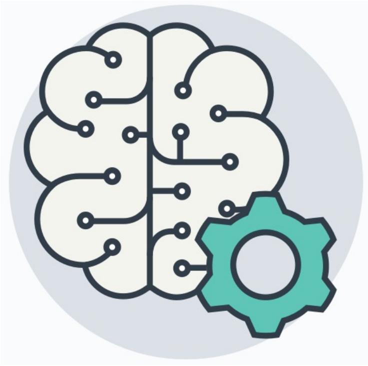
Aprendizado de Máquina

Medidas Estatísticas, Distribuições e Escalonamento de Features



Prof. Regis Pires Magalhães

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

Medidas estatísticas

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

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

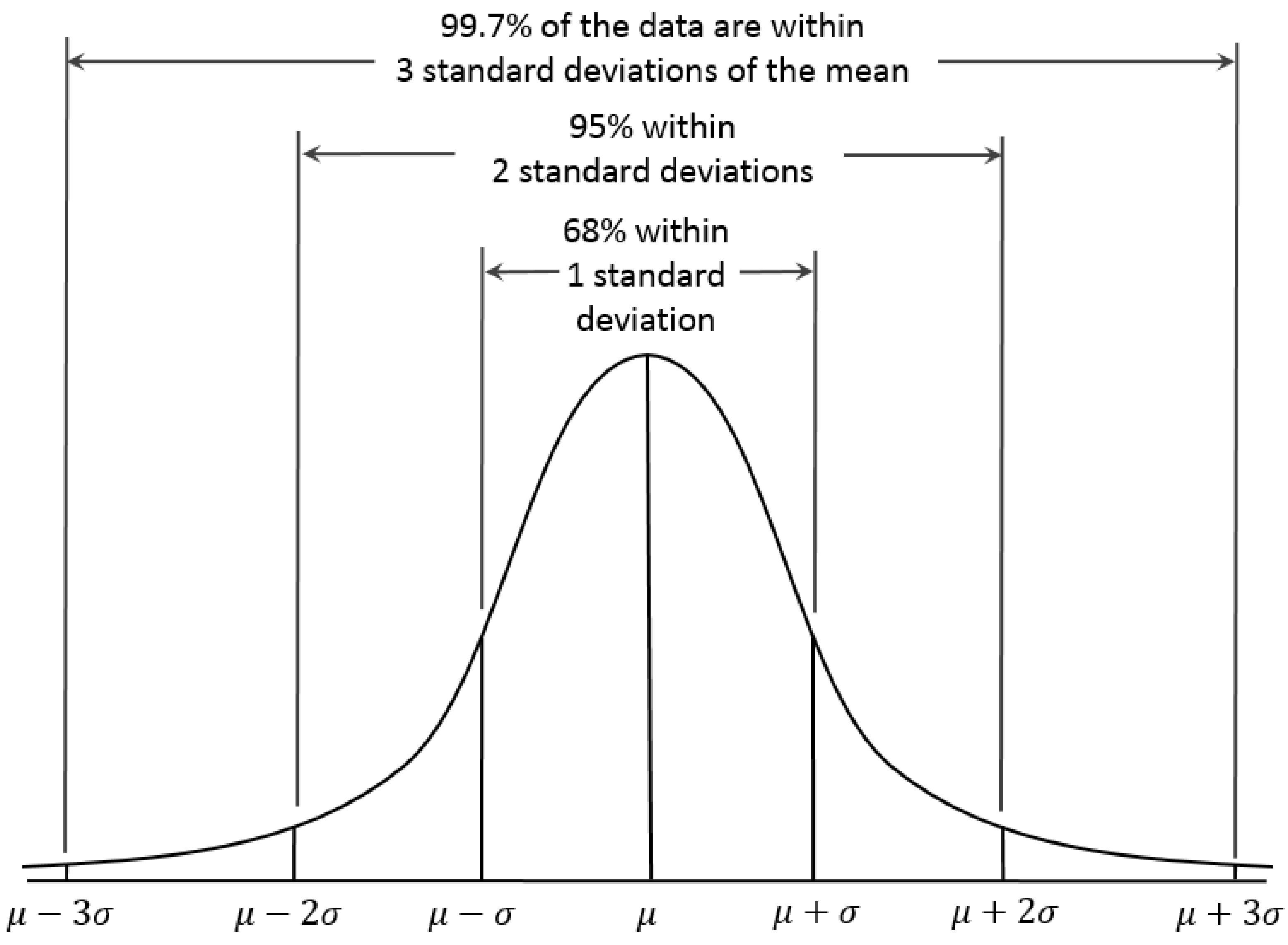
$$covariance = \sum_{i=1}^{n}((x_i - mean(x)) \times (y_i - mean(y)))$$

```python
# Example of Estimating Mean and Variance
# Calculate the mean value of a list of numbers
def mean(values):
  return sum(values) / float(len(values))

# Calculate the variance of a list of numbers
def variance(values, mean):
  return sum([(x-mean)**2 for x in values])

# calculate mean and variance
dataset = [[1, 1], [2, 3], [4, 3], [3, 2], [5, 5]]
x = [row[0] for row in dataset]
y = [row[1] for row in dataset]
mean_x, mean_y = mean(x), mean(y)
var_x, var_y = variance(x, mean_x), variance(y,
mean_y)
print('x stats: mean=%.3f variance=%.3f' % (mean_x,
var_x))
print('y stats: mean=%.3f variance=%.3f' % (mean_y,
var_y))
```

