

Machine Learning: Coursera (Stanford)

Aprendizado supervisionado (Supervised Learning)

Dizemos sempre qual é a resposta correta para o algoritmo.

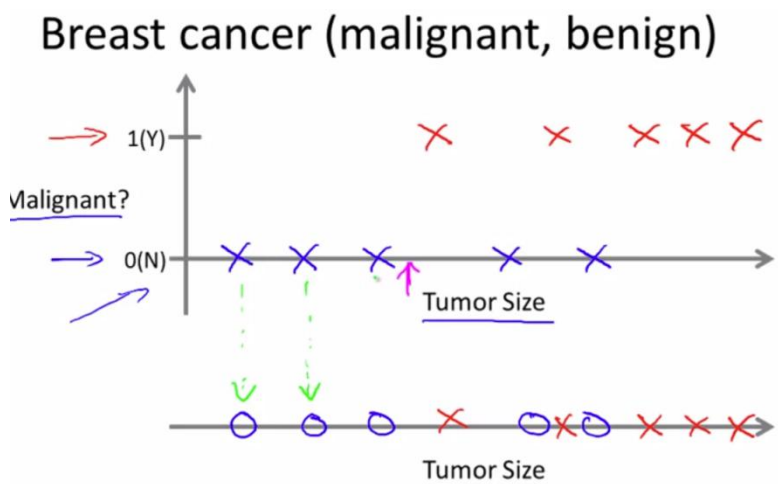
Problema de Regressão:

O objetivo é prever um valor contínuo de output (como o valor de uma casa, a partir de uma foto, descobrir qual é a idade da pessoa, etc). Quer dizer que usaremos funções contínuas para isso.

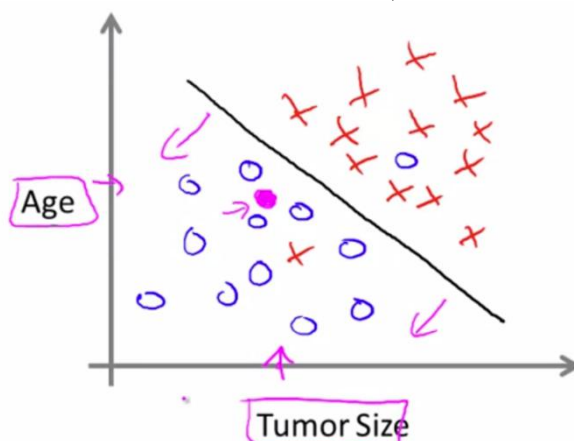
Problema de Classificação:

O objetivo é prever um valor discreto de output.

Podemos ilustrar de duas maneiras diferentes:



Podemos ter mais de um atributo (ou feature) para mensurar e classificar (podemos ter infinitas features):



Neste caso, o paciente (em roxo) tem uma maior probabilidade de ter um cancer benigno, visto a área em que se encontra.

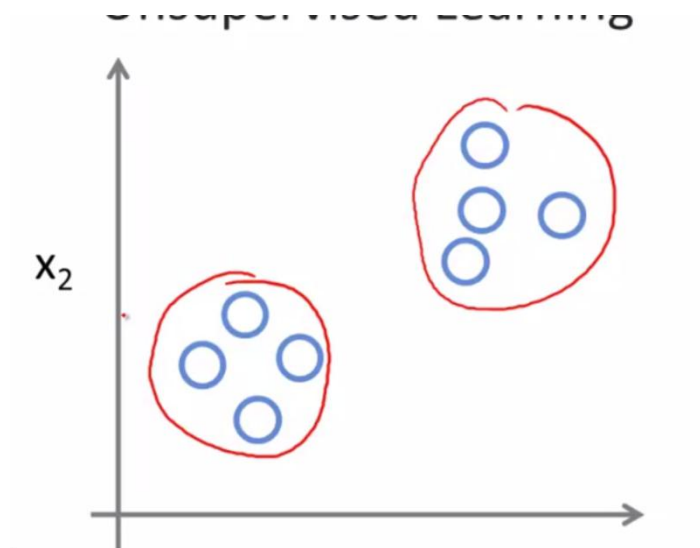
Com mais features, pode vir a pergunta: o computador não ficará sem memória ou algo assim? Existe um algoritmo chamado Support Vector Machine (SVM) que faz com que o computador consiga lidar com 'n' features.

Aprendizado sem supervisão (Unsupervised Learning)

Passamos um conjunto de dados, mas não dizemos ou não sabemos qual é a resposta correta. O algoritmo tem que prever isso.

