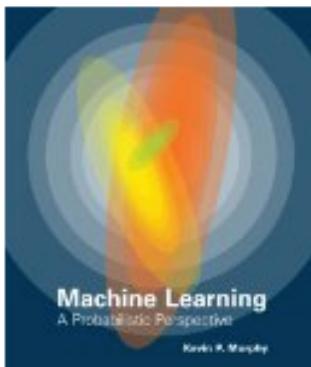


Recommendations for You in Books



[Machine Learning: A Probabilistic...](#)

› Kevin P. Murphy

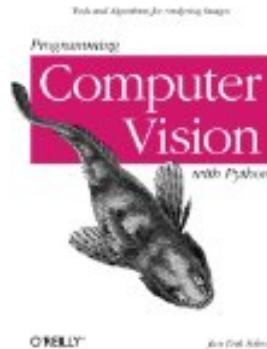
Hardcover

★★★★★ (25)

\$90.00 **\$77.41**

[Why recommended?](#)

› [See more recommendations](#)



[Programming Computer Vision with...](#)

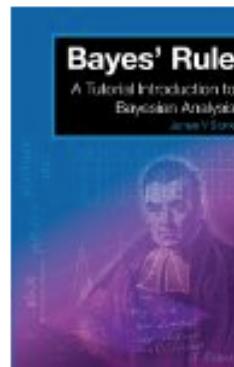
› Jan Erik Solem

Paperback

★★★★★ (9)

\$39.99 **\$26.38**

[Why recommended?](#)



[Bayes' Rule: A Tutorial Introduction...](#)

› James V. Stone

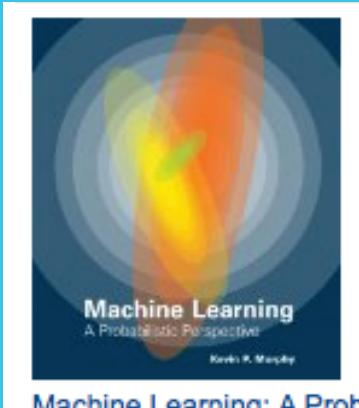
Paperback

★★★★★ (9)

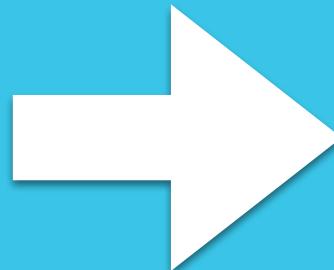
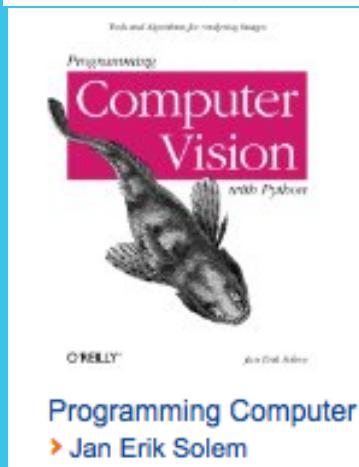
\$29.95 **\$26.96**

[Why recommended?](#)

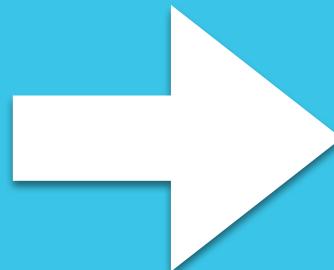
Previsão de nota



Machine Learning: A Prob
► Kevin P. Murphy



8,5
(oito e meio)



7,0
(sete)

Modelos preditivos

Régressão
predizer um valor

Regressão linear

- Modelar o relacionamento entre

(univariada)

