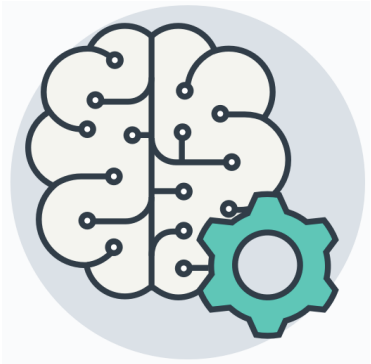
Aprendizado de Máquina

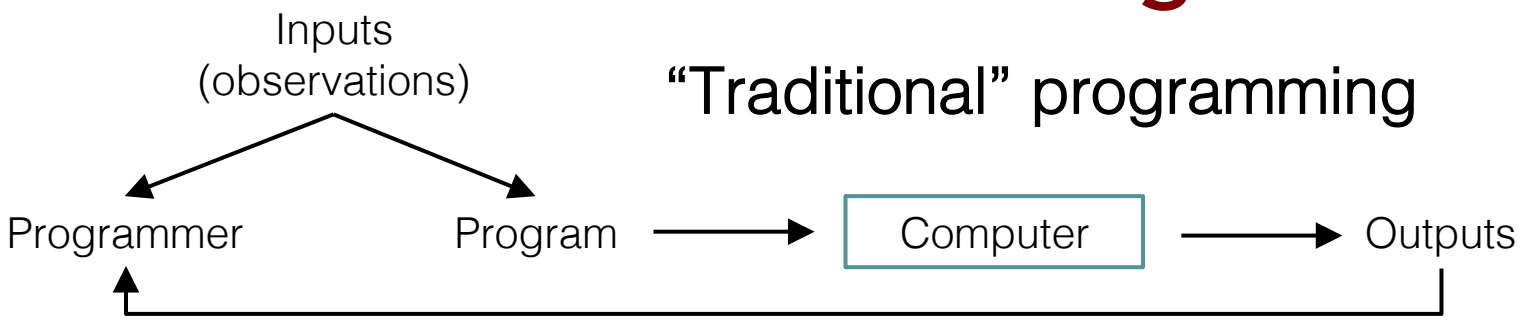
Visão Geral



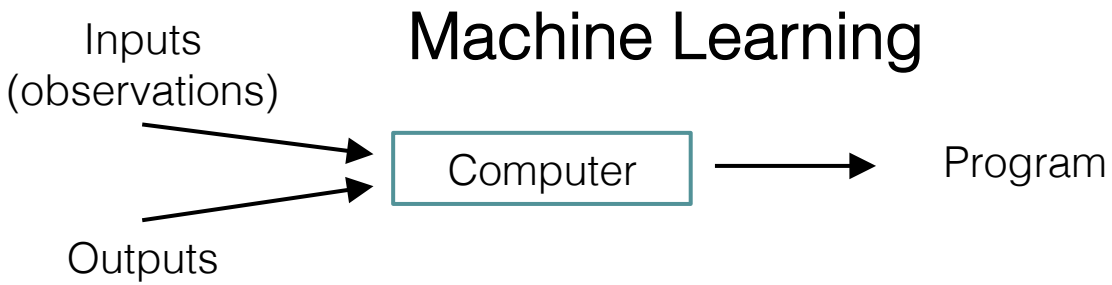
Prof. Regis Pires Magalhães

regismagalhaes@ufc.br - http://bit.ly/ufcregis

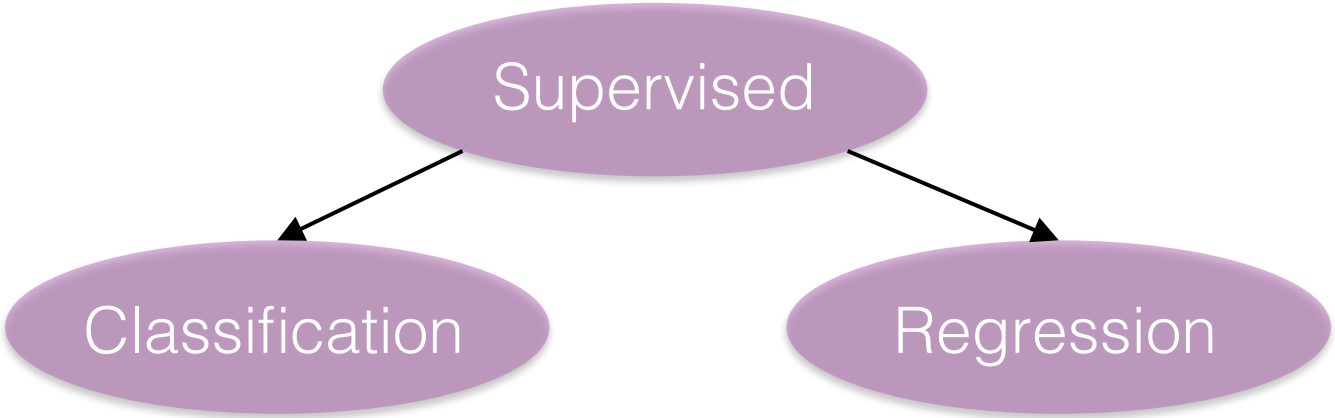
What is Machine Learning?

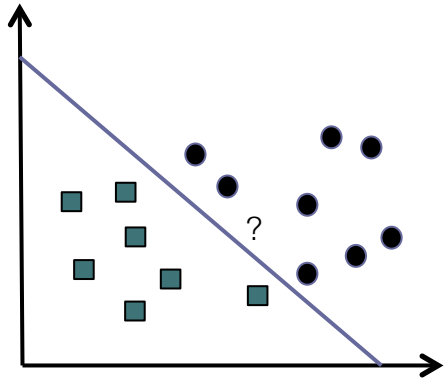


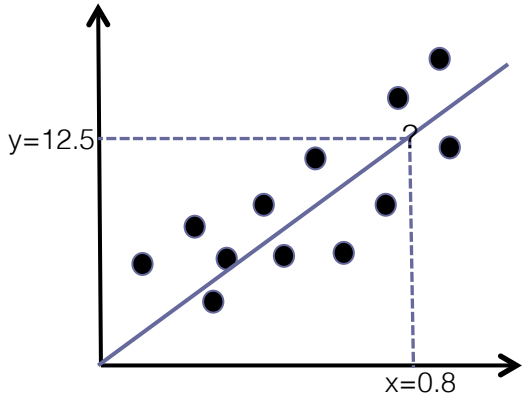
Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.
-- Arthur Samuel (1959)



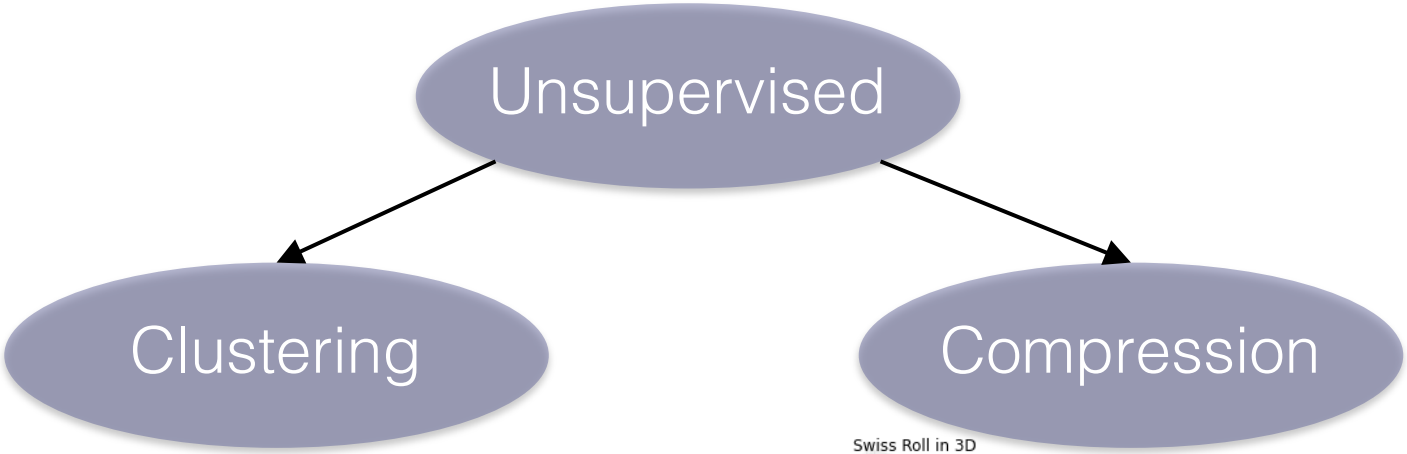
Supervised Learning

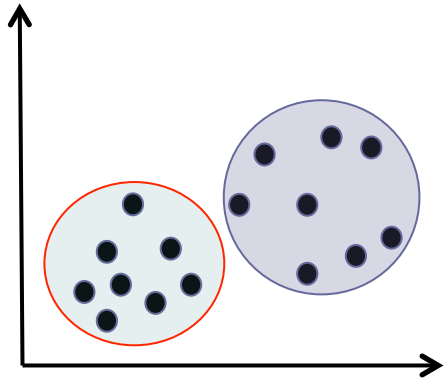


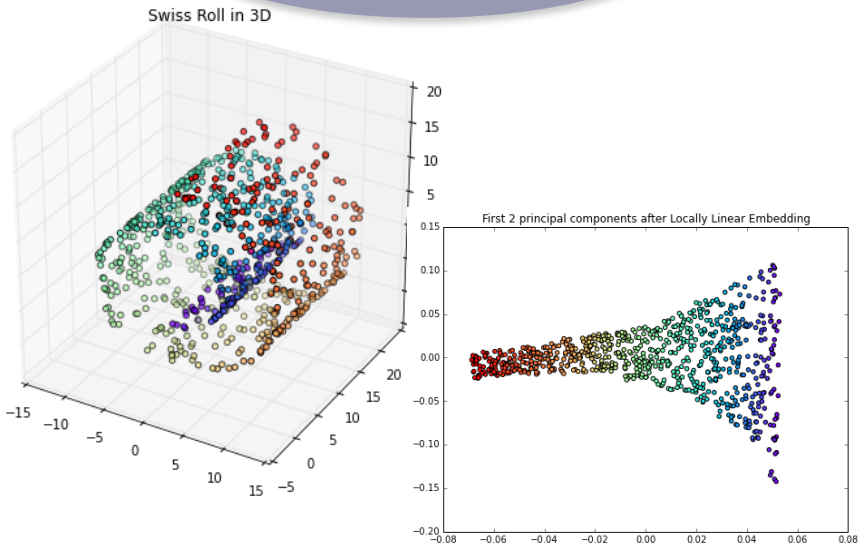




Unsupervised Learning







Statistical Learning Perspective

- Given some input variables (input), what is the predicted output variable (output).

Output = f(Input)

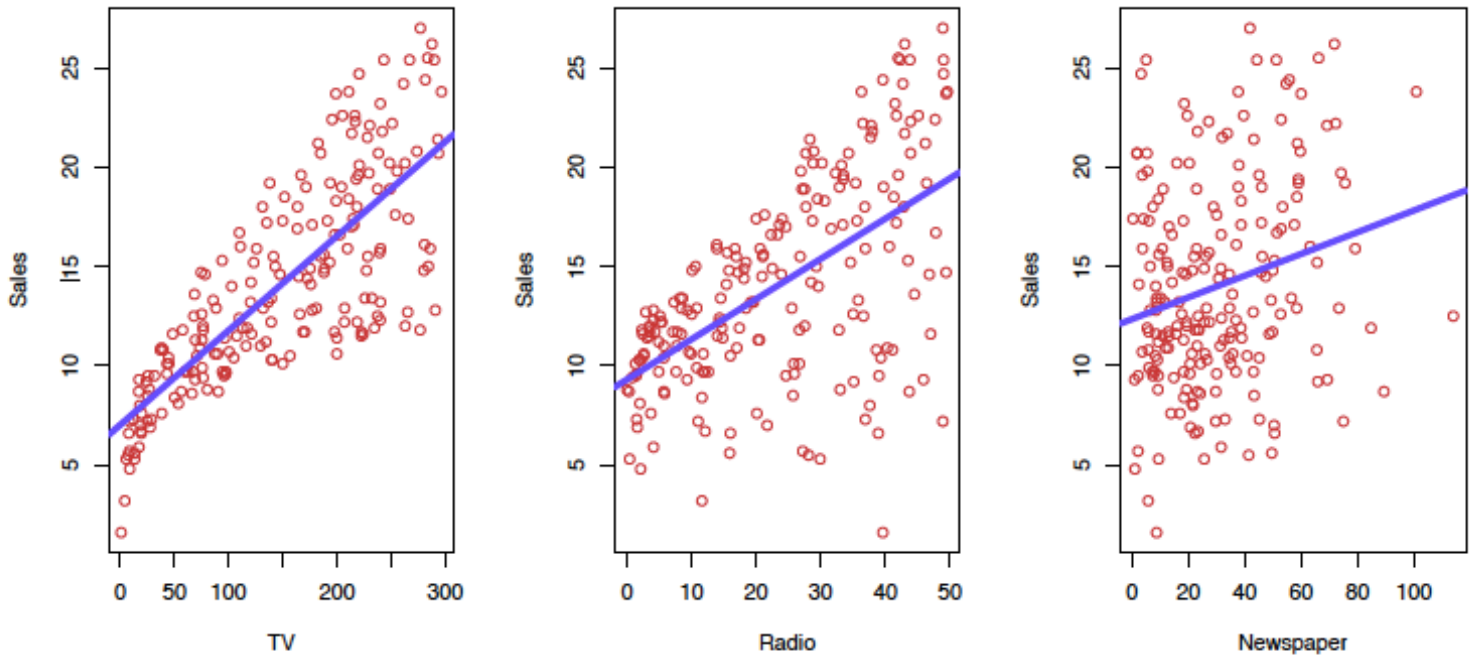
OutputVariable = f(InputVariables)

OutputVariable = f(InputVector)

DependentVariable = f(IndependentVariables)

$Y = f(X)$

Regressão Linear



Shown are `Sales` vs `TV`, `Radio` and `Newspaper`, with a blue linear-regression line fit separately to each.

Can we predict `Sales` using these three?

Perhaps we can do better using a model

$$\texttt{Sales} \approx f(\texttt{TV}, \texttt{Radio}, \texttt{Newspaper})$$

Notação

Here `Sales` is a *response* or *target* that we wish to predict. We generically refer to the response as Y.
`TV` is a *feature*, or *input*, or *predictor*; we name it X_1.
Likewise name `Radio` as X_2, and so on.
We can refer to the *input vector* collectively as

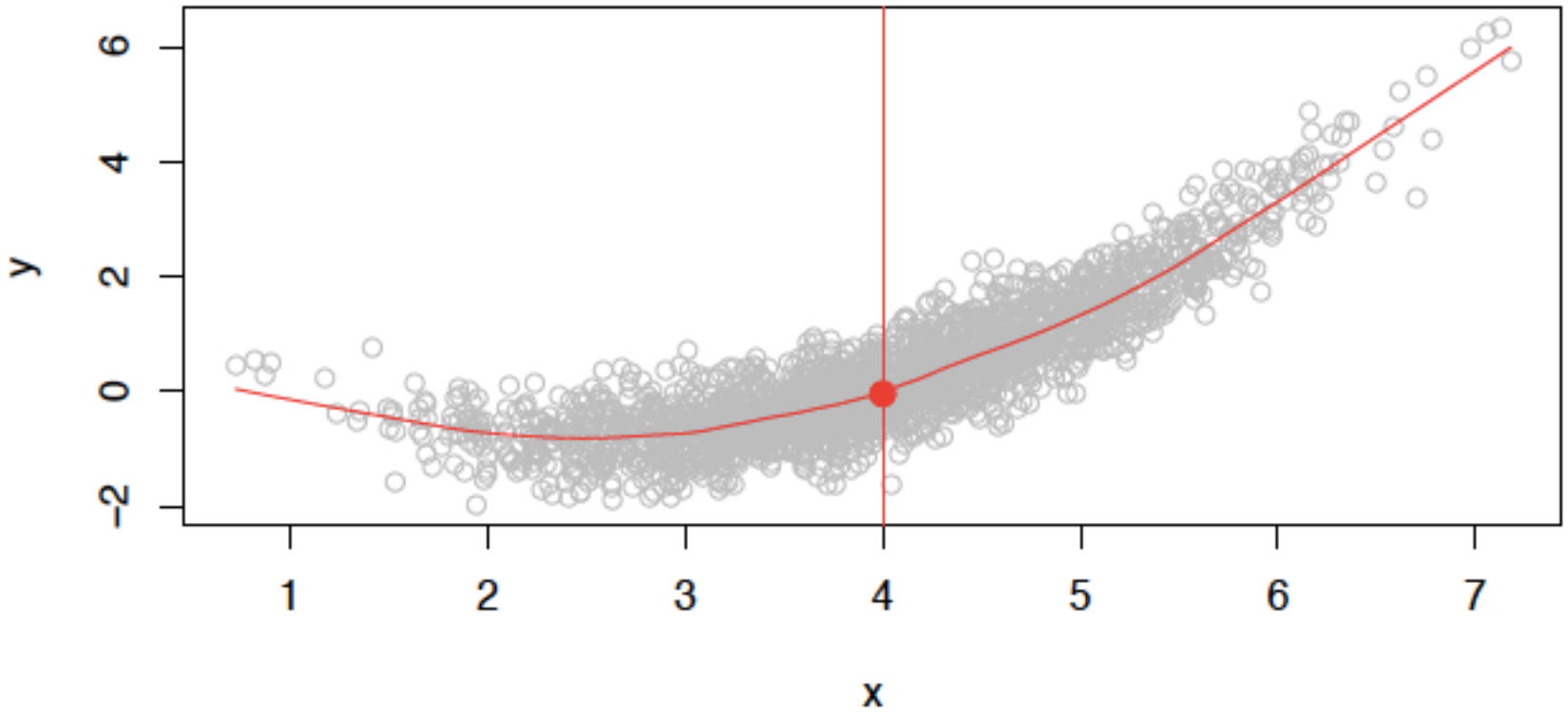
$$X = \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix}$$

Now we write our model as

$$Y = f(X) + \epsilon$$

where ϵ captures measurement errors and other discrepancies.

