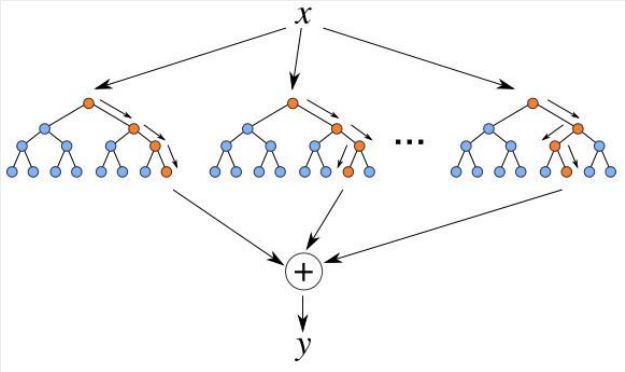
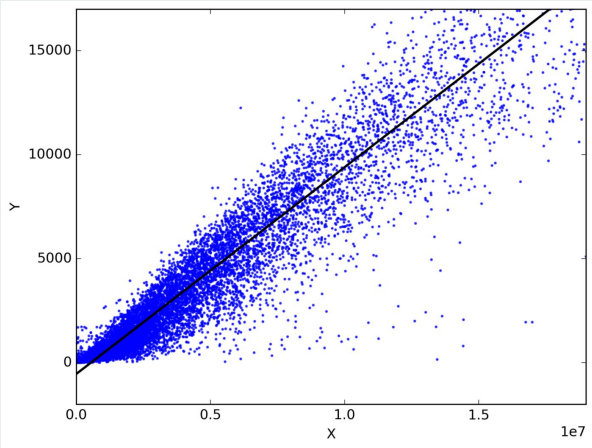
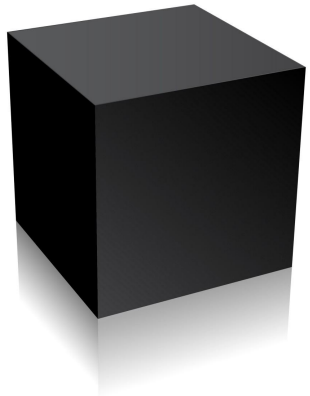
Funcionamento de Algoritmos





Até agora...

Até agora, estamos tratando algoritmos como caixas pretas, que recebem dados para treinamento, e após o aprendizado, conseguem prever resultados magicamente



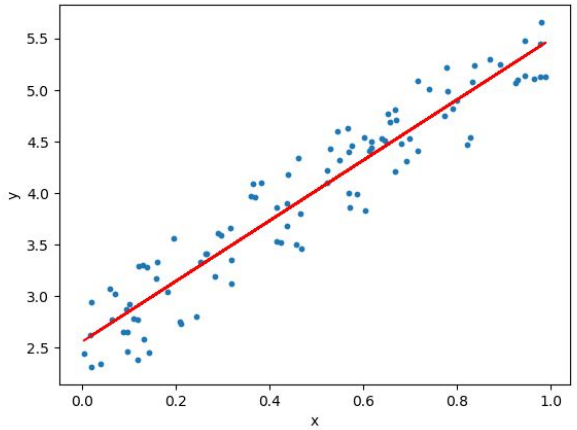
Linear Regression - Regressão Linear

Computa uma soma com pesos + uma constante chamada de *Bias Term*

Y -> Predição
B -> Pesos
X -> Features
B0 -> BIAS

$$Y_i = \beta_0 + \beta_1 X_i$$

Constant/Intercept
Independent Variable
Dependent Variable
Slope/Coefficient



Linear Regression - Regressão Linear

Ok, mas como chegamos nesse valor?