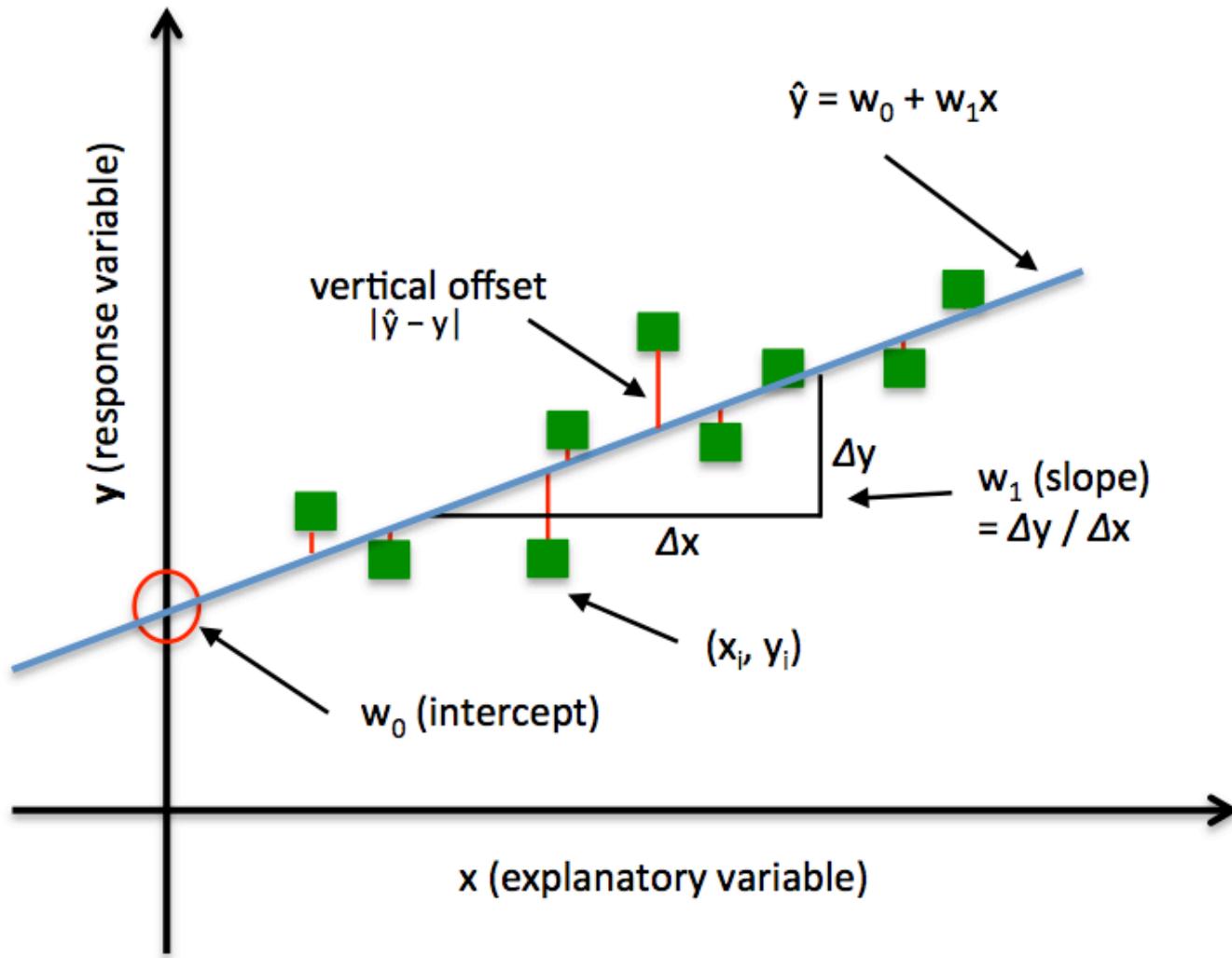
Variável explicativa (*explanatory*)

$$y = w_0 + w_1 + x$$

Variável de resposta (*response*)

$$y = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \sum_{i=0}^m w_i x_i = \mathbf{w}^T \mathbf{x}$$

(multivariada)



Exemplo: prever preços de casas

UCI: housing.data, 506 exemplos (subúrbio de Boston)

(explanatories)

(response)

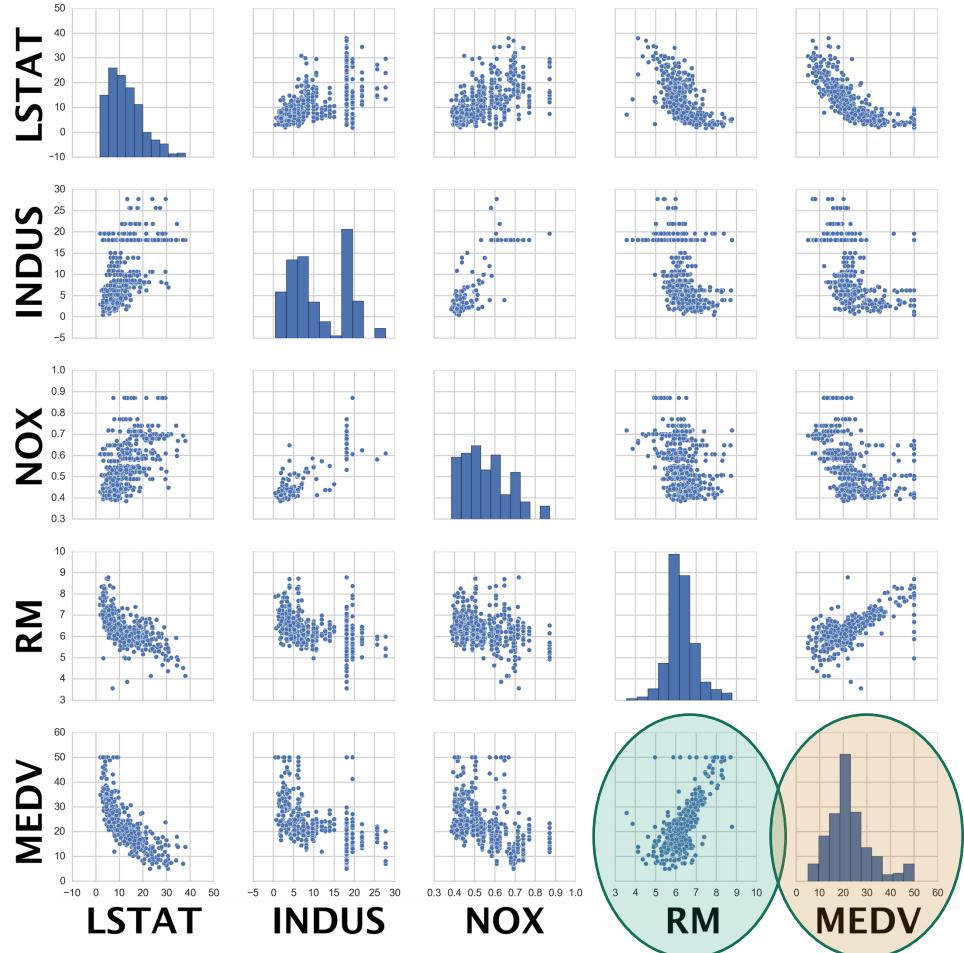
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