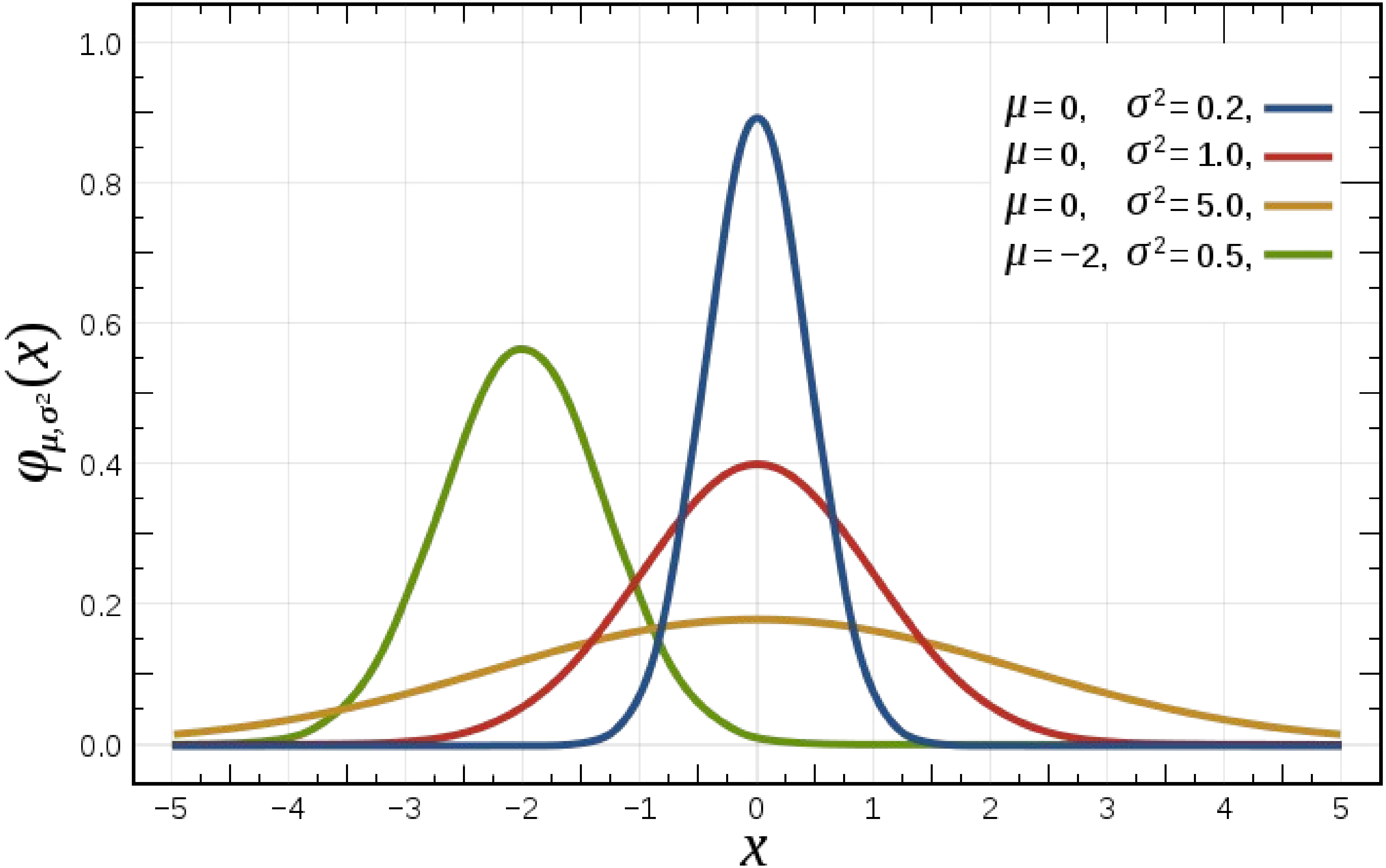
```python
# Calculate covariance between x and y
def covariance(x, mean_x, y, mean_y):
  covar = 0.0
  for i in range(len(x)):
    covar += (x[i] - mean_x) * (y[i] -
mean_y)
  return covar
```

```
x stats: mean=3.000 variance=10.000
y stats: mean=2.800 variance=8.800
Covariance: 8.000
```

