Machine Learning: Coursera (Stanford)

Aprendizado supervisionado (Supervised Learning)

Dizemos sempre qual é a resposta correta para o algoritmo.

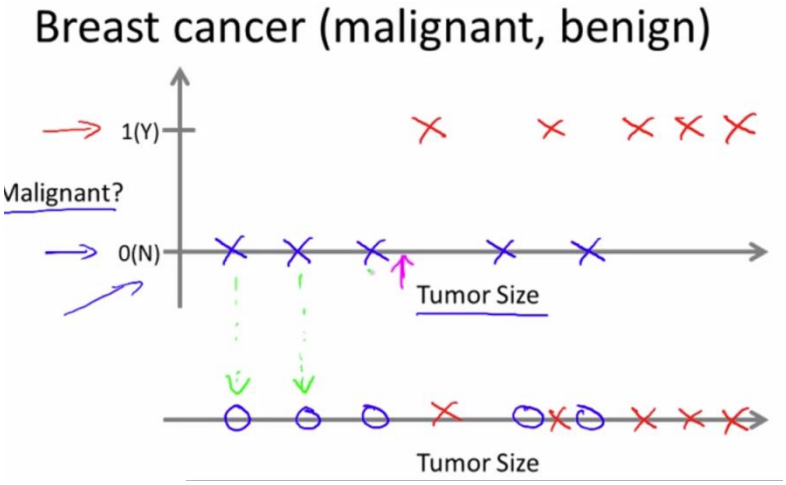
Problema de Regressão:

O objetivo é predizer 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.

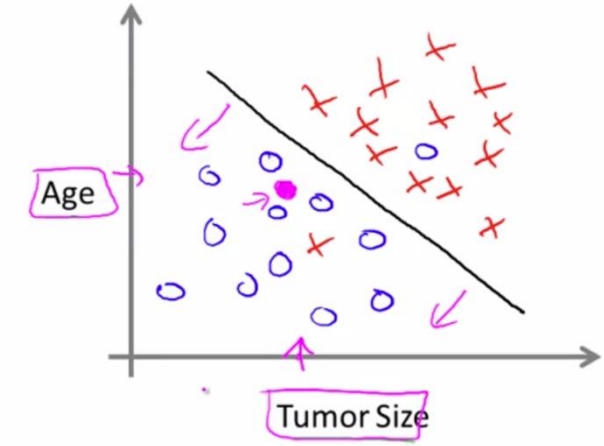
Problema de Classificação:

O objetivo é predizer 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.

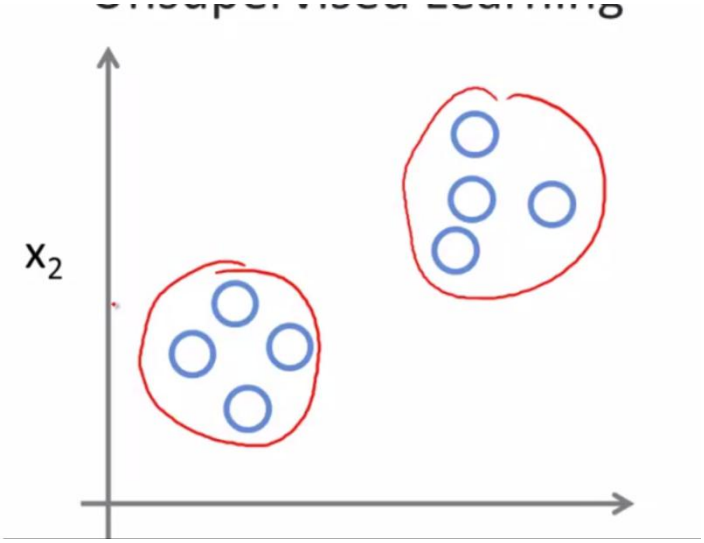
Com mais features, pode vir a pergunta: o computador não ficará sem memória ou algo assim? Existe um algoritmo chamado Support Vector Machine (SVM) que faz com que o computador consiga lidar com ‘n’ features.

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 predizer isso.