Primeiramente, começamos com um modelo tendo pesos qualquer e queremos calcular o quanto esse modelo está errando: MSE - Mean Square Error -> “Média do Erro ao Quadrado”

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2$$

n -> Quantidade de Previsões
yi -> Resultado calculado
~y -> Resultado Real

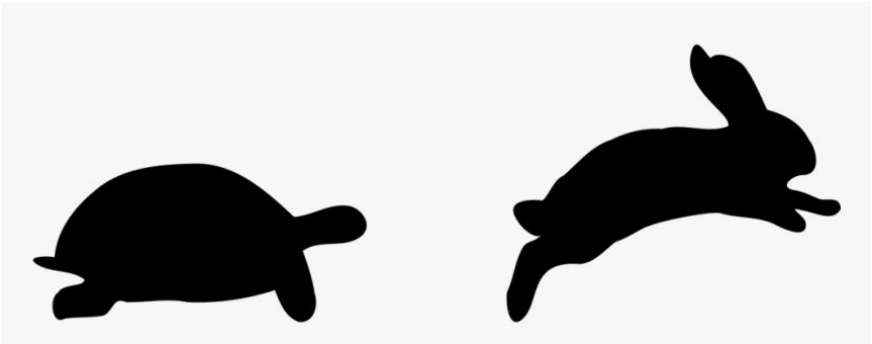
Linear Regression - Regressão Linear

Agora, precisamos calcular os pesos da nossa equação para reduzir ao máximo esse erro (MSE). Temos diferentes formas de fazer isso:

- Normal Equation - Equação Normal
- Gradient Descent
  - Batch Gradient Descent
  - Stochastic Gradient Descent

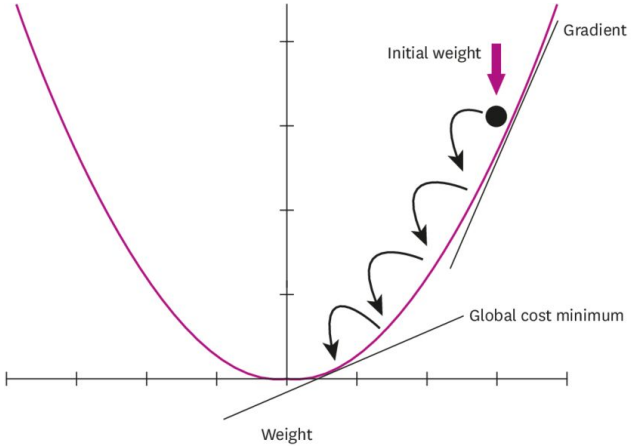
Linear Regression - Regressão Linear

No geral, todas funcionam e chegam em resultados muito próximos, porém cada uma é mais ou menos custosa dependendo do número de features das nossas instâncias e do tamanho do banco de dados



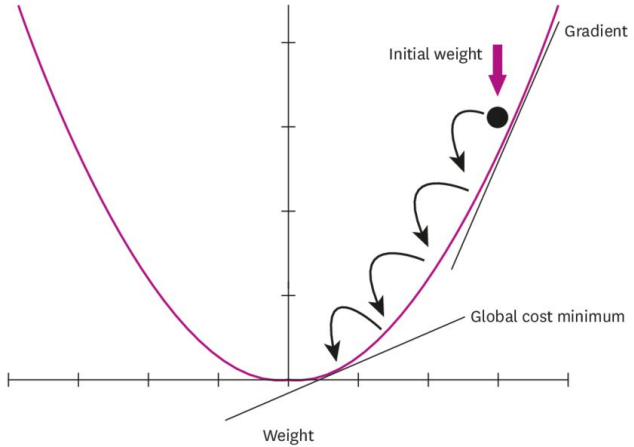
Gradient Descent

1. Colocamos valores random nos pesos (Random initialization)
2. Calculamos o MSE, queremos descobrir o quão errada está nossa randomização inicial
3. Alteramos os pesos da nossa função, para que eles reduzam o MSE
4. Voltamos ao passo 2 até que o erro seja tão pequeno quanto quisermos



Learning Rate

- É a velocidade com a qual o algoritmo aprende (muda seus pesos).
- Se for muito baixa, temos um problema computacional de tempo de execução, o algoritmo vai demorar muito
- Se for muito alta, o algoritmo não aprende nunca



Tipos de Gradient Descent (Como calcular o valor reduzido ou aumentado de cada peso)

- Batch Gradient Descent
- Stochastic Gradient Descent
- Alguns outros...

Batch Gradient Descent

Para calcular o valor de cada variável, o algoritmo tem que descobrir o quanto a mudança dela afeta o resultado, visando ter uma média de resultados o mais próximo possível da realidade (menor MSE)

Como calculamos o quanto a mudança de uma variável afeta o resultado? Derivadas Parciais

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

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

Batch Gradient Descent

Mas e o custo de calcular a derivada parcial de cada peso milhares de vezes? Alto

Resolução: Numpy faz isso com multiplicação de matrizes paralelizada e tá tudo certo!