Distribuição Normal

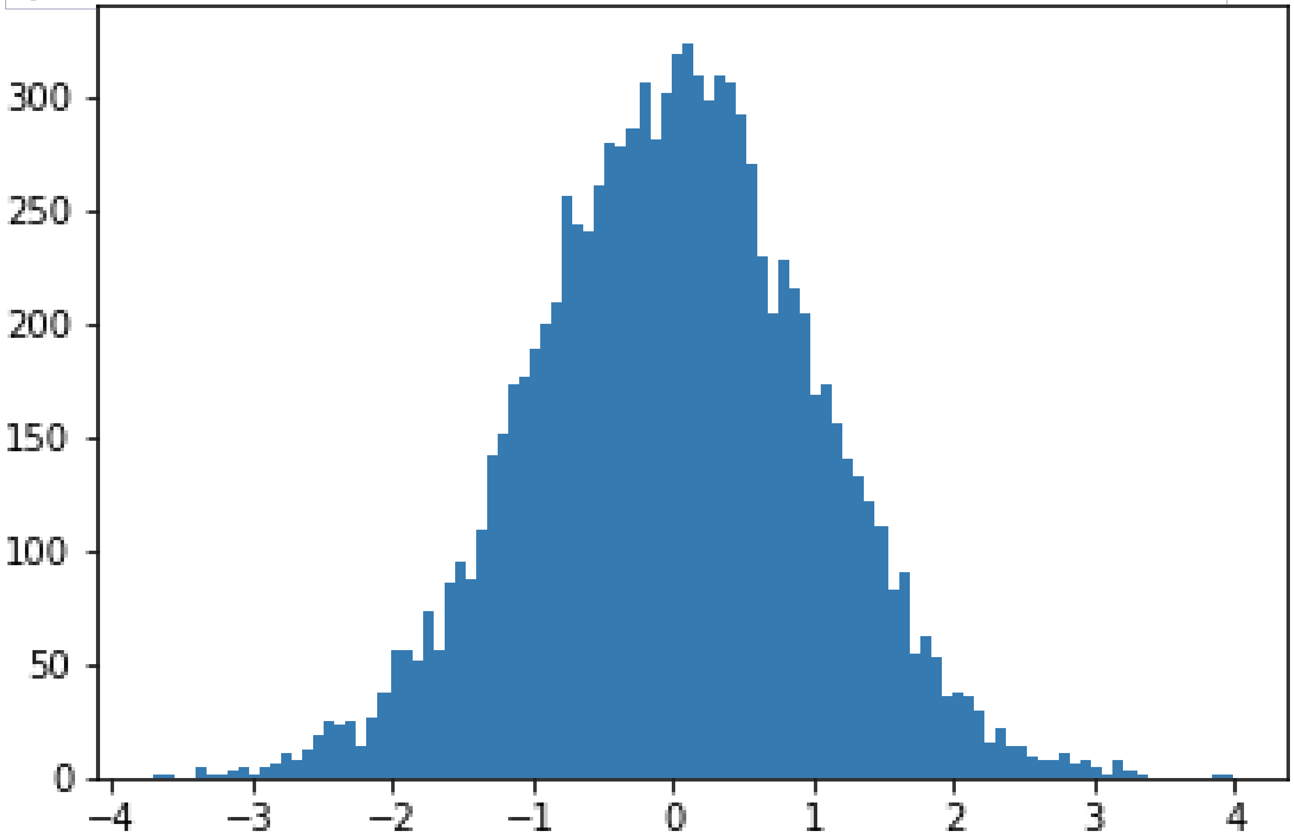


Distribuição Normal



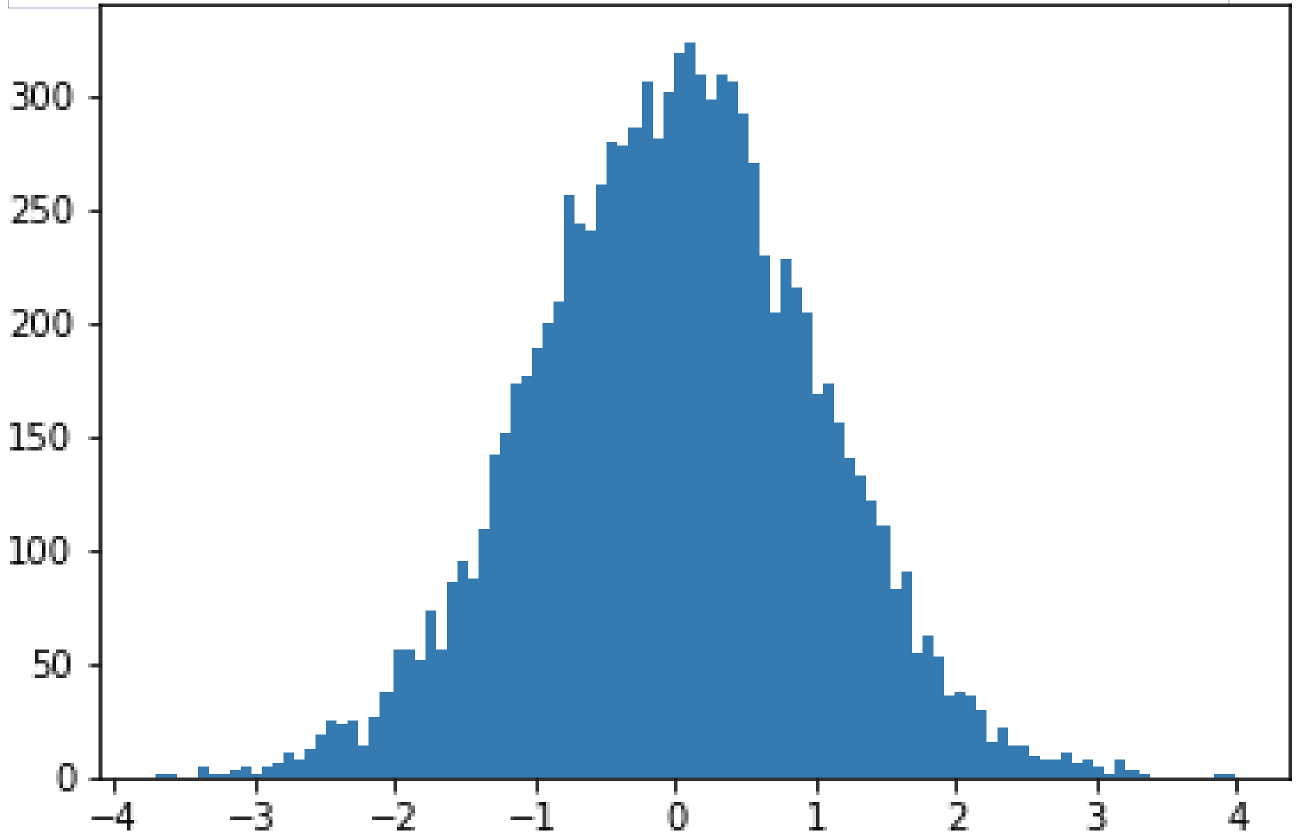
Distribuição Normal

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



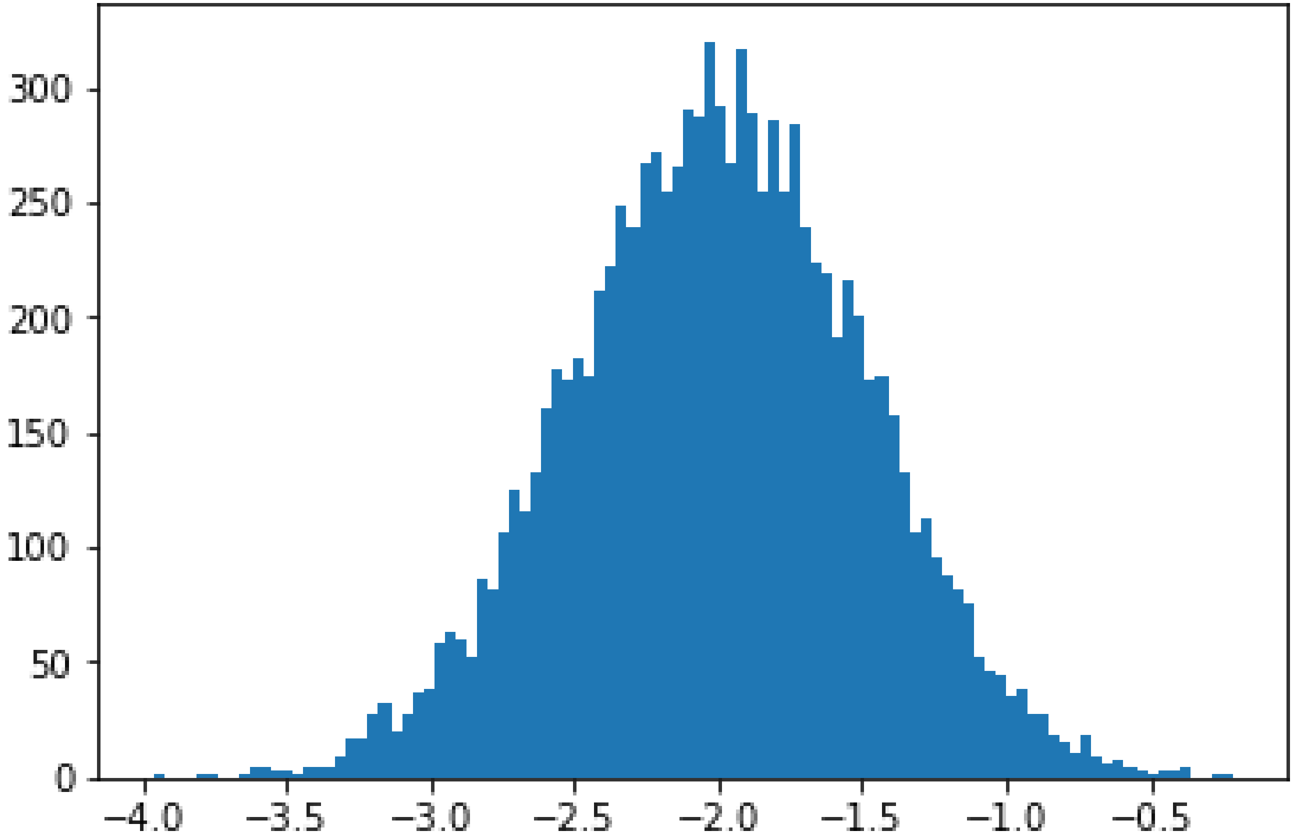