Função de Regressão



$E(Y|X = 4)$ means *expected value* (average) of Y given $X = 4$.

This ideal $f(x) = E(Y|X = x)$ is called the *regression function*.

Modelos paramétricos

A learning model that summarizes data with a set of parameters of fixed size (independent of the number of training examples) is called a parametric model.

Artificial Intelligence: A Modern Approach, page 737

Modelos paramétricos

Modelo de regressão linear

$B0 + B1 \times X1 + B2 \times X2 = 0,$

Where B0, B1 and B2 are the coefficients of the line that control the intercept and slope, and X1 and X2 are two input variables.

We only need to estimate the coefficients of the line equation and we have a predictive model for the problem.

Modelos paramétricos

Exemplos:

- Linear regression
- Logistic Regression
- Linear Discriminant Analysis
- Perceptron

Modelos paramétricos

Vantagens:

- Simpler
  - These methods are easier to understand and interpret results.
- Speed
  - Parametric models are very fast to learn from data.
- Less Data
  - They do not require as much training data and can work well even if the fit to the data is not perfect.

Modelos paramétricos

Desvantagens:

- Constrained
  - By choosing a functional form these methods are highly constrained to the specified form.
- Limited Complexity
  - The methods are more suited to simpler problems.
- Poor Fit
  - In practice the methods are unlikely to match the underlying mapping function.

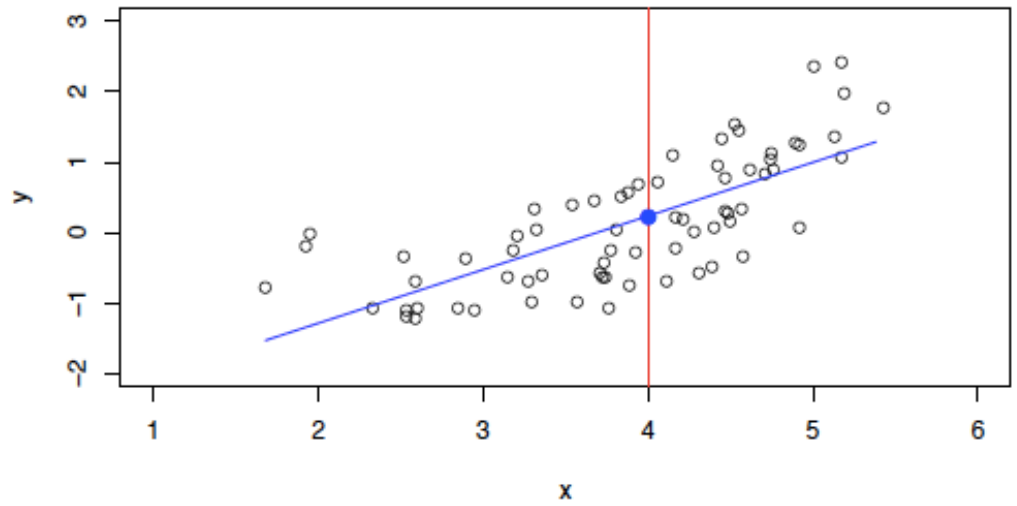
Modelos paramétricos

The *linear* model is an important example of a parametric model:

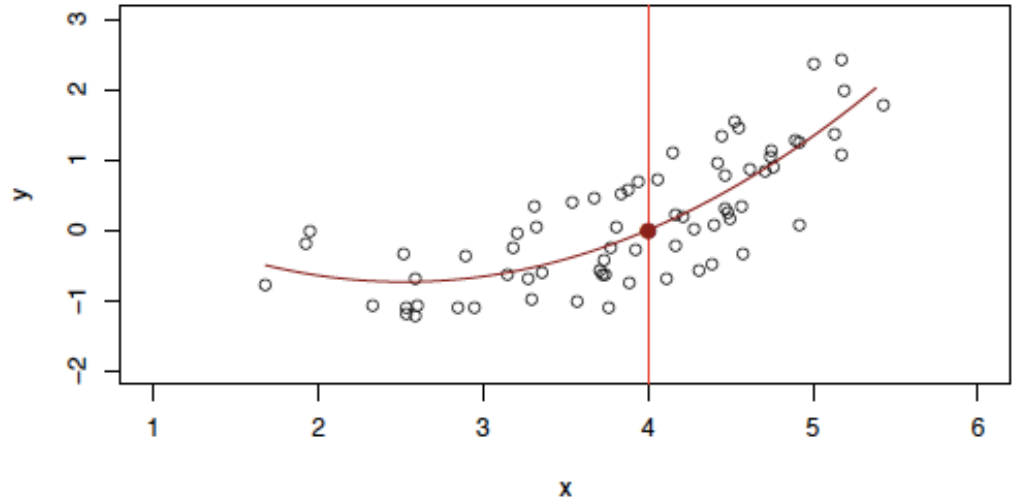
$$f_L(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots \beta_p X_p.$$

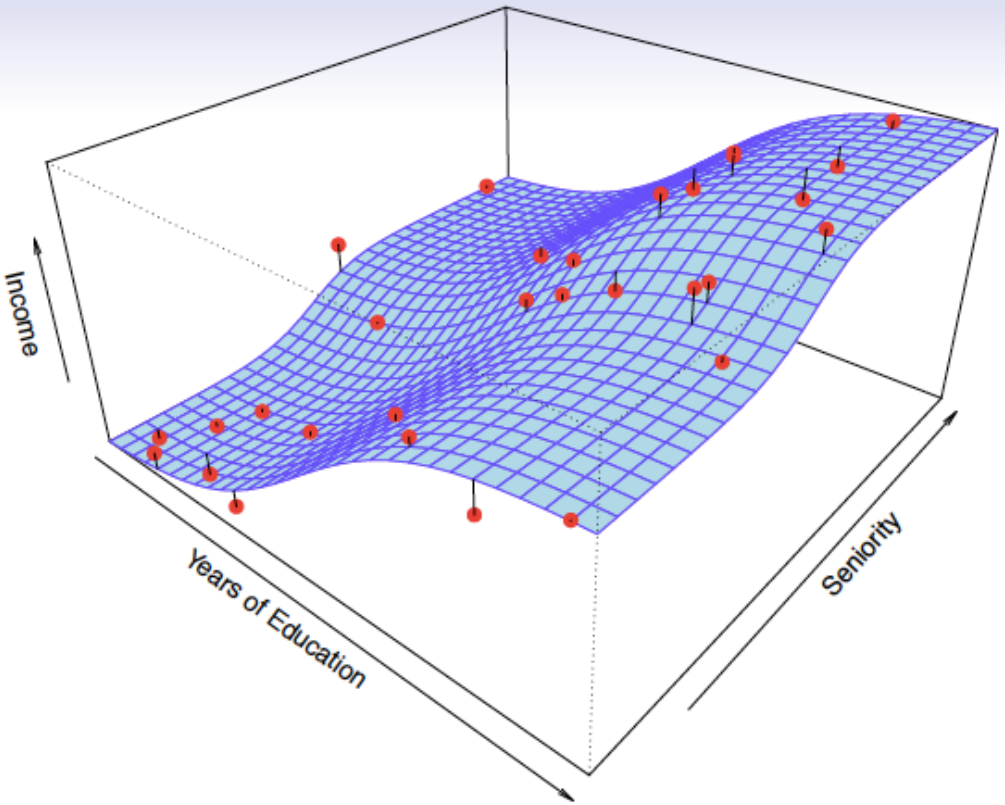
- A linear model is specified in terms of $p + 1$ parameters $\beta_0, \beta_1, \ldots, \beta_p$.
- We estimate the parameters by fitting the model to training data.
- Although it is *almost never correct*, a linear model often serves as a good and interpretable approximation to the unknown true function $f(X)$.

A linear model $\hat{f}_L(X) = \hat{\beta}_0 + \hat{\beta}_1 X$ gives a reasonable fit here



A quadratic model $\hat{f}_Q(X) = \hat{\beta}_0 + \hat{\beta}_1 X + \hat{\beta}_2 X^2$ fits slightly better.

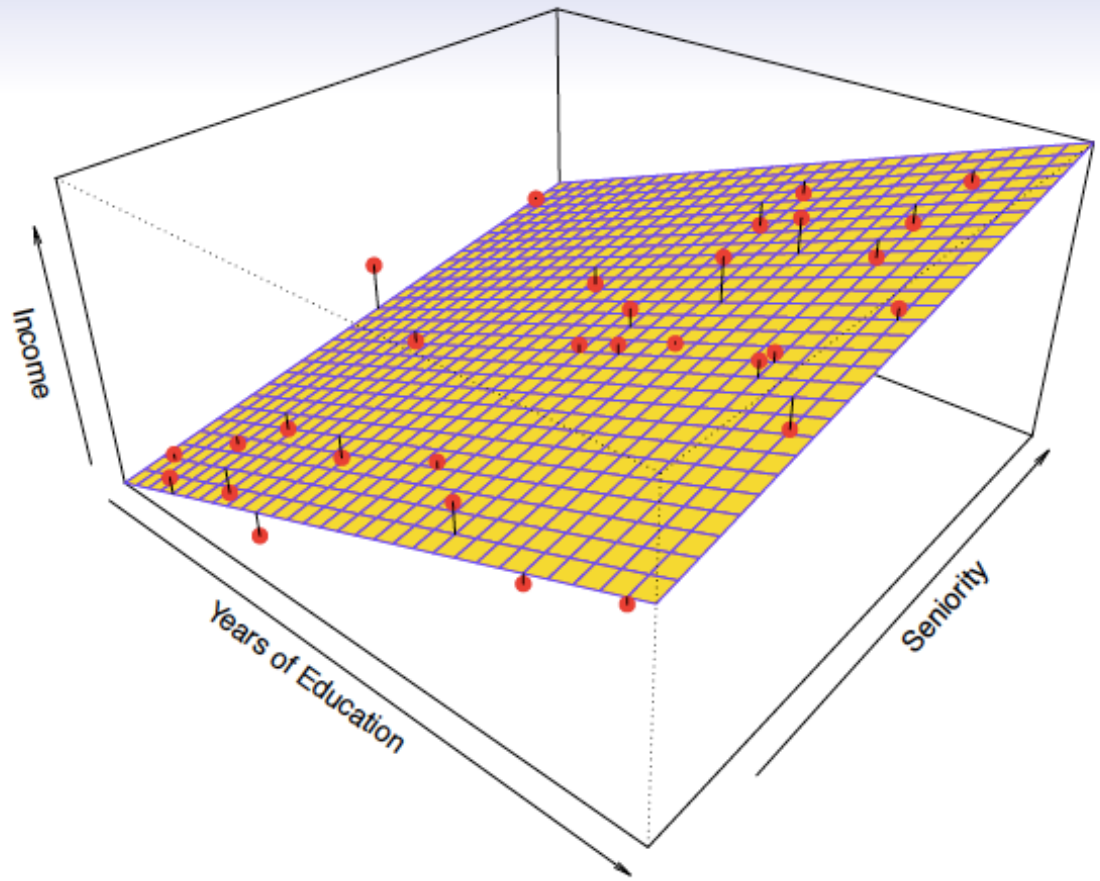




Simulated example. Red points are simulated values for `income` from the model

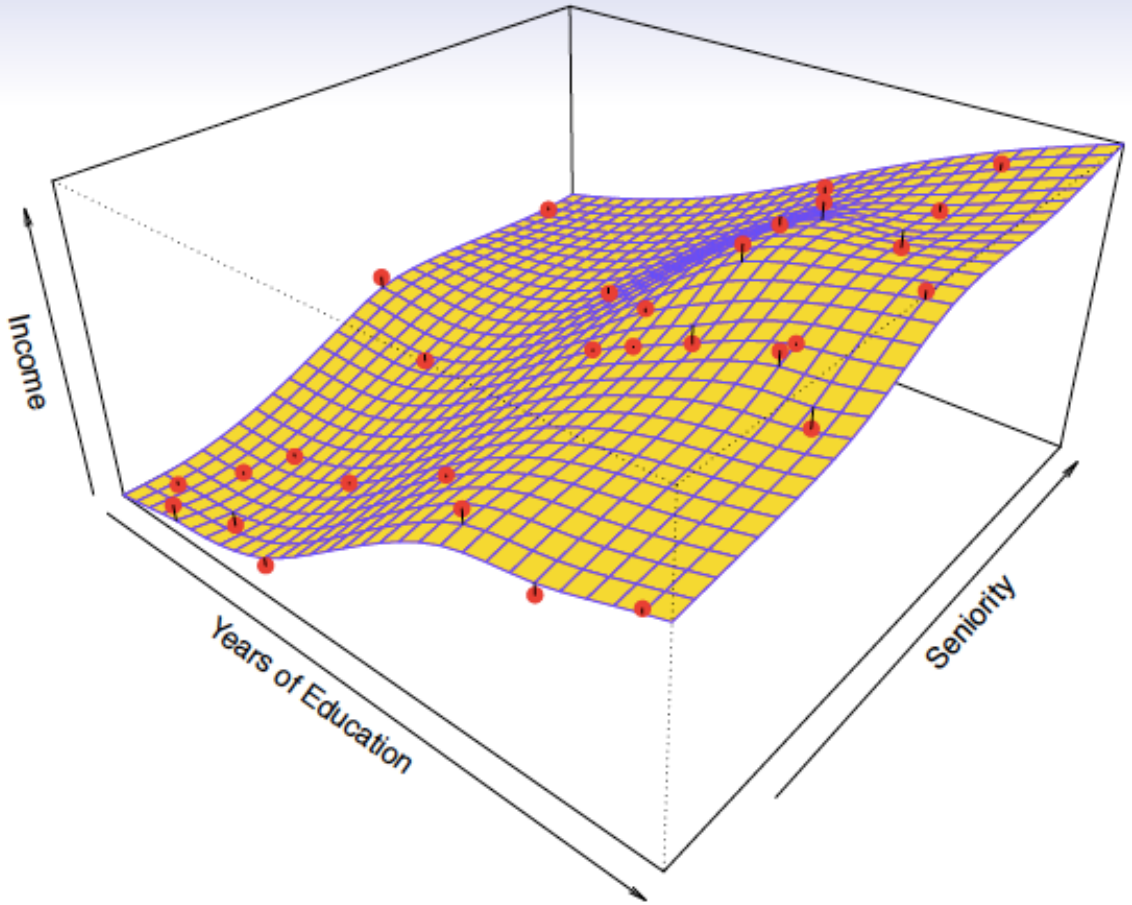
$$\texttt{income} = f(\texttt{education}, \texttt{seniority}) + \epsilon$$

f is the blue surface.

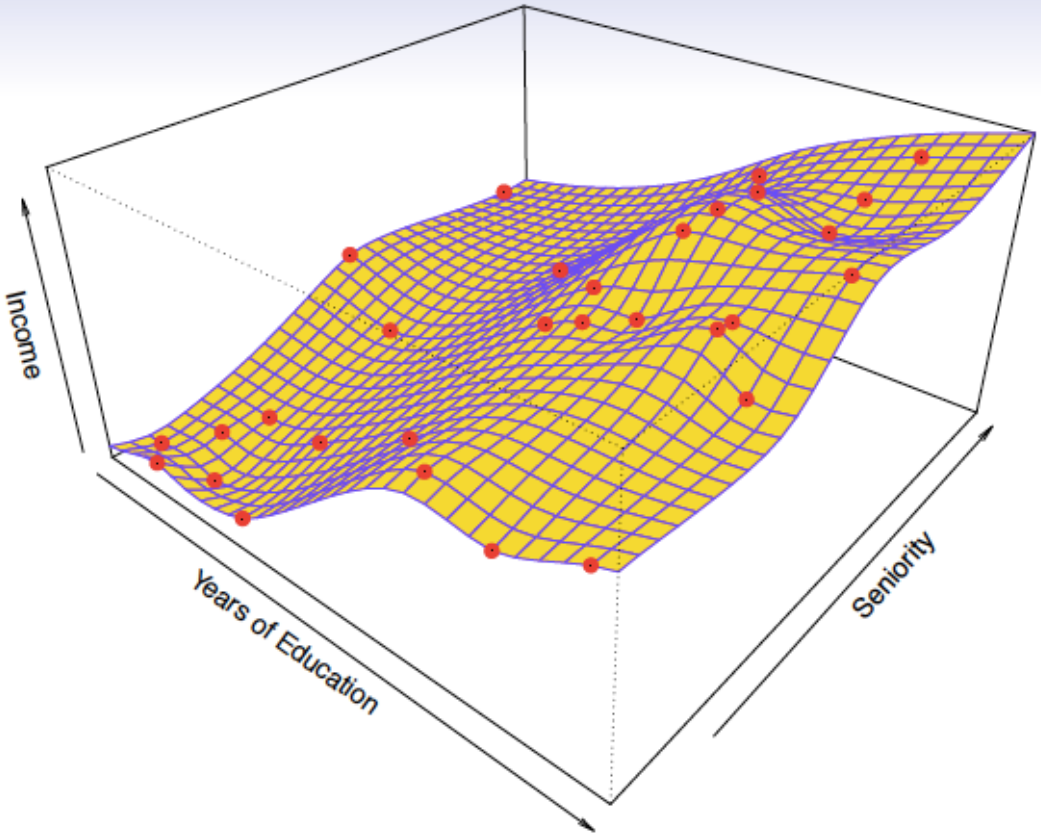


Linear regression model fit to the simulated data.

$$\hat{f}_L(\texttt{education}, \texttt{seniority}) = \hat{\beta}_0 + \hat{\beta}_1 \times \texttt{education} + \hat{\beta}_2 \times \texttt{seniority}$$



More flexible regression model $\hat{f}_S(\texttt{education}, \texttt{seniority})$ fit to the simulated data. Here we use a technique called a *thin-plate spline* to fit a flexible surface. We control the roughness of the fit (chapter 7).



Even more flexible spline regression model $\hat{f}_S(\texttt{education}, \texttt{seniority})$ fit to the simulated data. Here the fitted model makes no errors on the training data! Also known as *overfitting*.

