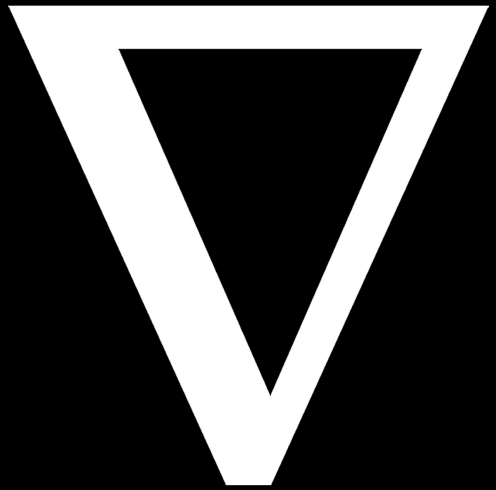


Boosting

Bruno Gomes Coelho
@brunogomescoelho.github.io

Aula de hoje: Boosting

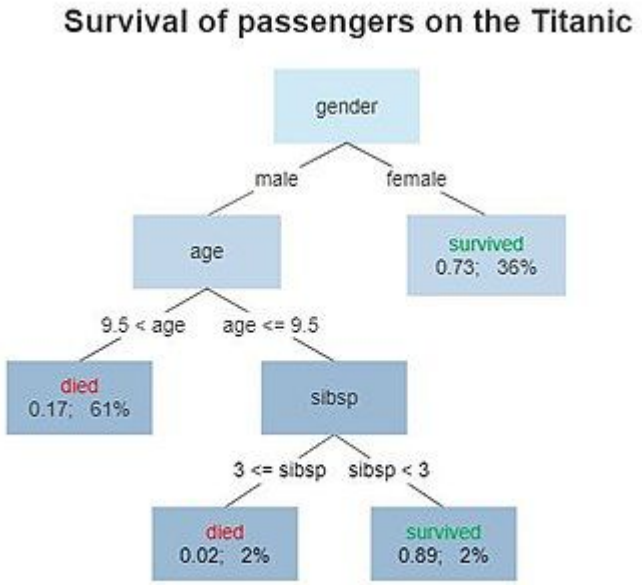
- Revisão
  - Árvores de decisão, Dilema Viés-Variância, Random Forest
- Introdução: Aprendiz fraco (“weak-learner”)
- Gradient descent & Boosting
- Na prática: XGBoost

Pré-requisitos: Conteúdo de revisão e algumas noções matemáticas

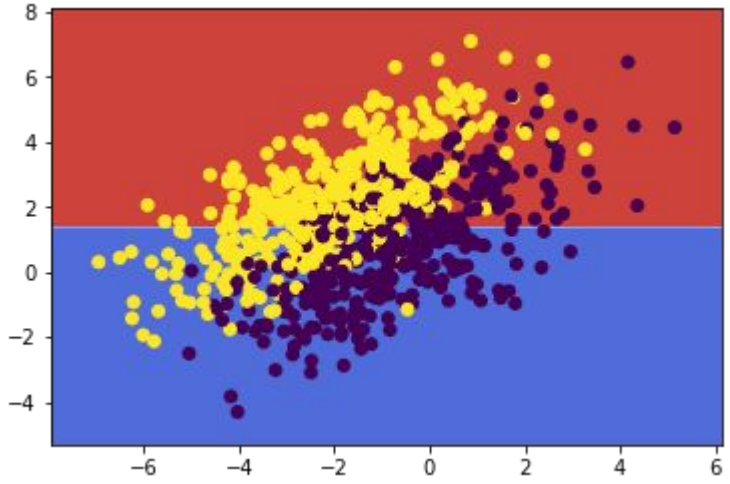
Relembrando Árvores de Decisão

Em árvores de decisão, nós usamos os nós em uma árvore para determinar a categoria a ser predita.

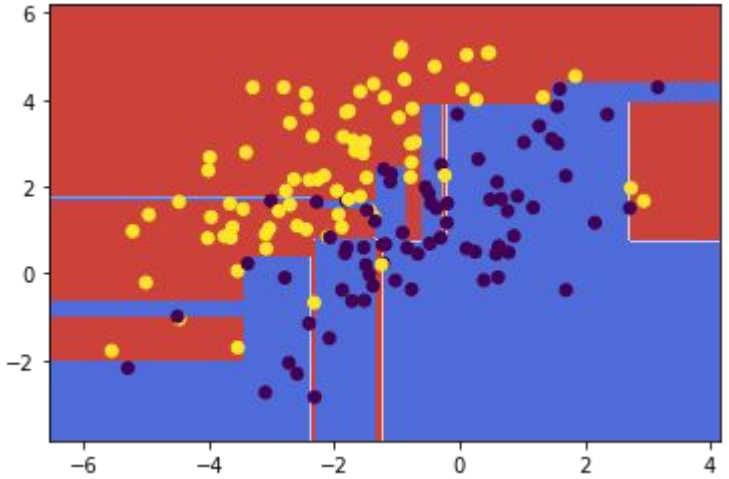
Para fazer predições, nós começamos na raíz e descemos os nós da árvore até chegarmos em uma folha, que determina a classe final.