Distribuição Normal

```
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])

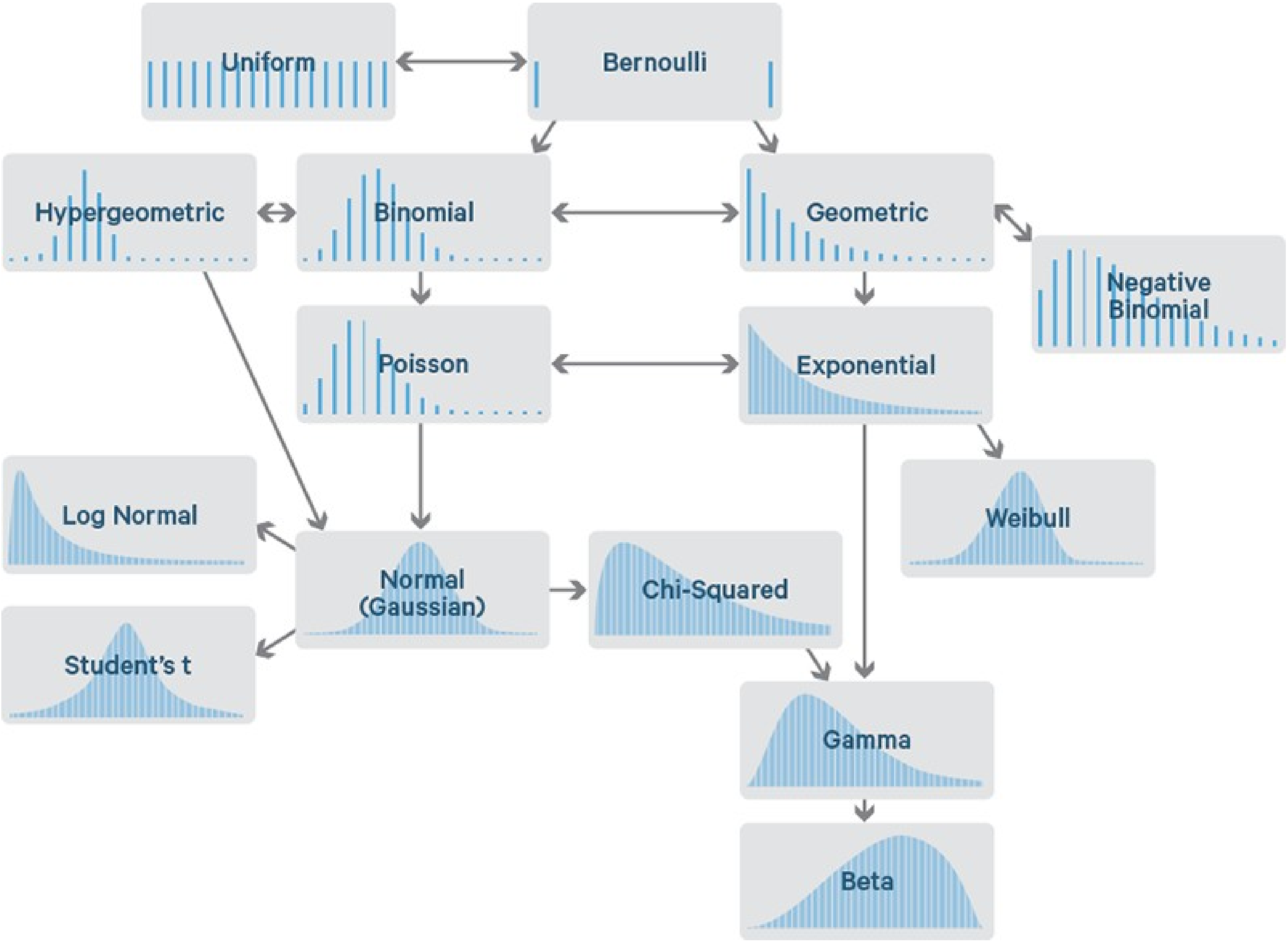
- `standard_t` (df[, size])

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

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