Stochastic Gradient Descent

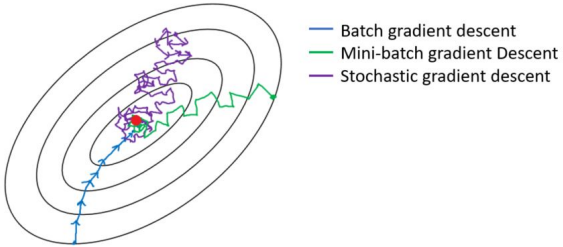
O numpy resolve muito nossos problemas quando o dataset é relativamente pequeno ou intermediário, mas em DataSets massivos o cálculo com Gradiente padrão ainda demora bastante.

Por esse motivo, em datasets grandes o Gradiente Estocástico se sobressai

Stochastic Gradient Descent

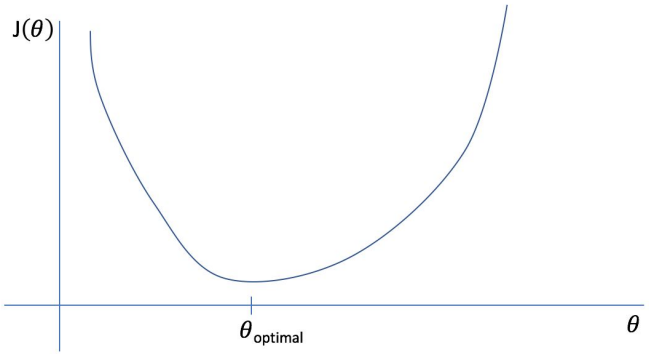
O gradiente estocástico é calculado pegando instâncias (dados) randômicos e calculando as parciais nesses dados. Conforme ele vai calculando, ele começa a tender para alguns pesos, que seriam então os pesos optimos.

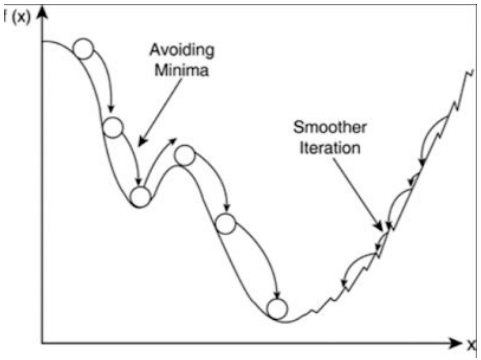
- A parte boa é que ele é muito rápido
- A parte ruim é que devido à sua natureza estocástica (não determinística), ele é menos regular que o gradiente normal, sua função de custo não é decrescente pura, ela vai decrescendo mas tem aumentos locais