Clustering

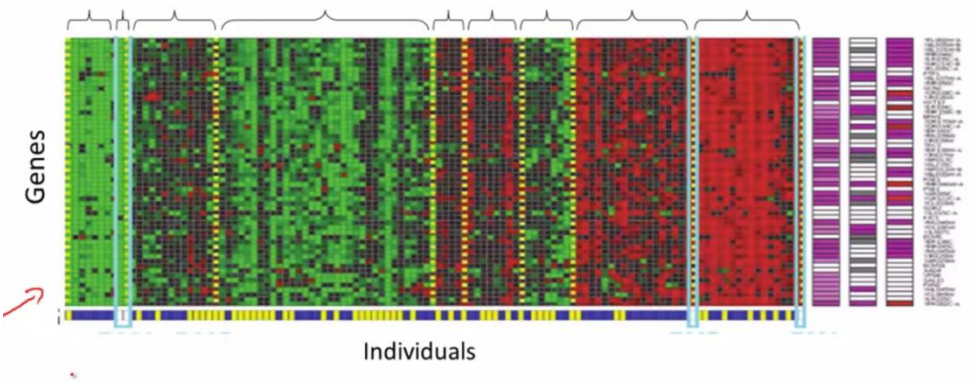
No exemplo, passamos dois cluters (conjuntos) de dados separados (usando um algoritmo de clusterização, ou seja, um algoritmo de aprendizado sem supervisão).



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 algortimo, apenas passamos os dados e ele por si só se vira para nos dar um conjunto de agrupamento baseado no padrão de dados.



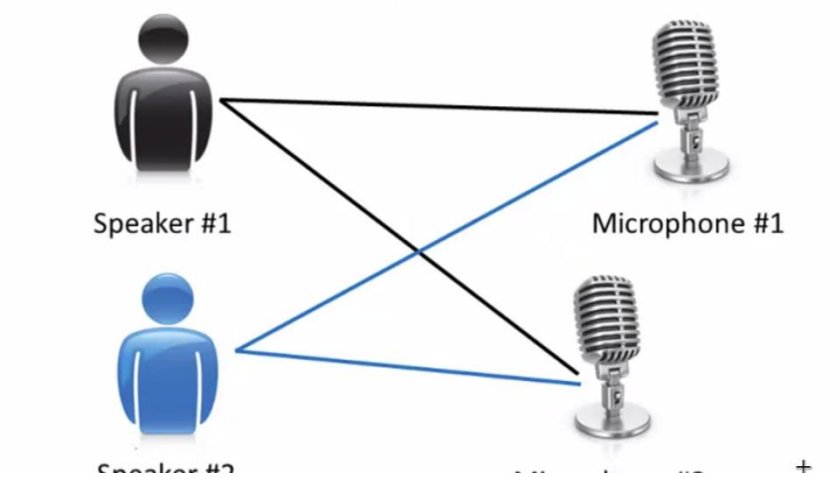
Esse tipo de clusterização é usado em dados em larga escala como Social networks, segmentação de mercado, análise de dados astronômicos, etc...

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:

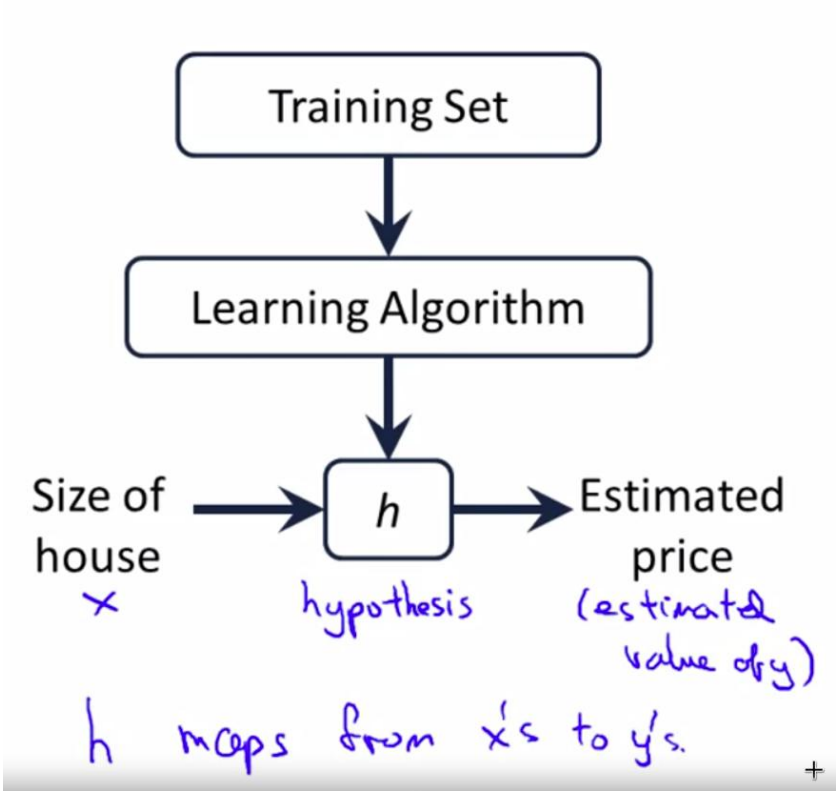
```
[W,s,v] = svd((repmat(sum(x.*x,1),size(x,1),1).*x)*x');
```

