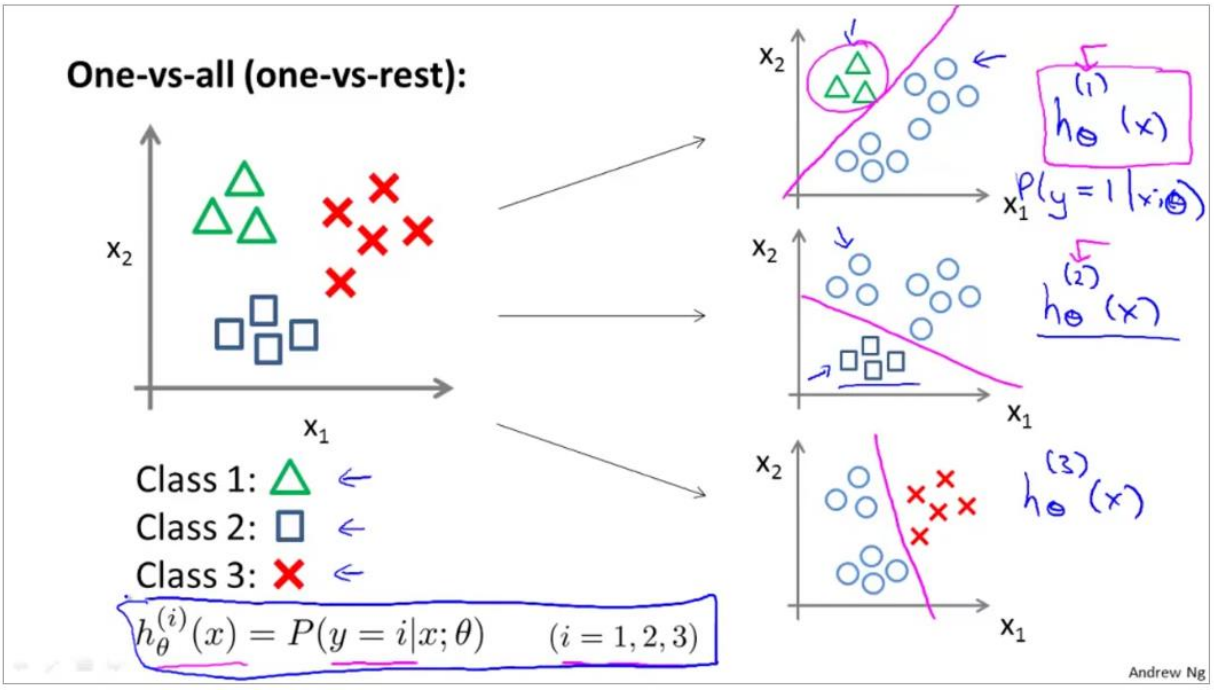
Then we can use octave's "fminunc()" optimization algorithm along with the "optimset()" function that creates an object containing the options we want to send to "fminunc()". (Note: the value for MaxIter should be an integer, not a character string - errata in the video at 7:30)

