Redes Auto-organizáveis de Kohonen

Gustavo Stein Santos Mattos Araújo dos Santos

Humberto da Silva Neto

Agenda

- Redes Auto-organizáveis de Kohonen
  - Introdução
  - Características
  - Funcionamento
- Aplicações
- Implementação
  - SOMTF
  - SOMPY
  - Testes
  - Problema proposto e resultados

Redes Auto-organizáveis de Kohonen

Introdução

- Proposta por Teuvo Kohonen, 1982
- Inspiração biológica no córtex cerebral
- Aprendizado competitivo
- Estrutura reticulada

Características

Redes Neurais usuais

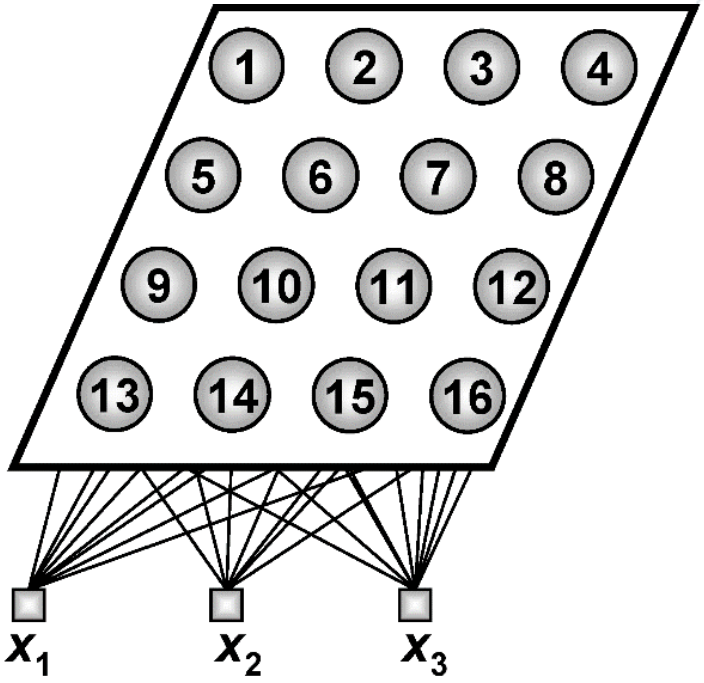
- ~~Série de camadas~~
- ~~Correção por erro~~
- ~~Aprendizado supervisionado~~

SOM

- Rede 2D*
- Por competição
- Aprendizado não supervisionado

Características

- Adapta-se à organização topológica de um dataset, permitindo a visualização de cluster's em potencial
- Lida melhor com variáveis contínuas



Características

Vantagens

- Representação visual (Redução da dimensionalidade)
- Organização de dados de forma a tornar inferências sobre os mesmos possível
- Grande número de variáveis

Desvantagens

- Não lida bem com variáveis categóricas
- Computacionalmente dispendiosa
- Soluções nem sempre facilmente correlacionáveis

Funcionamento: treinamento

Definir mapa topológico

Definir vizinhança — σ

Obter conjunto de amostras — x

Inicializar o vetor de pesos — w

NORMALIZAR OS VETORES!

TREINAMENTO

Identificar as regiões correspondentes

Decaimento linear da taxa de aprendizado

Encontrar o neurônio vencedor

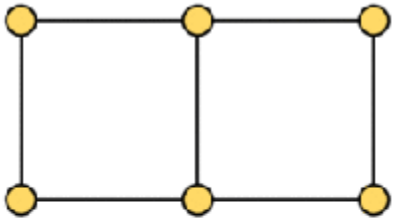
Ajustar o vetor de pesos

$w^{(v)} \leftarrow w^{(v)} + \eta(x^{(k)} - w^{(v)})$

$w^{(\Omega)} \leftarrow w^{(\Omega)} + \eta * \alpha^{(\Omega)} (w^{(v)} - w^{(\Omega)})$