Visualizando a superfície de decisão



1 nó



Vários nós

Problema: Overfit

Uma árvore consegue sempre ter 100% acurácia dos dados do treino?

Problema: Overfit

Uma árvore consegue sempre ter 100% acurácia dos dados do treino?

Sim*!

*Contanto que nossos dados sejam “consistentes” (sem exemplos com as mesmas features (X) mas predições (Y) diferentes), podemos criar árvores arbitrariamente grandes que **memorizam** todos os dados

Porque isso é ruim?

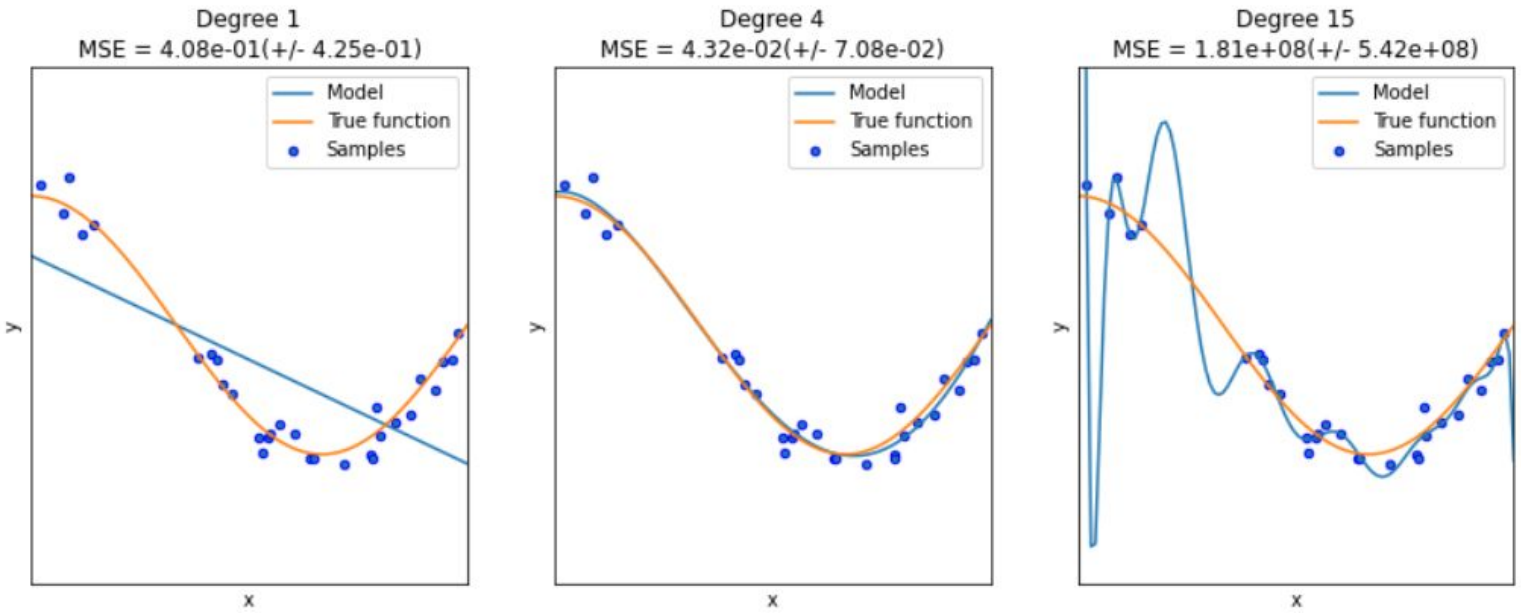
Generalização

Queremos que nossos modelos tenham bons resultados para **dados nunca antes vistos**

Não basta memorizar a lista de exercício para ir bem numa prova

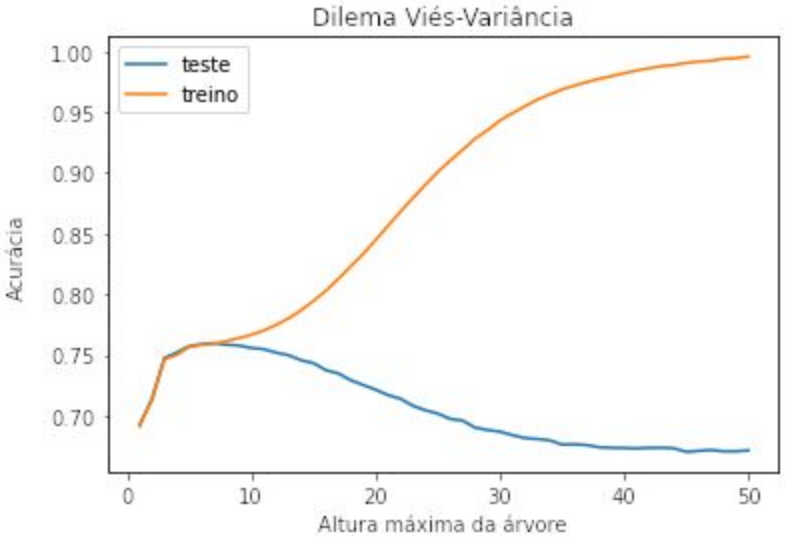
Existe uma escolha a ser feita: Queremos modelos complexos para capturar as nuances dos dados, mas que não memorizem os dados

