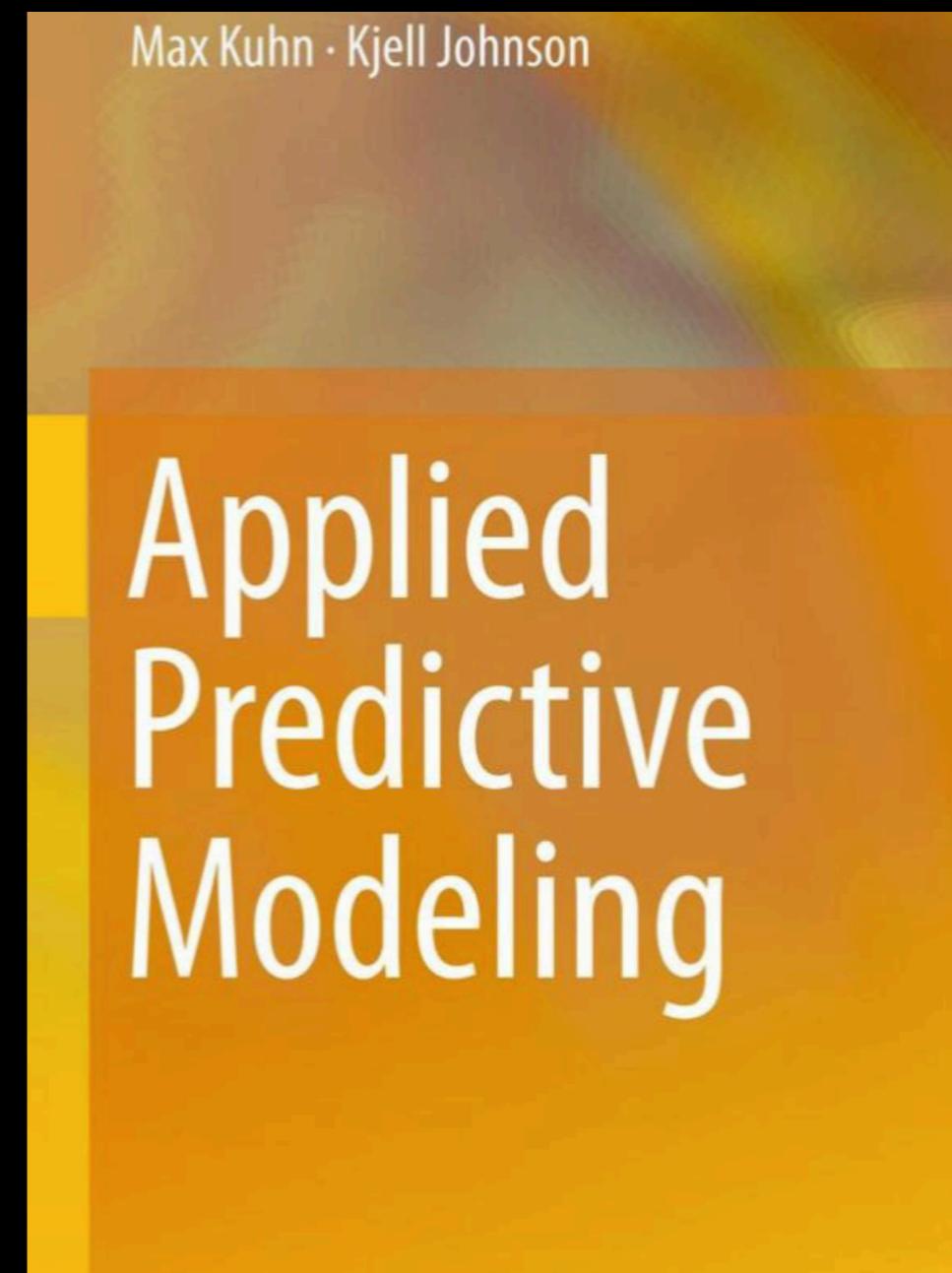
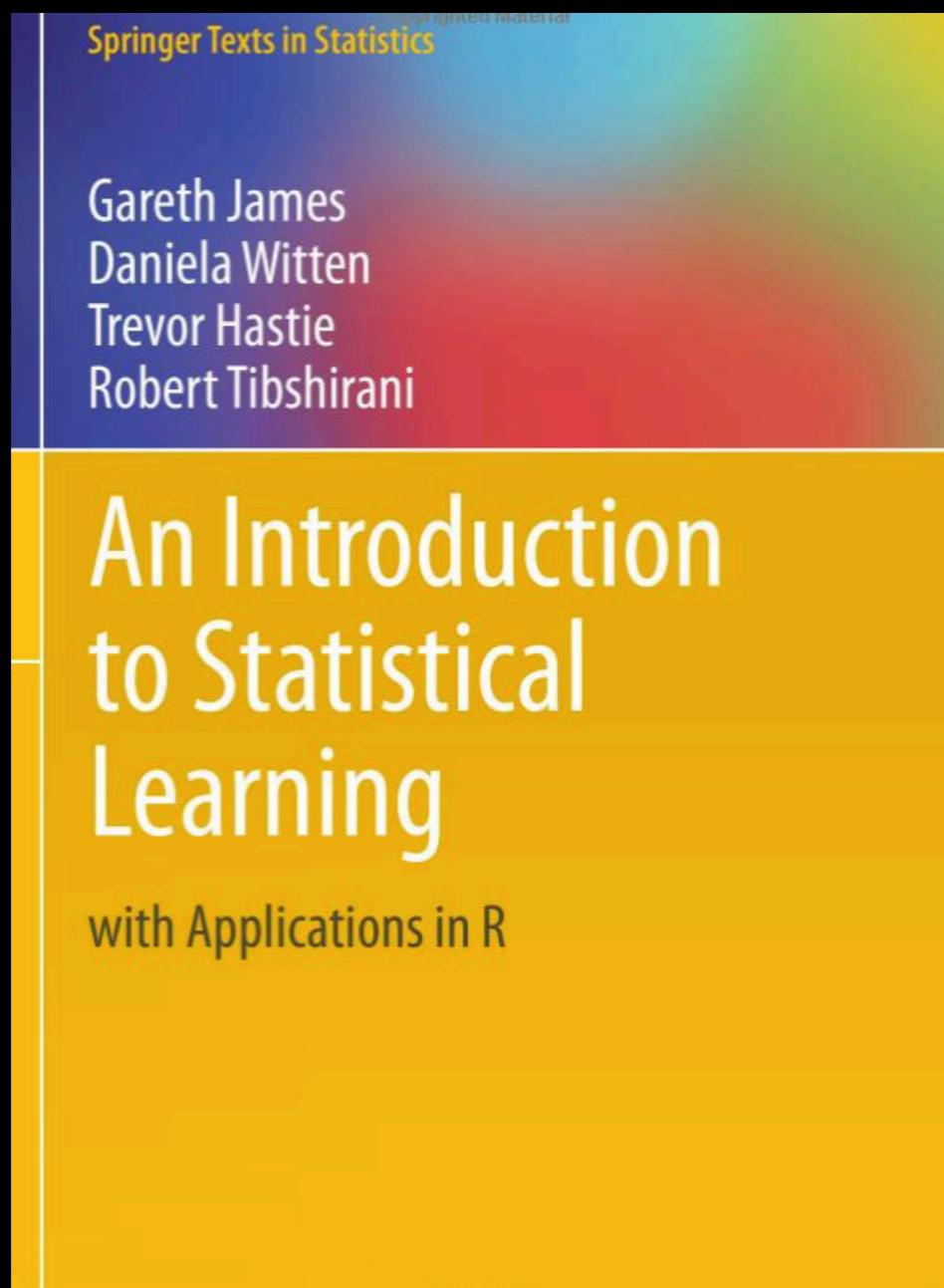