Clustering

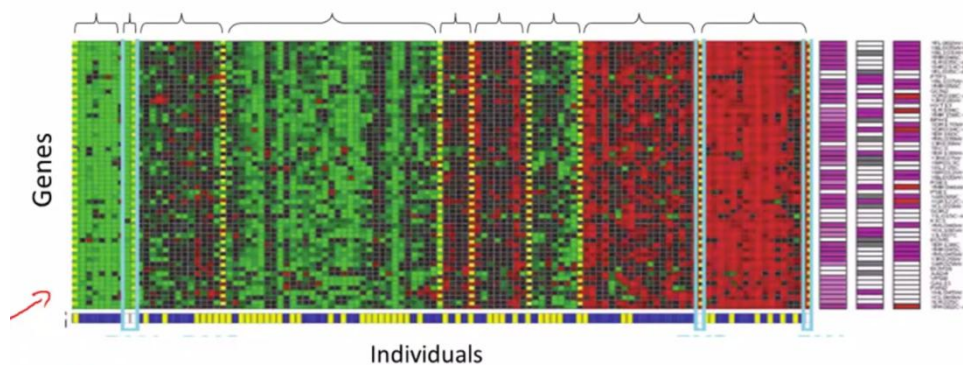
No exemplo, passamos dois clusters (conjuntos) de dados separados (usando um algoritmo de clusterização, ou seja, um algoritmo de aprendizado sem supervisão).



O google News usa esse tipo de algoritmo para agrupar notícias com o mesmo conteúdo.

Outro exemplo é com genes, onde podemos agrupar pessoas ou genes que possuem determinada característica ou não.

Este é um algoritmo sem supervisão pois não dizemos qual é o tipo de pessoa correto para o algoritmo, apenas passamos os dados e ele por si só se vira para nos dar um conjunto de agrupamento baseado no padrão de dados.



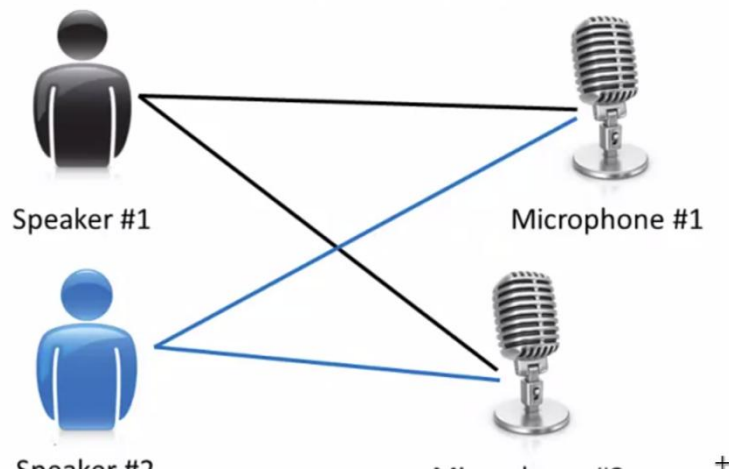
Esse tipo de clusterização é usado em dados em larga escala como Social networks, segmentação de mercado, análise de dados astronômicos, etc...

Non-clustering

Outro exemplo é o problema “Cocktail party”

Este problema consiste em pessoas falando ao mesmo tempo, logo não conseguimos entender o que cada um está falando.

Dispomos dois microfones com diferentes distâncias de duas pessoas diferentes, logo, cada um gravará um áudio com uma voz audível diferente de cada pessoa.



O algoritmo é capaz de definir uma estrutura e um padrão para conseguir separar esses áudios “com ruídos” e deixar apenas uma pessoa falando em cada output.

Este problema pode ser resolvido em uma linha de código:

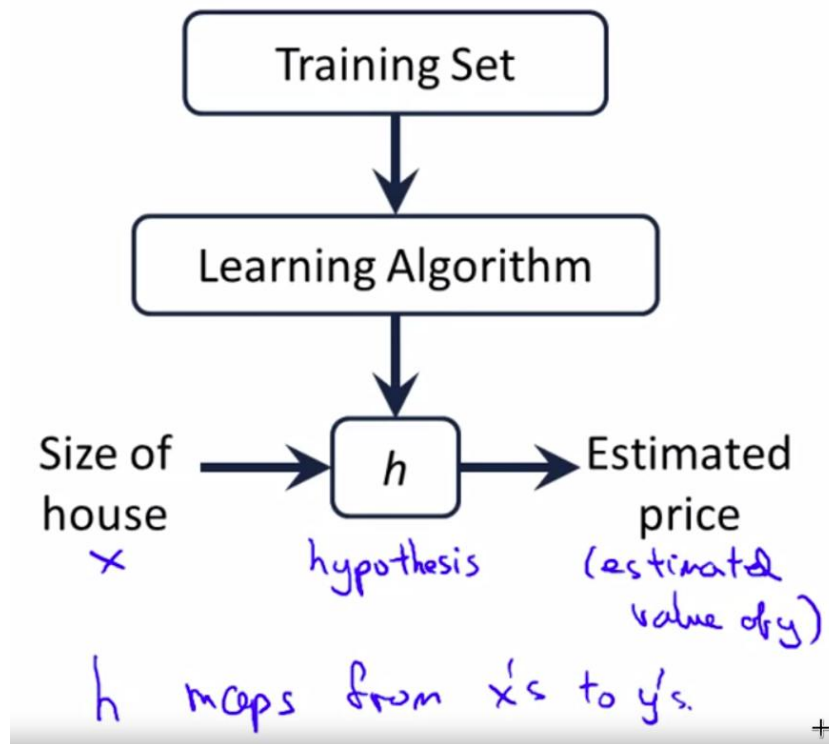
```
[W,s,v] = svd(( repmat(sum(x.*x,1),size(x,1),1).*x)*x');
```

Usando o Octave ou o Matlab.