Underfitting & Overfitting



Fonte

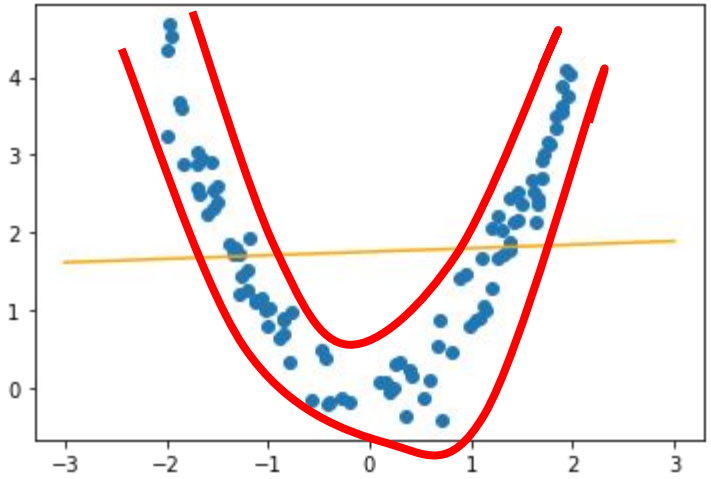
Dilema Viés-Variância



Classificadores Não-Enviesados

Classificadores não enviesados são aqueles que “no infinito”/em média, tendem a ter a mesma resposta ao melhor classificador possível - linha vermelha

Em laranja temos um classificador enviesado



Random Forests

Random Forests

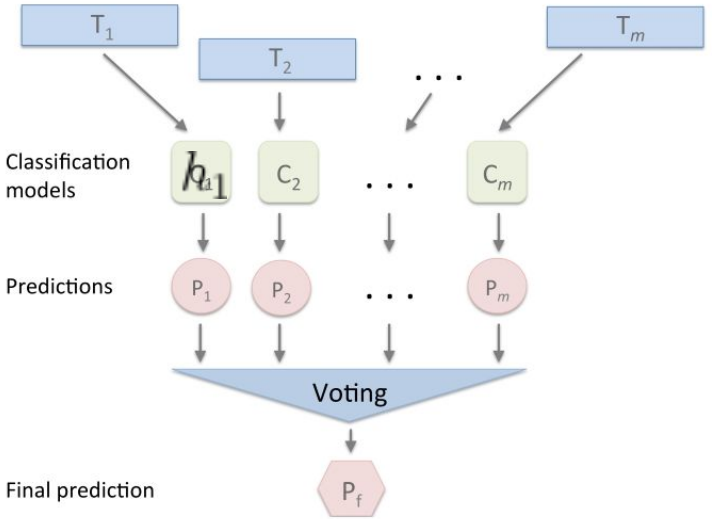
Aplicação de **bagging** com várias árvores diferentes (e não enviesadas) para reduzir a variância do modelo final

Uso de **bootstrap sampling** para simular aleatoriedade e diminuir a variância, com cada árvores usando sqrt(D) features, onde temos D features no total.

Random Forests

No final fazemos a média de M modelos, cada um $h_1, h_2, \ldots, h_m$, resultando no modelo final G_m

$$G_m = \frac{1}{M} \sum_{i=1}^{M} h_i(x_i)$$



Dúvidas até aqui?

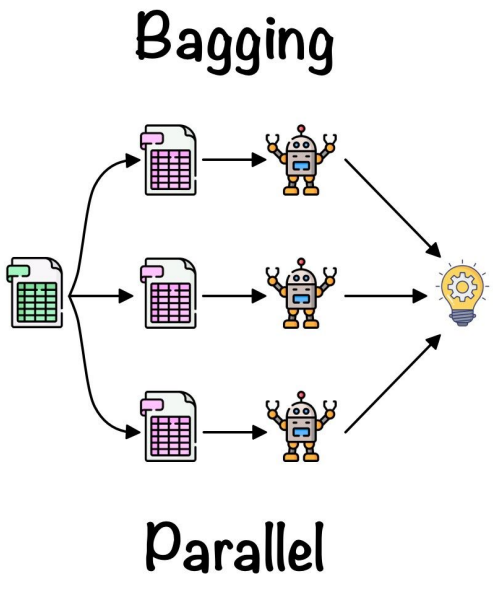
Random Forests

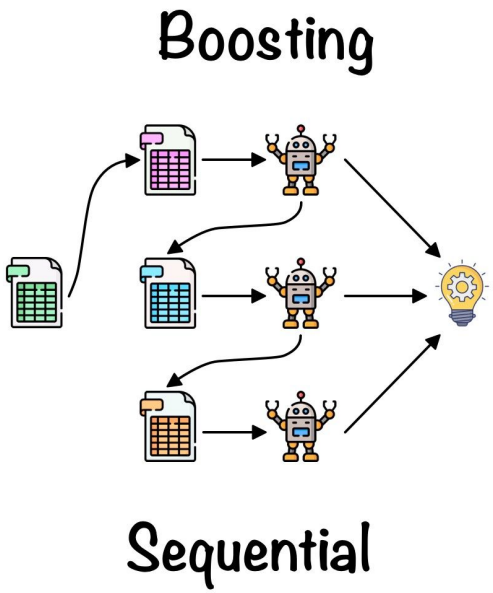
Cada modelo é independente e não enviesado - em geral 1 única árvore já possui um resultado decente;

Existe outra maneira de combinar vários modelo que não seja o bagging?