Problemas e Resoluções com funções não puramente convexas

Tratados em versões otimizadas desse algoritmos e outros algoritmos

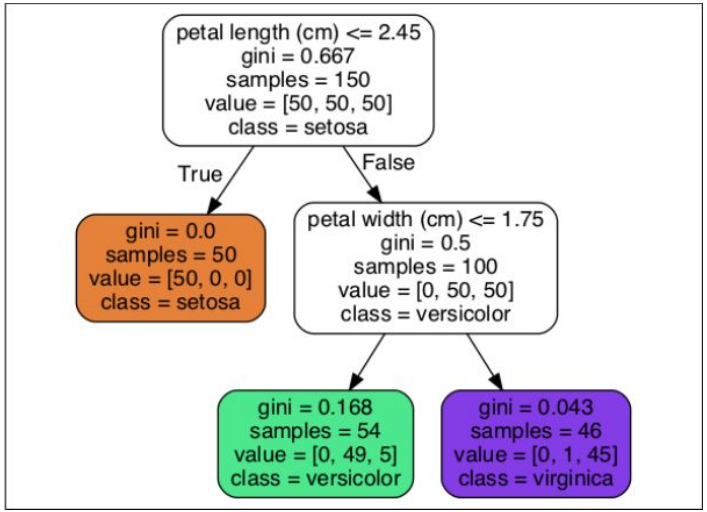




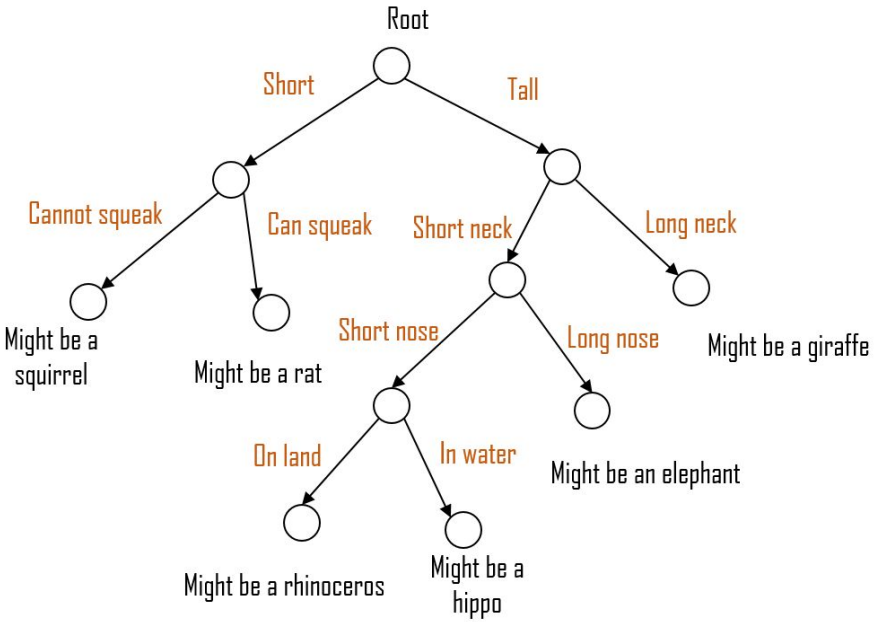
Decision Trees - Árvores de decisão

Chega em determinado resultado selecionando através de decisões pelos “nós”, partindo da raiz.

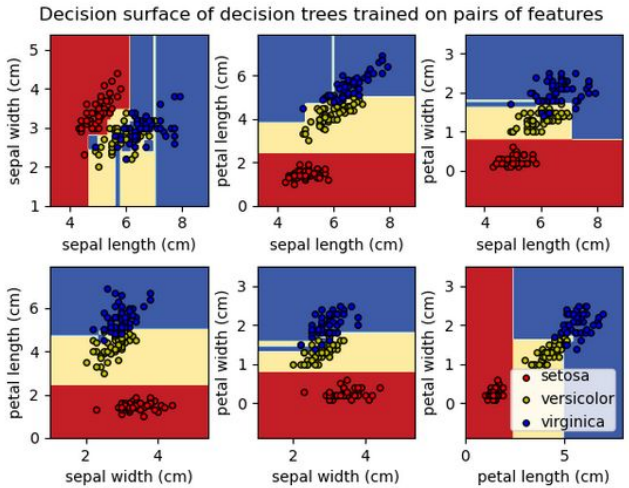
gini = “pureza” do nó
samples = total de samples que chegaram ao nó
value = número de samples por classe
class = provável classe da sample