Nonparametric models

- Algorithms that do not make strong assumptions about the form of the mapping function.
- They are free to learn any functional form from the training data.
- Nonparametric methods are good when you have a lot of data and no prior knowledge, and when you don't want to worry too much about choosing just the right features (Artificial Intelligence: A Modern Approach, page 757).

Nonparametric models

Exemplos:

- Decision Trees like CART and C4.5
- Naive Bayes
- Support Vector Machines
- Neural Networks

Nonparametric models

Vantagens:

- Flexibility
  - Capable of fitting a large number of functional forms.
- Power
  - No assumptions (or weak assumptions) about the underlying function.
- Performance
  - Can result in higher performance models for prediction.

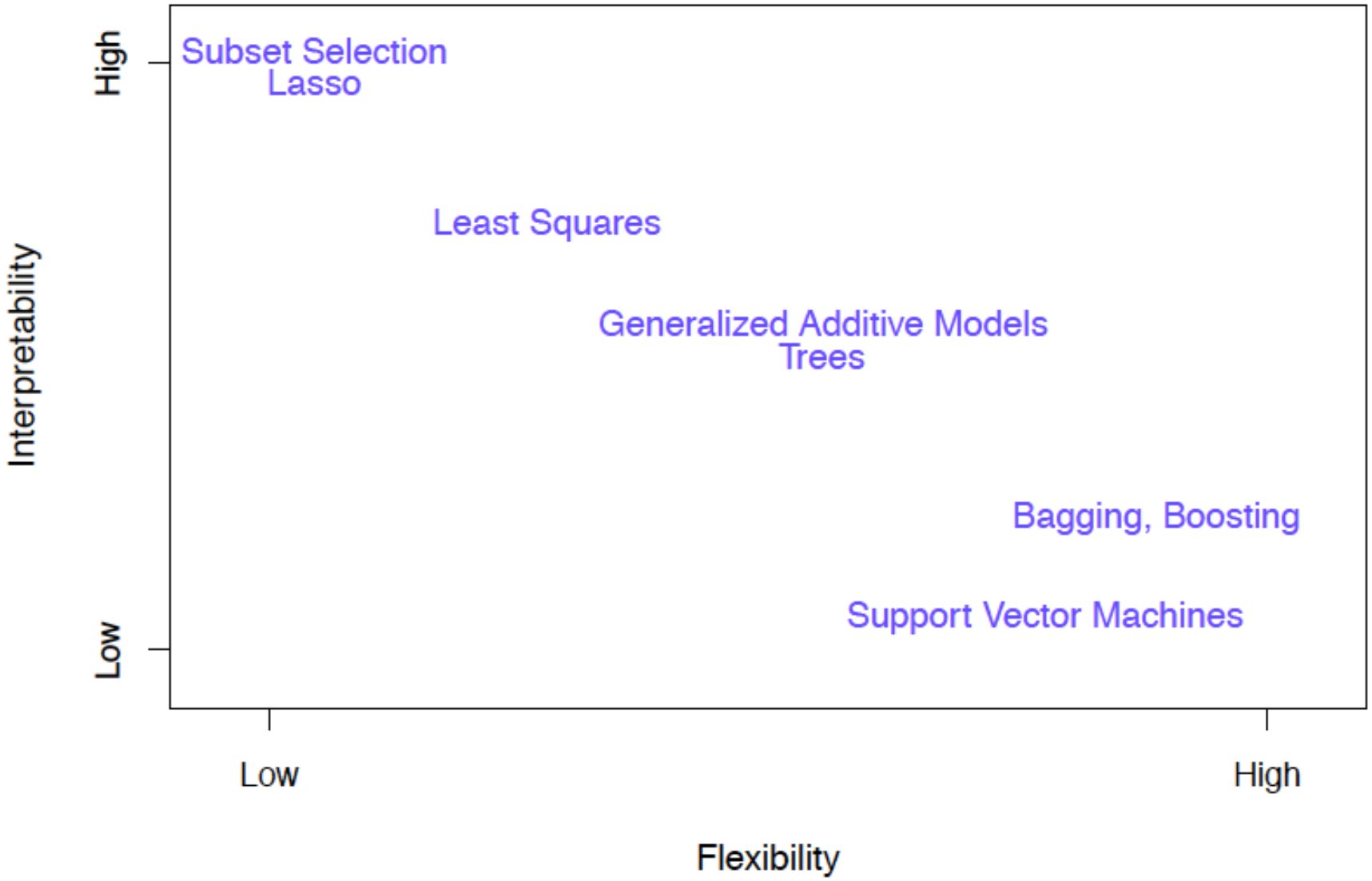
Nonparametric models

Desvantagens:

- More data
  - Require a lot more training data to estimate the mapping function.
- Slower
  - A lot slower to train as they often have far more parameters to train.
- Overfitting
  - More of a risk to overfit the training data and it is harder to explain why specific predictions are made.

Custo-benefício

- Prediction accuracy versus interpretability.
  - Linear models are easy to interpret;
  - thin-plate splines are not.
- Good fit versus over-fit or under-fit.
  - How do we know when the fit is just right?
- Parsimony versus black-box.
  - We often prefer a simpler model involving fewer variables over a black-box predictor involving them all.


Subset Selection
Lasso
Least Squares
Generalized Additive Models
Trees
Bagging, Boosting
Support Vector Machines
High
Low
Interpretability
Low
High
Flexibility

Precisão do modelo

Suppose we fit a model $\hat{f}(x)$ to some training data $\mathsf{Tr} = \{x_i, y_i\}_1^N$, and we wish to see how well it performs.

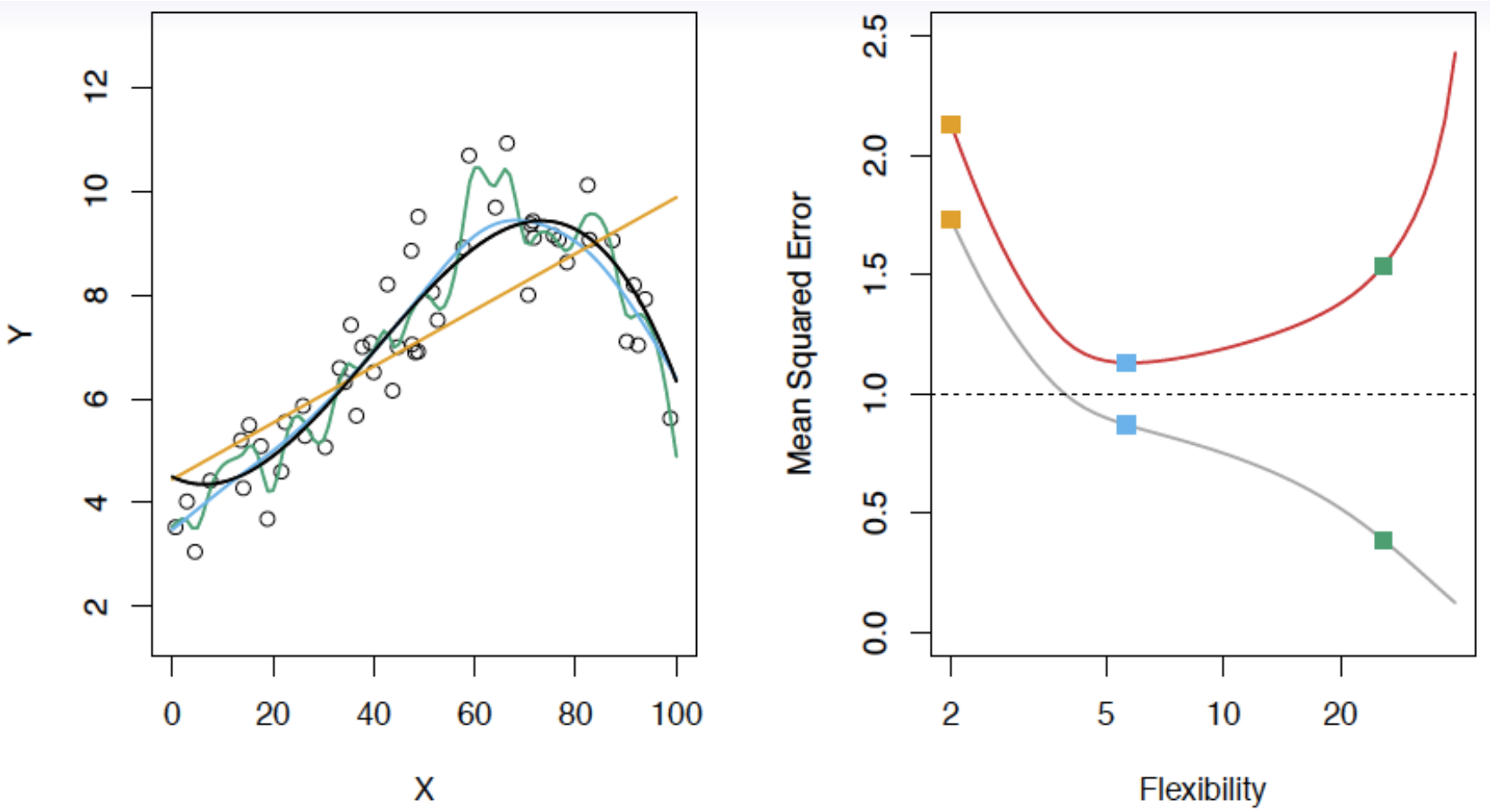
- We could compute the average squared prediction error over Tr:

$$\mathrm{MSE}_{\mathsf{Tr}} = \mathrm{Ave}_{i \in \mathsf{Tr}}[y_i - \hat{f}(x_i)]^2$$

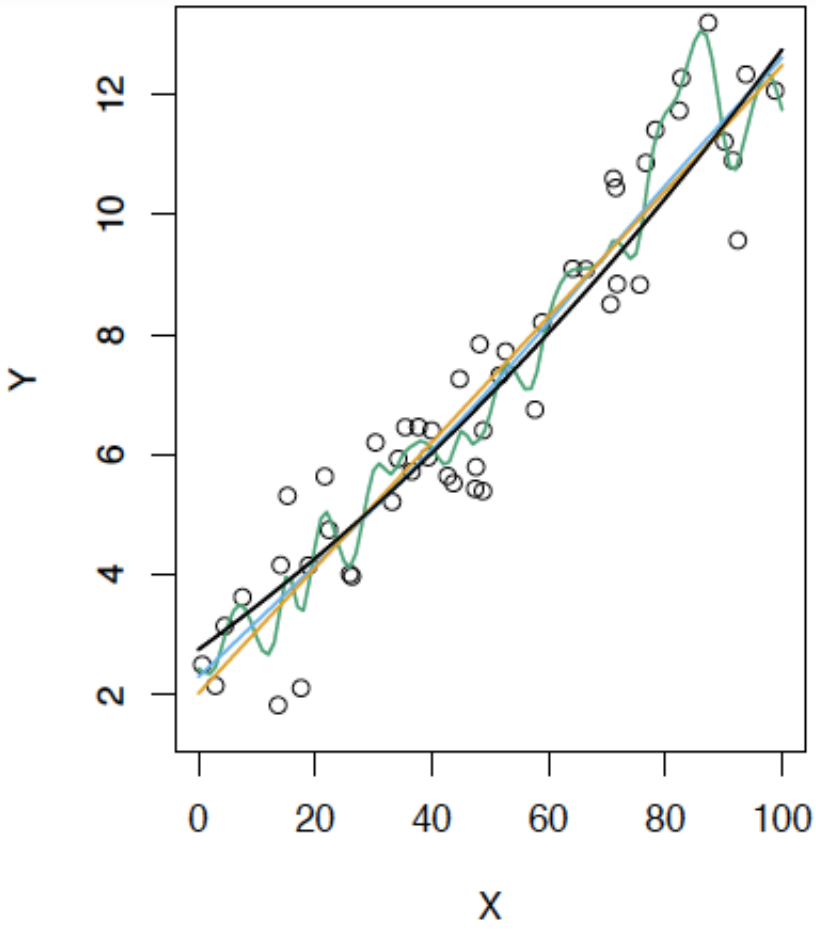
This may be biased toward more overfit models.

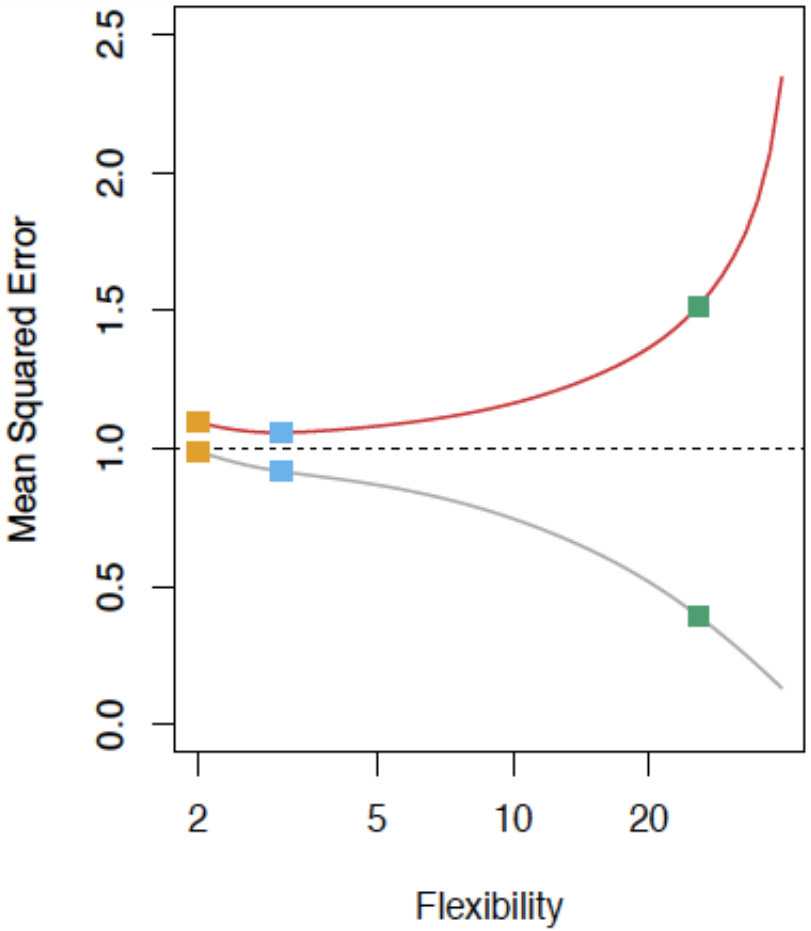
- Instead we should, if possible, compute it using fresh *test* data $\mathsf{Te} = \{x_i, y_i\}_1^M$:

$$\mathrm{MSE}_{\mathsf{Te}} = \mathrm{Ave}_{i \in \mathsf{Te}}[y_i - \hat{f}(x_i)]^2$$

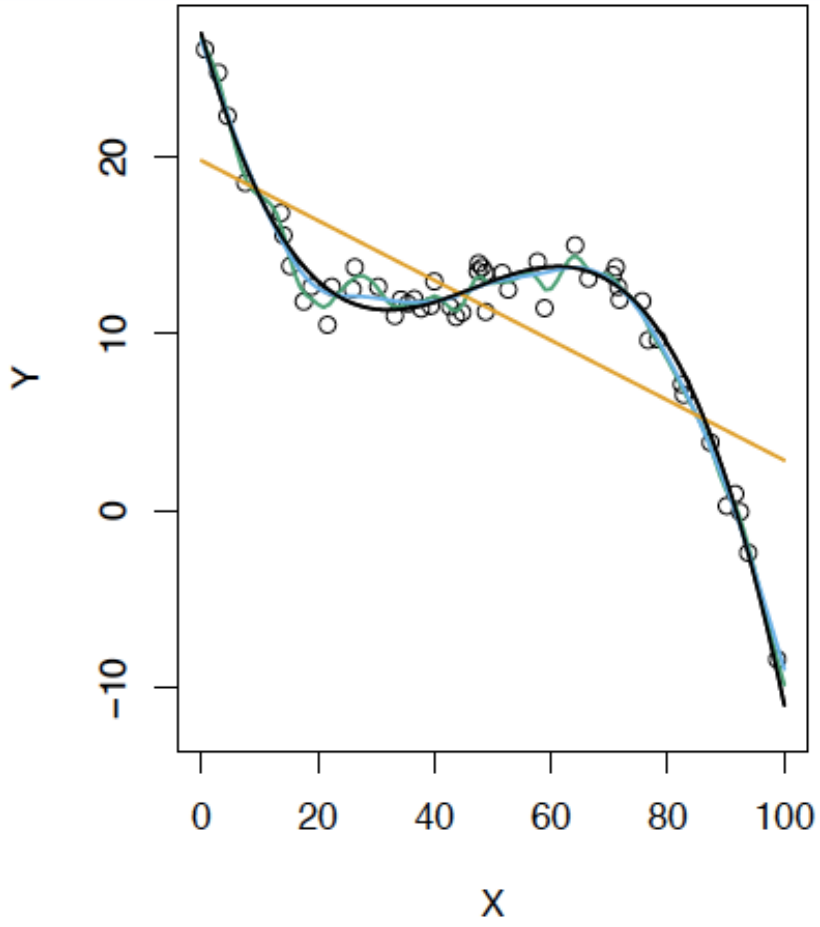


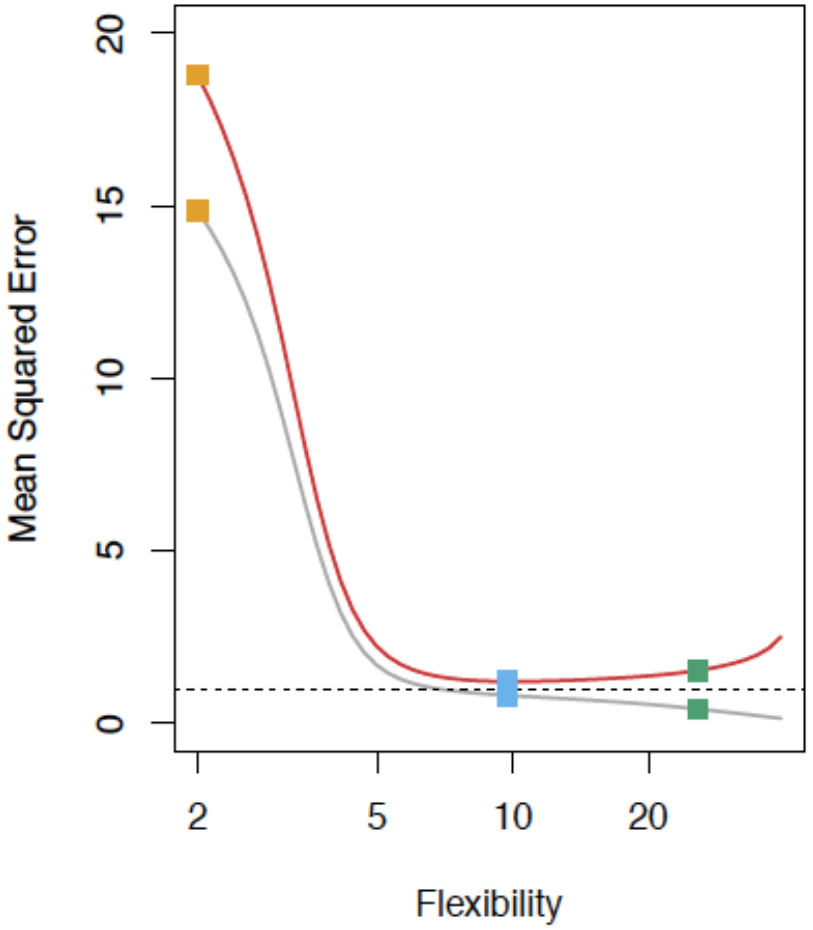
Black curve is truth. Red curve on right is MSE_{Te}, grey curve is MSE_{Tr}. Orange, blue and green curves/squares correspond to fits of different flexibility.