Usando o Octave ou o Matlab.

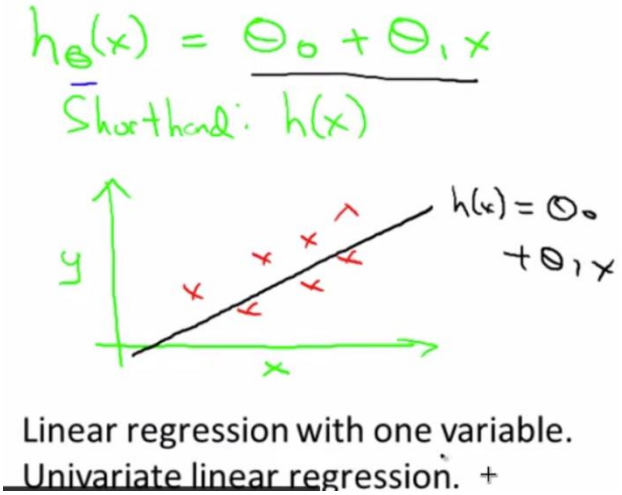
Já existem funções como o svd (single value decomposition), da Algebra Linear. Pode-se fazer em Java, C++, Python, porém é muito mais trabalhoso. Por isso, será usado o Octave como a aplicação de protótipos. O que acontece é que muitas empresas usam o Octave para fazer um protótipo e depois migram isso 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 algoithm) e, por sua vez, esse algoritmo nos "cospe" uma função h (hipotese, nomeado por convenção), baseado nos dados de entrada e saída que demos para treino.



Primeiramente começaremos utilizando uma regressão linear com uma varíavel, logo, nossa função h_theta seria da forma:

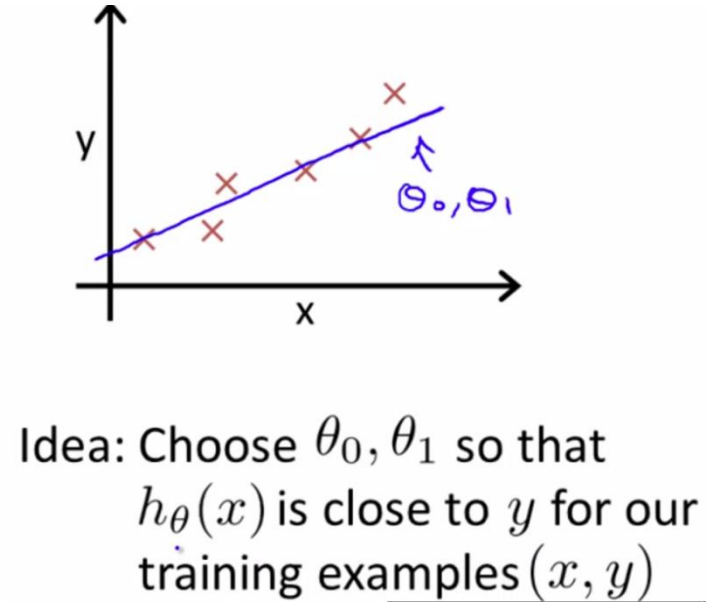


Linear regression with one variable.
Univariate linear regression.