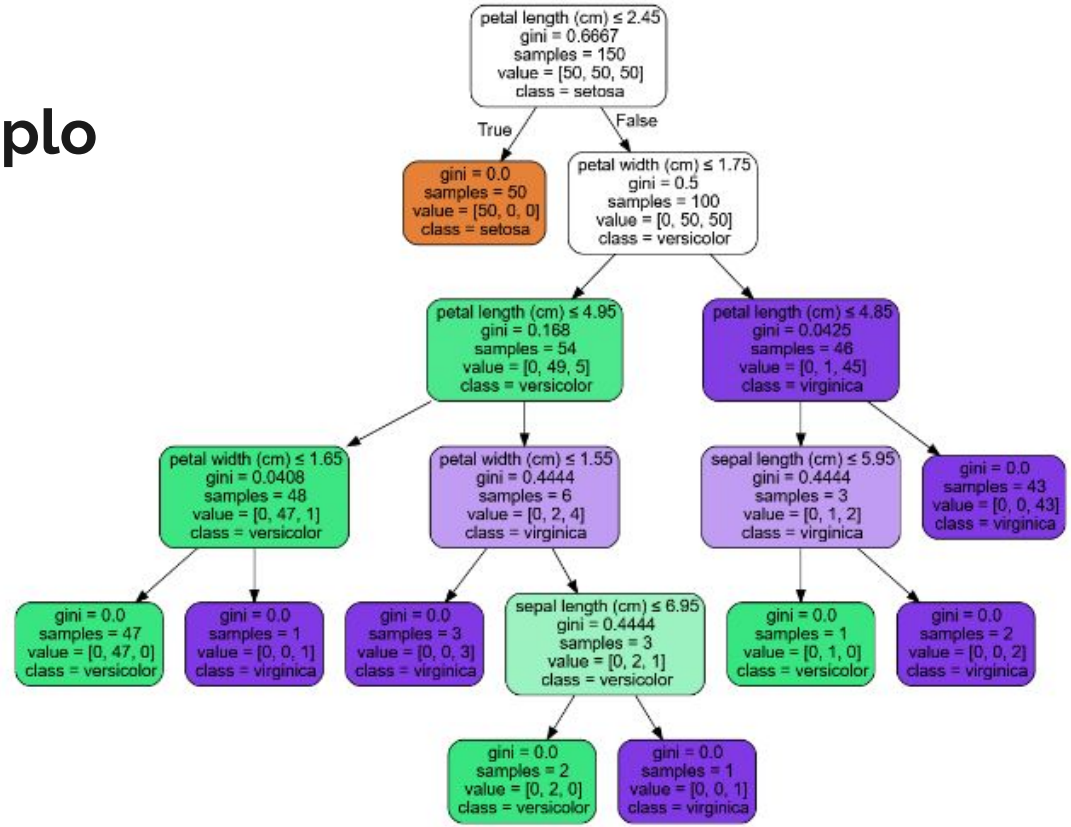


Decision Trees - Conceito

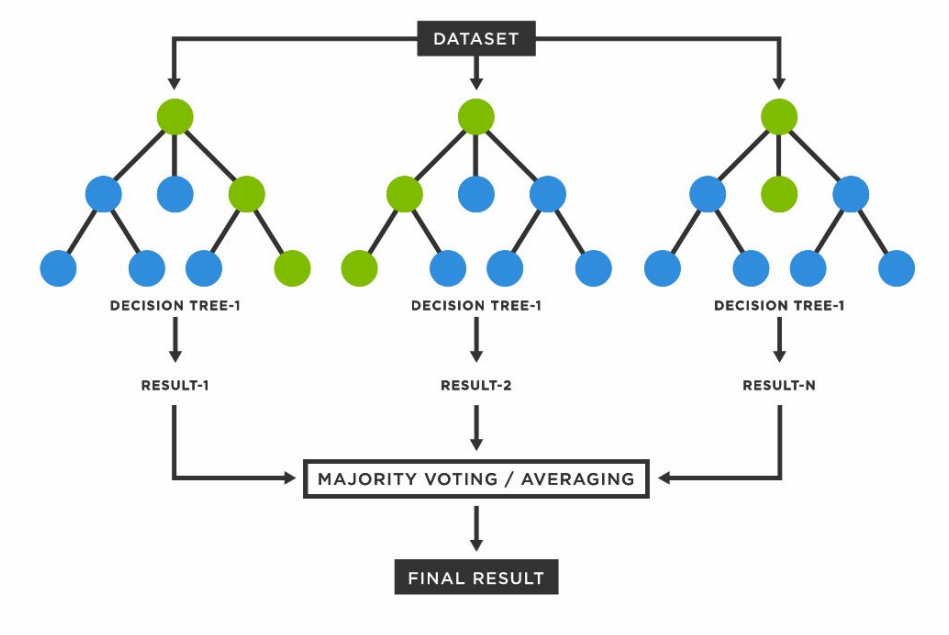


Decision Trees - Exemplo





Random Forests



Random Forests - Feature Importance

Uma das vantagens de Random Forests é a mediação da importância de cada atributo com base nas ocorrências dentro do dataset.

Hard Voting x Soft Voting

Maior número de votos x média das porcentagens