Here the truth is smoother, so the smoother fit and linear model do really well.





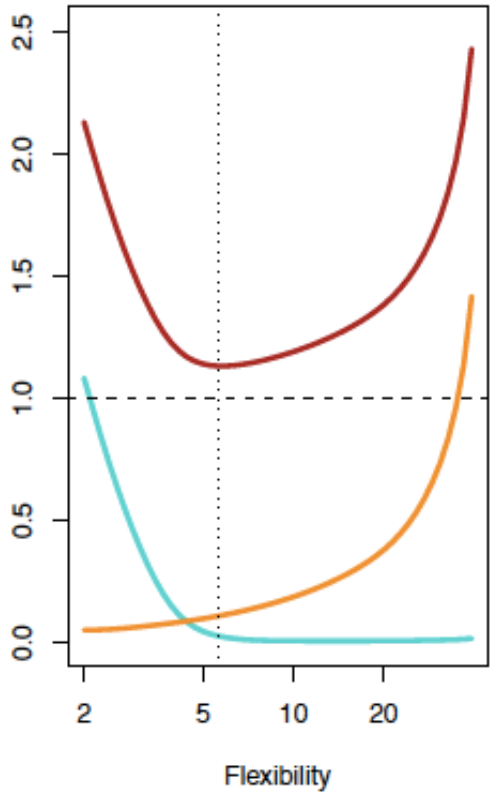
Here the truth is wiggly and the noise is low, so the more flexible fits do the best.

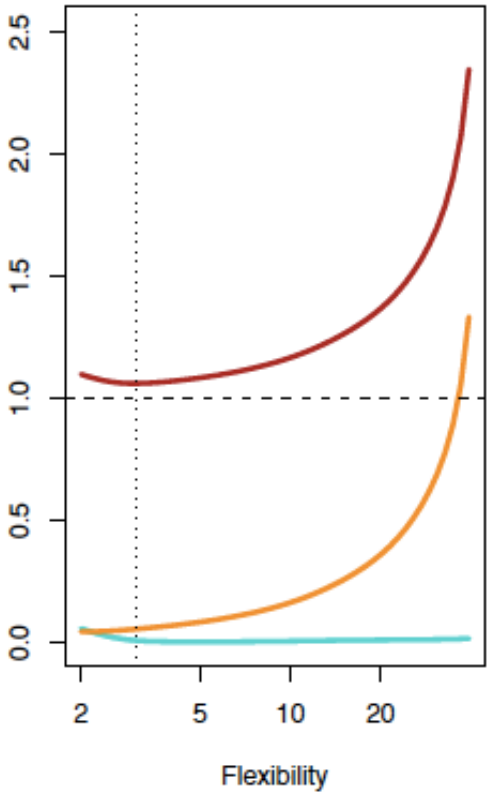
Errors

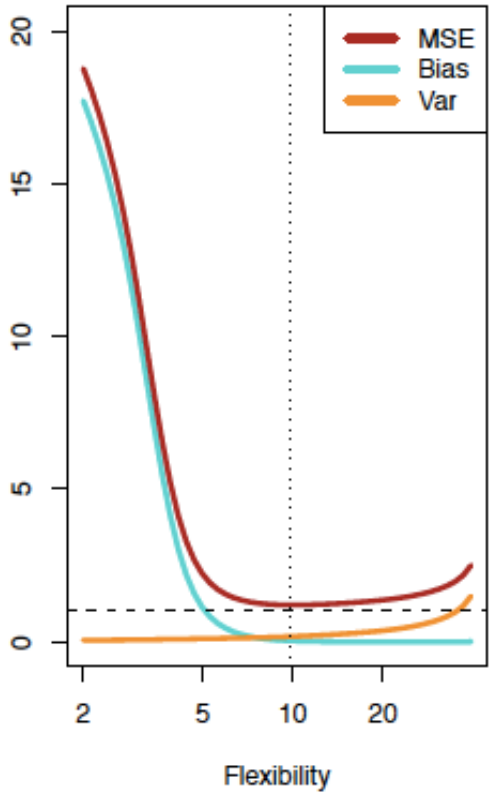
- Bias Error
  - Low Bias
    - Suggests less assumptions about the form of the target function.
  - High-Bias
    - Suggests more assumptions about the form of the target function.
- Variance Error
  - Low Variance
    - Suggests small changes to the estimate of the target function with changes to the training dataset.
  - High Variance
    - Suggests large changes to the estimate of the target function with changes to the training dataset.
- Irreducible Error

Bias-Variance Trade-off

Typically as the *flexibility* of $\hat{f}$ increases, its variance increases, and its bias decreases. So choosing the flexibility based on average test error amounts to a *bias-variance trade-off.*







Overfitting

- In statistics a fit refers to how well you approximate a target function.
- Overfitting refers to a model that models the training data too well.
  - Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance on the model on new data.

How to limit overfitting

1. Use a resampling technique to estimate model accuracy.
   - The most popular resampling technique is k-fold cross-validation.

2. Hold back a validation dataset.

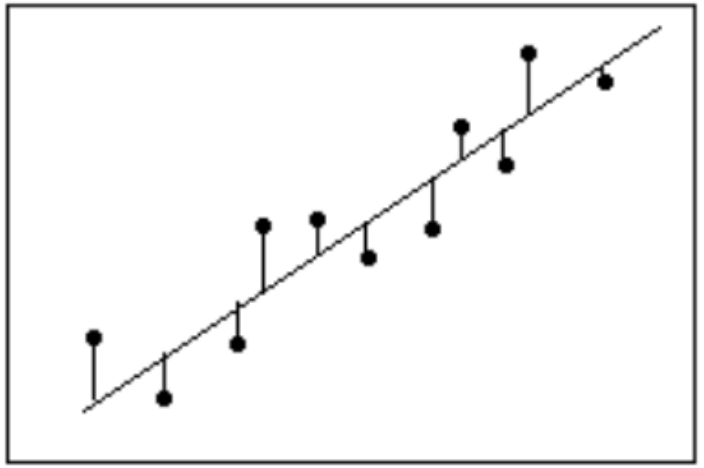
Underfitting

- Underfitting refers to a model that can neither model the training data nor generalize to new data.
- An underfit machine learning model is not a suitable model and it will have poor performance on the training data.
- Underfitting is often not discussed as it is easy to detect given a good performance metric.
- The remedy is to move on and try alternate machine learning algorithms.

Residuals

- Linear regression calculates an equation that minimizes the distance between the fitted line and all of the data points.

- Ordinary least squares (OLS) regression minimizes the sum of the squared residuals.



Definition: Residual = Observed value - Fitted value

R-squared (R^2)

- R-squared is a statistical measure of how close the data are to the fitted regression line
- R-squared = Explained variation / Total variation
- Values between 0% and 100%
  - 0% - model explains none of the variability of the response data around its mean.
  - 100% - model explains all the variability of the response data around its mean.
- In general, the higher the R-squared, the better the model fits your data.

T-Test

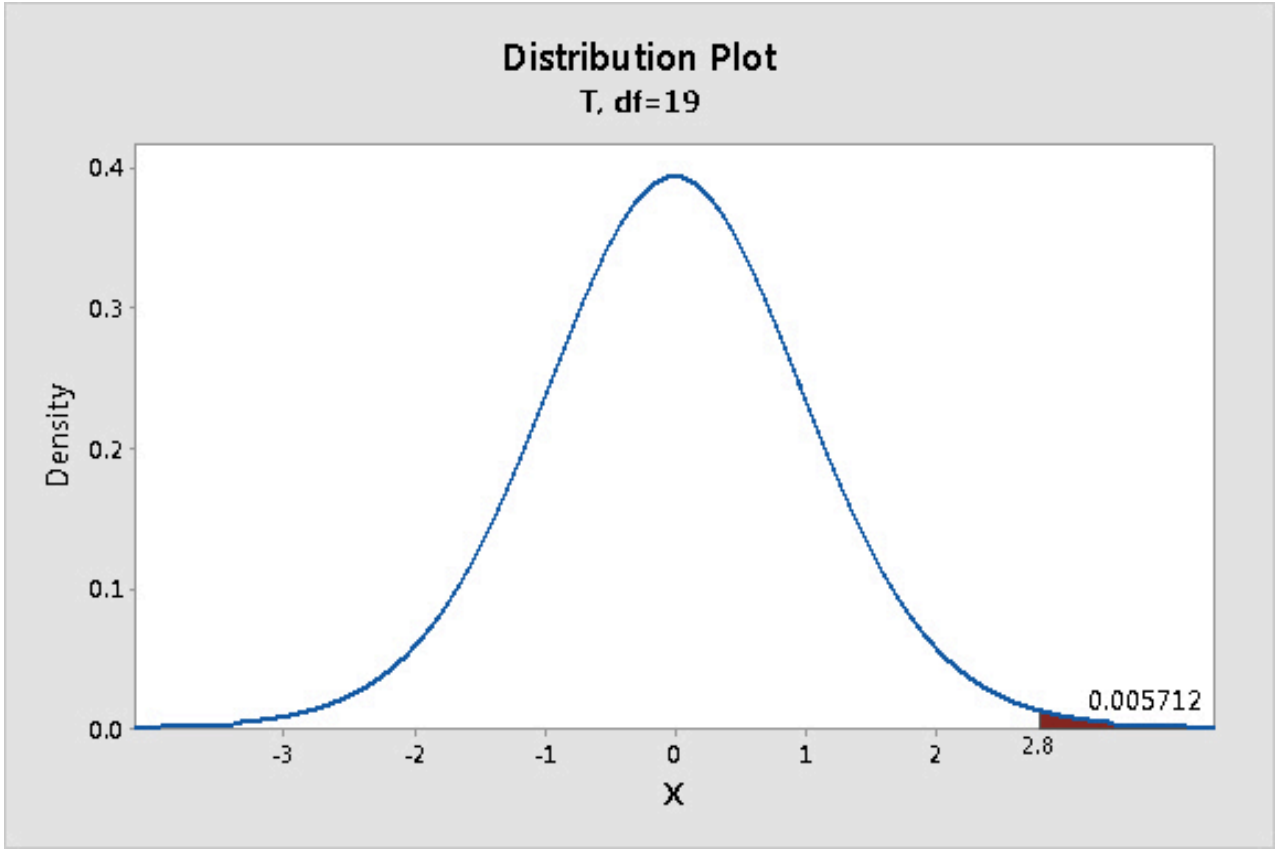
- When you perform a t-test, you're usually trying to find evidence of a significant difference between population means (2-sample t) or between the population mean and a hypothesized value (1-sample t).
- The t-value measures the size of the difference relative to the variation in your sample data.
- T is simply the calculated difference represented in units of standard error.
- The greater the magnitude of T (positive or negative), the greater the evidence against the null hypothesis that there is no significant difference.
- The closer T is to 0, the more likely there isn't a significant difference.

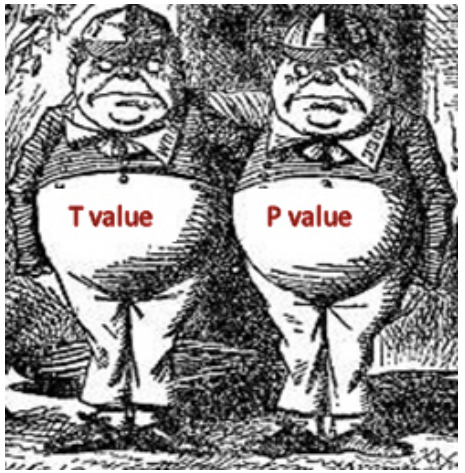
T Values e P Values

One-Sample T: C1

Test of μ = 5 vs > 5

Variable	N	Mean	StDev	SE Mean	95% Lower Bound	T	P
C1	20	5.790	1.262	0.282	5.302	2.80	0.006



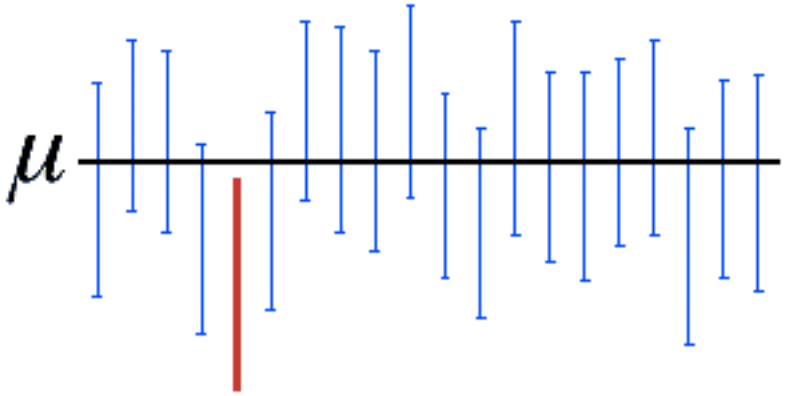


For a 1-sample t test, the degrees of freedom equals the sample size minus 1.

The probability of obtaining a t-value of 2.8 or higher, when sampling from the same population (here, a population with a hypothesized mean of 5), is approximately 0.006.

Confidence Intervals

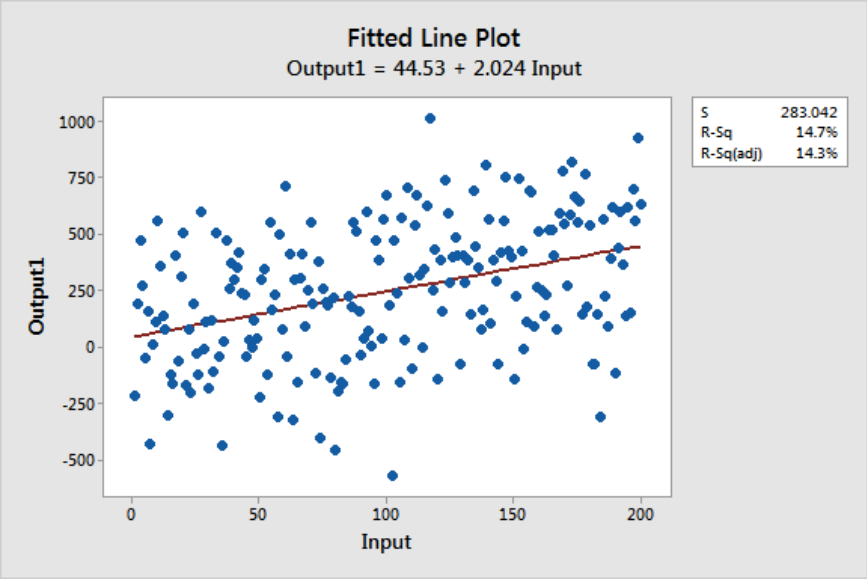
- Confidence interval is a range of values, derived from sample statistics, that is likely to contain the value of an unknown population parameter.

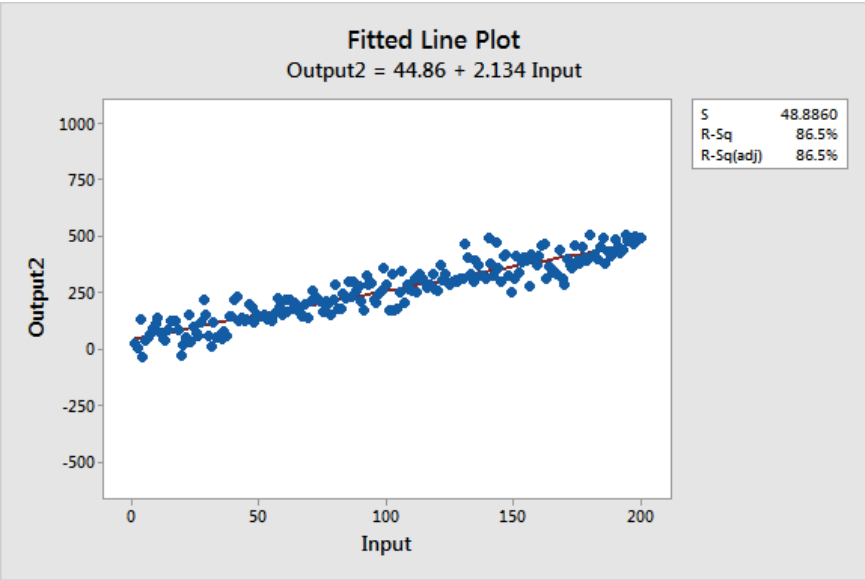


A 95% confidence interval indicates that 19 out of 20 samples (95%) from the same population will produce confidence intervals that contain the population parameter.

- Most frequently, we use confidence intervals to bound the mean or standard deviation, but we can also use regression coefficients, proportions, rates of occurrence (Poisson), and for the differences between populations.