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 theta_1 e theta_0 a fim de que nossa função seja algo próximo dos valores de output 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.

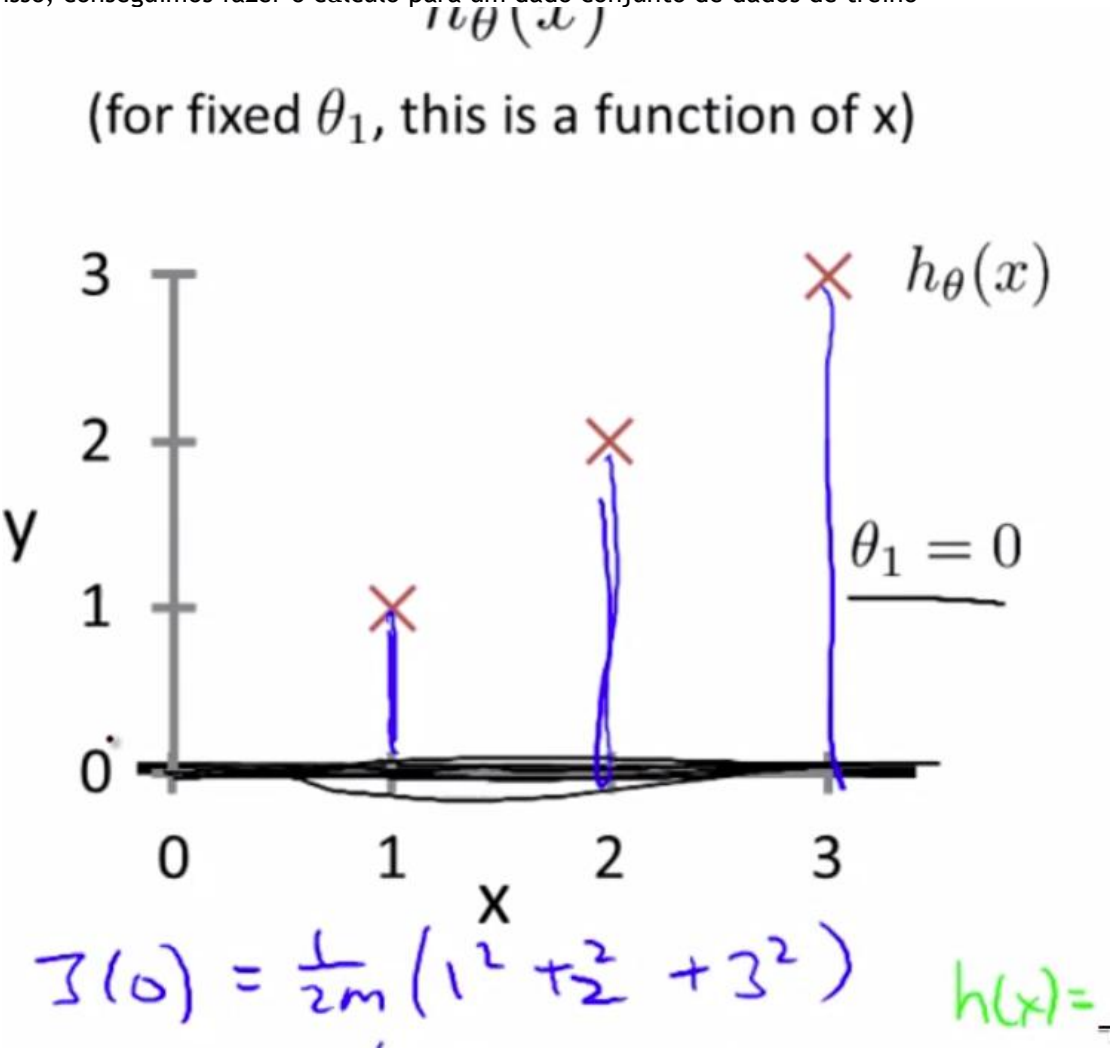
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

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 predição do conjunto de dados, menos o valor atual das casas). Logo, minimizar o J (Squared error Function ou Squared cost Function).