Modelagem Analítica com Machine Learning

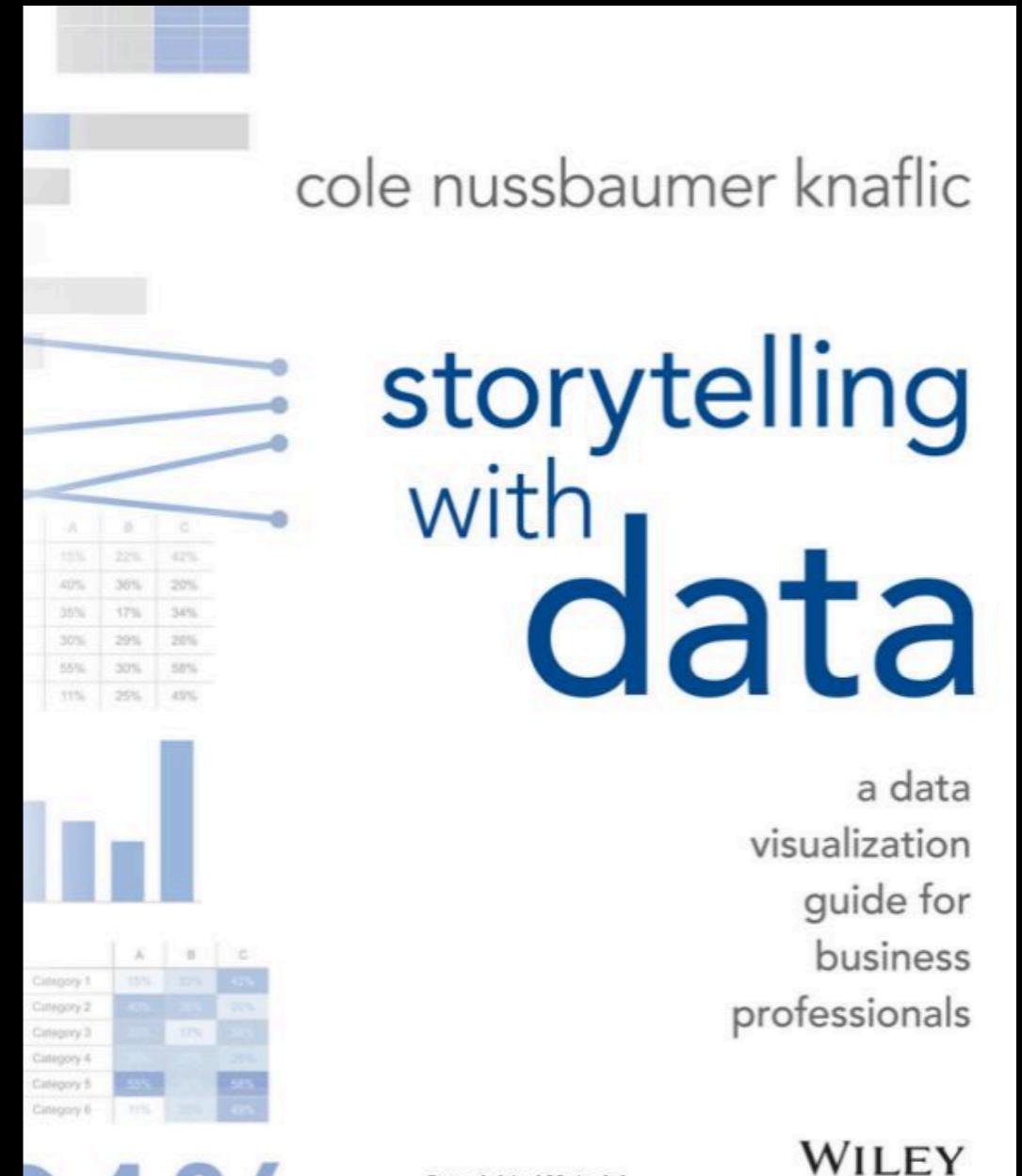
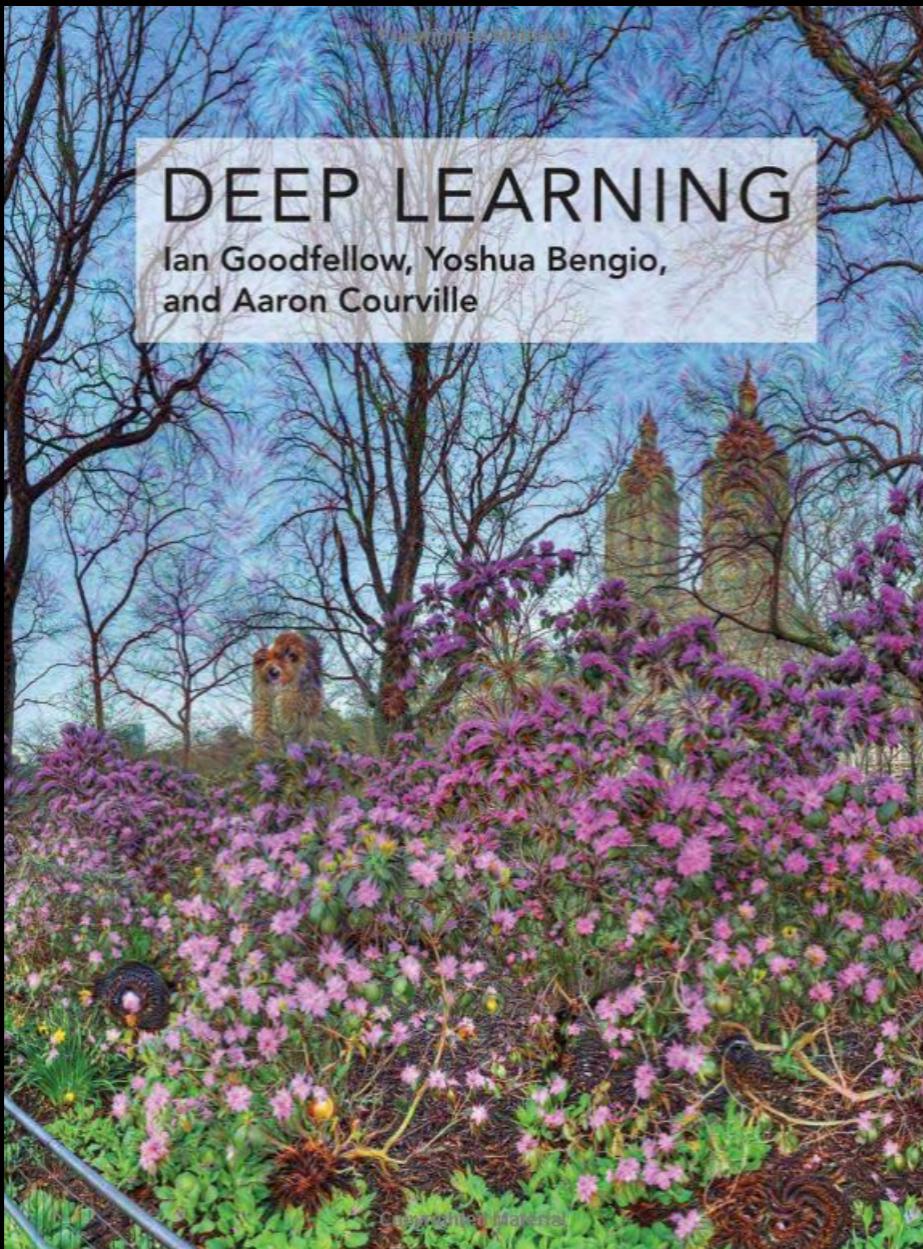
Terceiro Dia

Paulo Cysne Rios, Jr.