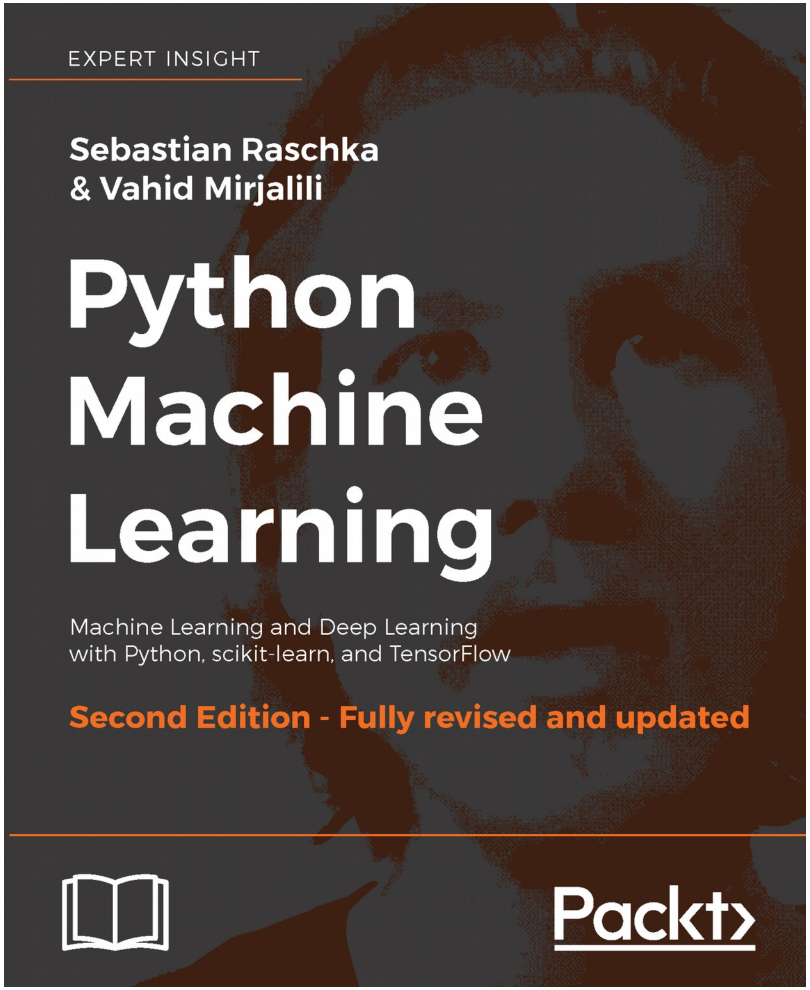
- `weibull` (a[, size])
- `zipf` (a[, size])

Distribuições



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

Main Reference



Python Machine Learning

Chapter 4 - Building Good Training Sets – Data Preprocessing

Bringing features onto the same scale

Feature Scaling

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

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

standardizatio
n

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.  ]]
```