Boosting

Bagging vs Boosting





Fonte

Boosting

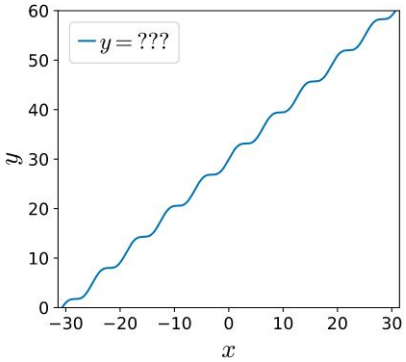
Abordagem semelhante, mas diferente: vários modelos ruins (“weak learners”);

Definimos um weak-learner como um modelo que vai levemente melhor que um chute aleatório.

Conseguimos juntar vários desses modelos? Sim!

Boosting: Intuição

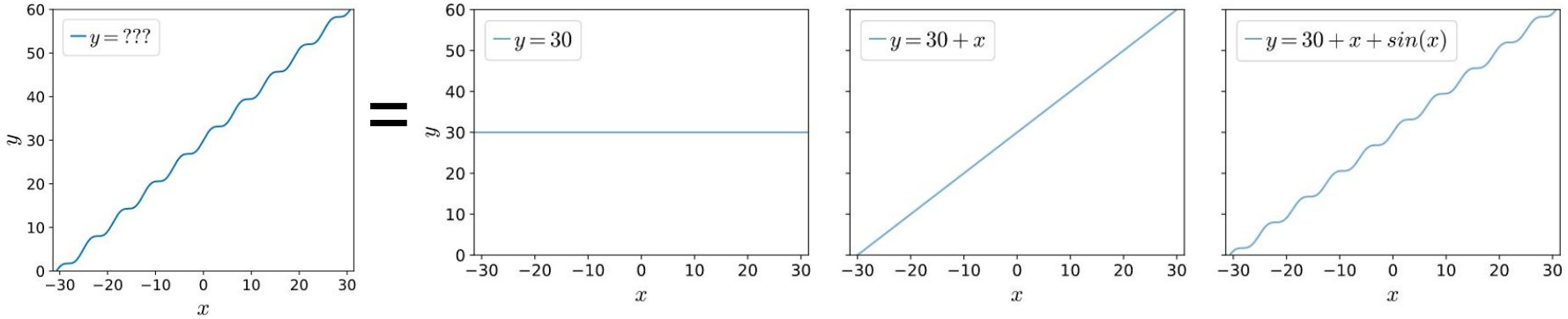
Queremos aprender uma função complexa qualquer



= ???

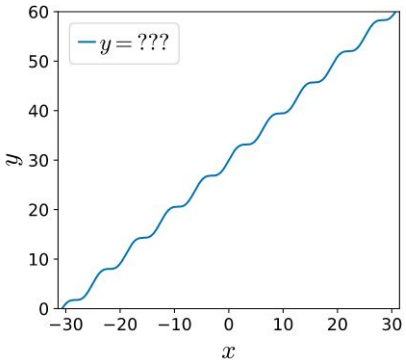
Boosting: Intuição

Fazemos uma soma de várias funções diferentes



Boosting: Intuição

Fazemos uma soma de várias funções diferentes



=

$$F(x) = f_1(x) + f_2(x) + f_3($$

where:

$$f_1(x) = 30$$
$$f_2(x) = x$$
$$f_3(x) = sin(x)$$

Boosting

Queremos ter um único modelo que é a combinação de M modelos diferentes;

Vamos fazer isso recursivamente, a cada passo acrescentando mais 1 modelo;

Poderíamos no passo **m** mudar todos os (**m-1)** modelos anteriores mas isso é custoso demais - Como adaptar?

Boosting

Implementamos uma solução gulosa, a cada passo, fixamos os modelos anteriores e mudamos apenas o modelo novo :)

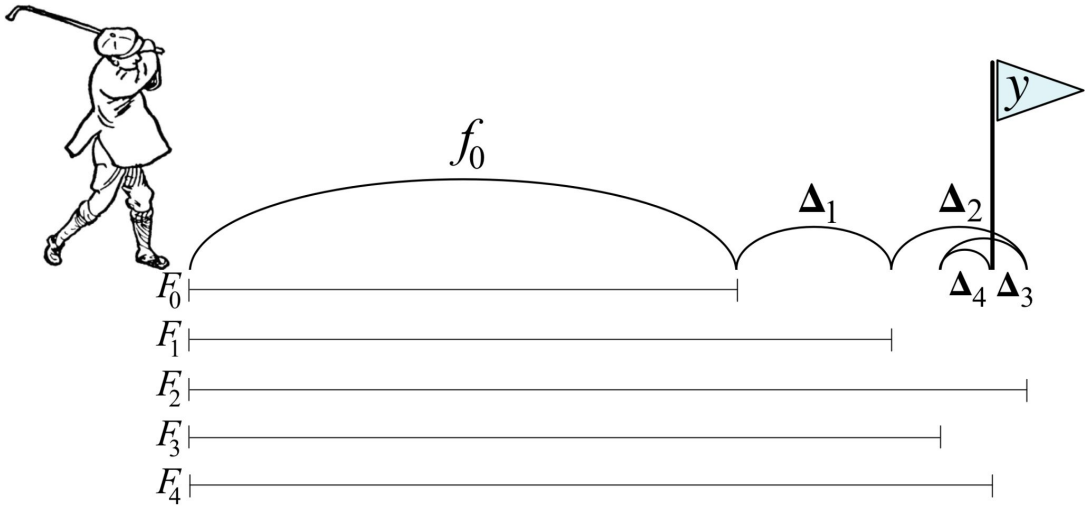
Ou numa notação mais formal, no passo m aprendemos um modelo h_m(x):