Matriz de dispersão

- Visualizar as correlações entre pares de características
 - Como dados estão distribuídos
 - Presença de ruídos

Relação linear entre RM e MEDV

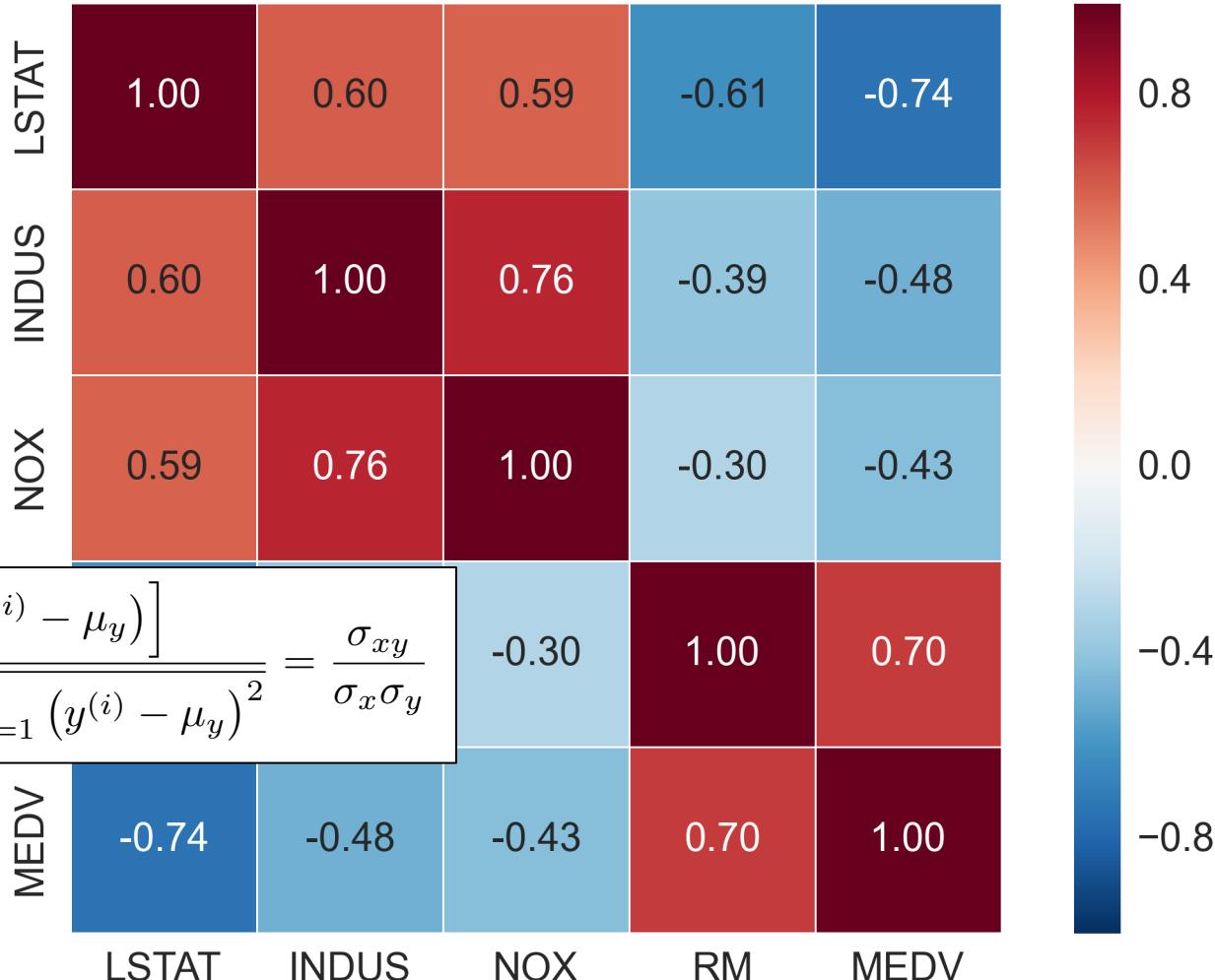
MEDV tem distribuição Normal,
mas contém ruídos