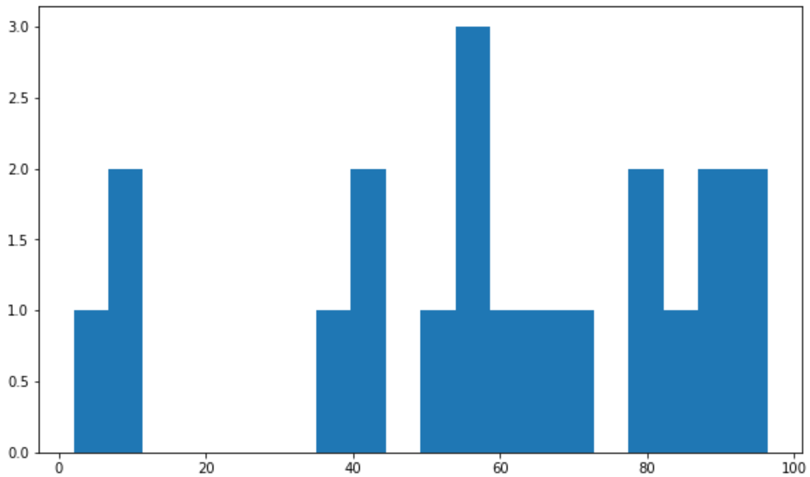
$$x_{norm}^{(i)} = \frac{x^{(i)} - \mathbf{x}_{min}}{\mathbf{x}_{max} - \mathbf{x}_{min}}$$

```python
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]))
```