$$G_m(x) = G_{m-1}(x) + \alpha h_m(x)$$

Onde alpha é hiper-parâmetro que vamos ver daqui a pouco

Boosting: Intuição

A cada passo nosso modelo aprender a ir um pouco melhor



Boosting

Como treinar esses vários modelos? Iniciamos com um chute inicial e a cada iteração treinamos um modelo no erro do modelo anterior

Vamos ver um exemplo que queremos predizer o aluguel (*rent*) de uma casa a partir da sua área (*sqfeet,* pés quadrados);

Vamos usar árvores de decisão com apenas 1 corte como o modelo fraco.

Boosting

Dados exemplos:

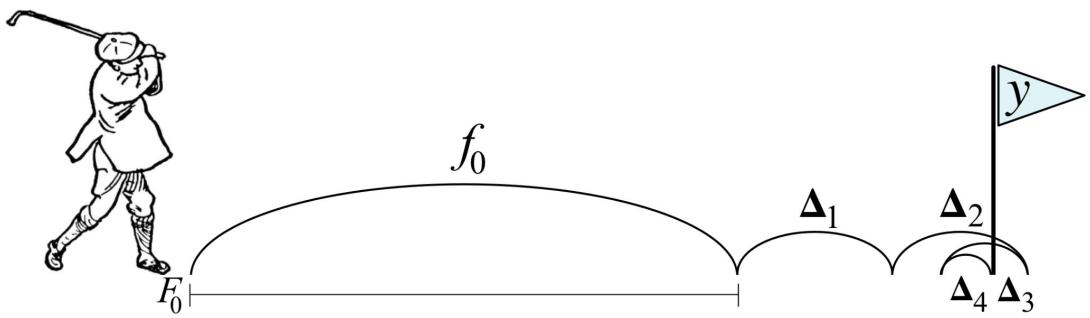
sqfeet	rent
750	1160
800	1200
850	1280
900	1450
950	2000

Boosting

A partir dos meus dados, gero um chute inicial que é só a média dos valores;

Isso equivale a nosso primeiro modelo $$F_0 = \frac{1}{n}\sum_{i=1}^{n} y_i = 1418$$

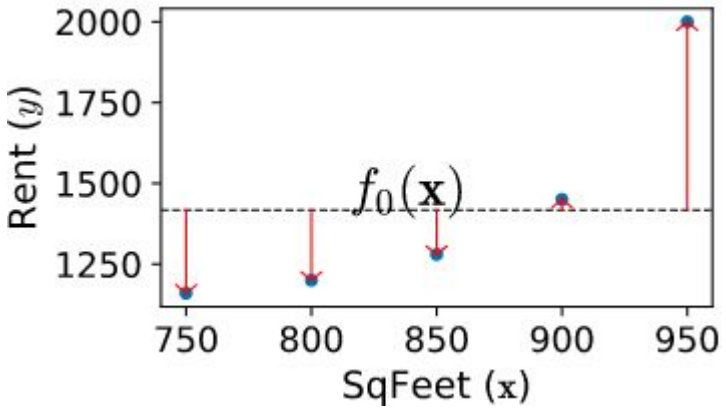
sqfeet	rent	F_0
750	1160	1418
800	1200	1418
850	1280	1418
900	1450	1418
950	2000	1418



Boosting

Treinamos o próximo modelo sobre o erro do modelo anterior, mais especificamente sobre o **Residual** do modelo F definido como (y - F(x))

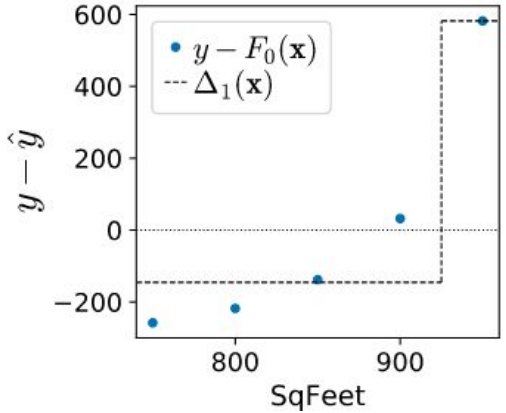
$$F_0 = \frac{1}{n}\sum_{i=1}^{n} y_i = 1418$$