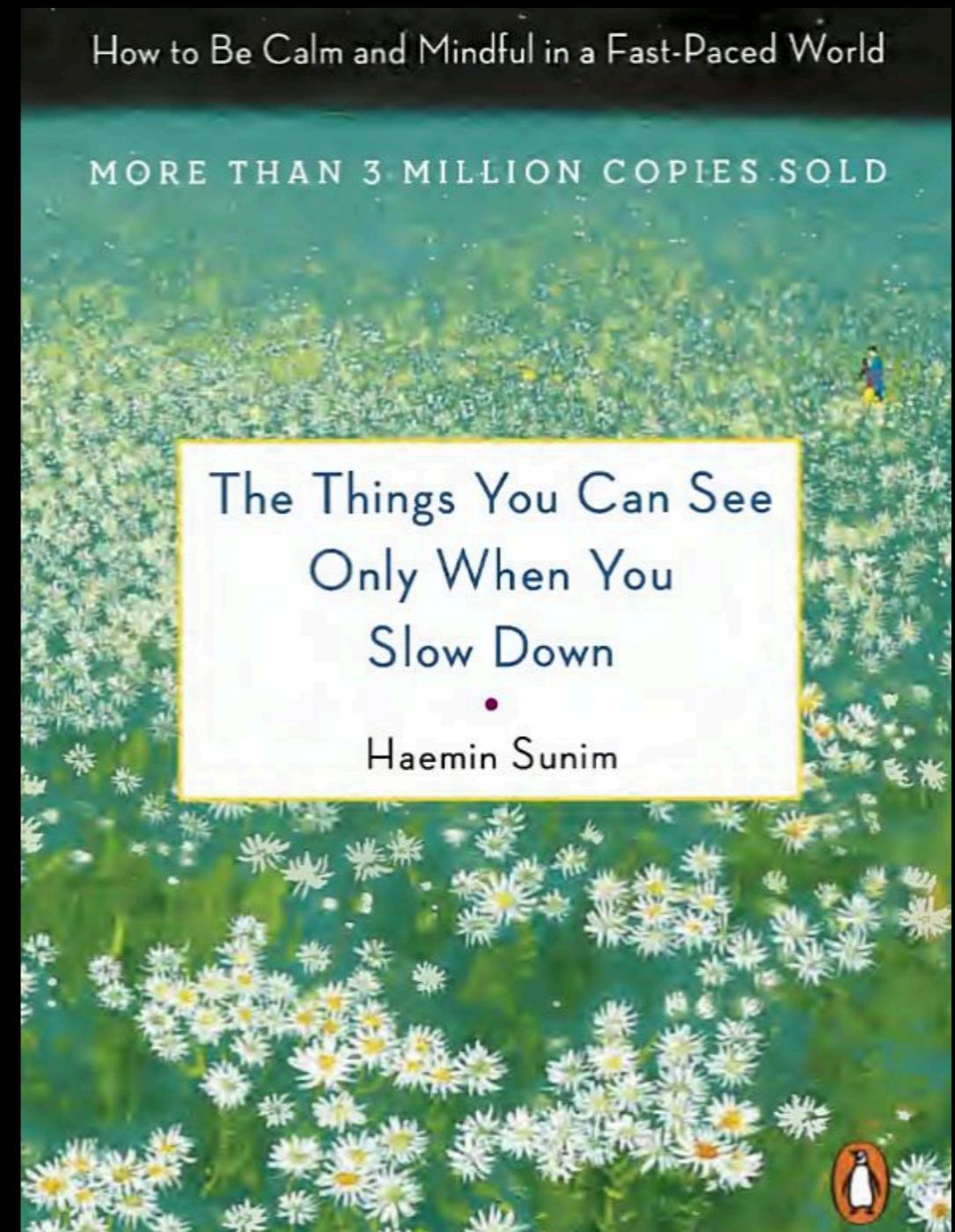
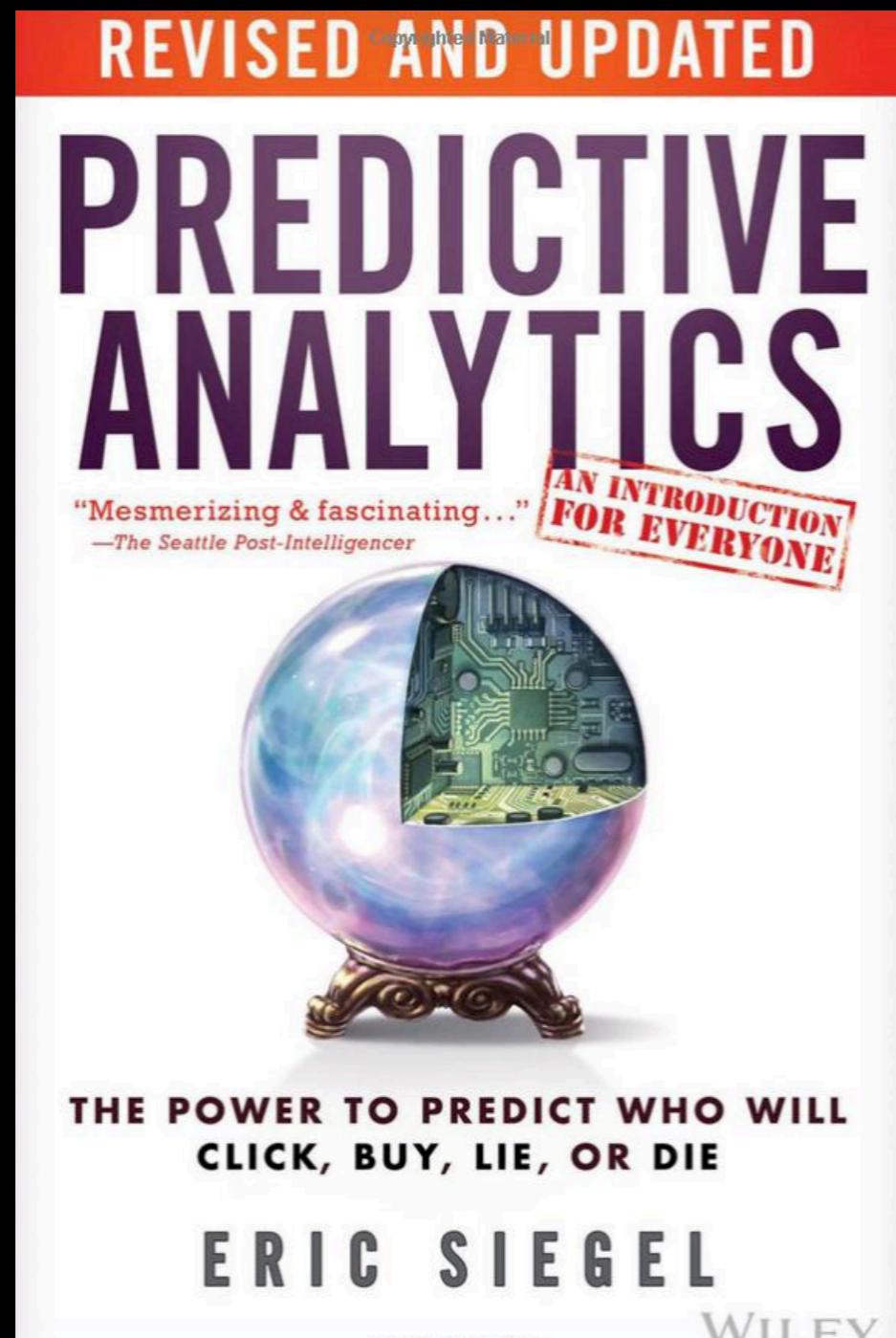
Livros Recomendados