Comparação entre modelos





Prediction for Output1

Regression Equation

Output1 = 44.5 + 2.024 Input

Variable	Setting
Input	10

Fit	SE Fit	95% CI	95% PI
64.7766	37.2129	(-8.60793, 138.161)	(-498.190, 627.743)

PI – Prediction Interval

Prediction for Output2

Regression Equation

Output2 = 44.86 + 2.1343 Input

Variable	Setting
Input	10

Fit	SE Fit	95% CI	95% PI
66.2076	6.42728	(53.5329, 78.8823)	(-31.0260, 163.441)

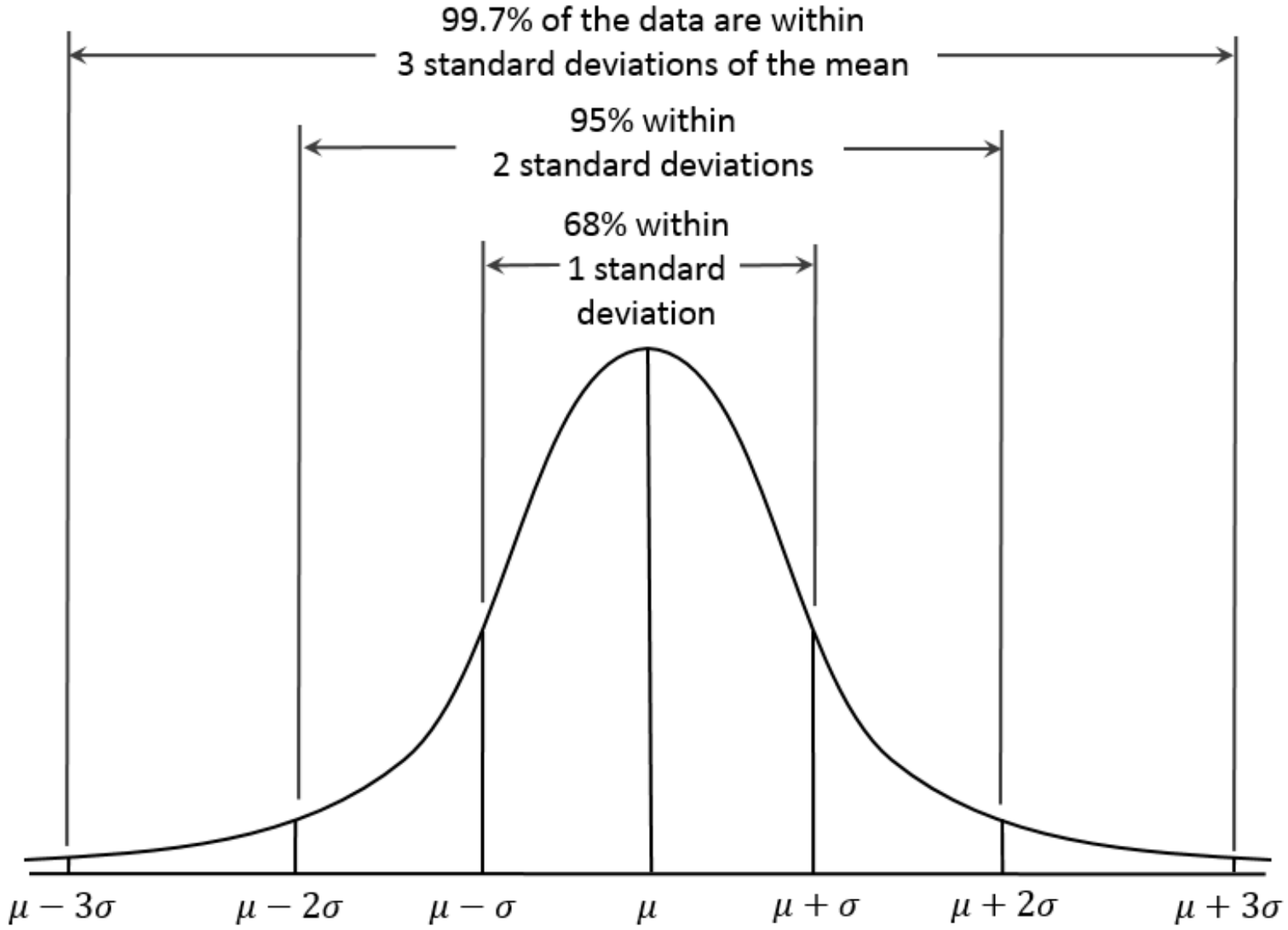
Medidas estatísticas

$$mean(x) = \frac{\sum_{i=1} x_i}{count(x)}$$

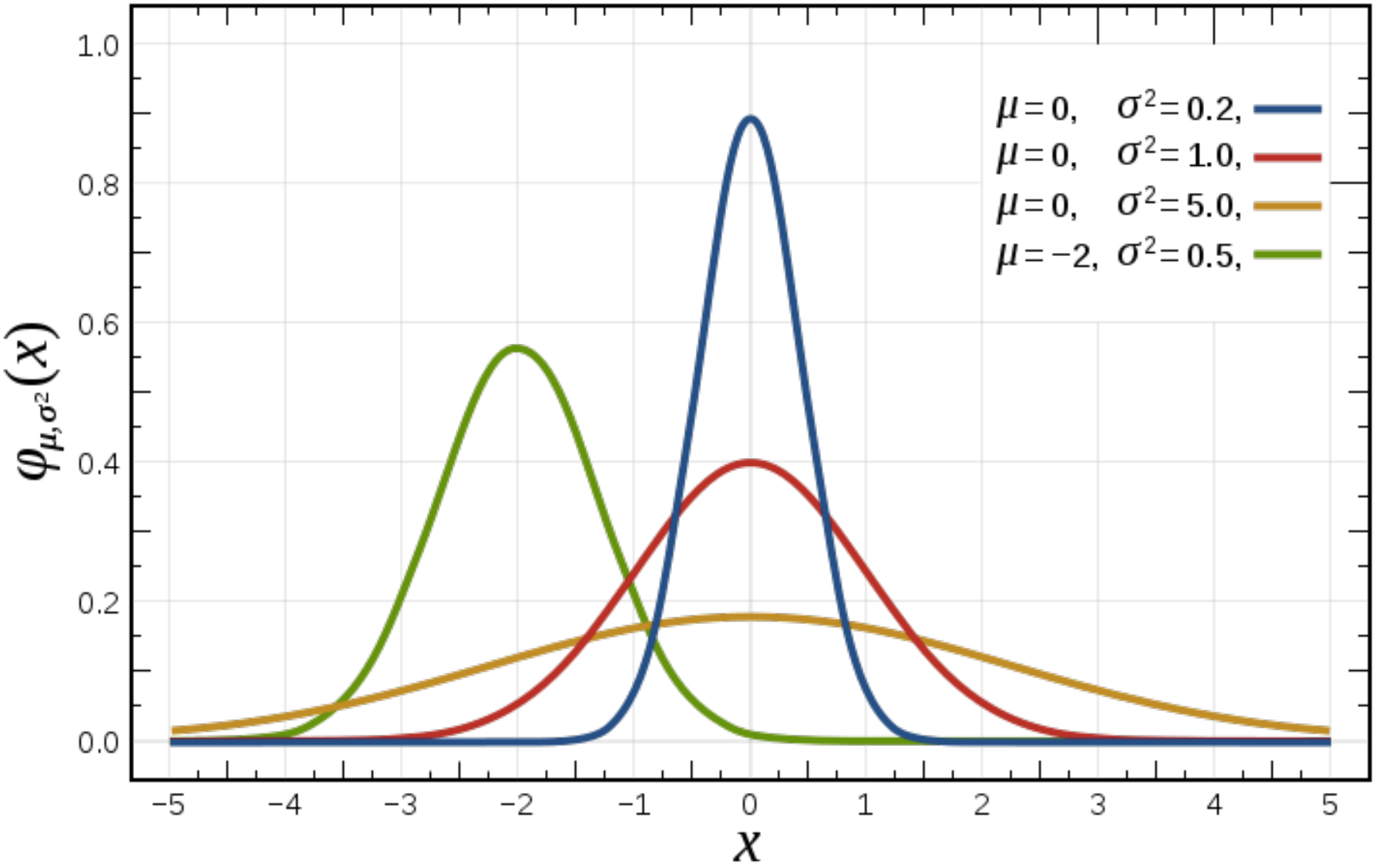
$$variance = \sum_{i=1}^{n} (x_i - mean(x))^2$$

$$covariance = \sum_{i=1}^{n} ((x_i - mean(x)) \times (y_i - mean(y)))$$

Distribuição Normal

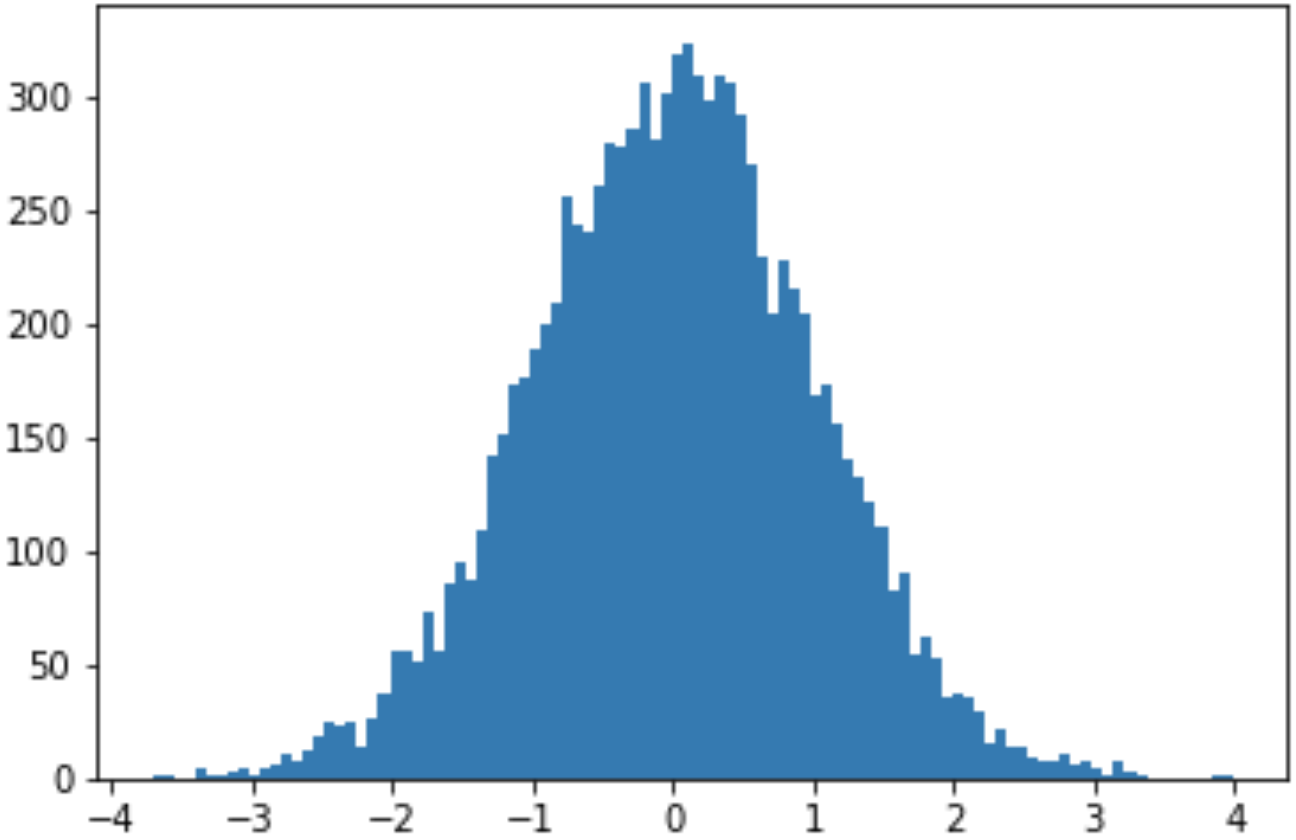


Distribuição Normal



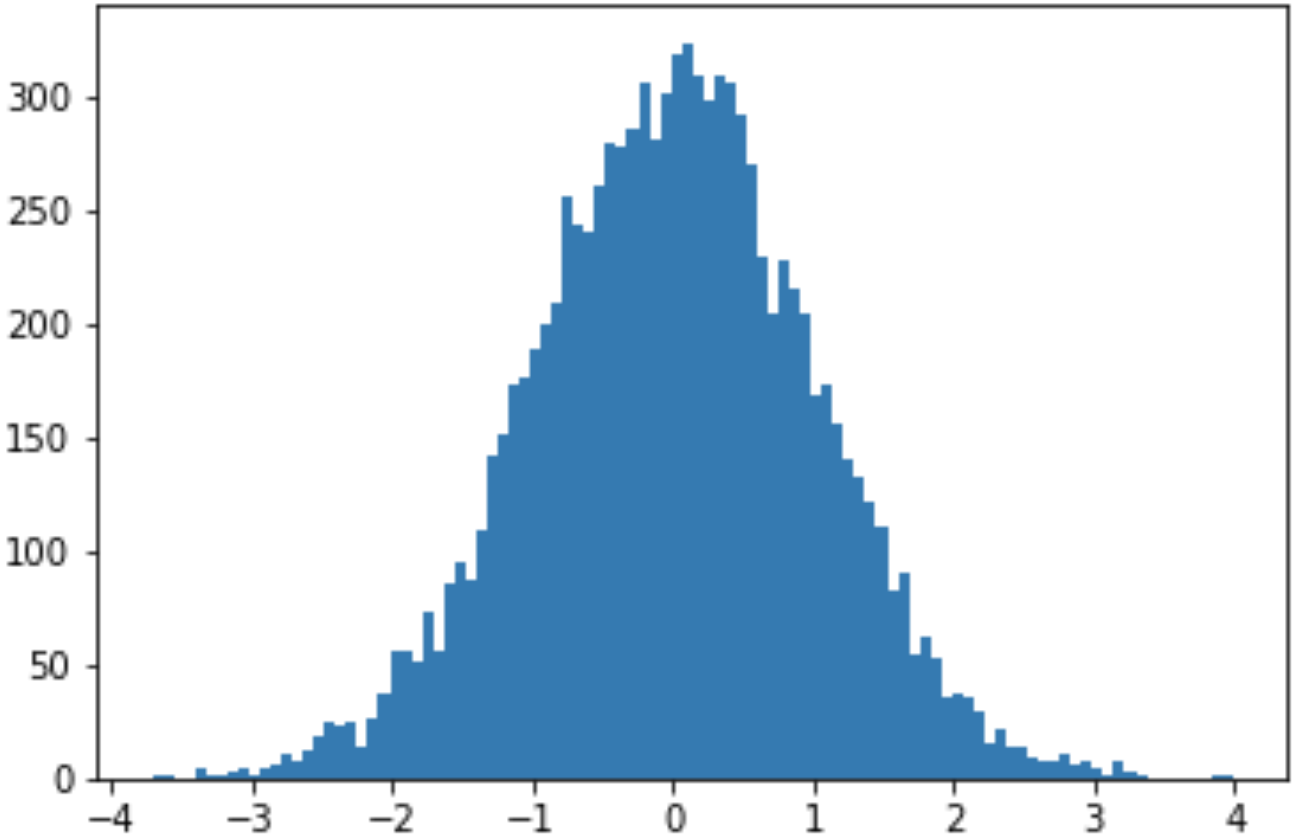
Distribuição Normal

```
s = np.random.normal(size=10000)
plt.hist(s, bins=100);
```



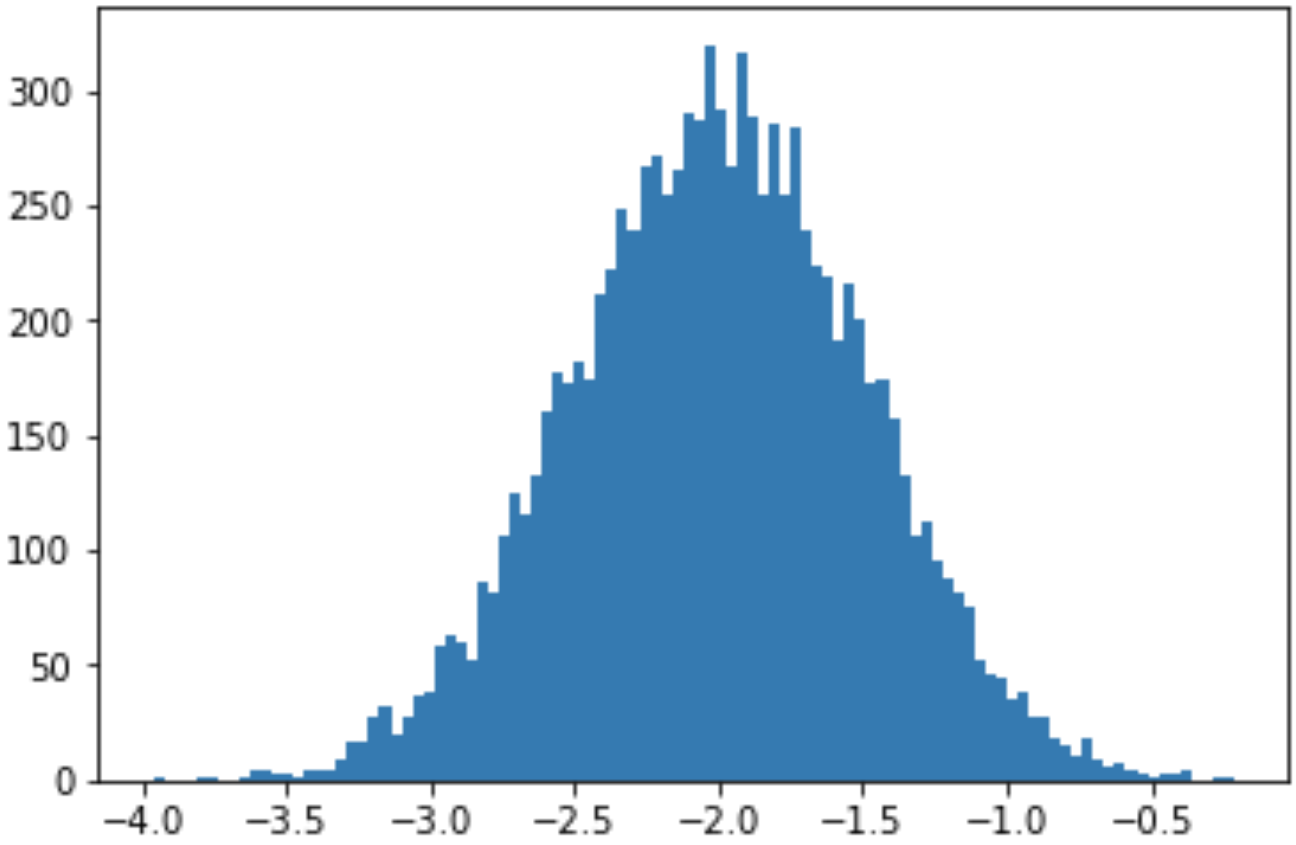
Distribuição Normal

```
s = np.random.normal(size=10000)
plt.hist(s, bins=100);
```



Distribuição Normal

```
s = np.random.normal(loc=-2, scale=0.5, size=10000)
plt.hist(s, bins=100);
```



loc : Mean ("centre") of the distribution.

scale : Standard deviation (spread or "width") of the distribution.