Boosting

Próximo modelo prediz y-F0, tenta “acertar o erro” do modelo anterior

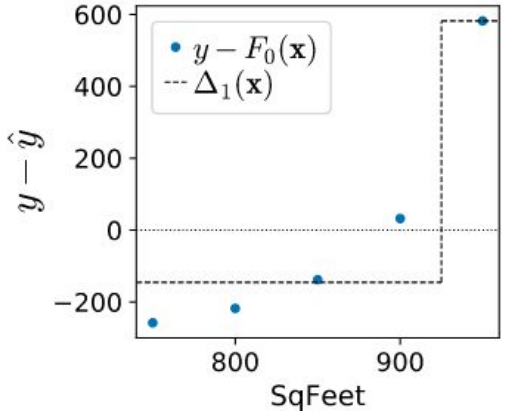
sqfeet	rent	F_0	$\mathbf{y} - F_0$
750	1160	1418	-258
800	1200	1418	-218
850	1280	1418	-138
900	1450	1418	32
950	2000	1418	582



Boosting

Próximo modelo prediz y-F0, tenta “acertar o erro” do modelo anterior

sqfeet	rent	F_0	$\mathbf{y} - F_0$
750	1160	1418	-258
800	1200	1418	-218
850	1280	1418	-138
900	1450	1418	32
950	2000	1418	582



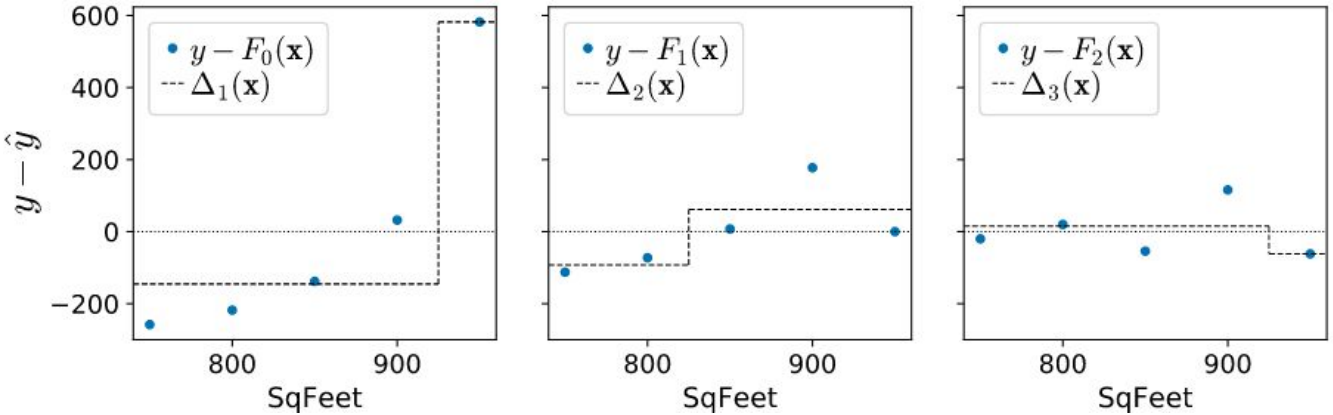
Mas nossos modelos são “fracos”, então não vamos ter um resultado perfeito

Vamos ter **um novo residual**...