Já existem funções como o svd (single value decomposition), da Algebra Linear. Pode-se fazer em Java, C++, Python, porém é muito mais trabalhoso. Por isso, será usado o Octave como a aplicação de protótipos. O que acontece é que muitas empresas usam o Octave para fazer um protótipo e depois migram isso para outra linguagem (Java, C++, etc).

Model and cost Function

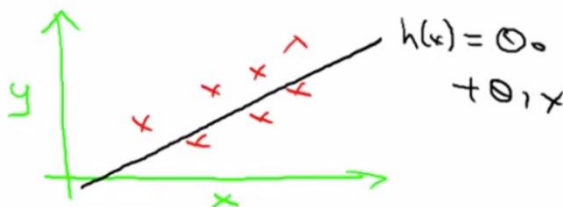
Num treinamento **supervisionado** usamos um conjunto de dados de treino (training set) para ser usado em um algoritmo de aprendizado (learning algorithm) e, por sua vez, esse algoritmo nos “cospe” uma função h (hipótese, nomeado por convenção), baseado nos dados de entrada e saída que demos para treino.



Primeiramente começaremos utilizando uma regressão linear com uma variável, logo, nossa função h_{θ} seria da forma:

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

Shorthand: $h(x)$



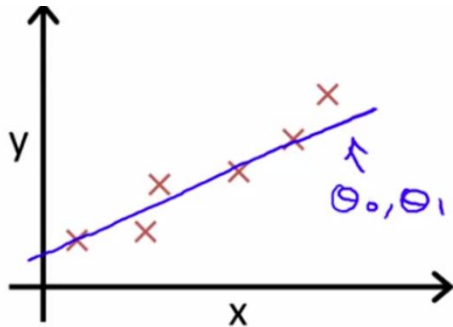
Linear regression with one variable.

Univariate linear regression. +

E, o nome desse modelo é **regressão linear com uma variável** ou **regressão linear univariável**, no caso, a variável x .

Cost Function

A ideia é escolher valores de θ_1 e θ_0 a fim de que nossa função seja algo próximo dos valores de output y .



Idea: Choose θ_0, θ_1 so that
 $h_{\theta}(x)$ is close to y for our
training examples (x, y)

Para isso, precisamos **minimizar** a função para que a diferença $h(x) - y$ seja a menor possível, ou seja, mais próxima de 0. No caso, precisamos minimizar o **quadrado dessa diferença**: $(h(x) - y)^2$, porém queremos fazer isso para todos os i -ésimos dados.

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

A divisão por $2m$ é uma convenção utilizada para computação do **gradient descent**, o termo derivativo da função quadrática irá cancelar com o $1/2$.

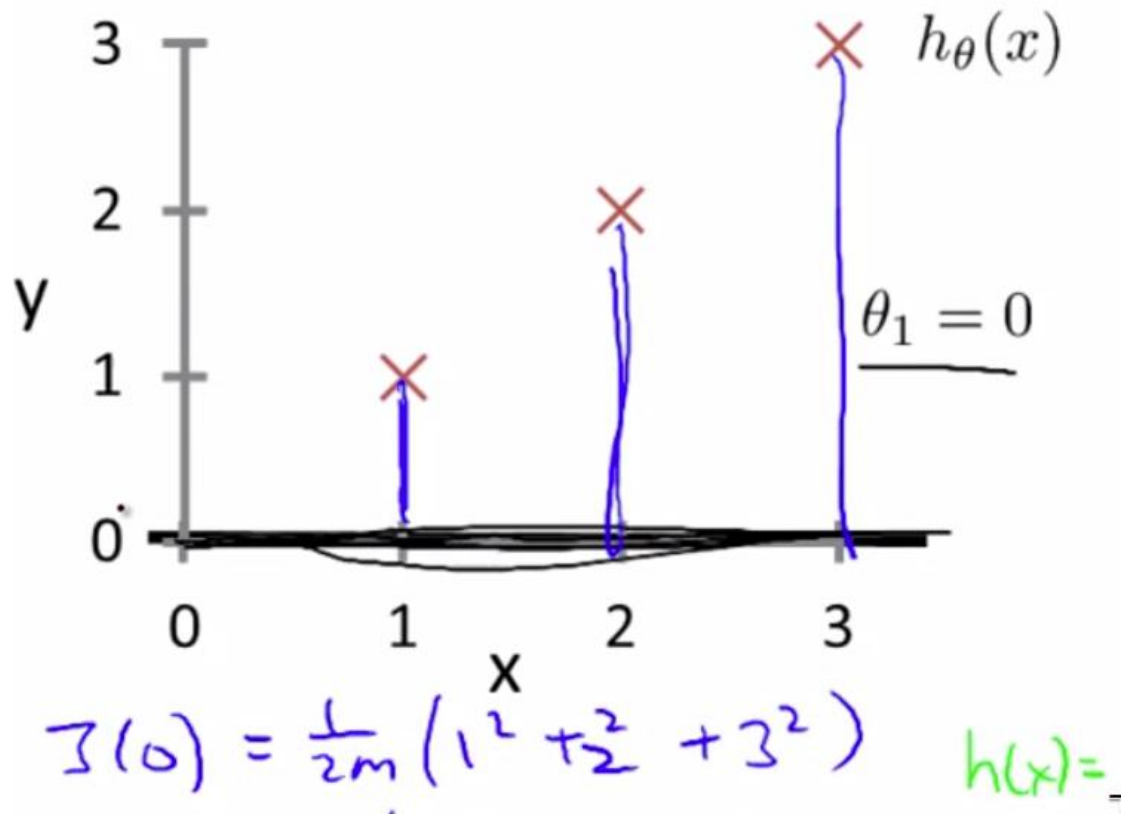
Recapitulando, minimizaremos a função h , usando esse somatório (que é o quadrado da soma dos erros a partir da previsão do conjunto de dados, menos o valor atual das casas). Logo, minimizar o J (Squared error Function ou Squared cost Function).