Distribuições

numpy.random

`beta` (a, b[, size])
`binomial` (n, p[, size])
`chisquare` (df[, size])
`dirichlet` (alpha[, size])
`exponential` ([scale, size])
`f` (dfnum, dfden[, size])
`gamma` (shape[, scale, size])
`geometric` (p[, size])
`gumbel` ([loc, scale, size])
`hypergeometric` (ngood, nbad, nsample[, size])
`laplace` ([loc, scale, size])

`logistic` ([loc, scale, size])
`lognormal` ([mean, sigma, size])
`logseries` (p[, size])
`multinomial` (n, pvals[, size])
`multivariate_normal` (mean, cov[, size, ...)

`negative_binomial` (n, p[, size])
`noncentral_chisquare` (df, nonc[, size])
`noncentral_f` (dfnum, dfden, nonc[, size])
`normal` ([loc, scale, size])

`pareto` (a[, size])

`poisson` ([lam, size])
`power` (a[, size])

`rayleigh` ([scale, size])
`standard_cauchy` ([size])

`standard_exponential` ([size])
`standard_gamma` (shape[, size])
`standard_normal` ([size])

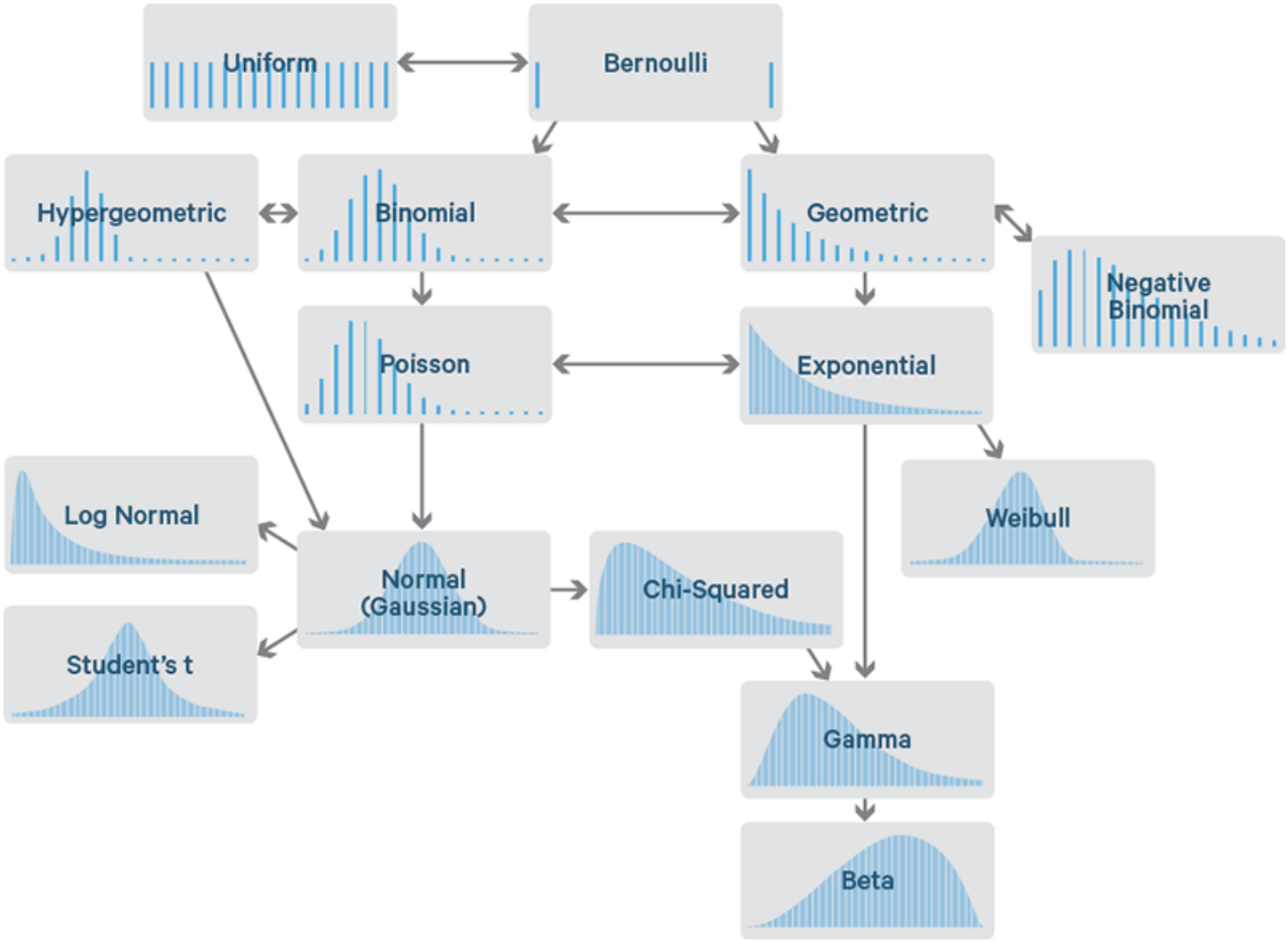
`standard_t` (df[, size])

`triangular` (left, mode, right[, size])

`uniform` ([low, high, size])
`vonmises` (mu, kappa[, size])
`wald` (mean, scale[, size])

`weibull` (a[, size])
`zipf` (a[, size])

Distribuições



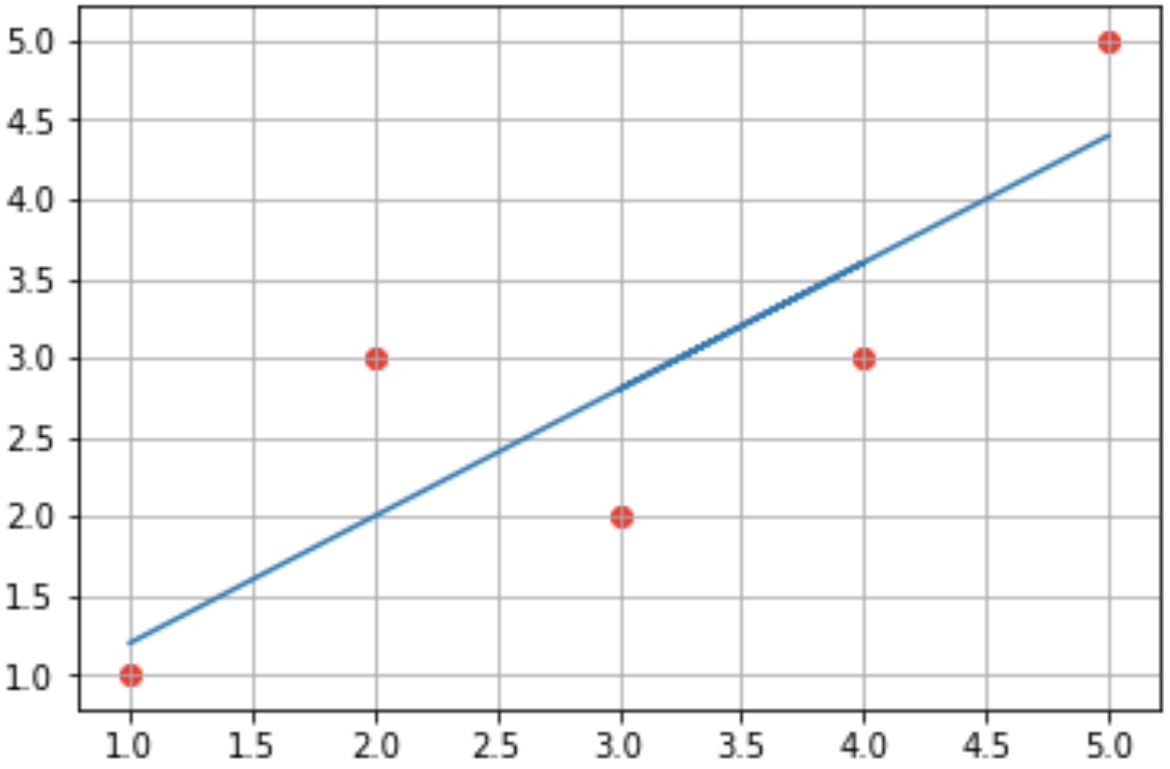
http://blog.cloudera.com/blog/2015/12/common-probability-distributions-the-data-scientists-crib-sheet/

scikit-learn datasets

datasets.load_boston ([return_X_y])	Load and return the boston house-prices dataset (regression).
datasets.load_breast_cancer ([return_X_y])	Load and return the breast cancer wisconsin dataset (classification).
datasets.load_diabetes ([return_X_y])	Load and return the diabetes dataset (regression).
datasets.load_digits ([n_class, return_X_y])	Load and return the digits dataset (classification).
datasets.load_files (container_path[, ...])	Load text files with categories as subfolder names.
datasets.load_iris ([return_X_y])	Load and return the iris dataset (classification).
datasets.load_linnerud ([return_X_y])	Load and return the linnerud dataset (multivariate regression).
datasets.load_mlcomp (name_or_id[, set_, ...])	DEPRECATED: since the http://mlcomp.org/ website will shut down in March 2017, the load_mlcomp function was deprecated in version 0.19 and will be removed in 0.21.
datasets.load_sample_image (image_name)	Load the numpy array of a single sample image
datasets.load_sample_images ()	Load sample images for image manipulation.
datasets.load_svmlight_file (f[, n_features, ...])	Load datasets in the svmlight / libsvm format into sparse CSR matrix
datasets.load_svmlight_files (files[, ...])	Load dataset from multiple files in SVMlight format
datasets.load_wine ([return_X_y])	Load and return the wine dataset (classification).

Tutorial

x	y
1	1
2	3
4	3
3	2
5	5



Simple Linear Regression

- $y = B0 + B1 \times x$

$$B1 = \frac{\sum_{i=1}^{n}(x_i - mean(x)) \times (y_i - mean(y))}{\sum_{i=1}^{n}(x_i - mean(x))^2}$$

$$B1 = \frac{covariance(x, y)}{variance(x)}$$

$$B0 = mean(y) - B1 \times mean(x)$$

$$mean(x) = 3$$
$$mean(y) = 2.8$$

$$B1 = \frac{8}{10}$$
$$B1 = 0.8$$

$$B0 = mean(y) - B1 \times mean(x)$$
$$B0 = 2.8 - 0.8 \times 3$$
$$B0 = 0.4$$

$$y = B0 + B1 \times x$$
$$y = 0.4 + 0.8 \times x$$

RMSE = 0.692820323

Simple Linear Regression

- Atalho

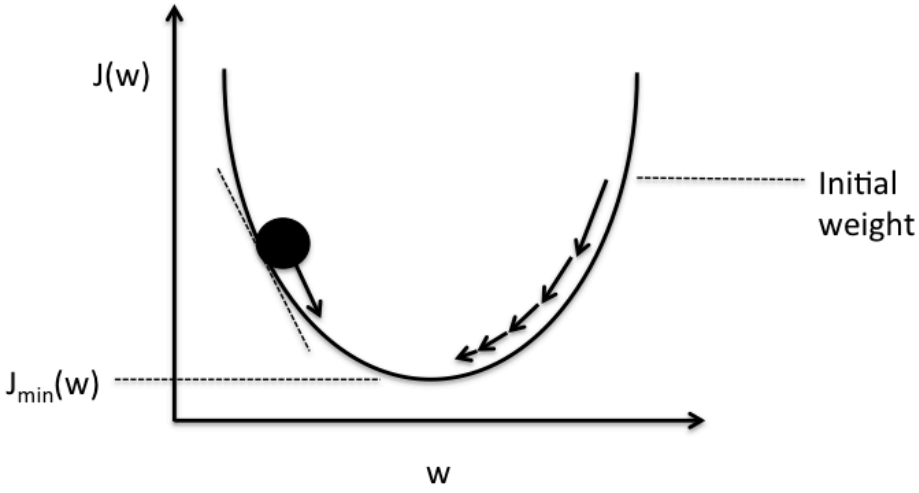
$$B1 = corr(x, y) \times \frac{stdev(y)}{stdev(x)}$$

$$B1 = 0.852802865 \times \frac{1.483239697}{1.58113883}$$

$$B1 = 0.8$$

Gradient Descent

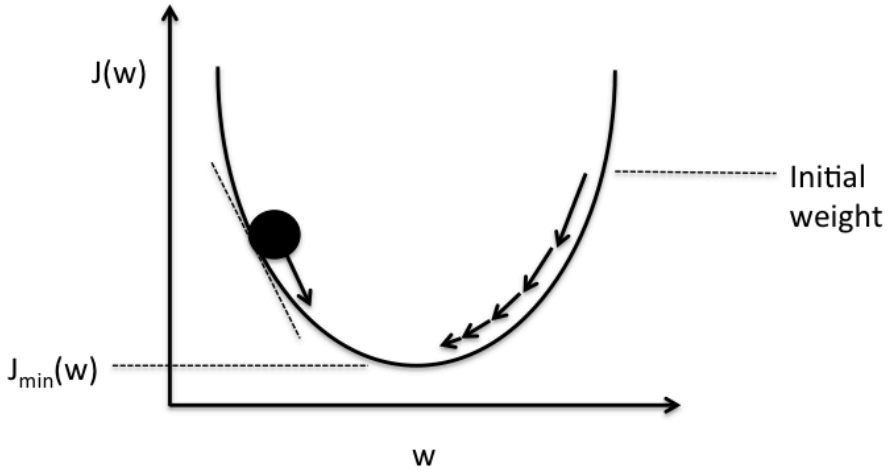
- Imagine uma bola rolando ao longo do gráfico de uma função de custo.



Schematic of gradient descent.

- A medida que a bola rola, ela segue a rota mais íngreme, eventualmente chegando ao fundo.
- Em resumo, é isso que ocorre com o gradiente descendente.

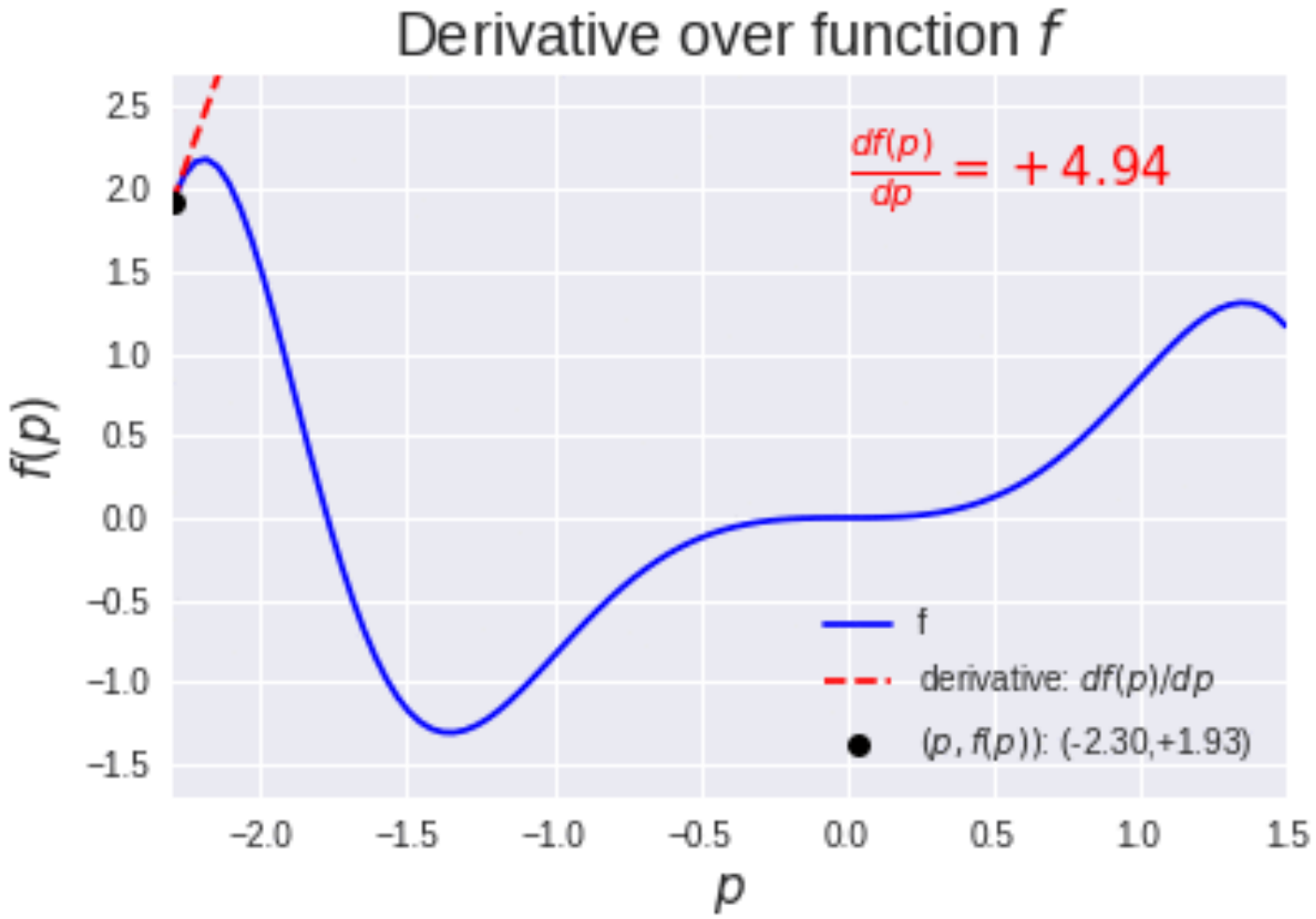
Gradient Descent



Schematic of gradient descent.