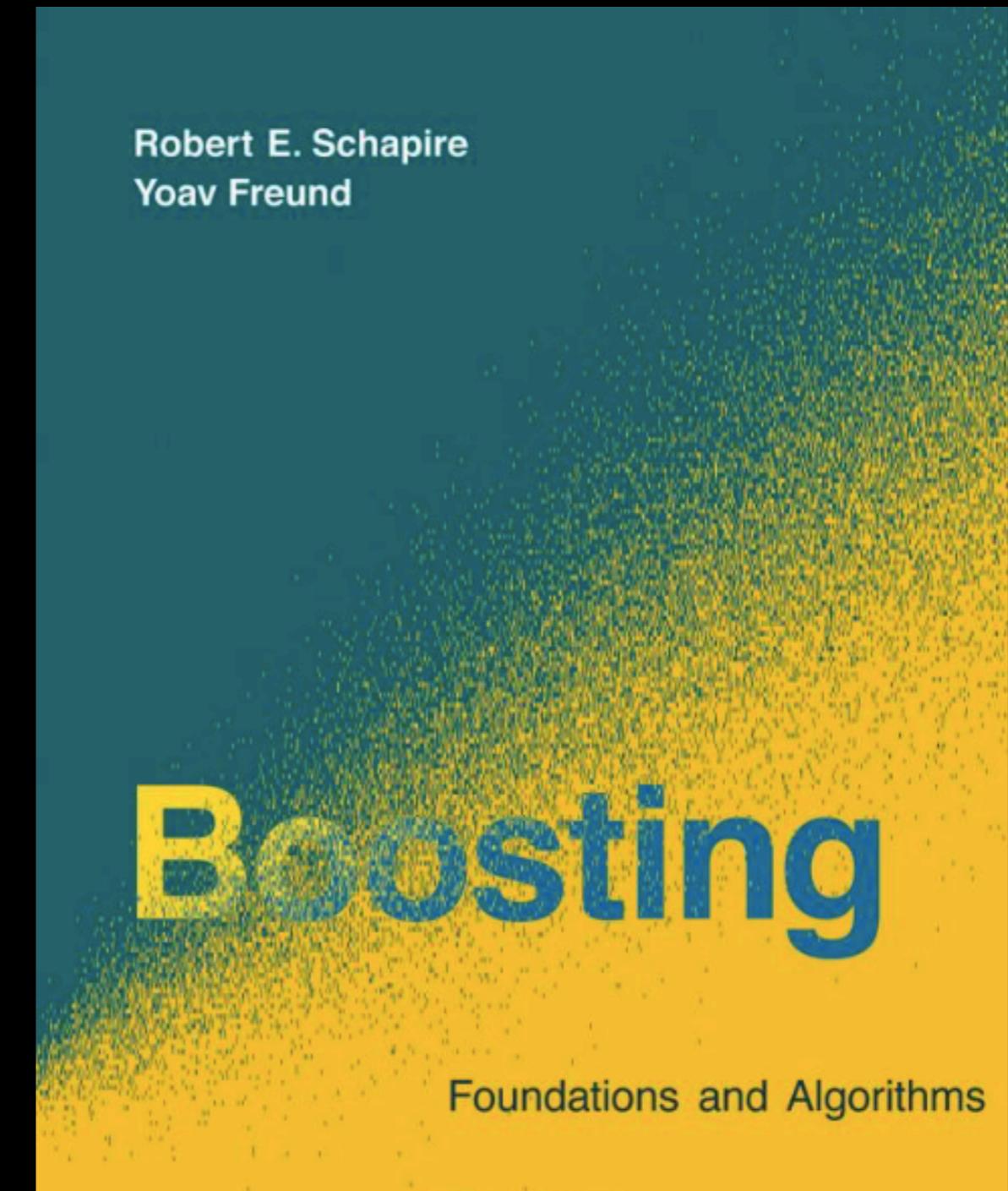
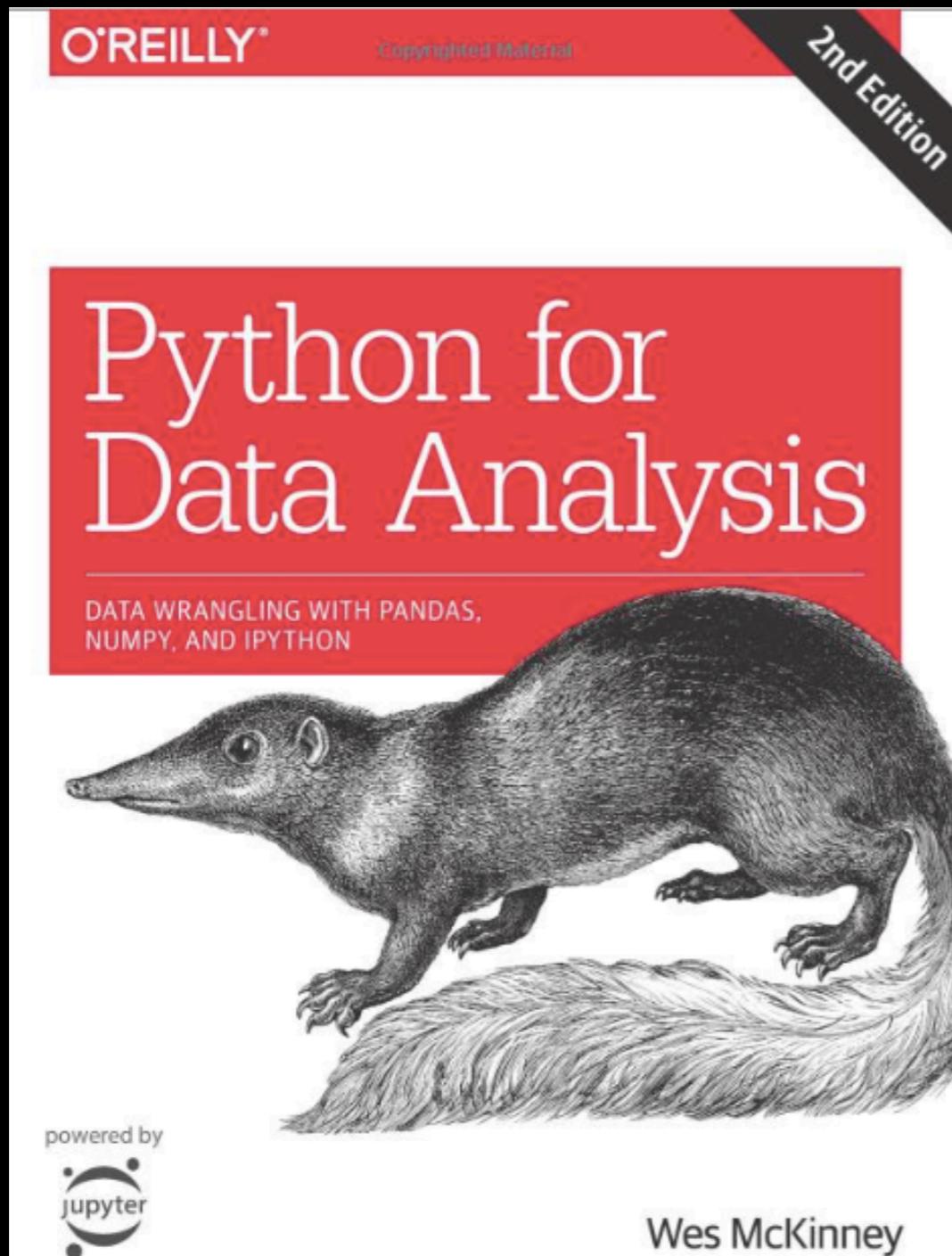
Livros Recomendados



Livros Recomendados



Livros Recomendados



Revisão de Ontem

Regressão Linear com mais de uma variável independente

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

De forma matricial

$$\hat{y} = h_{\theta}(\mathbf{x}) = \boldsymbol{\theta}^T \cdot \mathbf{x}$$

Diferença

A diferença entre o
valor real e o **valor que predizemos**

A diferença entre o valor predito e o valor real
pode ser expresso através do mean square error (MSE)

$$\text{MSE}(\mathbf{X}, h_{\theta}) = \frac{1}{m} \sum_{i=1}^m \left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)} \right)^2$$

Valor Preditivo
Valor Real

Nosso objetivo é **minimizar** este valor!!