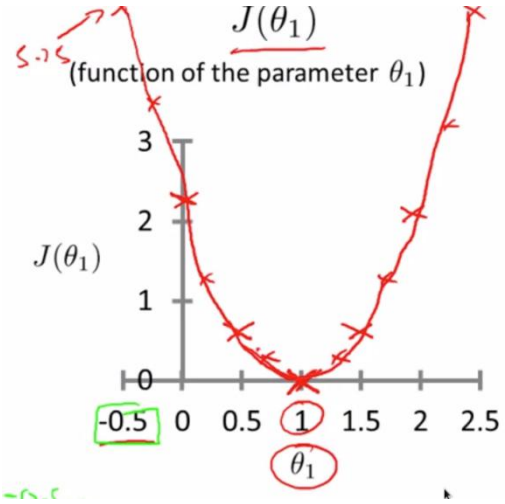
Intuição da função de custo

Primeiramente, passamos theta_0 = 0 apenas para ficarmos com theta_1 como variável, com isso, conseguimos fazer o cálculo para um dado conjunto de dados de treino



E a partir disso, conseguimos calcular a função de custo para ‘n’ valores de theta_1, apenas substituindo os valores na função de custo J.

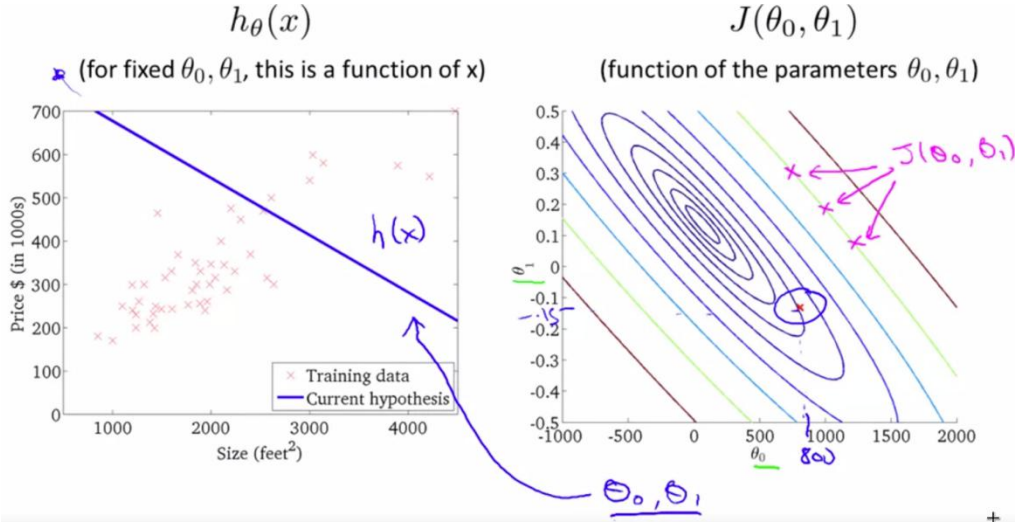
Conseguimos o seguinte gráfico dos valores de theta_1 em função de J.



Logo, a partir deste gráfico, sabemos que a melhor minimização (otimização) é quando theta_1 = 1 (minimo global). Apenas para simplificar e entender a função de custo, o theta_0 foi adotado como 0.