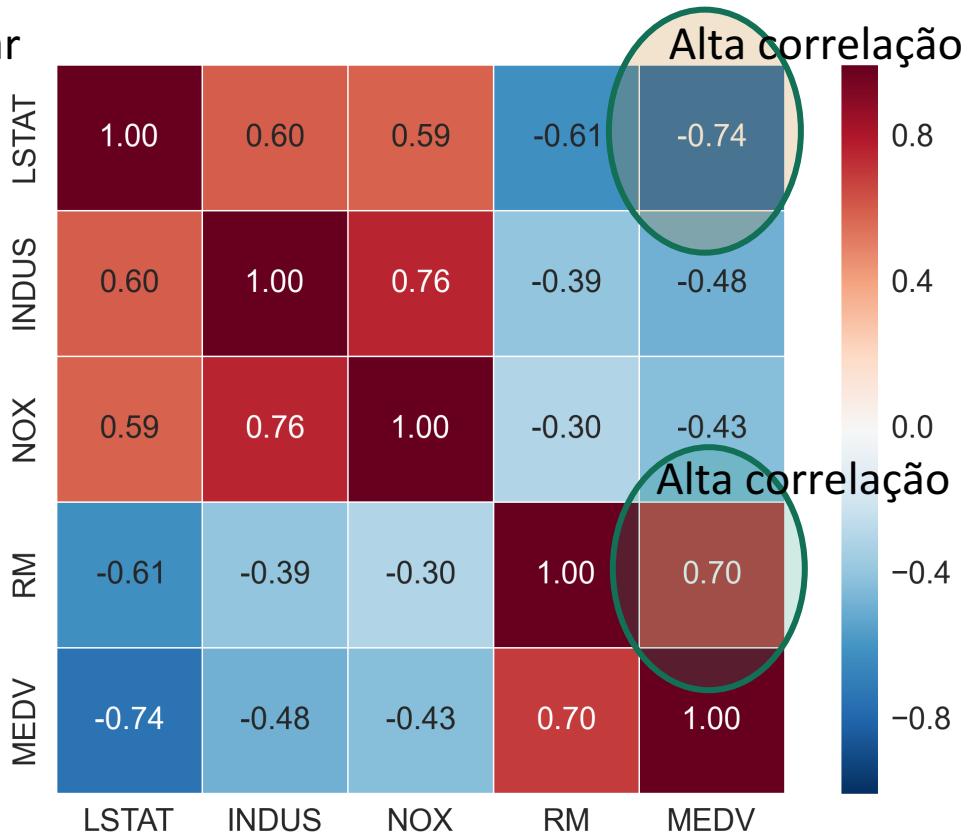
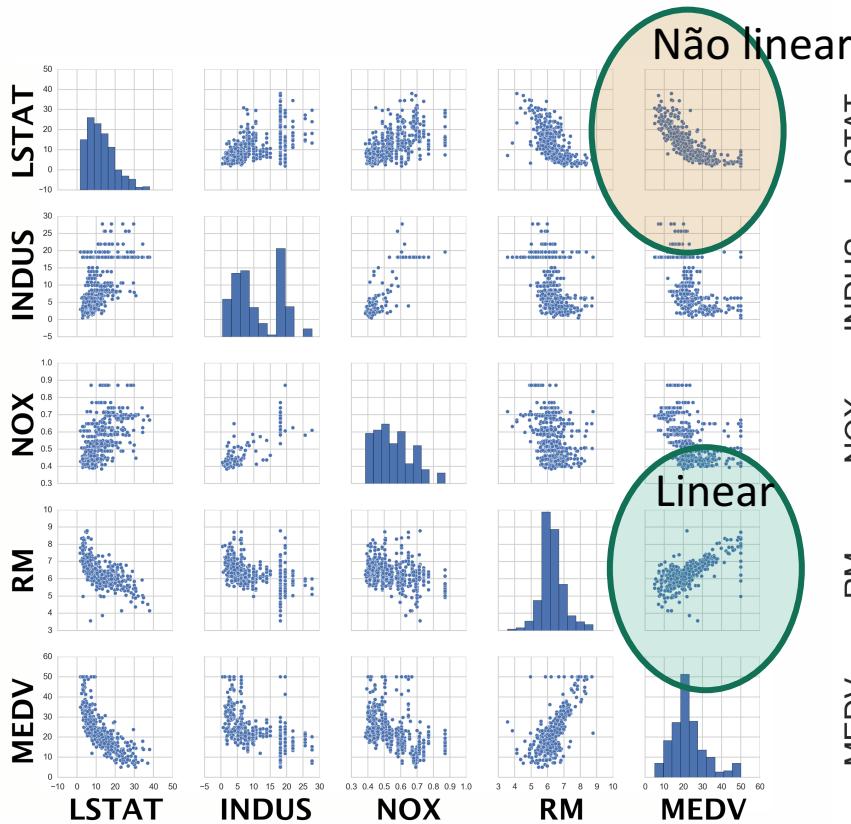
Matriz de correlação

- Quantificar a dependência linear entre pares de características
- r (coef. Pearson) $\in [-1, +1]$

$$r = \frac{\sum_{i=1}^n [(x^{(i)} - \mu_x)(y^{(i)} - \mu_y)]}{\sqrt{\sum_{i=1}^n (x^{(i)} - \mu_x)^2} \sqrt{\sum_{i=1}^n (y^{(i)} - \mu_y)^2}} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$



Qual considerar p/ modelo de regressão linear univariada?