Minimizar esta diferença = **predizer melhor!!**

Minimizando o custo

Esta diferença é a nossa função custo

Podemos matematicamente minimizar esta função custo

Escolhendo coeficientes que a minimizem

Matematicamente se pode provar que **estes valores dos coeficientes minimizam** esta função custo

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

Consequências da Equação de Minimização do Custo

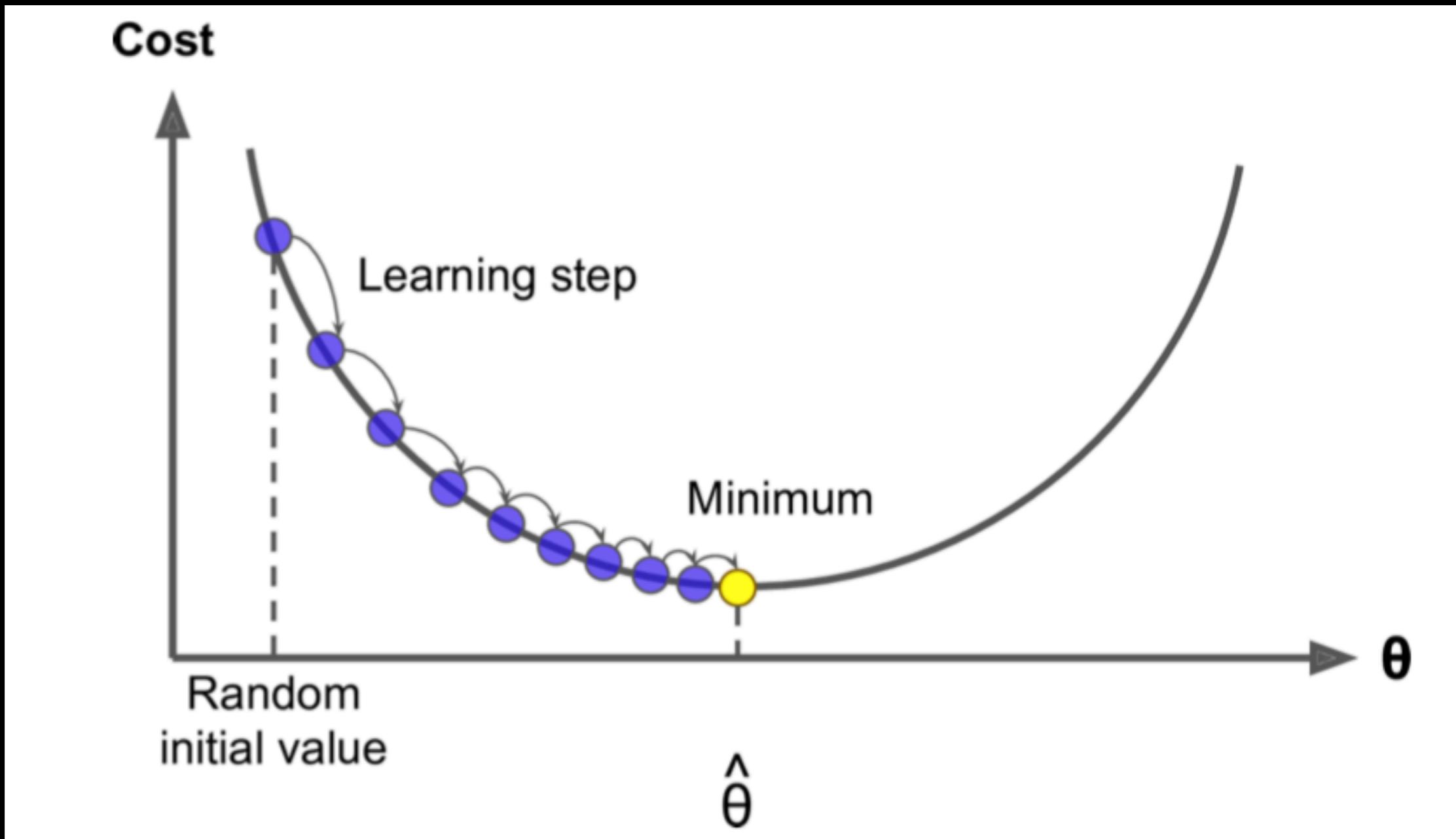
$$\hat{\theta} = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$

- Esta equação depende do produto de matrizes X.
- Sua computação se torna bastante **lenta** quando o **número de variáveis independentes** se torna muito grande (por exemplo, acima de 100 mil como no caso de aplicações em biologia molecular).

Gradient Descent

- GD mede o gradiente (taxa de mudança) local da função de custo/erro em relação ao vetor de coeficientes θ .
- E segue na direção do **gradiente descendente**.
- Uma vez que o gradiente (taxa de mudança) é zero, você atingiu um **mínimo!**

Gradient Descent



Cálculo de GD

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\theta) = \frac{2}{m} \sum_{i=1}^m (\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)}$$

Derivada da função custo em relação a cada variável

$$\nabla_{\theta} \text{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y})$$

O vetor gradiente da função custo com todas as derivadas

n é a taxa de aprendizagem

Cada passo

$$\theta^{(\text{next step})} = \theta - \eta \nabla_{\theta} \text{MSE}(\theta)$$

Tipos de GD

- **GD de Lote (Batch)** = é o que vimos até agora. Ele usa **todo** o conjunto de treinamento para calcular os gradientes em cada etapa. O que o torna muito **lento** quando o conjunto de treinamento é grande.
- **GD Estocástico** = No extremo oposto, Stochastic Gradient Descent escolhe **apenas uma instância aleatória** no conjunto de treinamento **em cada etapa** e calcula os gradientes com base apenas na única instância. Obviamente, isso torna o algoritmo **muito mais rápido**, pois tem dados muito pequenos para manipular em cada iteração. Também **permite** treinar em conjuntos de treinamento enormes, uma vez que apenas uma instância precisa estar na memória em cada iteração!

Sumário: Algoritmos de Regressão Linear

	Muitas instâncias, observações	Muitas variáveis independentes	Variáveis na mesma escala	Classe na Scikit-Learn
Descida de Gradiente Estocástica (SGD)	Rápido	Rápido	Sim	SGDRegressor
Descida de Gradiente de Lote (BGD)	Lento	Rápido	Sim	Nenhuma
Solução pela equação de minimização	Rápido	Lento	Não	LinearRegression

Exercício Prático 1:

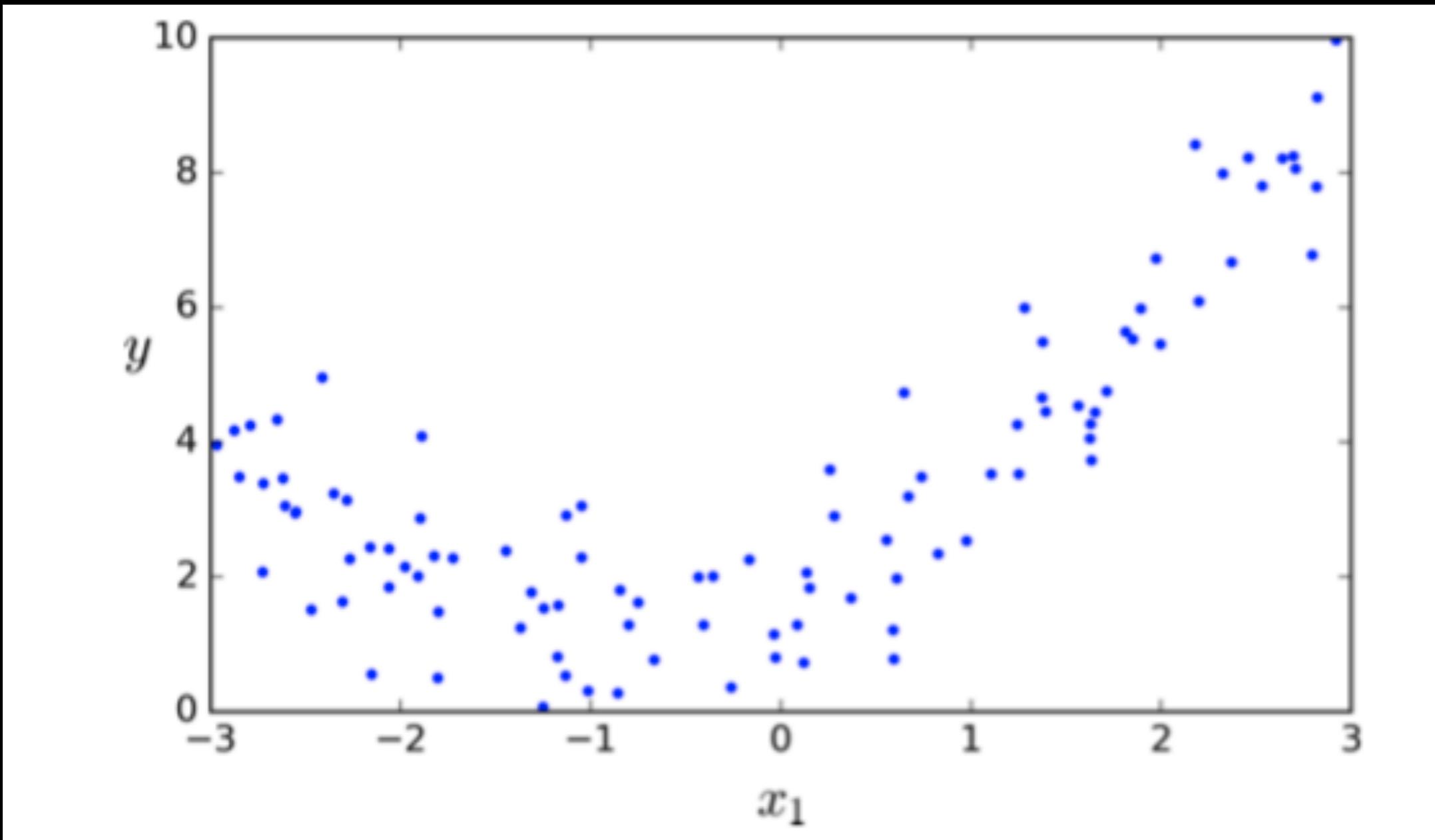
Regressão Linear

**Apresentação de
Solução feita por vocês**

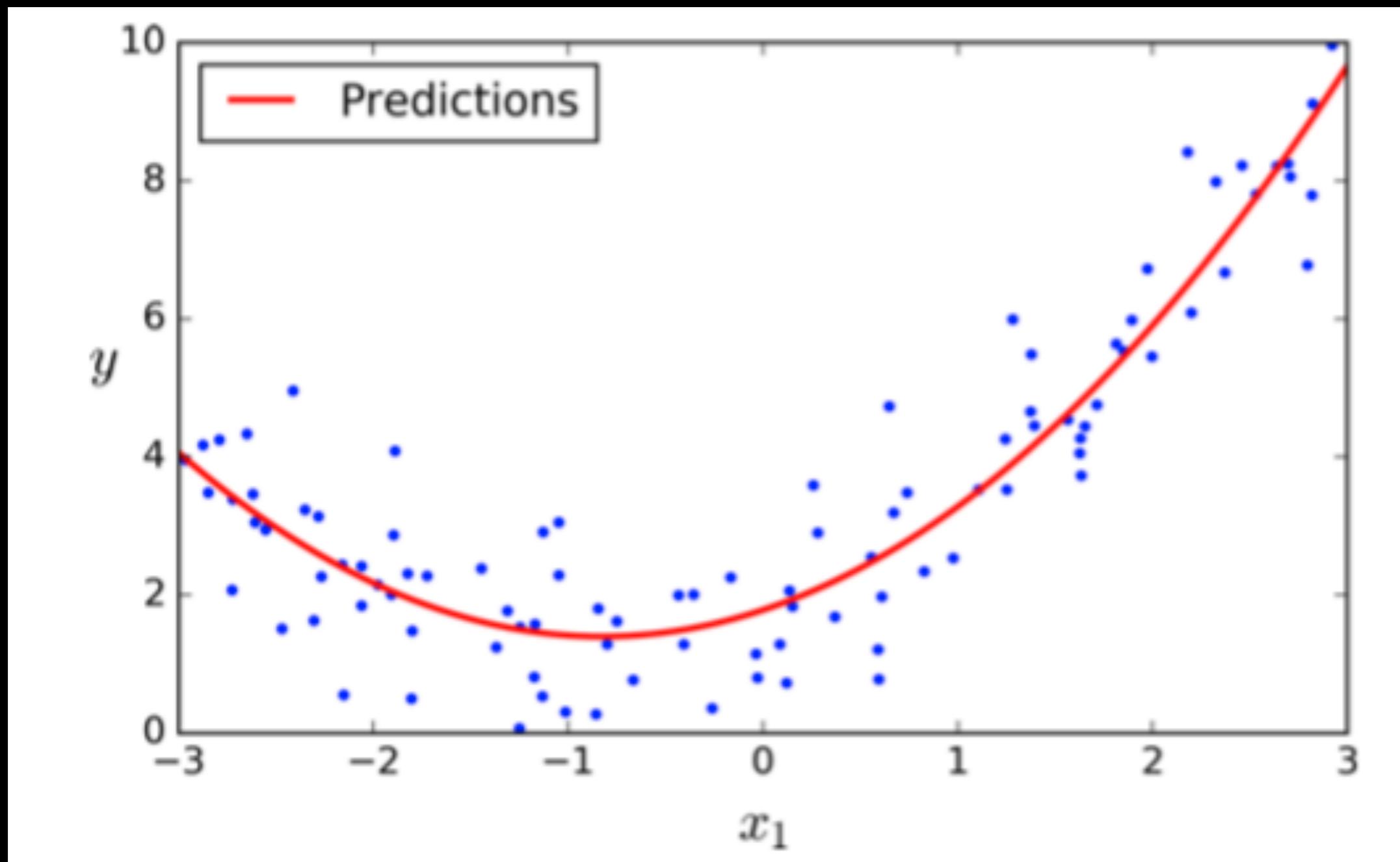
Regressão Polinomial

- Podemos usar também Regressão Linear quando o relacionamento entre as variáveis independentes e a variável objetivo (a dependente) não é “linear”,
- Este modelo funciona também bem em relacionamentos **não lineares**,
- Se o relacionamento for **proporcional!**

Regressão Polinomial



Regressão Polinomial



Regressão Polinomial

- Podemos adicionar graus mais altos a cada variáveis como segundo ou terceiro grau.
- Em Python temos para isto a classe `PolynomialFeatures`.
- Esta classe faz pré-processamento!
- Transformando os dados, adicionando, neste caso, 2o. grau seria bom, o quadrado de cada variável no conjunto de treinamento.

Aprendizagem Supervisionada: Classificação

Ensembles

- Quando você está diante de um problema complexo de **classificação**, diferentes abordagens podem aparecer ao procurar uma solução.
- Embora essas abordagens possam estimar a classificação, às vezes **nenhuma** deles é melhor que o resto.
- Mas um time brilhante pode ser feito de jogadores fracos. A **união** faz a força!

Ensembles

- Ou seja, uma escolha razoável é **mantê-los todos** e, em seguida, criar **um sistema final, integrando as peças**. Este método de diversificação é uma das práticas mais convenientes: **dividir a decisão entre vários sistemas**, a fim de evitar colocar todos os seus ovos em uma cesta.
- Uma vez que eu tenho uma série de estimativas para um caso, como posso **combinar as decisões dos subsistemas N?** Como uma resposta rápida, posso tomar a **média** da decisão e usar isso. Mas existem diferentes maneiras de aproveitar ao máximo os meus subsistemas? **Claro que sim!**

Ensembles

- Vários classificadores com um objetivo comum são chamados de **multiclassificadores**.
- Em Machine Learning, os multiclassificadores são conjuntos de diferentes classificadores que fazem **estimativas que são fundidos**, obtendo um resultado que é **uma combinação deles**.
- Muitos termos são utilizados para se referir a multiclassificadores: multi-modelos, sistemas de classificação múltipla, combinando classificadores, comitês de decisão, **ensembles**, etc. Eles podem ser divididos em dois grupos principais.