Intuição da função de custo

Primeiramente, passamos $\theta_0 = 0$ apenas para ficarmos com θ_1 como variável, com isso, conseguimos fazer o cálculo para um dado conjunto de dados de treino

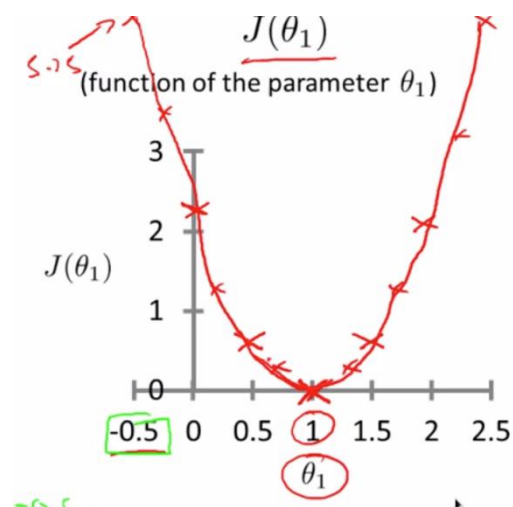
$$h_{\theta}(x)$$

(for fixed θ_1 , this is a function of x)



E a partir disso, conseguimos calcular a função de custo para 'n' valores de θ_1 , apenas substituindo os valores na função de custo J .

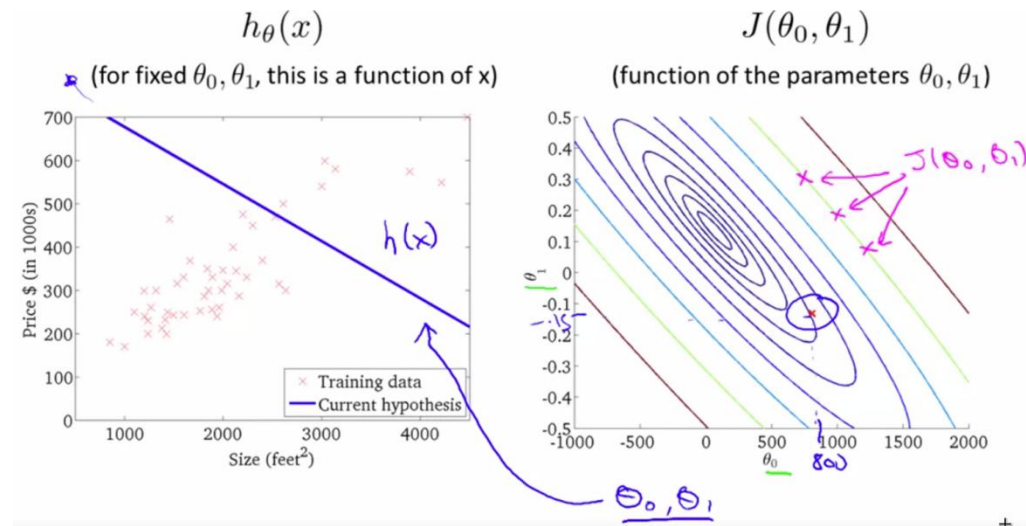
Conseguimos o seguinte gráfico dos valores de θ_1 em função de J .



Logo, a partir deste gráfico, sabemos que a melhor minimização (otimização) é quando $\theta_1 = 1$ (mínimo global). Apenas para simplificar e entender a função de custo, o θ_0 foi adotado como 0.