Solução da regressão

- ADALINE

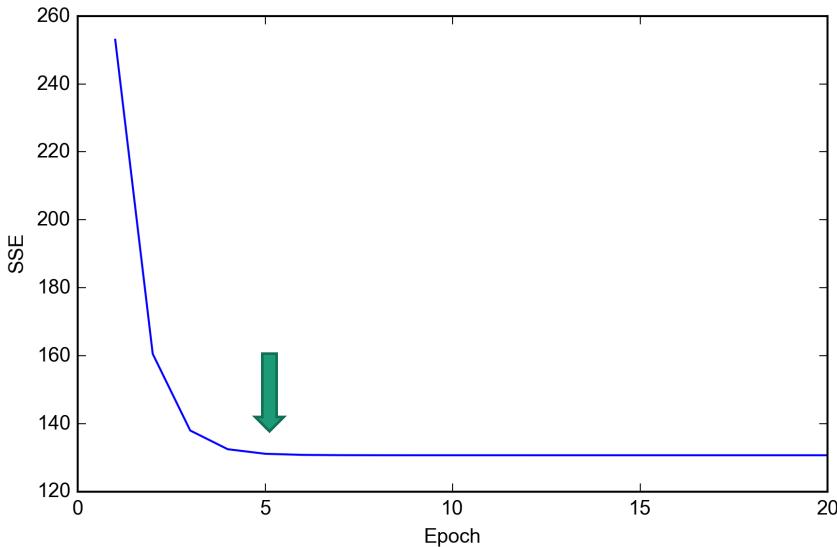
- Uso de GD (ou SGD) para minimizar função de custo

$$J(w) = \frac{1}{2} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$
$$\hat{y} = \mathbf{w}^T \mathbf{x}$$

```

1 class LinearRegressionGD(object):
2
3     def __init__(self, eta=0.001, n_iter=20):
4         self.eta = eta
5         self.n_iter = n_iter
6
7     def fit(self, X, y):
8         self.w_ = np.zeros(1 + X.shape[1])
9         self.cost_ = []
10
11    for i in range(self.n_iter):
12        output = self.net_input(X)
13        errors = (y - output)
14        self.w_[1:] += self.eta * X.T.dot(errors)
15        self.w_[0] += self.eta * errors.sum()
16        cost = (errors**2).sum() / 2.0
17        self.cost_.append(cost)
18    return self
19
20    def net_input(self, X):
21        return np.dot(X, self.w_[1:]) + self.w_[0]
22
23    def predict(self, X):
24        return self.net_input(X)

```



```

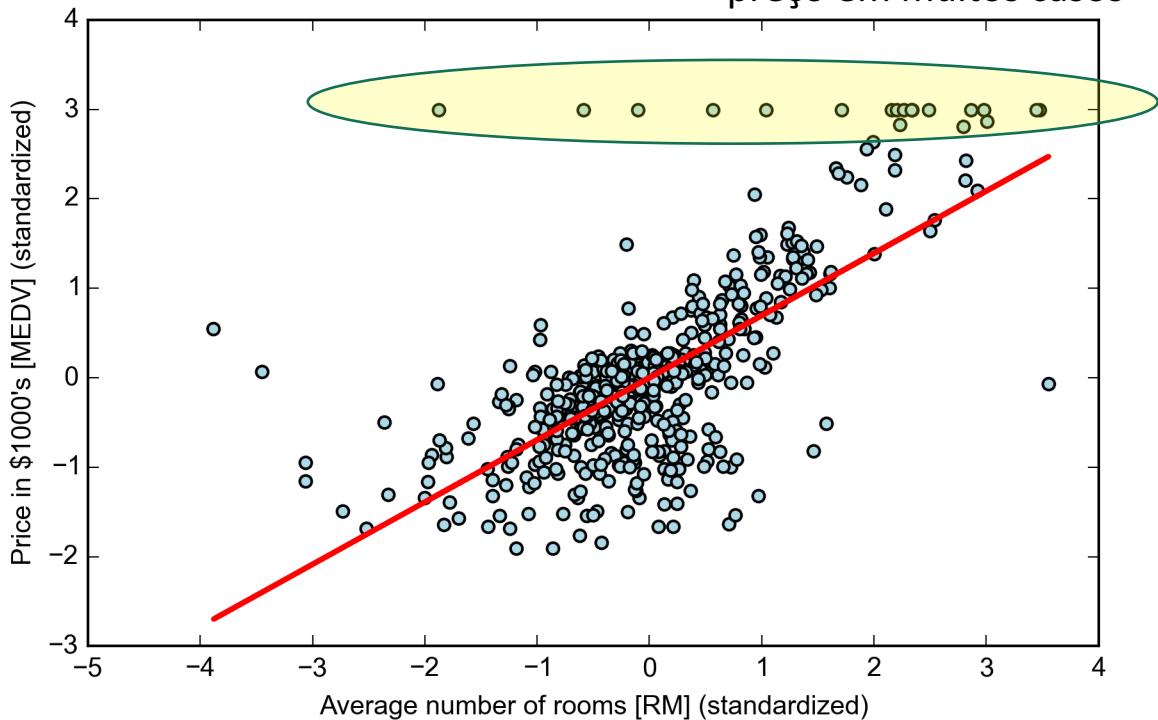
1 from sklearn.preprocessing import StandardScaler
2 X = df[['RM']].values
3 y = df['MEDV'].values
4 sc_x = StandardScaler()
5 sc_y = StandardScaler()
6 X_std = sc_x.fit_transform(X)
7 y_std = sc_y.fit_transform(y)
8 lr = LinearRegressionGD()
9 lr.fit(X_std, y_std)

```

Regressão
univariada

Quão bem a linha de regressão se ajusta aos dados de treinamento...