Ensembles

- **Métodos de conjuntos (ensemble methods)**: refere-se a conjuntos de sistemas que se combinam para criar um novo sistema usando a **mesma** técnica de aprendizagem. **Bagging** e **Boosting** são os mais usados.
- **Métodos híbridos**: toma um conjunto de aprendedores **diferentes** e os combina usando **novas** técnicas de aprendizado. O empilhamento (**Stacking**) ou a generalização empilhada (**Stacked Generalization**) é um dos principais multiclassificadores (**multiclassifiers**) **híbridos**.

Ensembles

- As principais causas de erro na aprendizagem são devidas ao **ruído, viés e a variância**.
- Ensembles ajudam a **minimizar** esses fatores. Esses métodos são projetados para melhorar a **estabilidade e a precisão** dos algoritmos de Machine Learning.

Bagging e Boosting

- As combinações de vários classificadores **diminuem** a variação, especialmente no caso de classificadores instáveis, e podem produzir uma classificação mais **confiável** do que um único classificador.
- Para usar **Bagging** ou **Boosting**, você deve selecionar **um** algoritmo de aprendizado básico. Por exemplo, se escolhermos uma árvore de classificação (decision tree), Bagging e Boosting consistiria em um **grupo** de árvores **tão grande quanto queremos**.

Bagging e Boosting

- Bagging e Boosting criam N aprendedores, gerando dados adicionais na fase de treinamento.
- N novos conjuntos de dados de treinamento são produzidos por amostragem aleatória com substituição a partir do conjunto original.

Bagging e Boosting

- Amostragem com substituição = algumas observações podem ser repetidas em cada novo conjunto de dados de treinamento.
- Esses conjuntos múltiplos são usados para treinar o mesmo algoritmo do aprendedor e, portanto, diferentes classificadores são produzidos.

Bagging e Boosting

- Neste ponto, começamos a lidar com a principal diferença entre os dois métodos.
- Enquanto o estágio de treinamento é **paralelo** no caso de **Bagging** (ou seja, cada modelo é construído de forma independente),
- o **Boosting** constrói o novo aluno de forma **seqüencial**.

Boosting

- Nos algoritmos de Boosting, cada classificador é treinado em dados, levando em consideração o sucesso dos classificadores anteriores.
- Após cada passo de treinamento, os pesos são redistribuídos. Dados mal classificados ajudam a enfatizar os casos mais difíceis.
- Desta forma, os aprendedores subsequentes se concentrarão neles durante seu treinamento.

Boosting

- Algumas das técnicas de **Boosting** incluem uma condição extra para **manter ou descartar** um único aluno. Por exemplo, no AdaBoost, o mais conhecido, é necessário um erro inferior a 50% para manter o modelo; Caso contrário, a iteração é repetida até alcançar um aluno melhor do que um palpite aleatório.
- As modelagens mais conhecidas de Boosting são: **AdaBoost, XGBoost, GradientBoost**.

Bagging e Boosting

- Como funciona o **estágio de classificação**?
- Para prever a classe de novos dados, só precisamos aplicar os N aprendedores nas novas observações.
- Em **Bagging**, o resultado é obtido pela **média** das respostas dos N aprendedores (ou **voto maioritário**).
- No entanto, **Boosting** atribui um **segundo** conjunto de pesos, desta vez para os N classificadores, a fim de obter uma **média ponderada** de suas estimativas.

Qual é o melhor? Bagging ou Boosting

- Não há um vencedor absoluto. Depende dos dados, da simulação e das circunstâncias.
- Ambos Bagging e Boosting **diminuem a variância de sua estimativa única**, pois combinam várias estimativas de diferentes modelos. Portanto, o resultado pode ser um modelo com maior **estabilidade**.

Qual é o melhor? Bagging ou Boosting

- Se o problema com o modelo único é que ele tem um desempenho muito baixo, o Bagging raramente obterá um melhor viés. No entanto, o **Boosting** pode gerar um **modelo combinado com erros menores**, pois ele optimiza as vantagens e reduz os problemas do modelo único.
- Em contrapartida, se a dificuldade do modelo único for **overfitting**, então o **Bagging** é a melhor opção. **Boosting** por sua parte não ajuda a evitar o excesso de ajuste (overfitting); na verdade, essa técnica é confrontada com esse problema. Por este motivo, o **Bagging** é efetivo com mais freqüência do que Boosting!

Comparação

Em Comum	Diferenças
Ambos são métodos de conjunto para obter N aprendedores a partir de 1 aluno	Mas, enquanto eles são construídos de forma independente para Bagging, o Boosting tenta adicionar novos modelos que funcionam bem, onde modelos anteriores falharam.
Ambos geram vários conjuntos de dados de treinamento por amostragem aleatória	Mas apenas Boosting determina pesos para os dados para inclinar a escala em favor dos casos mais difíceis.
Ambos tomam a decisão final pela média dos N aprendedores (ou a maioria deles)	Mas é uma média igualmente ponderada para Bagging e uma média ponderada para Boosting, mais peso é dado para aqueles com melhor desempenho em dados de treinamento
Ambos são bons em reduzir a variância e proporcionar maior estabilidade	Mas apenas o Boosting tenta reduzir o viés. Por outro lado, Bagging pode resolver o problema de overfitting, enquanto o Boosting pode aumentá-lo .

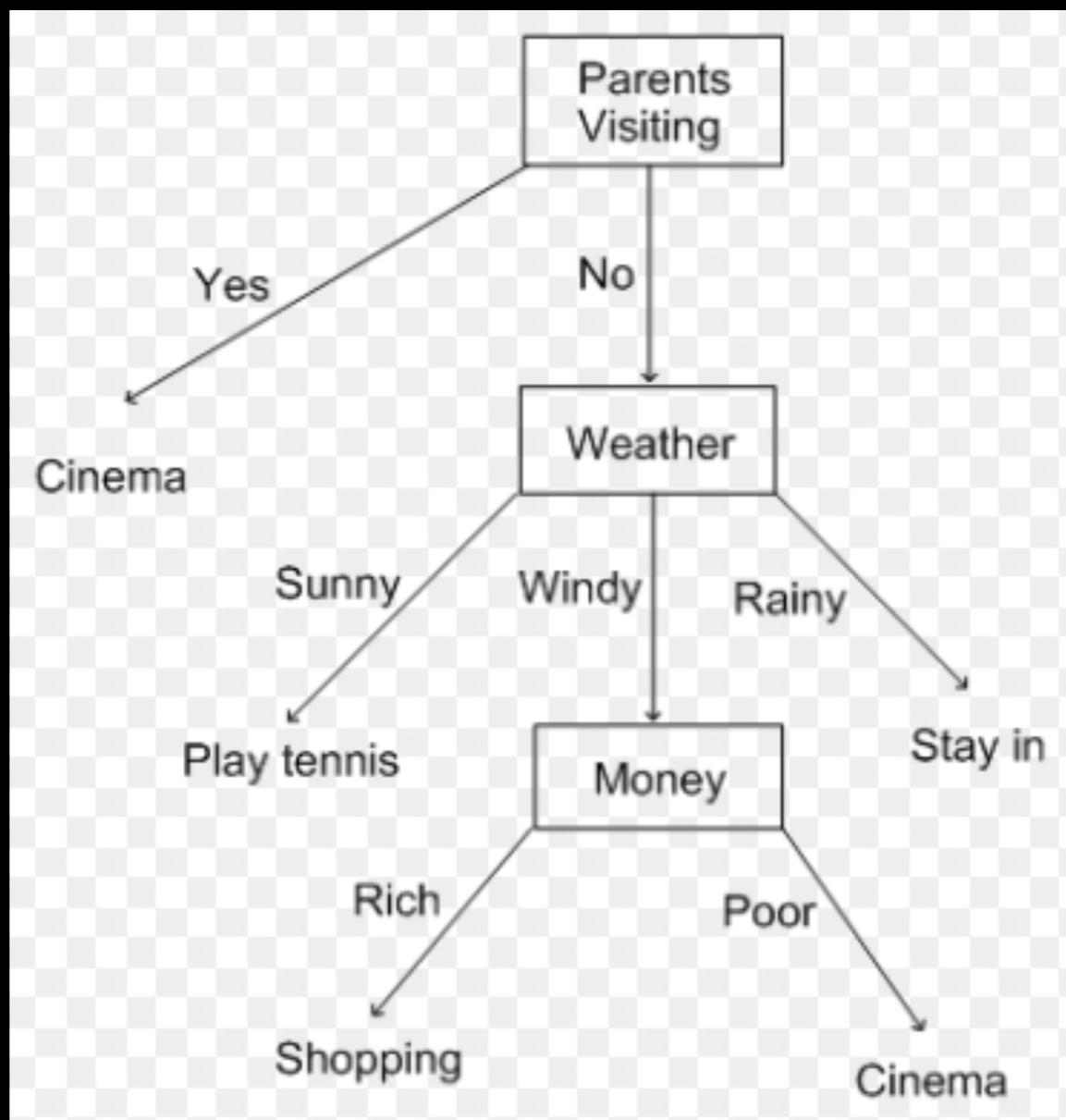
Random Forests (Árvores Aleatórias)