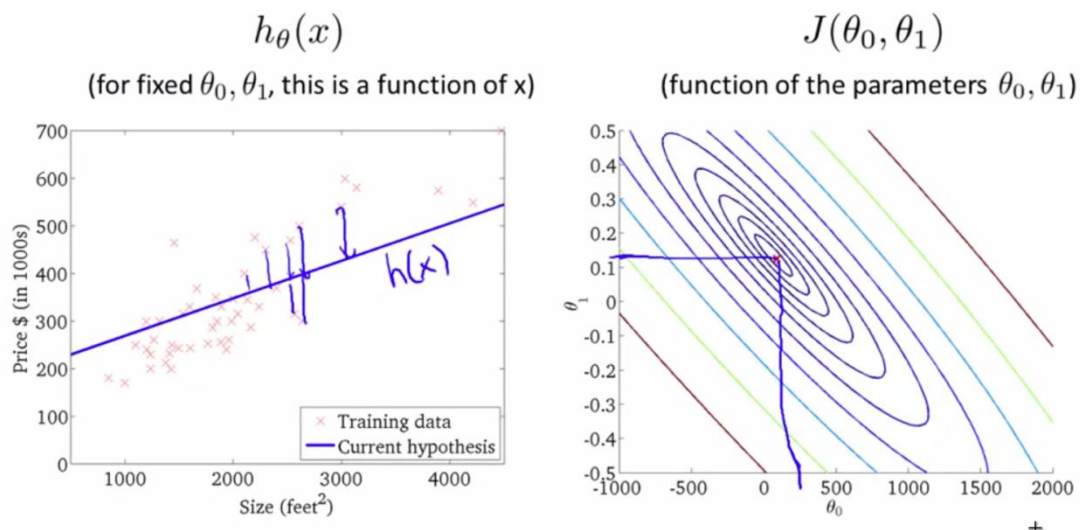
Cost function - Intuition II

Contour Plots -> Curvas de nível para 'n' Dimensões, com 'n' maior que 2.



Em magenta, para diversos valores de θ_0 e θ_1 , o valor de $J(\theta_0, \theta_1)$ é o mesmo.

Pegando um ponto mais próximo do centro, chegamos perto do mínimo (a soma do quadrado das distâncias entre os exemplos de treinamento e a hipótese (soma do quadrado dos erros)).



Gradient Descent

Método usado para encontrar a função de custo mínima. Usado amplamente em ML, não apenas em regressão linear.

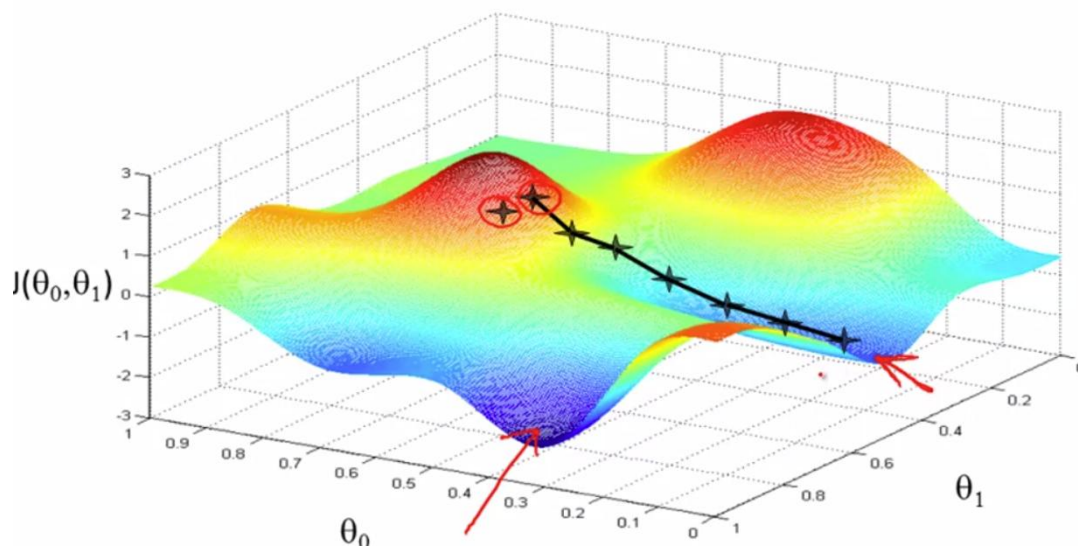
Ele é usado para resolver problemas mais amplos do que apenas uma regressão linear de 2 variáveis, pode-se utilizar para 'n' variáveis, por exemplo.

Funciona da seguinte maneira:

1. Primeiramente, definimos estimativas iniciais para θ_0 e θ_1 (uma escolha comum é começar com 0 nos parâmetros);
2. Trocamos os valores das variáveis (θ_0 e θ_1) a fim de diminuir o valor da função de custo J .

A partir de um ponto inicial, o algoritmo verifica todas as possibilidades da próxima iteração e escolhe a que faz a função de custo ter um resultado menor. Faz isso 'n' vezes até que nenhuma outra possibilidade exista.

(Pergunta: Caso caia em um mínimo local?!). Irá ser abordado posteriormente!



Algoritmo:

: = quer dizer atribuição de uma variável.

= é uma assertiva, $a = b$, estamos afirmando que o valor de 'a' é igual ao valor de 'b'.

Gradient descent algorithm

```
repeat until convergence {  
→  $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$  (for  $j = 0$  and  $j = 1$ )  
}
```

Correct: Simultaneous update

$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

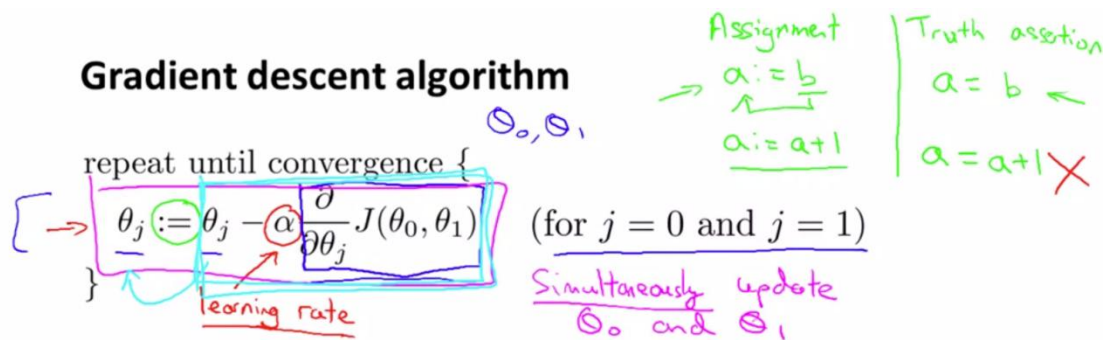
$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