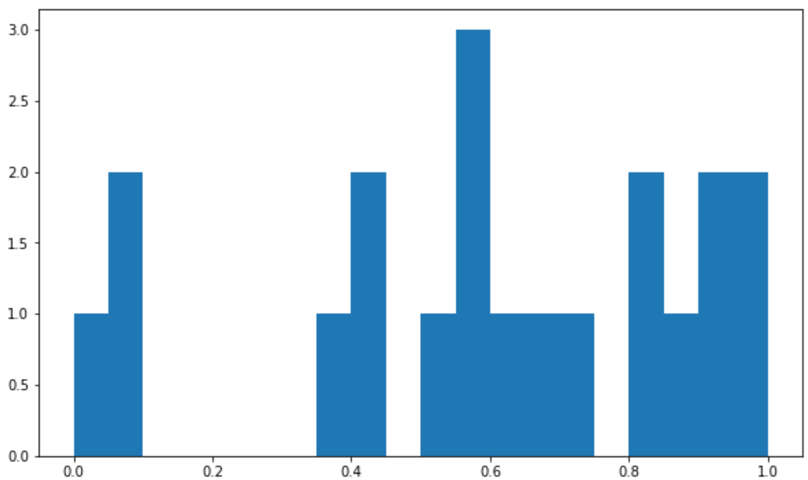
Normalization

```
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

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

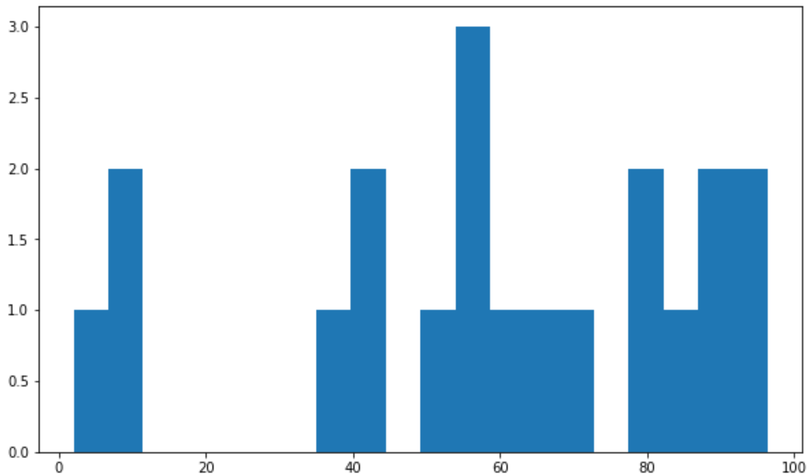
```
def standardize(X):
    X_std = np.copy(X)
    n_cols = X.shape[1]
    for i in range(n_cols):
        X_std[:, i] = (X[:, i] - np.mean(X[:, i])) /
                              np.std(X[:, i])
    return X_std
```

Standardization

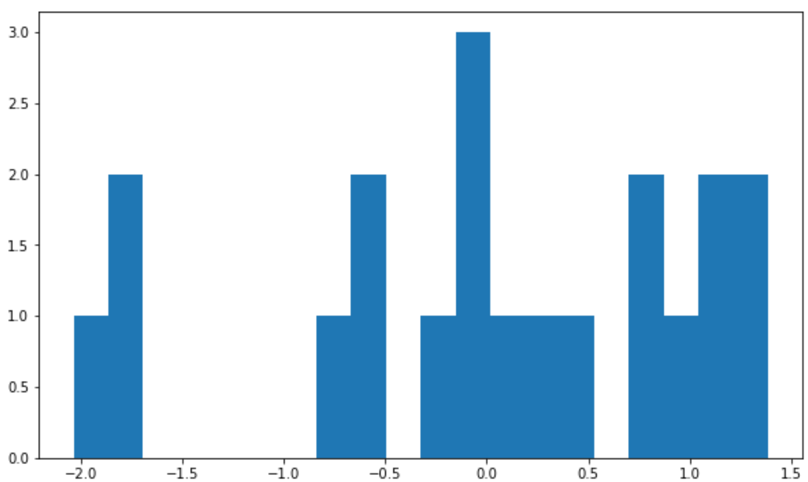
```
x_std = standardize(x)
```

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

```
[[-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]]
```




```
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
```

Obrigado!

Dúvidas, comentários, sugestões?

Regis Pires Magalhães
regismagalhaes@ufc.br
UFC