RM não explica o preço em muitos casos



```
1 print('Slope: %.3f' % lr.w_[1])
2 print('Intercept: %.3f' % lr.w_[0])
```

Slope: 0.695
Intercept: -0.000

```
1 num_rooms_std = sc_x.transform([5.0])
2 price_std = lr.predict(num_rooms_std)
3 print("Price in $1000's: %.3f" % sc_y
```

Price in \$1000's: 10.840

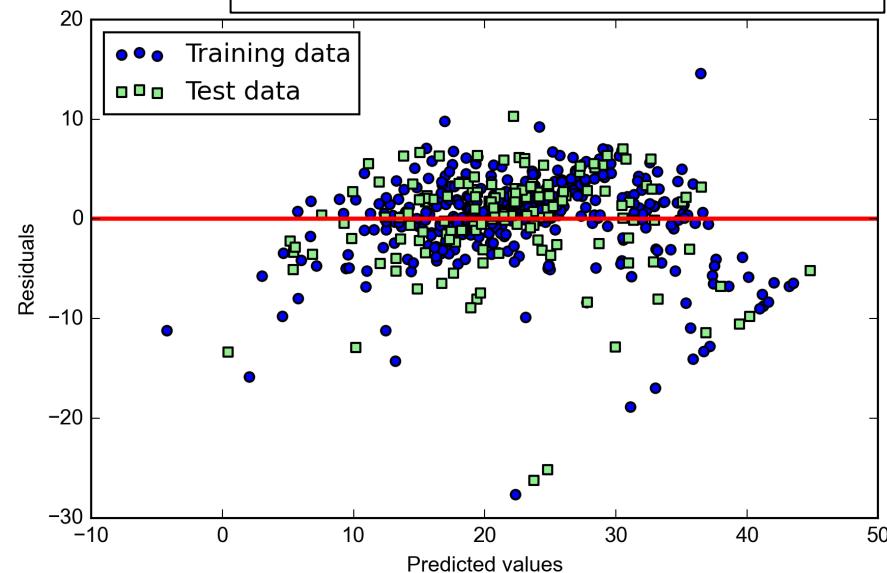
Avaliando a performance do modelo

```
1 from sklearn.cross_validation import train_test_split  
2  
3 X = df.iloc[:, :-1].values  
4 y = df['MEDV'].values  
5  
6 X_train, X_test, y_train, y_test = train_test_split(  
7     X, y, test_size=0.3, random_state=0)  
8  
9 slr = LinearRegression()  
10  
11 slr.fit(X_train, y_train)  
12 y_train_pred = slr.predict(X_train)  
13 y_test_pred = slr.predict(X_test)
```

Regressão multivariada

Residual plot

(distâncias verticais entre o valor real e o previsto)



Avaliando a performance do modelo

```
1 from sklearn.cross_validation import train_test_split
2
3 X = df.iloc[:, :-1].values
4 y = df['MEDV'].values
5
6 X_train, X_test, y_train, y_test = train_test_split(
7     X, y, test_size=0.3, random_state=0)
8
9 slr = LinearRegression()
10
11 slr.fit(X_train, y_train)
12 y_train_pred = slr.predict(X_train)
13 y_test_pred = slr.predict(X_test)
```

Regressão multivariada

$$MSE = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

MSE (Mean Squared Error)

```
1 from sklearn.metrics import r2_score
2 from sklearn.metrics import mean_squared_error
3 print('MSE train: %.3f, test: %.3f' % (
4     mean_squared_error(y_train, y_train_pred),
5     mean_squared_error(y_test, y_test_pred)))
6 print('R^2 train: %.3f, test: %.3f' % (
7     r2_score(y_train, y_train_pred),
8     r2_score(y_test, y_test_pred)))
```

MSE train: 19.958, test: 27.196

Exemplo: prever preços de casas

UCI: housing.data, 506 exemplos (subúrbio de Boston)

(explanatories)

(response)

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

Regressão linear: uso da regularização

- Abordagem **Ridge Regression**

$$J(\mathbf{w})_{ridge} = \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 + \lambda \|\mathbf{w}\|_2^2$$

L2

$$\lambda \|\mathbf{w}\|_2^2 = \lambda \sum_{j=1}^m w_j^2$$

Assim como nos modelos para classificação, ao aumentar o valor de λ , comprimimos os pesos aprendidos.

Regressão linear: uso da regularização

- Abordagem Least Absolute Shrinkage and Selection Operator (LASSO)

$$J(\mathbf{w})_{LASSO} = \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 + \lambda \|\mathbf{w}\|_1$$

L1

$$\lambda \|\mathbf{w}\|_1 = \lambda \sum_{j=1}^m |w_j|$$

```
1 from sklearn.linear_model import Lasso
2 lasso = Lasso(alpha=0.5)
3 lasso.fit(X_train, y_train)
4 y_train_pred = lasso.predict(X_train)
5 y_test_pred = lasso.predict(X_test)
6 print(lasso.coef_)
```

```
[-0.0926693  0.04820685 -0.01406582  0.          -0.          2.44619324
 -0.00279111 -0.99274477  0.20992275 -0.01388192 -0.85539024  0.00750462
 -0.62386874]
```

Regularização mais forte. É possível que ao final do treinamento, muitos pesos possuam valor zero, fazendo com que LASSO seja também útil como selecionador de features.