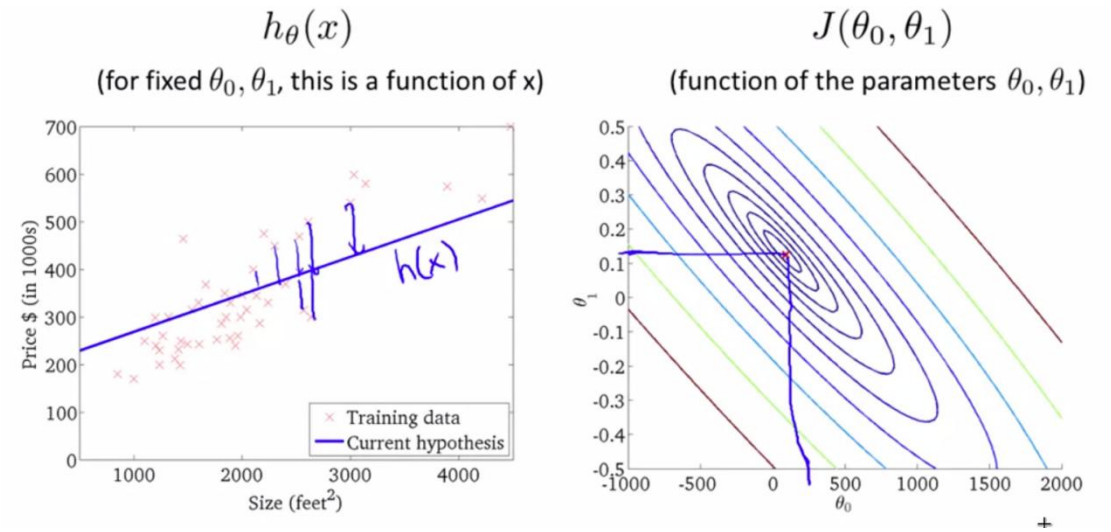
Cost function - Intuition II

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



Em magenta, para diversos valores de theta_0 e theta_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

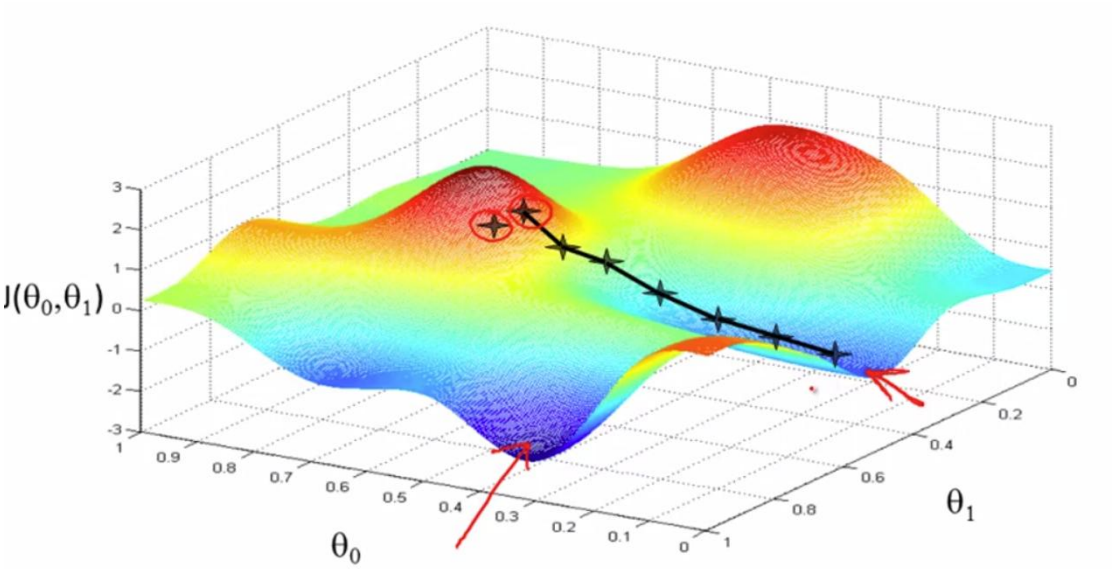
Método usado para encontrar a função de custo mínima. Usado amplamente em ML, não apenas em regressão linear.
Ele é usado para resolver problemas mais amplos do que apenas uma regressão linear de 2 variáveis, pode-se utilizar para 'n' variáveis, por exemplo.

Funciona da seguinte maneira:

1. Primeiramente, definimos estimativas iniciais para theta_0 e theta_1 (uma escolha comum é começar com 0 nos parâmetros);
2. Trocamos os valores das variáveis (tetha_0 e tetha_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