Boosting

Após alguns passos

F1 F2 F3



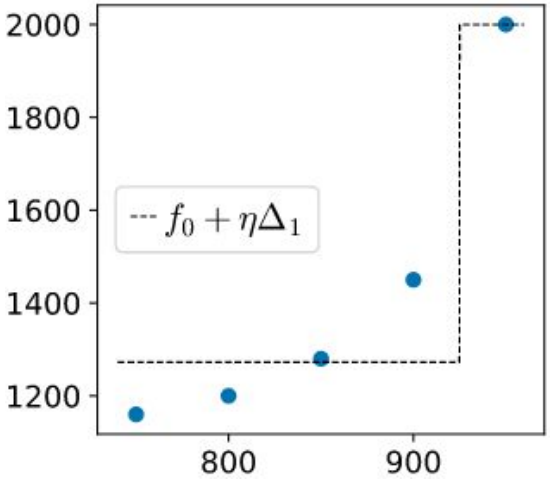
Boosting

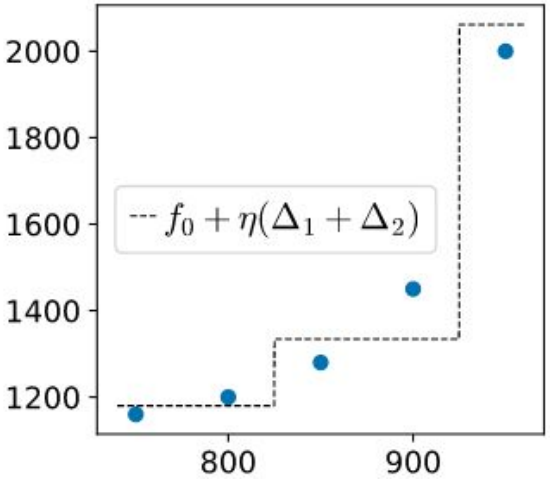
Após vários passos

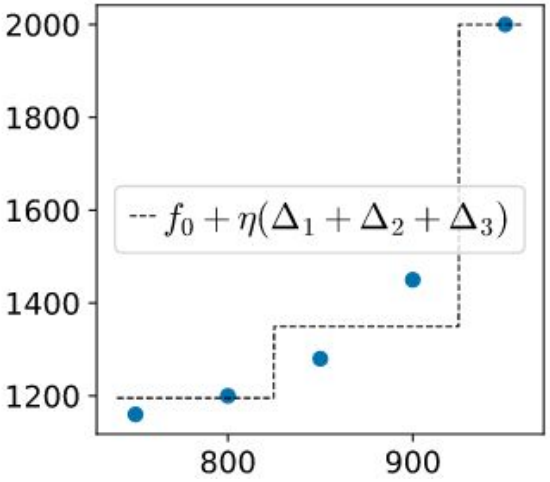
sqfeet	rent	F_0	$y - F_0$	Δ_1	F_1	y-F_1	Δ_2	F_2	y - F_2	Δ_3	F_3
750	1160	1418	-258	-145.5	1272.5	-112.5	-92.5	1180	-20	15.4	1195.4
800	1200	1418	-218	-145.5	1272.5	-72.5	-92.5	1180	20	15.4	1195.4
850	1280	1418	-138	-145.5	1272.5	7.5	61.7	1334.2	-54.2	15.4	1349.6
900	1450	1418	32	-145.5	1272.5	177.5	61.7	1334.2	115.8	15.4	1349.6
950	2000	1418	582	582	2000	0	61.7	2061.7	-61.7	-61.7	2000

Boosting

Resultado da soma das funções a cada passo







Dúvidas até aqui?

Boosting: Algoritmo

Comece com um chute inicial F0 (por exemplo, média)

Para cada modelo *m* de 1 a M, faça:

Calcule o residual $Rm = y - Fm\text{-}1$

Treine uma árvore *Hm* que tente predizer Rm

Seu modelo agora é $Fm = Fm\text{-}1(X) + n\ Hm(x)$

Retorne Fm

Boosting

Vimos que então aprendemos um modelo do tipo $G_m(x) = G_{m-1}(x) + \alpha h_m(x)$

Na teoria gostaríamos de ter o melhor *hm(x)* possível a cada passo, mas isso não é computacional viável, então adotamos a solução gulosa de termos um *hm(x)* qualquer.

Boosting

Visão Estatística: Boosting é uma maneira **genérica** de ter um modelo final baseado em vários **modelos fracos**, através de uma **estratégia aditiva gulosa**.

Visão Otimização: Boosting é gradient descent no espaço de funções

Boosting

Então vimos como boosting funciona no caso específico de regressão usando árvores de decisão

Mas boosting pode ser estendido para qualquer modelo fraco e várias tarefas

Para isso, precisamos de uma leve introdução a gradientes

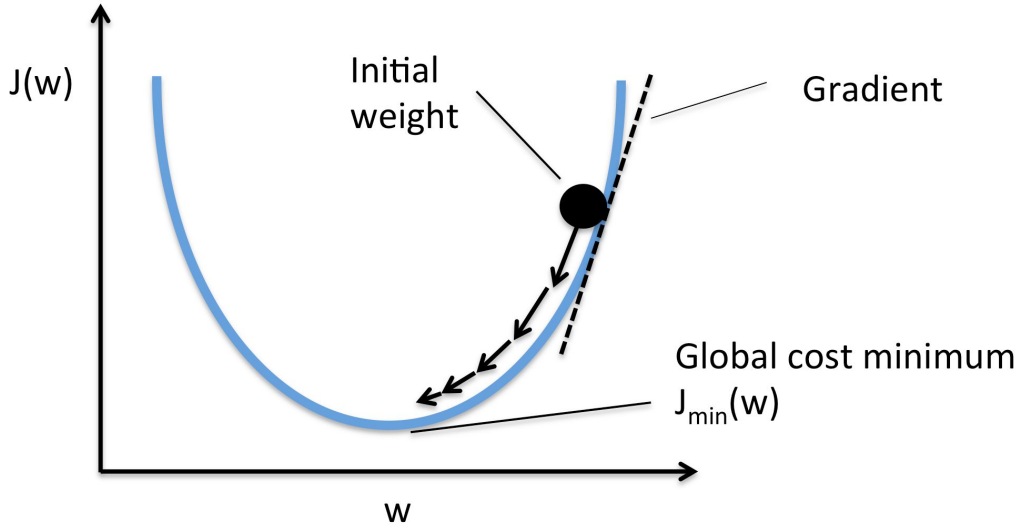
Gradientes

Direção em que devemos mudar nosso parâmetro para maximizar algo

Se queremos descer uma montanha, andamos na direção inversa do gradiente por exemplo