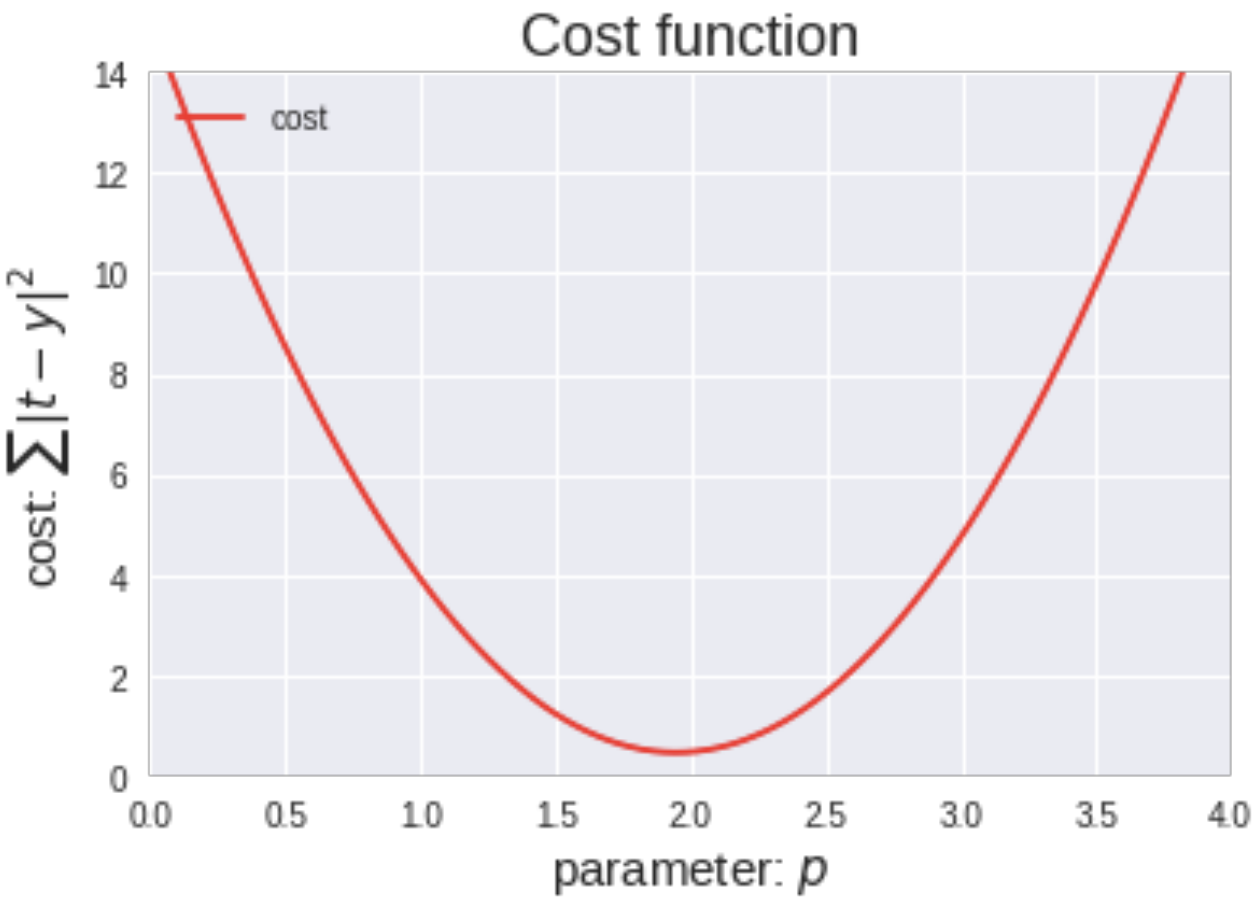
- Escolha um ponto no gráfico, encontre a direção que tem a inclinação mais íngreme, naquela direção, e repita o processo.
- Eventualmente, encontraremos um mínimo da função de custo.
- E porque aquele ponto é o mínimo, ele possui os parâmetros necessários para desenhar nossa linha.

Derivada de uma função f



Função de custo

objetivo: minimizar o erro quadrado



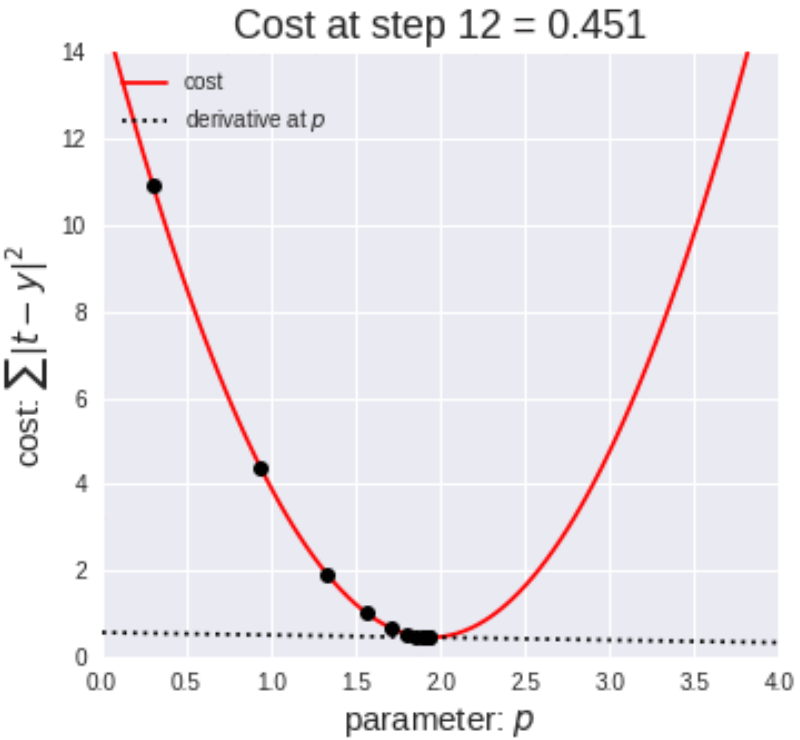
Função de Erro

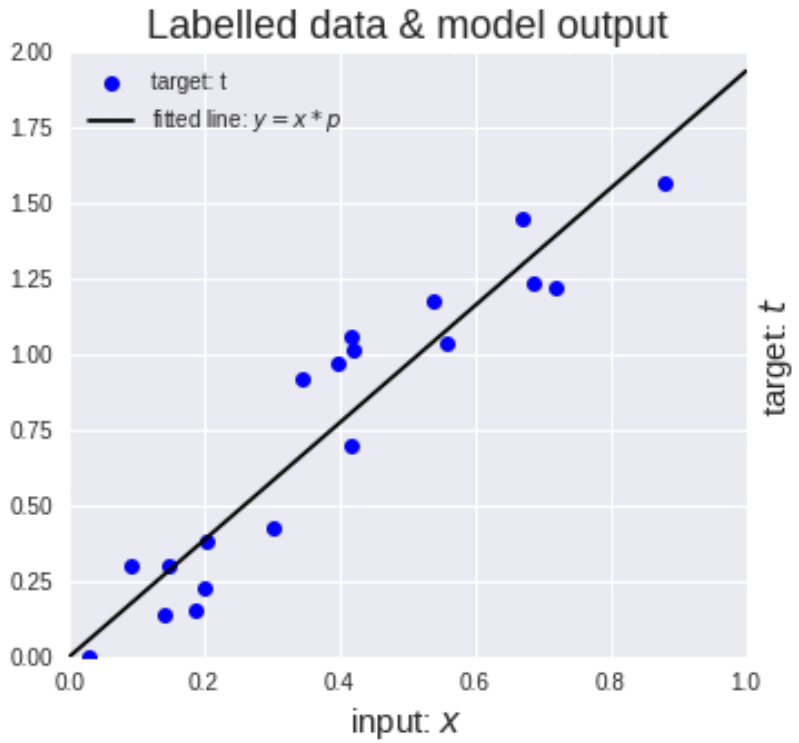
$y = b_1x + b_0$
b_1 é a inclinação
b_0 é onde y é interceptado.

$$Error_{\beta_0,\beta_1} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (\beta_1 x_i + \beta_0))^2$$

Minimizando a função de custo

$$Error_{\beta_0,\beta_1} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (\beta_1 x_i + \beta_0))^2$$





Fonte: https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/

Minimizando a função de custo

$$Error_{\beta_0,\beta_1} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (\beta_1 x_i + \beta_0))^2$$

É necessário calcular a derivada parcial para b0 e b1:

$$\frac{\partial}{\partial\beta_1} = \frac{2}{N}\sum_{i=1}^{N} -x_i(y_i - (\beta_1 x_i + \beta_0))$$

$$\frac{\partial}{\partial\beta_0} = \frac{2}{N}\sum_{i=1}^{N} -(y_i - (\beta_1 x_i + \beta_0))$$

- O erro diminui a cada interação.
- A direção do movimento em cada interação é calculada a partir das 2 derivadas parciais.

Fonte: https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/

Gradient Descent

```
def compute_error_for_line_given_points(b0, b1, x, y):
    N = len(y)
    totalError = 1/N * np.sum((y - (b1 * x + b0)) ** 2)
    return totalError

def step_gradient(b0_current, b1_current, x, y, learning_rate):
    N = len(y)
    b0_gradient = 2/N * np.sum(-(y - ((b1_current * x) + b0_current)))
    b1_gradient = 2/N * np.sum(-x * (y - ((b1_current * x) + b0_current)))
    new_b0 = b0_current - (learning_rate * b0_gradient)
    new_b1 = b1_current - (learning_rate * b1_gradient)
    return new_b0, new_b1

def gradient_descent_runner(x, y, b0, b1, learning_rate, num_iterations):
    for _ in range(num_iterations):
        b0, b1 = step_gradient(b0, b1, x, y, learning_rate)
    return b0, b1
```

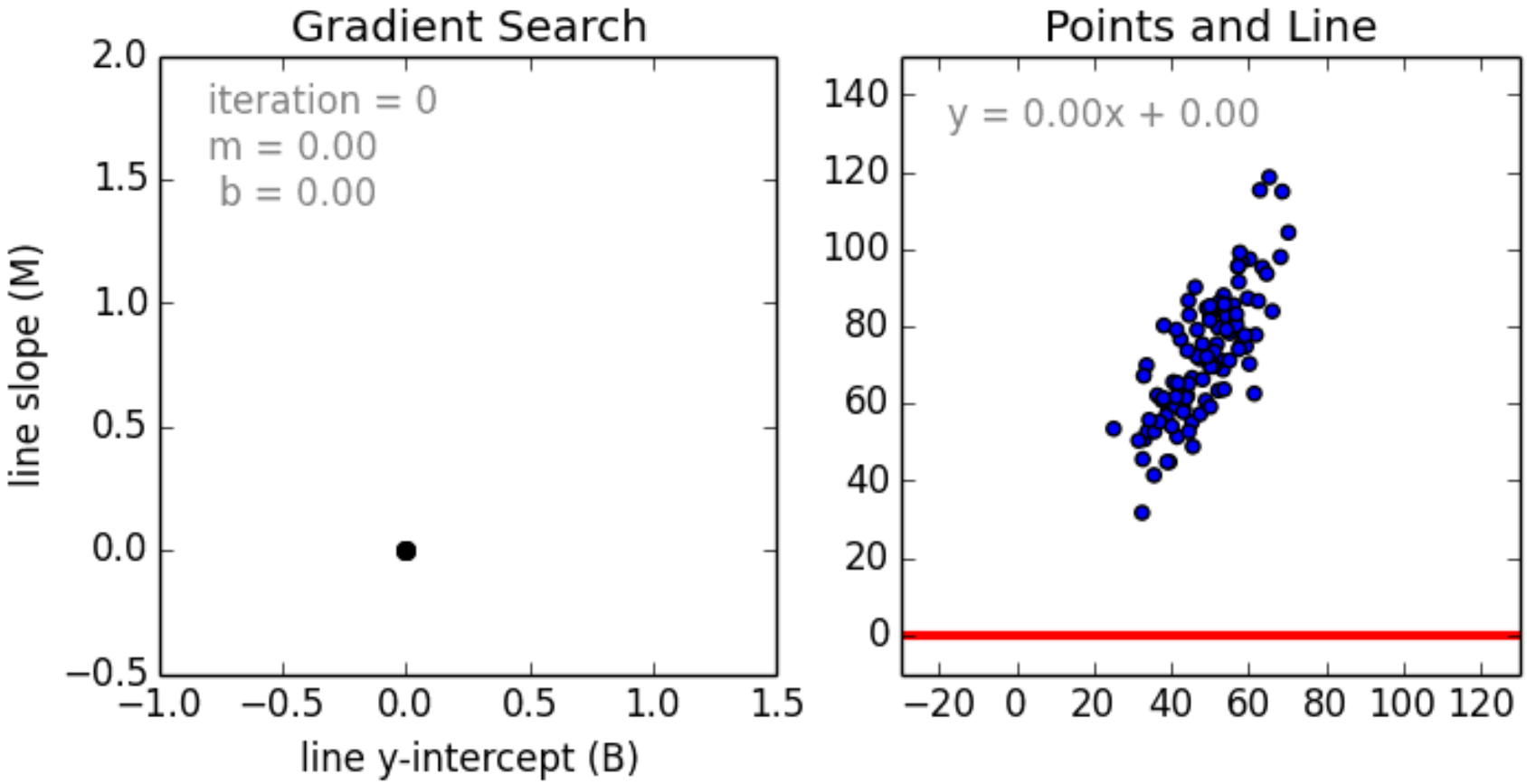
```
b0, b1 = gradient_descent_runner(x, y, initial_b0, initial_b1,
learning_rate, num_iterations)
```

Gradient Descent

- Após 100000 iterações, obtemos
  - b0 = 4.247984440219184
  - b1 = 1.3959992655297515
  - error = 110.78631929745077

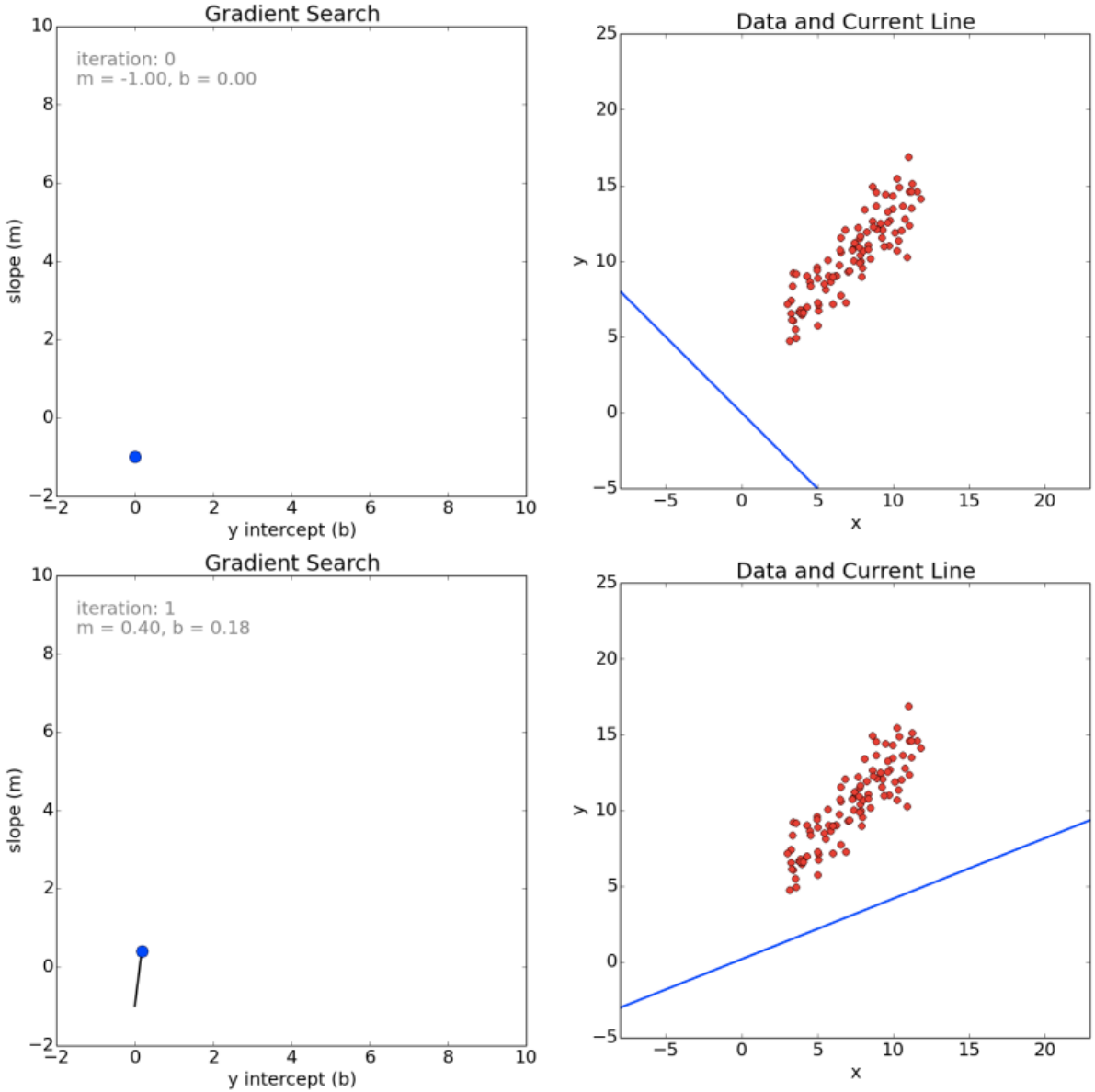
- Linear Regression do Scikit Learn
  - b0 = 7.991020982270399
  - b1 = 1.32243102
  - error = 110.25738346621316