$\theta_0 := \text{temp0}$

$\theta_1 := \text{temp1}$

denotar atribuição, este é

Gradient descent algorithm



Correct: Simultaneous update

$\rightarrow \text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
 $\rightarrow \text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
 $\rightarrow \theta_0 := \text{temp0}$
 $\rightarrow \theta_1 := \text{temp1}$

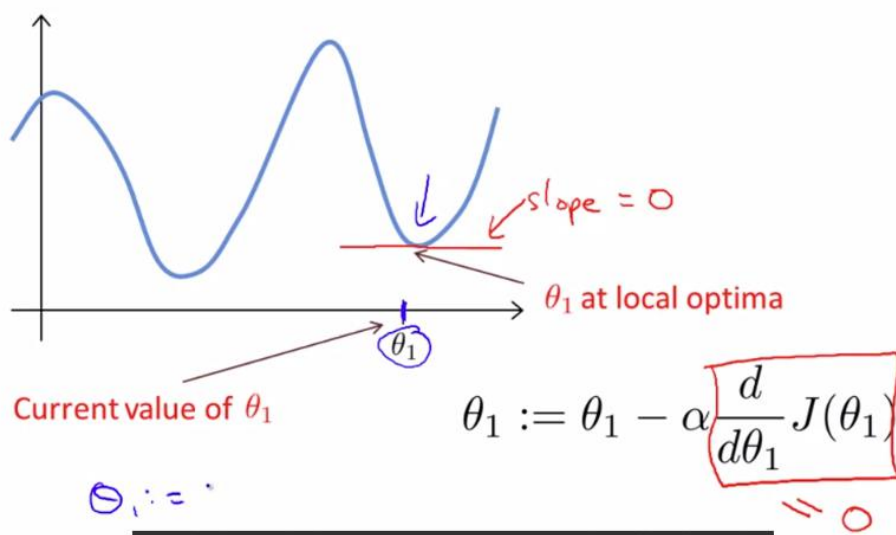
Incorrect:

$\rightarrow \text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
 $\rightarrow \theta_0 := \text{temp0}$
 $\rightarrow \text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
 $\rightarrow \theta_1 := \text{temp1}$

O alpha é o 'passo' que daremos ao algoritmo, o quanto ele vai 'andar'. Passos grandes permitem iterações menores, porém, não é tão granular quanto passos pequenos.

Gradient Descent Intuition

* A derivada no mínimo é zero!!! (A inclinação da tangente é zero, é uma constante)



Logo, é por isso que o gradient descent, com um alpha fixo, acaba chegando no mínimo local, pois à medida em que a função vai sendo iterada, a derivada vai diminuindo, (posteriormente, se tornará zero), com isso, o valor multiplicado por alpha vai diminuindo e o seu passo, consequentemente, também.

Termos das derivadas parciais resolvidos:

Gradient descent algorithm

$$\begin{aligned} &\text{repeat until convergence } \{ \\ &\quad \theta_0 := \theta_0 - \alpha \left(\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \right) \\ &\quad \theta_1 := \theta_1 - \alpha \left(\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)} \right) \\ &\} \end{aligned}$$

$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

Termo 'batch': quando acaba-se todos os conjuntos de teste (ou seja, quando chega-se ao 'm' do somatório). "Batch Gradient Descent": Quando acabou de rodar todos os conjuntos de treinamento do Gradient Descent. Existem versões do Gradient Descent que não são "batches", nós não olhamos para o conjunto completo e sim para pequenos subconjuntos de treinamento.

Instalando o Octave

No linux, basta: `$sudo apt install octave;`

Normalização média

Para ajudar no Gradient Descent, aplicamos a normalização média dentro de cada feature dos dados de teste.

(Não aplicamos sobre x_0)

E.g. $x_1 = \frac{\text{size} - 1000}{2000}$ Average size = 1000
 $x_2 = \frac{\# \text{bedrooms} - 2}{5}$ 1-5 bedrooms
 $-0.5 \leq x_1 \leq 0.5$ $-0.5 \leq x_2 \leq 0.5$
 $x_1 \leftarrow \frac{x_1 - \mu_1}{S_1}$ $x_2 \leftarrow \frac{x_2 - \mu_1}{S_2}$
Annotations: μ_1 is the avg value of x_1 in training set. S_1 is the range (max-min) or standard deviation.

Onde S_j ($1 \leq j \leq n$) é o desvio padrão, ou seja (o valor máximo menos o valor mínimo).

Para “debugarmos” o Gradient Descent, devemos ver a função $J(\theta)$ x número de iterações feitas.