Gradientes

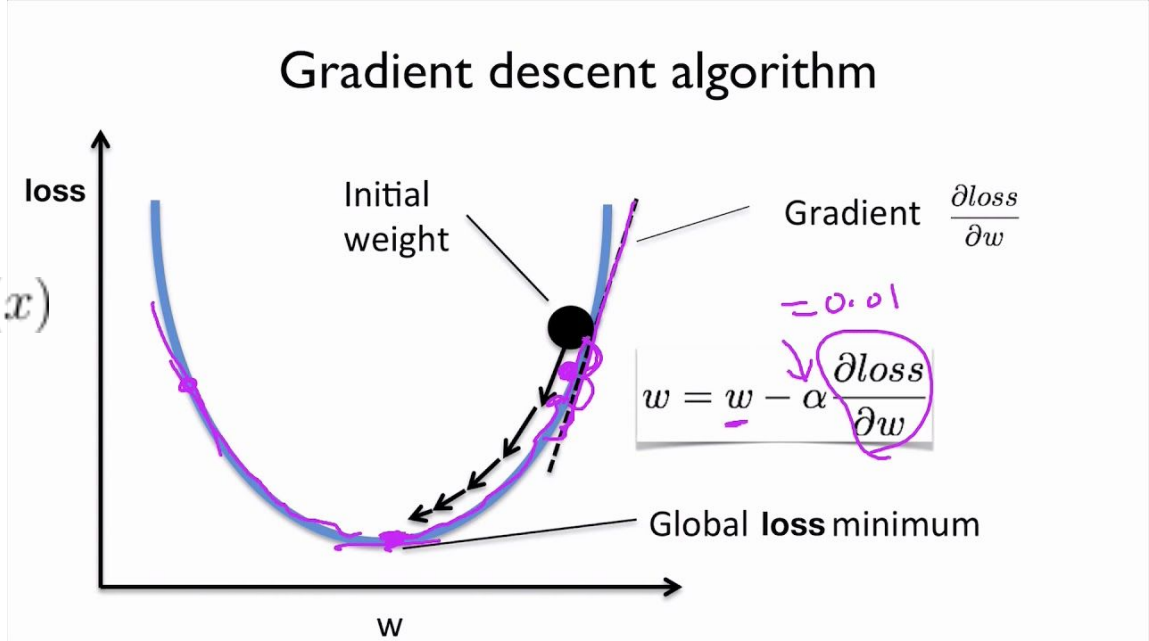
Se quero o parâmetro **w** que minimiza minha função J(w), devo atualizar w com o inverso do gradiente



Boosting & gradientes

Boosting

$$G_m(x) = G_{m-1}(x) + \alpha h_m(x)$$



Boosting & gradientes

A derivada do erro quadrático médio é o próprio residual;

Dessa maneira, Boosting pode ser visto como uma otimização de gradiente.

Demorou anos para se descobrir isso

- ~1990 Adaboost: como combinar modelos fracos
- ~2000 Boosting: Adaboost é só um exemplo de boosting
- ~2014 XGBoost: Boosting em esteróides

Boosting: Material extra

Em ordem de mais recomendado pra menos:

- How to explain gradient boosting
- Documentação do XGBoost
- Aulas teóricas Killian
- Paper original XGboost
- Material das aulas 10 e 11 do curso da NYU
- Paper original sobre Boosting

Prática: XGBoost

XGBoost

XGBoost (“eXtreme Gradient Boosting”) é uma implementação muito popular de boosting aplicado com árvores de decisão.

Possui diversas vantagens interessantes: **Boa escolha como modelo complexo padrão para dados tabulares.**

XGBoost: Modificações

- Aproximação de taylor de segunda ordem em vez de só derivada/residual
  - Cálculo da “melhor” árvore a cada passo de uma maneira mais otimizada
  - É o análogo d*o método de newton* de *redes neurais* para boosting
- Feature sampling
- Row sampling

XGBoost: Modificações

Usa feature sampling: Usar apenas algumas das colunas

Mesma ideia e motivação da RF, utilizar apenas um subconjunto das features para cada modelo

Também torna o algoritmo mais rápido

XGBoost: Modificações

Usa data sampling: Usar apenas algumas das linhas

Utilizar apenas um subconjunto dos dados para fazer a estimação do residual/direção da derivada

Também torna o algoritmo mais rápido, pode ser visto como equivalente de um “mini-batch” nas redes neurais

Vantagens

Aplicável a várias tarefas diferentes, incluindo ranking e análise de sobrevivência

Inst	Age	Sex	ph.ecog	ph.karno	pat.karno	meal.cal	wt.loss	Time to death (days)
3	74	1	1	90	100	1175	N/A	306
3	68	1	0	90	90	1225	15	455
3	56	1	0	90	90	N/A	15	$[1010, +\infty)$
5	57	1	1	90	60	1150	11	210
1	60	1	0	100	90	N/A	0	883
12	74	1	1	50	80	513	0	$[1022, +\infty)$
7	68	2	2	70	60	384	10	310

Vantagens

Altamente escalonável para milhões de dados/milhares de features e ambientes de programação diferentes

- Implementações em Spark/Hadoop/MPI
- Otimização com GPU
- Acesso com C/C++/Julia/Java/Ruby/Python/R
    - Interface “scikit-learn” para Python; *model.fit()/model.predict()*

Desvantagens

Método não muito interpretável (assim como a maioria dos modelos complexos)

Atualmente só é estado da arte para problemas difíceis para dados tabulares

Derivação matemática complexa

XGBoost na prática

Necessário especificar:

- A quantidade de modelos total M
  - Quanto mais modelos maior o poder de modelagem, mas maior a tendência a overfitting
- Qual a *learning rate*, isto é, o nosso parâmetro alpha/eta.
  - Quanto menor, maior o tempo para convergir pois menos cada atualização;
  - Em geral, quanto menor possível tende a resultados bons

Fin