Regressão linear: uso da regularização

- Abordagem **Elastic Net**

$$J(\mathbf{w})_{ElasticNet} = \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 + \lambda_1 \sum_{j=1}^m w_j^2 + \lambda_2 \sum_{j=1}^m |w_j|$$

L1 + L2

Combina Ridge com LASSO. Objetivo é usar L1 para permitir geração de esparsidade e L2 para superar limitação do LASSO no tocante ao número de variáveis selecionadas.

Regressão polinomial

- A regressão linear **assume relação de linearidade** entre variáveis explanatórias e variáveis de resposta
- Uma forma de lidar com esse pressuposto é tentar modelar uma **relação não linear** é **adicionar termos polinomiais** (d = grau do polinômio)

$$y = w_0 + w_1x + w_2x^2 + \cdots + w_dx^d$$

Exemplo simples de Regressão polinomial

- Adicionando um termo polinomial (ex: grau 2) e aplicando a transformação nas características

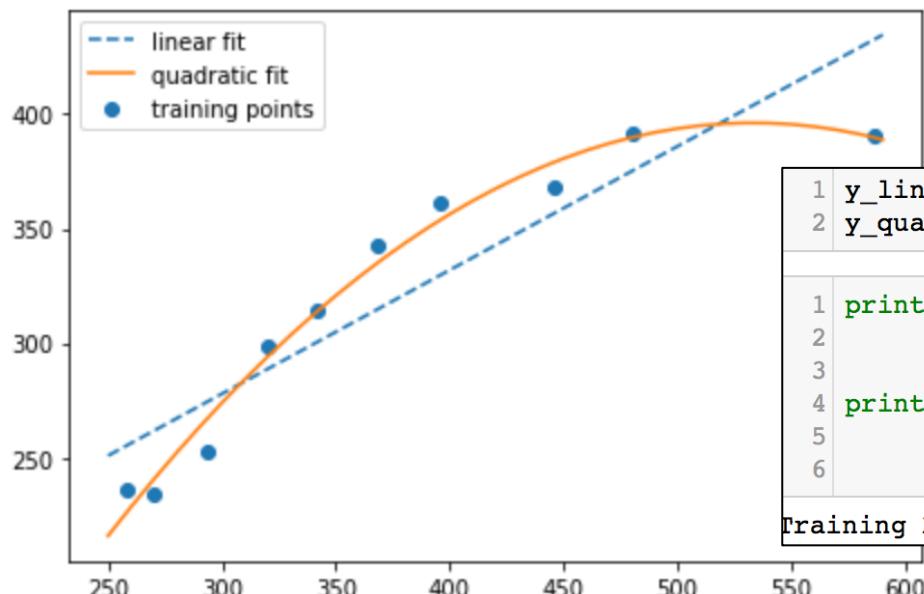
```
1 from sklearn.preprocessing import PolynomialFeatures
2
3 X = np.array([258.0, 270.0, 294.0,
4                 320.0, 342.0, 368.0,
5                 396.0, 446.0, 480.0, 586.0])[:, np.newaxis]
6
7 y = np.array([236.4, 234.4, 252.8,
8                 298.6, 314.2, 342.2,
9                 360.8, 368.0, 391.2,
10                390.8])
11
12 lr = LinearRegression()
13 pr = LinearRegression()
14 quadratic = PolynomialFeatures(degree=2)
15 X_quad = quadratic.fit_transform(X)
```

Exemplo simples de Regressão polinomial

- Treinando um modelo com adição das características quadráticas

```
1 # fit linear features
2 lr.fit(X, y)
3 X_fit = np.arange(250,600,10)[:, np.newaxis]
4 y_lin_fit = lr.predict(X_fit)
5
6 # fit quadratic features
7 pr.fit(X_quad, y)
8 y_quad_fit = pr.predict(quadratic.fit_transform(X_fit))
```