Caso esteja fazendo uma assíntota tendendo à zero, está ok! O número de iterações pode mudar bastante, dependendo do conjunto e do algoritmo. O interessante é dizer uma margem para o algoritmo, onde quando chegar nesse valor (épsilon), declaramos convergido, algo como 10^{-3} .

Podemos usar funções polinomiais (**regressão polinomial**) para que a função de hipótese se encaixe melhor nos dados propostos. (E.g: temos a largura e a profundidade de um terreno, podemos multiplicar os dois e fazer uma nova feature, onde contemple as duas features antigas, e a partir disso, usar um modelo quadrático, cúbico, etc...);

Dúvidas: `sum(vetor x vetor - vetor).^2`

Primeiramente, o octave faz o que está dentro dos parênteses para depois fazer o somatório! O somatório soma todos os valores do vetor, ou seja, sai um **escalar**.

O J_cost é apenas para ver se o valor de J está diminuindo, para validar o modelo!!! Não é usado no gradient descent.

A normalização das features é usado somente em múltiplas features!

LOGISTIC REGRESSION

A regressão linear **não** é utilizada para problemas de classificação, pois um dado a mais já pode acabar com o algoritmo. Ele também não expressa muito bem o modelo, pois $h(\theta)$ deveria estar entre 0 e 1. A **classificação não é uma função linear**.

Logo, para problemas de classificação usaremos a **logistic regression**.

O **decision boundary** é definido pelo vetor θ ! (Não o training set. O training set é usado para ajustar os parâmetros θ);

* Usar uma função sigmoide indica que, teremos uma função custo $J(\theta)$ não convexa, isso quer dizer que se usarmos um algoritmo tipo o gradient descent, não podemos garantir que ele convergirá para o mínimo global.

* Para isso, transformamos a função de custo em uma logarítmica.

Caso o algoritmo (com esse ajuste da função logarítmica) tentar prever algo que não é verdade (ou seja $P(y=1|x;\theta) = 0$), e o y é realmente 1, penalizaremos com um custo tendendo ao infinito (assintota vertical tendendo a 0).

(Da mesma forma o inverso $\rightarrow P(y=0|x;\theta) = 1$ é algo “impossível” de acontecer logo, penalizaremos com um custo alto, com uma assintota vertical tendendo a 1).

$H(\theta) = y$, logo $\text{cost}(h(\theta), y) = 0$;

Após sintetizar a função cost... Ela deriva de um principio estatístico (Maximum likelihood estimation), que, eficientemente, acha parâmetros (θ) para diferentes modelos de dados e tem uma propriedade de convex.

A função para atualizarmos o valor de θ é a mesma da regressão linear, porém, a função de hipótese ($h(\theta)$) nesse modelo é diferente!

****Função:**

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

(simultaneously update all θ_j)

Para calcularmos e vermos se o algoritmo está funcionando, o valor de $J(\theta)$ tem que estar diminuindo, neste caso,

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

Para motivos de comparação:

We can fully write out our entire cost function as follows:

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

A vectorized implementation is:

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

Remember that the general form of gradient descent is:

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

We can work out the derivative part using calculus to get:

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

Notice that this algorithm is identical to the one we used in linear regression. We still have to simultaneously update all values in theta.

A vectorized Implementation is:

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



TODO: Passar para a forma vetorizada na mao! (E fazer a derivada) [ok]

G = função sigmoide ($1/(1+e^{-(\theta^T x)})$);

*VER > Line search algorithm (Conjugate gradient, BFGS, L-BFGS); Funcao **fminunc** no octave;
(para thetas com dimensao >= 2);

A unica coisa que precisamos para estes algoritmos é passar uma função que calcule o J(theta)
e o resultado de suas derivadas em um vetor;