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

```

We give to the function "fminunc()" our cost function, our initial vector of theta values, and the "options" object that we created beforehand.

Multi-class classification: one vs All



1. Split into n number of classes,
2. Make ‘n’ number of classifiers for ‘n’ number of Classes.

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

$$\max_i h_\theta^{(i)}(x)$$

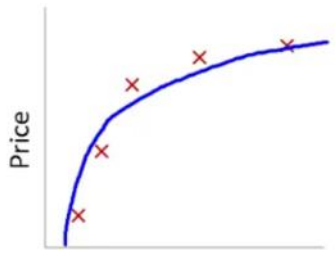
Overfitting and Regularization

Options:

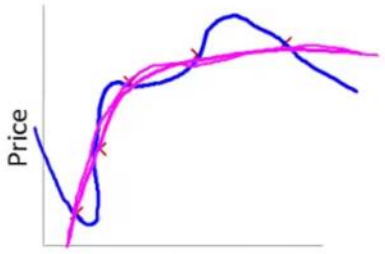
1. Reduce number of features.
   - — Manually select which features to keep.
   - — Model selection algorithm (later in course).
2. Regularization.
   - — Keep all the features, but reduce magnitude/values of parameters θ_j.
   - — Works well when we have a lot of features, each of which contributes a bit to predicting y.

Under fitting, 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. At the other extreme, 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.

Intuition



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



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

Suppose we penalize and make θ_3, θ_4 really small.

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

$\theta_3 \approx 0 \qquad \theta_4 \approx 0$

This gives:

- A simpler hypothesis
- Also less prone to overfitting.

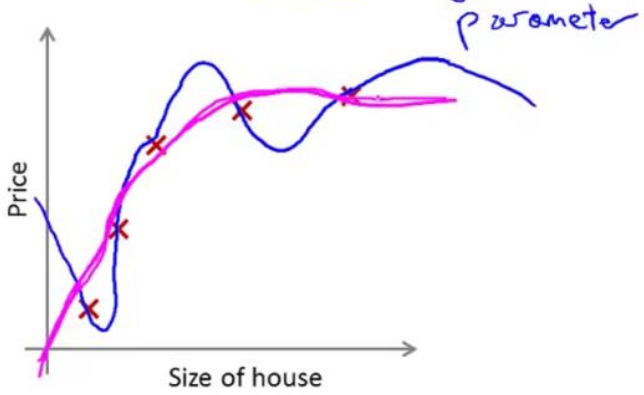
How to select which parameter we apply the penalization to:

For this we use regularization technique, we add a **regularization parameter** – 'λ', which will help controlling the under fitting/overfitting problem. It determines how much the costs of our theta parameters are inflated.

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

regularization parameter

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



What happens when Lambda is very large?

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.

Gradient descent for regularized linear regression

Repeat {

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

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j\right] \quad (j = 1, 2, 3, \ldots, n)$$

}

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

$1 - \alpha\frac{\lambda}{m} < 1$ 0.99

Normal equation for regularized linear regression

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix} \quad m \times (n+1) \qquad y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad \mathbb{R}^m$$

$$\rightarrow \min_\theta J(\theta) \qquad \frac{\partial}{\partial \theta_j} J(\theta) \overset{\text{set}}{=} 0 \ \forall j$$

$$\rightarrow \theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

$(n+1) \times (n+1)$

E.g. $n = 2$ $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Non invertibility problem solved:

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

$$\theta = (X^T X)^{-1} X^T y$$

non-invertible / singular

pinv inv

If $\lambda > 0$,

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

Recall that if m < n, then $X^T X$ is non-invertible. However, when we add the term λ·L, then $X^T X$ + λ·L becomes invertible.

Advanced optimization

fminunc (@costFunction, $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$, theta(1), theta(2), theta(n+1))

```
function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute J(θ)];
```

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

```
    gradient(1) = [code to compute ∂/∂θ0 J(θ)];
```

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

```
    gradient(2) = [code to compute ∂/∂θ1 J(θ)];
```

$$\left(\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)} \right) + \frac{\lambda}{m} \theta_1$$

```
    gradient(3) = [code to compute ∂/∂θ2 J(θ)];
```

$$\left(\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)} \right) + \frac{\lambda}{m} \theta_2$$

```
    ⋮
    gradient(n+1) = [code to compute ∂/∂θn J(θ)];
```

J(θ)

Cost function for Logistic regression with Regularization term

Recall that our cost function for logistic regression was:

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

We can regularize this equation by adding a term to the end:

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

The second sum, $\sum_{j=1}^{n}\theta_j^2$ **means to explicitly exclude** the bias term, θ_0. I.e. the θ vector is indexed from 0 to n (holding n+1 values, θ_0 through θ_n), and this sum explicitly skips θ_0, by running from 1 to n, skipping 0. Thus, when computing the equation, we should continuously update the two following equations:

Gradient descent

Repeat {

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

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j\right] \quad (j = 1, 2, 3, \ldots, n)$$

}

$\theta_1 \ldots \theta_n$

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

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