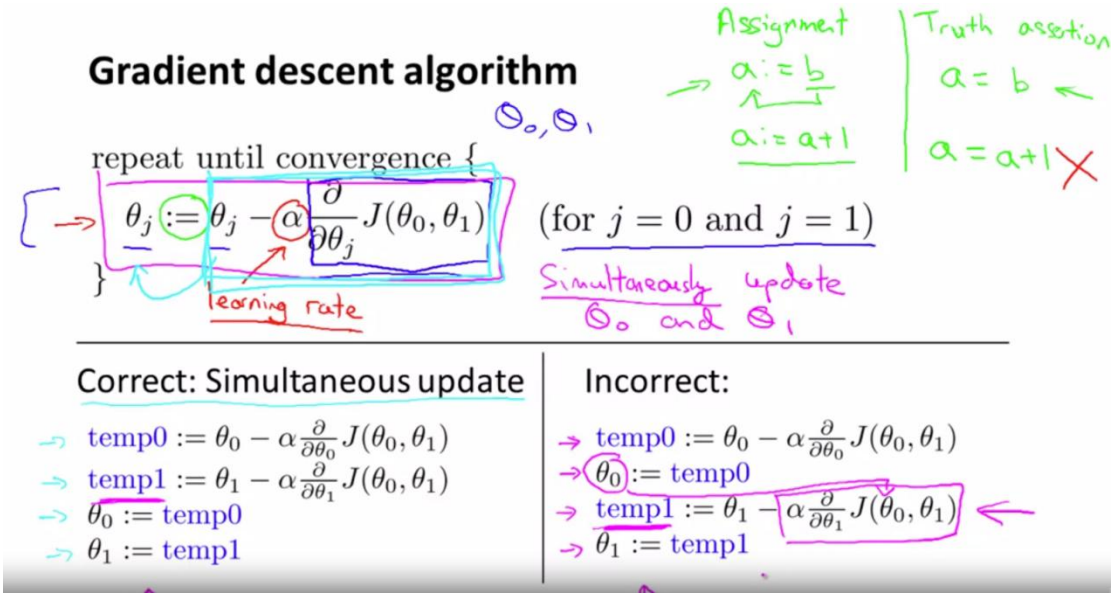
$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

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

$\theta_0 := \text{temp0}$

$\theta_1 := \text{temp1}$

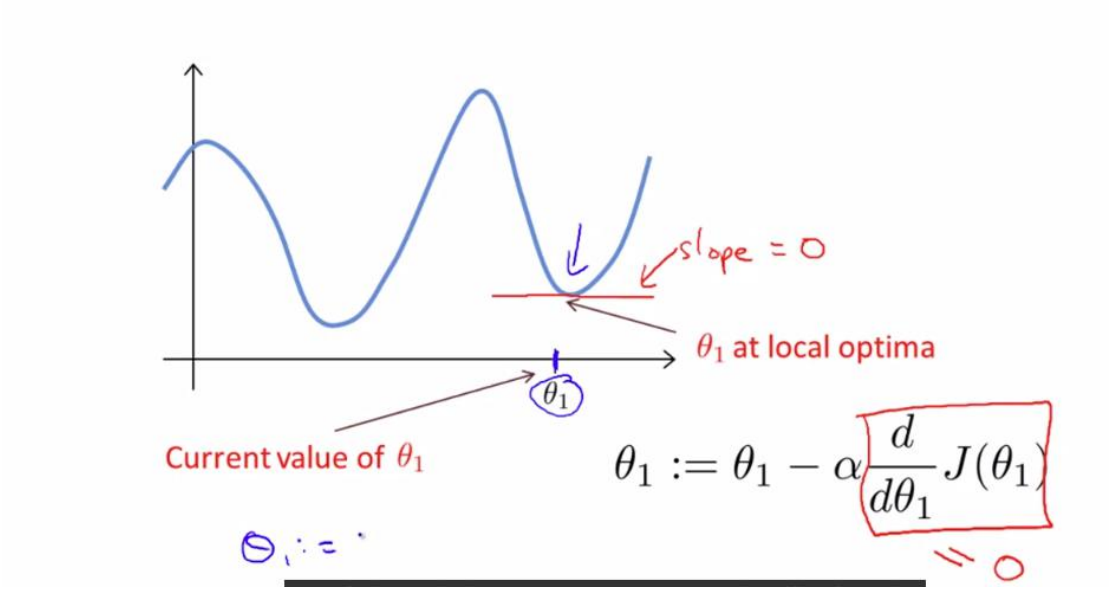
denotar atribuição, este é



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.