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);
```

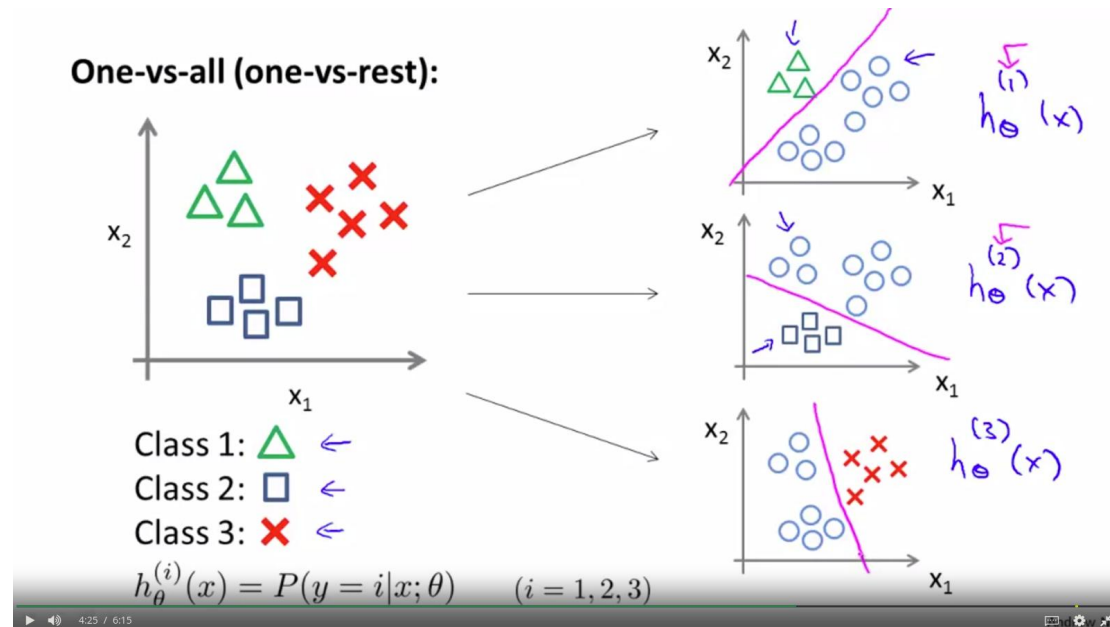
```
> options = optimset('GradObj', 'on', 'MaxIter', '100');  
> initialTheta = zeros(2,1);  
[optTheta, functionVal, exitFlag] ...  
    = fminunc(@costFunction, initialTheta, options);
```

com um Gradiente Descendente "turbinado",

* O numero de iterações tem que ser Number e não String!!

Regressão logística multiclasse

Para problemas com poucas classes (um n° finito e conhecido) devemos deixar uma classe em específico constante e usar as outras como hipótese negativa, com isso, teremos um array de hipóteses ($h(\theta)$);

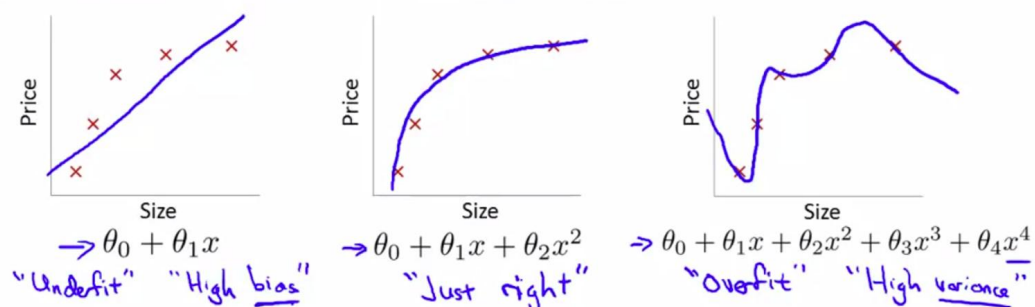


Para prever um teste, basta pegarmos o valor máximo da função de hipótese após rodar em todas as classes. (Isso indica a maior probabilidade).

*Overfitting: quer dizer que o modelo selecionado funciona bem para os dados de treinamento, mas para teste, ele não funciona tão bem, justamente por conta do modelo, que pode encaixar em qualquer situação

Exemplo em regressão linear:

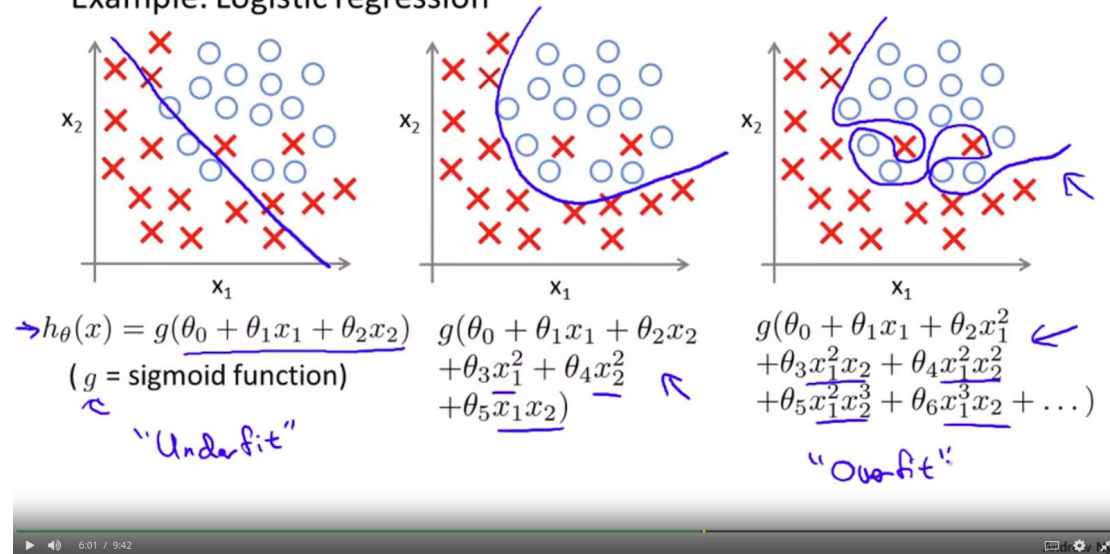
Example: Linear regression (housing prices)



Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

Exemplo em regressão logística:

Example: Logistic regression



O **overfitting** também pode acontecer quando temos um conjunto de variáveis maior que o conjunto de testes.

* Podemos transformar uma matrix $X^T X$ não-invertível em uma invertível usando o termo de regularização, $(\lambda * L)$ onde L é uma matriz apenas com a diagonal com 1 (com o primeiro termo $a_{00} = 0$)