Gradient Descent



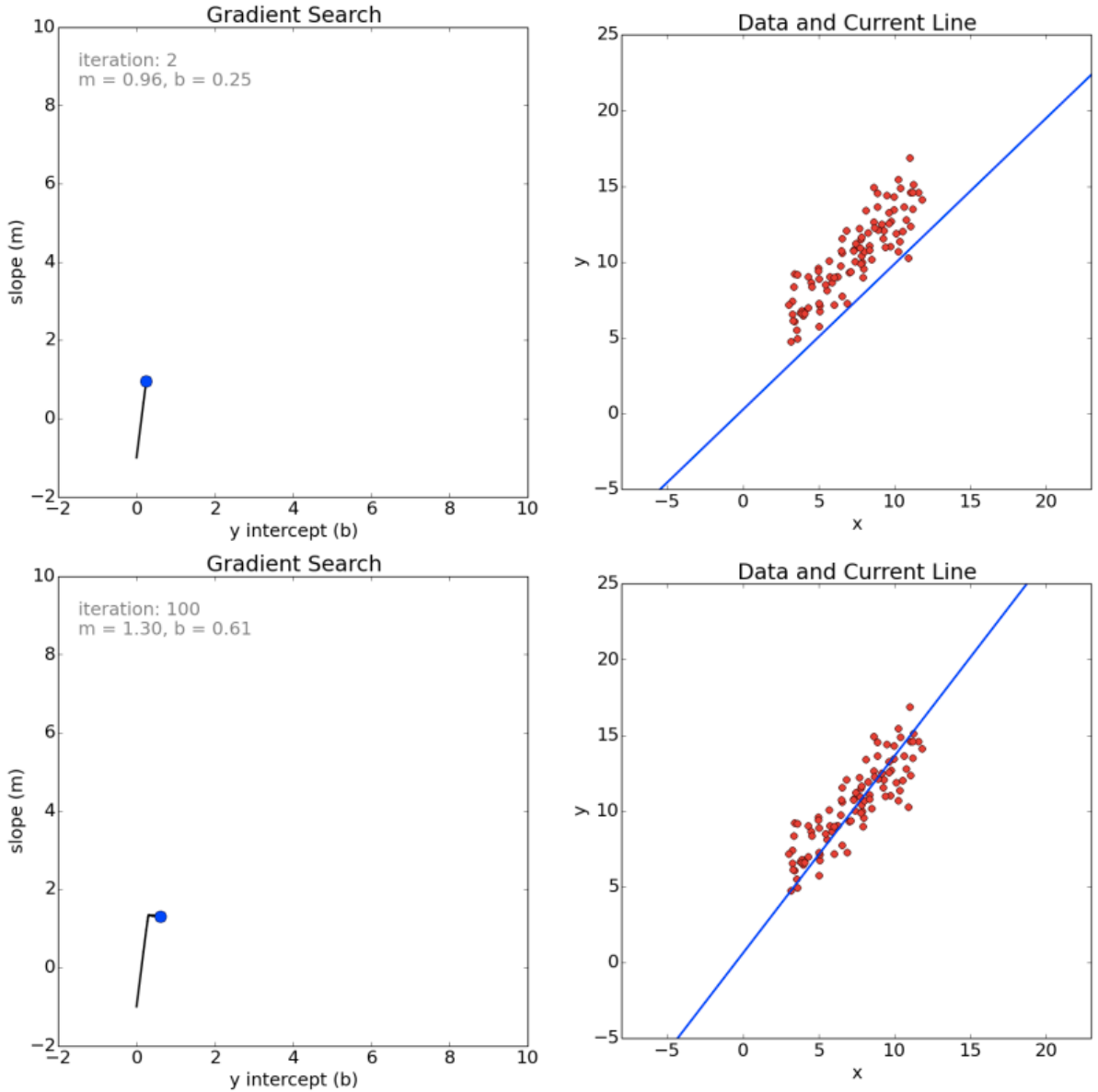
Fonte: https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/

Gradient Descent



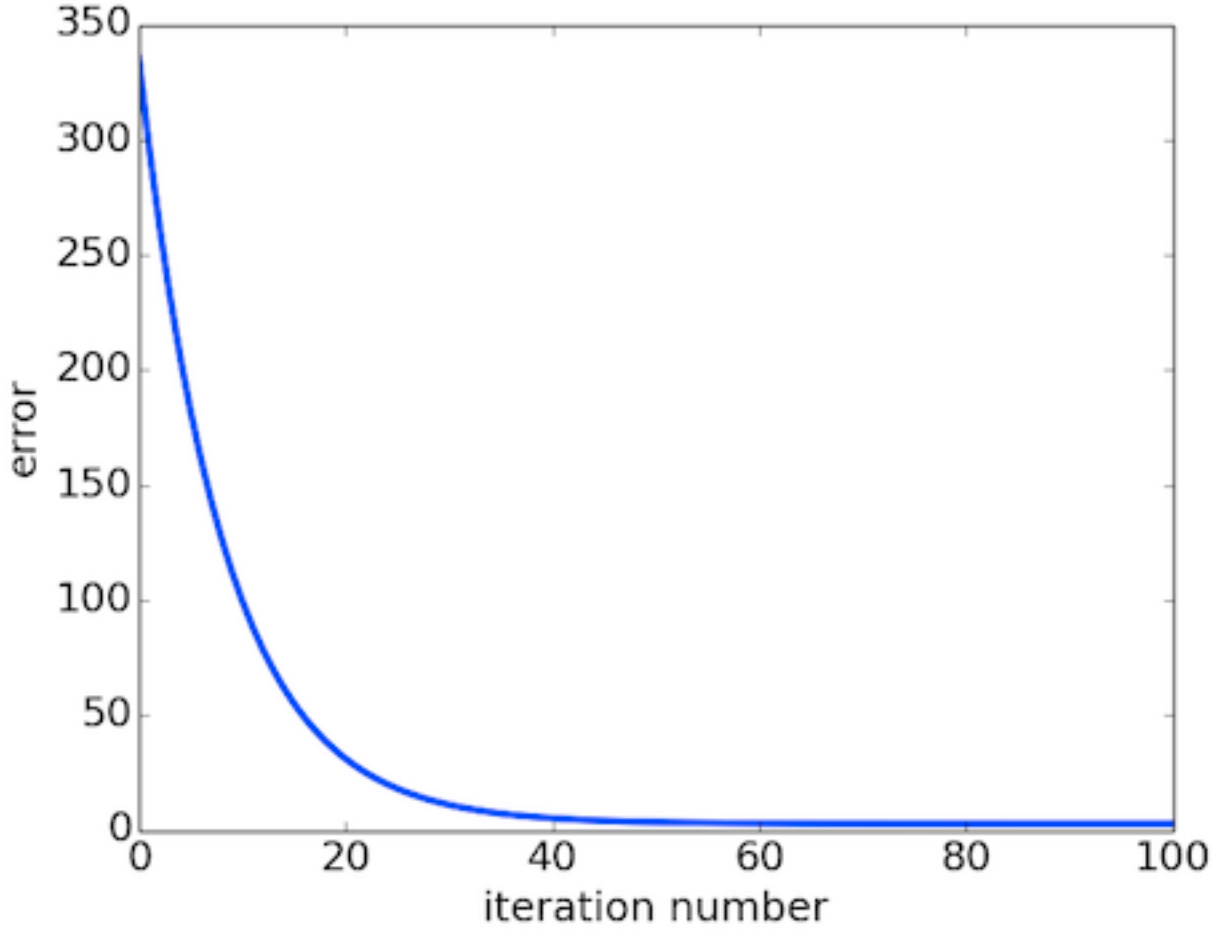
Fonte: https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/

Gradient Descent



Fonte: https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/

Gradient Descent



Fonte: https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/

Multivariate Linear Regression

$$y = b0 + b1 \times x1 + b2 \times x2 + ...$$

Stochastic Gradient Descent

- Gradient Descent is the process of minimizing a function following the slope or gradient of that function.

- Stochastic gradient descent evaluates and updates the coefficients every iteration to minimize the error of a model on our training data.

$$b = b - \text{learning rate} \times \text{error} \times x$$

Stochastic Gradient Descent

Parâmetros

- Learning Rate
  - Used to limit the amount that each coefficient is corrected each time it is updated.

- Epochs
  - The number of times to run through the training data while updating the coefficients.

$$b_1(t+1) = b_1(t) - \text{learning rate} \times \text{error}(t) \times x_1(t)$$

$$b_0(t+1) = b_0(t) - \text{learning rate} \times \text{error}(t)$$

Feature Scaling

$$x^{(i)}_{std} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

$$x^{(i)}_{norm} = \frac{x^{(i)} - \mathbf{x}_{min}}{\mathbf{x}_{max} - \mathbf{x}_{min}}$$

standardization

min-max scaling ("normalization")

	input	standardized	normalized
0	0	-1.46385	0.0
1	1	-0.87831	0.2
2	2	-0.29277	0.4
3	3	0.29277	0.6
4	4	0.87831	0.8
5	5	1.46385	1.0

Standardization

$$\text{standardized\_value}_i = \frac{\sum_{i=1}^{n}(value_i - mean)}{stdev}$$

- Standardization is a rescaling technique that refers to centering the distribution of the data on the value 0 and the standard deviation to the value 1.
- The mean and the standard deviation summarize a normal distribution.
- Standardization is a scaling technique that assumes your data conforms to a normal distribution.
- If a given data attribute is normal or close to normal, this is probably the scaling method to use.

Normalization

$$\text{scaled value} = \frac{value - min}{max - min}$$

- Normalization can refer to different techniques depending on context.
- Here, we use normalization to refer to rescaling an input variable to the range between 0 and 1.
- Normalization is a scaling technique that does not assume any specific distribution.
- If your data is not normally distributed, consider normalizing it prior to applying your machine learning algorithm.

Normalization

```
np.random.seed(0)
x = np.random.rand(20)
x = (x * 100).round(2)
x = np.resize(x, (20, 1))
```

```
[[ 54.88]
 [ 71.52]
 [ 60.28]
 [ 54.49]
 [ 42.37]
 [ 64.59]
 [ 43.76]
 [ 89.18]
 [ 96.37]
 [ 38.34]
 [ 79.17]
 [ 52.89]
 [ 56.8 ]
 [ 92.56]
 [  7.1 ]
 [  8.71]
 [  2.02]
 [ 83.26]
 [ 77.82]
 [ 87.  ]]
```

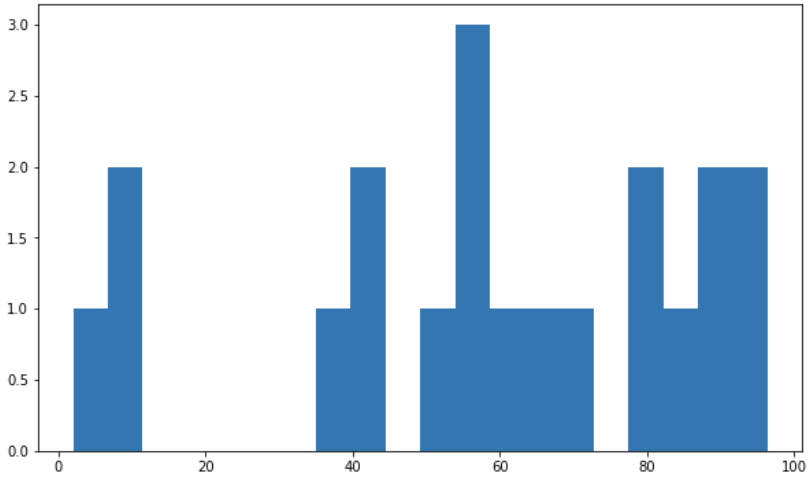
$$x_{norm}^{(i)} = \frac{x^{(i)} - \mathbf{x}_{min}}{\mathbf{x}_{max} - \mathbf{x}_{min}}$$

```
def normalize(X):
    X_norm = np.copy(X)
    n_cols = X.shape[1]
    for i in range(n_cols):
        X_norm[:, i] = (X[:, i] - np.min(X[:, i])) /
                       (np.max(X[:, i]) - np.min(X[:, i]))
    return X_norm
```

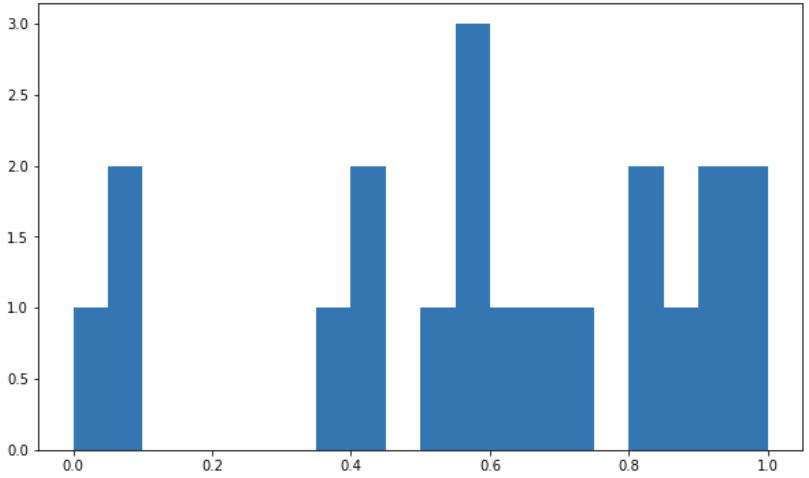
Normalization

```
x_norm = normalize(x)
```

```
plt.hist(x, bins=20)
```




```
x
---
mean: 58.16,
std: 27.59,
min: 2.02,
max: 96.37
```




```
x_norm
---
mean: 0.59,
std: 0.29,
min: 0.0,
max: 1.0
```

```
[[ 0.56025437],
 [ 0.73661897],
 [ 0.61748808],
 [ 0.55612083],
 [ 0.42766296],
 [ 0.66316905],
 [ 0.44239534],
 [ 0.92379438],
 [ 1.        ],
 [ 0.38494966],
 [ 0.81770005],
 [ 0.53916269],
 [ 0.58060413],
 [ 0.95961844],
 [ 0.05384208],
 [ 0.0709062 ],
 [ 0.        ],
 [ 0.86104928],
 [ 0.80339163],
 [ 0.90068892]]
```

Standardization

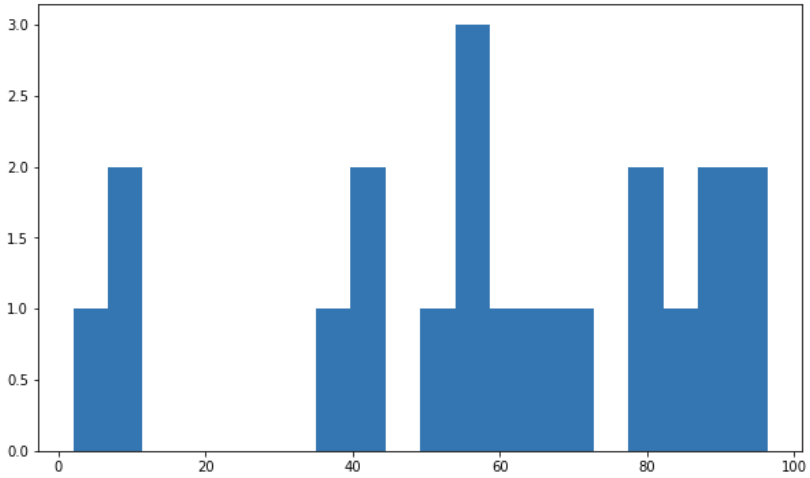
$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

```
def standardize(X):
    X_std = np.copy(X)
    n_cols = X.shape[1]
    for i in range(n_cols):
        X_std[:, i] = (X[:, i] - np.mean(X[:, i])) /
                      np.std(X[:, i])
    return X_std
```

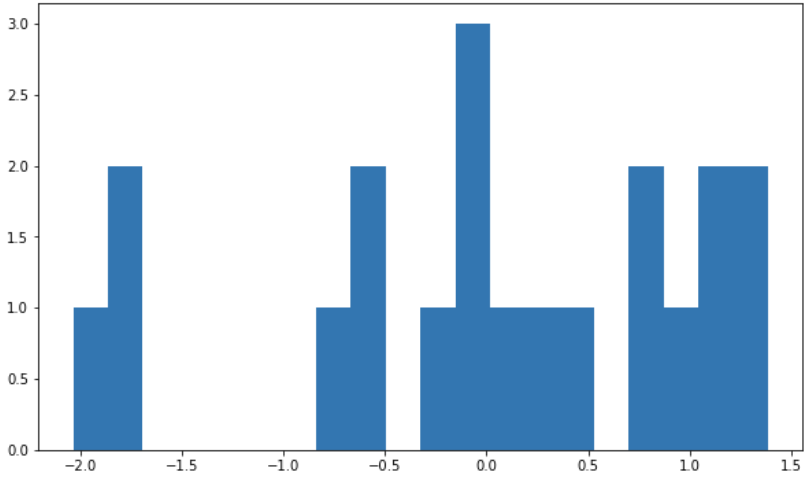
Standardization

```
x_std = standardize(x)
```

```
plt.hist(x, bins=20)
```




```
x
---
mean: 58.16,
std: 27.59,
min: 2.02,
max: 96.37
```




```
x_std
---
mean: 0.0,
std: 1.0,
min: -2.03,
max: 1.38
```

```
[[-0.11870903],
 [ 0.48434953],
 [ 0.07699507],
 [-0.13284322],
 [-0.5720902 ],
 [ 0.23319593],
 [-0.52171451],
 [ 1.12437442],
 [ 1.38495081],
 [-0.71814345],
 [ 0.761597  ],
 [-0.19082962],
 [-0.04912535],
 [ 1.24687069],
 [-1.85032791],
 [-1.79197909],
 [-2.03443473],
 [ 0.90982474],
 [ 0.71267098],
 [ 1.04536795]]
```

Referências

- https://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html

Obrigado!

Dúvidas, comentários, sugestões?

Regis Pires Magalhães
regismagalhaes@ufc.br

UFC