- É um ensemble de Árvores de Decisões (decision trees).
- Em geral treinado com o método de Bagging.

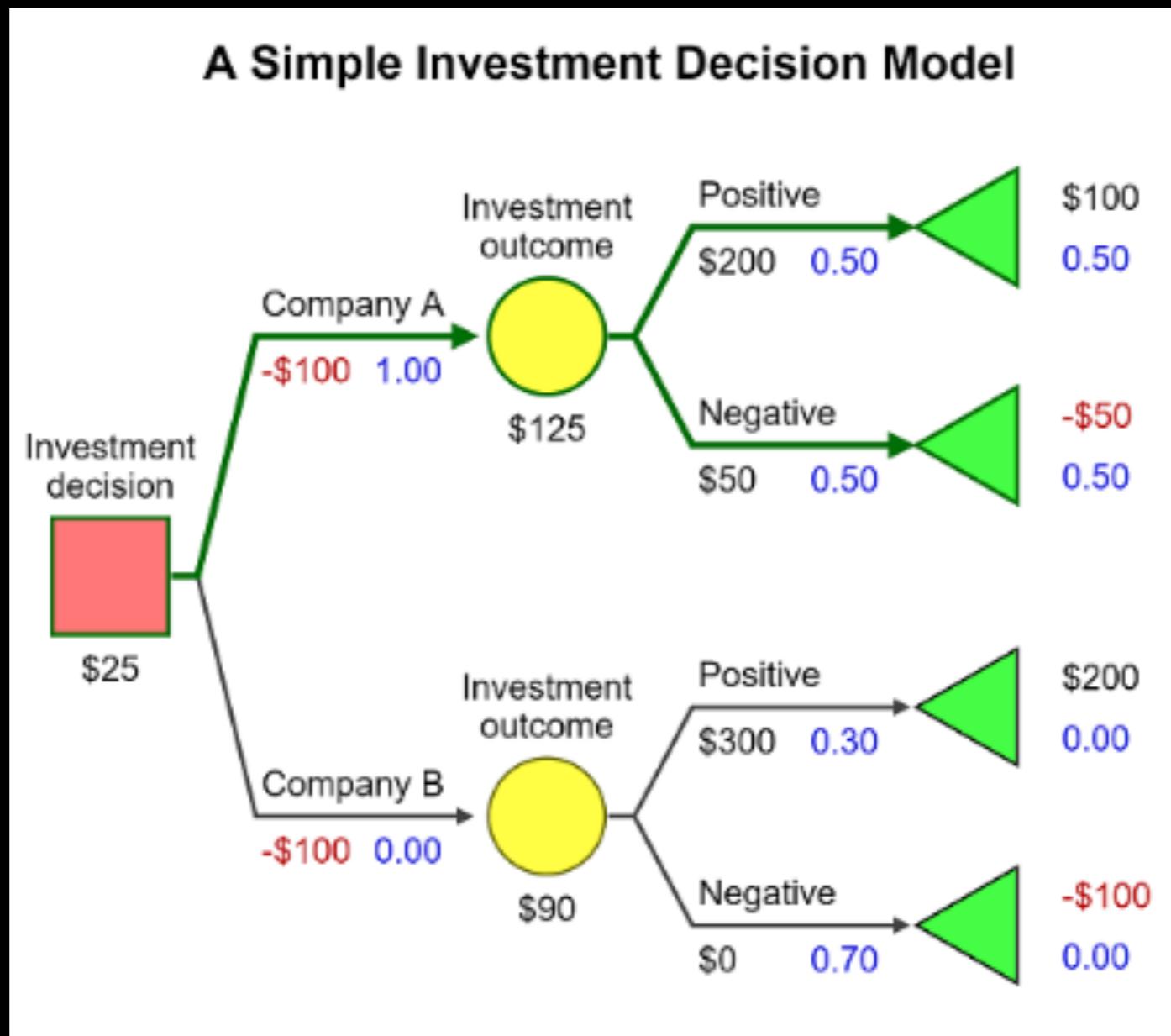
Decision Trees (Árvores de Decisão)

- Simples
- Interpretáveis
- Simula um processo de decisão humano

Decision Trees (Árvores de Decisão)



Decision Trees (Árvores de Decisão)



Decision Trees

- ```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
```
- ```
iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target
```
- ```
tree_clf = DecisionTreeClassifier(max_depth=2)
```
- ```
tree_clf.fit(X, y)
```

Random Forests (Árvores Aleatórias)

- O seguinte código treina um **classificador** de Random Forests.
- com **500 árvores**,
- cada limitada a um máximo de **16** nós (nodes) folhas em todos cores de CPU disponíveis.

Random Forests (Árvores Aleatórias)

- `from sklearn.ensemble import RandomForestClassifier`
- `rfc_md = RandomForestClassifier(n_estimators=500,
max_leaf_nodes=16, n_jobs=-1)`
- `rfc_md.fit(X_train, y_train)`
- `y_pred_rf = rfc_md.predict(X_test)`

Random Forests (Árvores Aleatórias)

- `from sklearn.datasets import load_iris`
- `iris = load_iris()`
- `rfc_m = RandomForestClassifier(n_estimators=500,
n_jobs=-1)
rfc_m.fit(iris["data"], iris["target"])`
- `for name, score in zip(iris["feature_names"],
rfc_m.feature_importances_):
print(name, score)`

Random Forests

Exercício Prático 1

- Use o conjunto de dados **Red Wines** e faça uma predição de sua qualidade usando **Decision Trees** e **Random Forests**. Compare os resultados.
- Este conjunto de dados está no Github no diretório day3.
- Divida o conjunto em **Treinamento** (80%) e **Teste** (20%).
- Use **Validação Cruzada**.
- Faça uma **Tabela de Confusão**.

Random Forests

Exercício Prático 1

- Use o conjunto de dados **White Wines** e faça uma predição de sua qualidade usando **Decision Trees** e **Random Forests**. Compare os resultados.
- Este conjunto de dados está no Github no diretório day3.
- Divida o conjunto em **Treinamento** (80%) e **Teste** (20%).
- Use **Validação Cruzada**.
- Faça uma **Tabela de Confusão**.