Exemplo simples de Regressão polinomial



- Modelo linear vs. quadrático

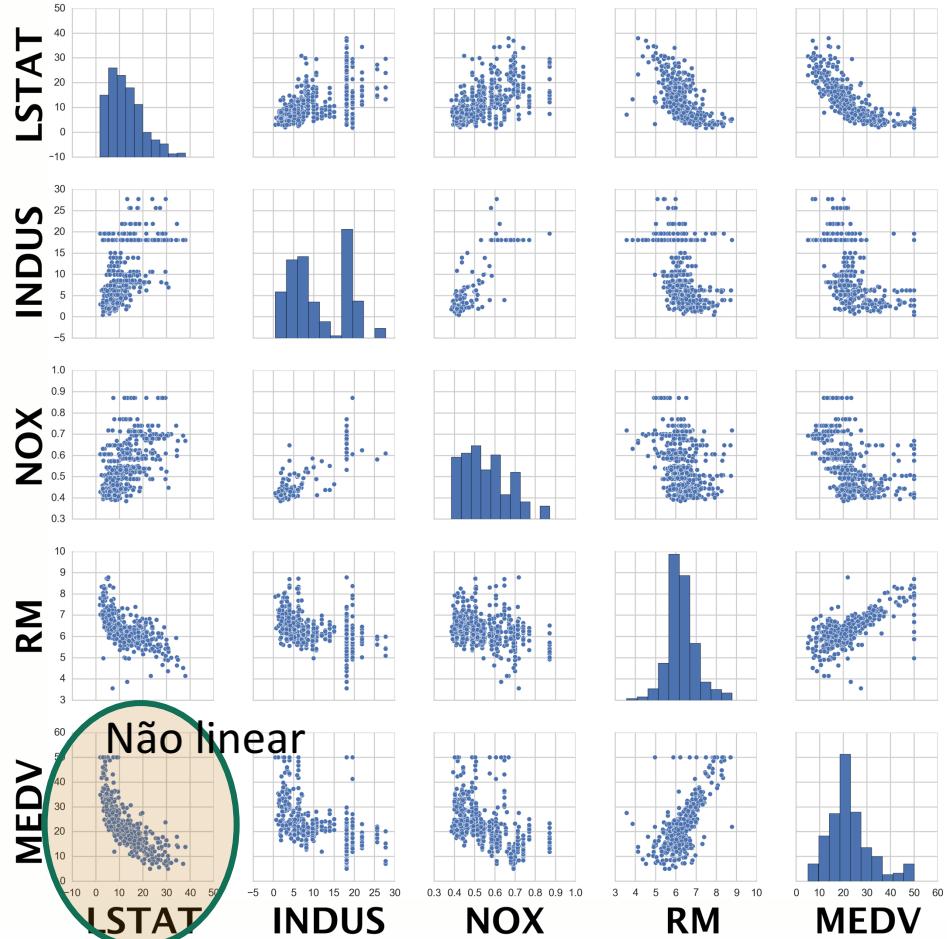
```
1 y_lin_pred = lr.predict(X)
2 y_quad_pred = pr.predict(X_quad)

1 print('Training MSE linear: %.3f, quadratic: %.3f' % (
2     mean_squared_error(y, y_lin_pred),
3     mean_squared_error(y, y_quad_pred)))
4 print('Training R^2 linear: %.3f, quadratic: %.3f' % (
5     r2_score(y, y_lin_pred),
6     r2_score(y, y_quad_pred)))
```

Training MSE linear: 569.780, quadratic: 61.330

MSE

Matriz de dispersão



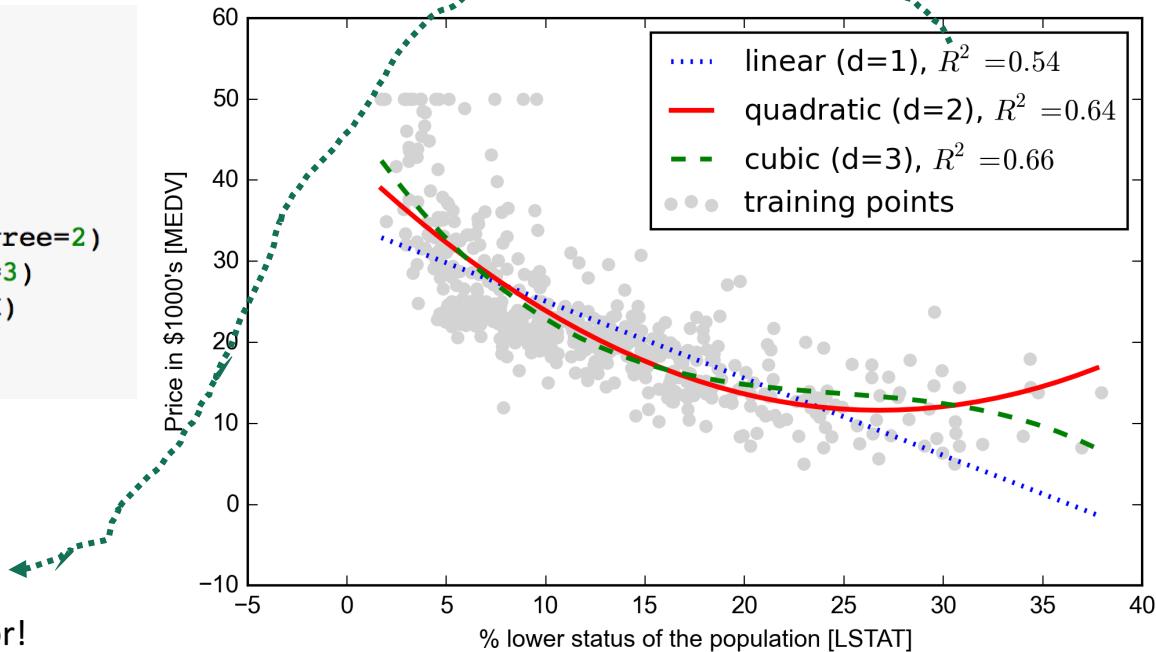
Aplicando para o dataset housing.data

- Características Lineares vs. Quadráticas vs. Cúbicas

```
1 x = df[['LSTAT']].values
2 y = df['MEDV'].values
3
4 regr = LinearRegression()
5
6 # create quadratic features
7 quadratic = PolynomialFeatures(degree=2)
8 cubic = PolynomialFeatures(degree=3)
9 X_quad = quadratic.fit_transform(X)
10 X_cubic = cubic.fit_transform(X)
11
```

$$R^2 = 1 - \frac{MSE}{Var(y)}$$

Versão normalizada do MSE:
quanto mais próximo de 1, melhor!





Hendrik Macedo

Escreve sobre Inteligência Artificial no Saense.

<http://www.saense.com.br/autores/artigos-publicados-por-hendrik-macedo/>