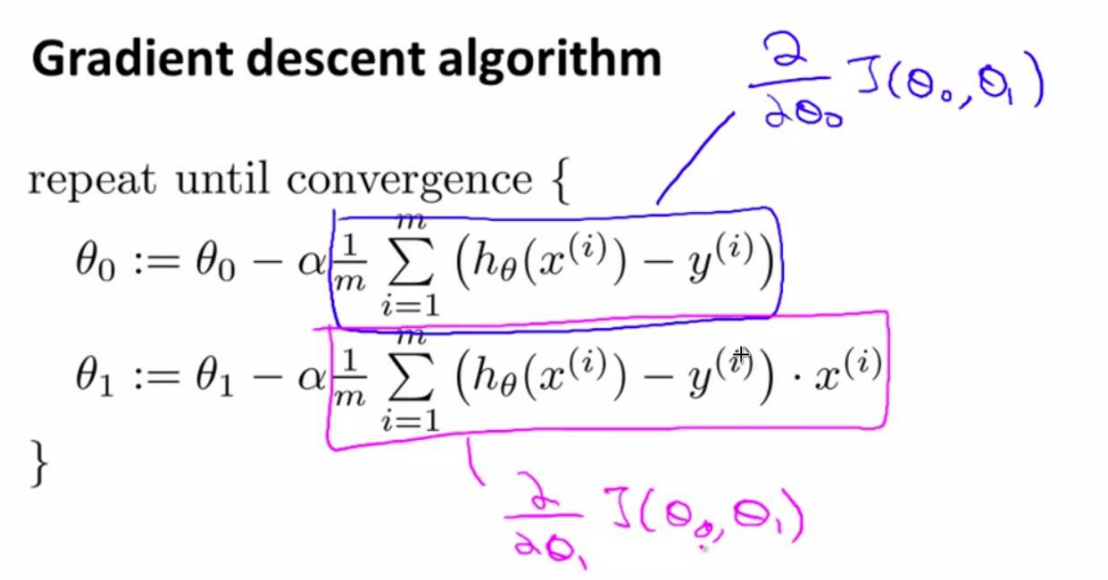
Gradient Descent Intuition

* 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 minimo 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:



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 “batchs”, nós não olhamos para o conjunto completo e sim para pequenos subconjuntos de treinamento.

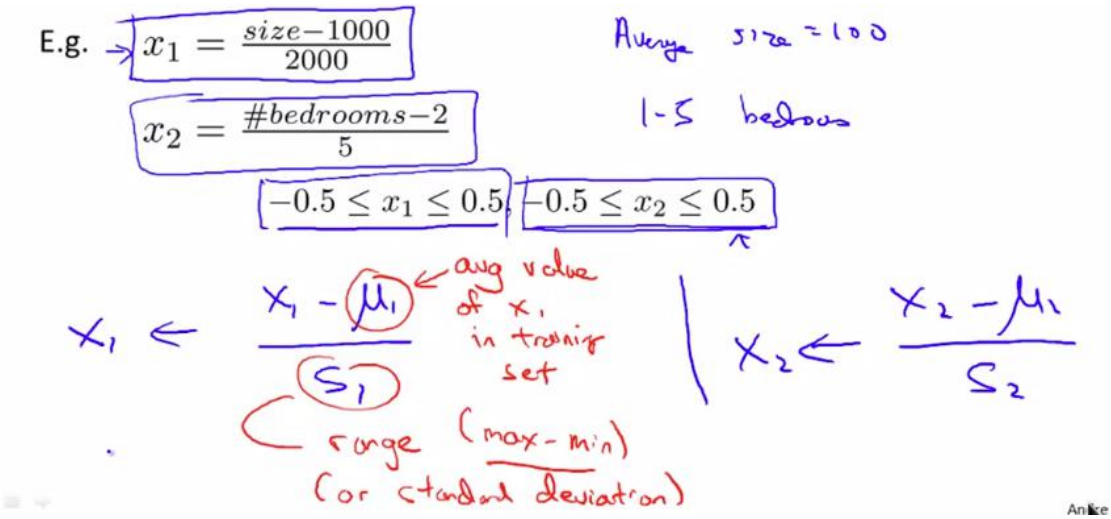
Instalando o Octave

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)



Onde S_j (1 <= j <= 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 numero 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 convergid, 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...);

Duvidas: 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 multiplas features!