$\alpha^{(\Omega)} = \exp(-||w^{(v)} - w^{(\Omega)}||^2 / 2\sigma^2)$

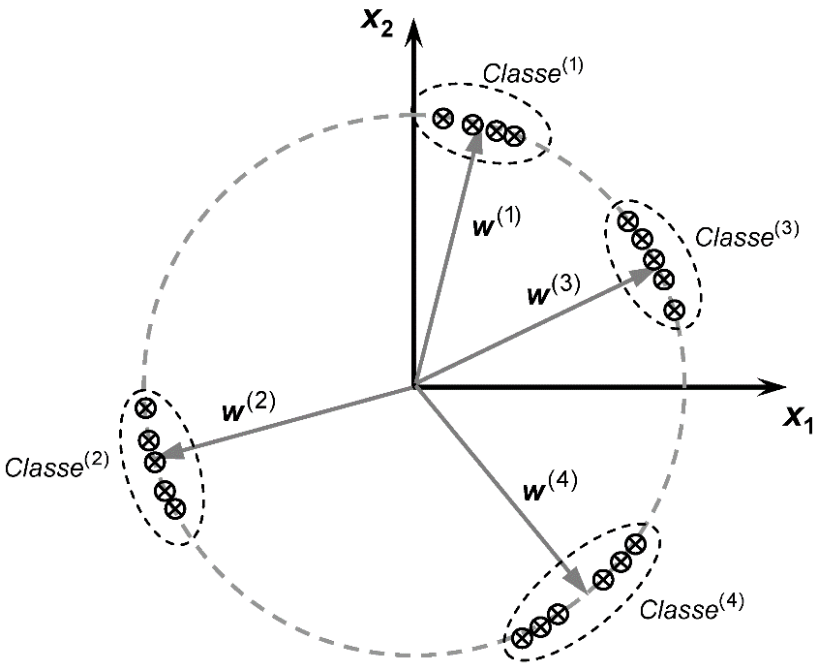
Funcionamento: treinamento

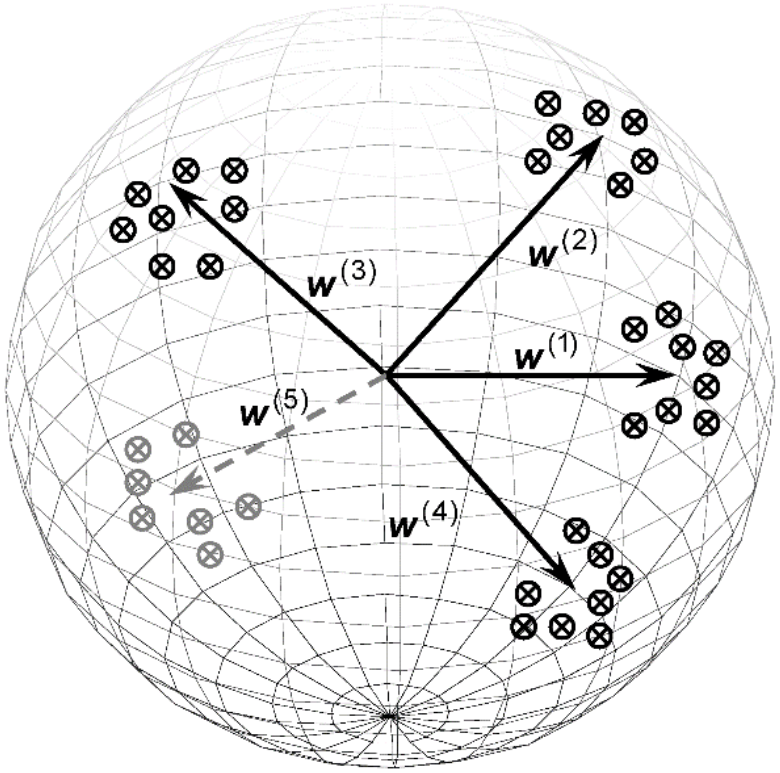


Step 0: Position neurons (orange) in data space.

https://algobeans.com/2017/11/02/self-organizing-map/

Funcionamento: agrupamento





Funcionamento: algoritmos

Início {Algoritmo Kohonen – Fase de Treinamento}

<1> Definir o mapa topológico da rede;

<2> Montar os conjuntos de vizinhança $\{\Omega^{(j)}\}$;

<3> Iniciar o vetor de pesos de cada neurônio $\{\boldsymbol{w}^{(j)}\}$ considerando os valores das n_1 primeiras amostras de treinamento;

<4> Obter o conjunto de amostras de treinamento $\{\boldsymbol{x}^{(k)}\}$;

<5> Normalizar os vetores de amostras e de pesos;

<6> Especificar a taxa de aprendizagem $\{\eta\}$;

<7> Iniciar o contador de número de épocas { *época* ← 0 };

<8> Repetir as instruções:

- <8.1> Para todas as amostras de treinamento $\{\boldsymbol{x}^{(k)}\}$, fazer:
  - <8.1.1> Calcular as distâncias euclidianas entre $\boldsymbol{x}^{(k)}$ e $\boldsymbol{w}^{(j)}$, conforme a expressão (8.1);
  - <8.1.2> Declarar como vencedor o neurônio *j* que contenha a menor distância euclidiana:

    $$vencedor = \arg \min_{j}\{|| \boldsymbol{x}^{(k)} - \boldsymbol{w}^{(j)} ||\}$$
  - <8.1.3> Ajustar o vetor de pesos do vencedor, conforme a regra 1 da expressão (8.4);
  - <8.1.4> Ajustar o vetor de pesos dos neurônios vizinhos ao vencedor, definidos em $\Omega^{(j)}$, conforme regra 2 da expressão (8.4) ou (8.5);
  - <8.1.5> Normalizar o vetor de pesos que foi ajustado na instrução anterior;
- <8.2> *época* ← *época* + 1;

Até que: não haja mudanças significativas nos vetores de pesos;

<9> Analisar o mapa visando extração de características;

<10> Identificar regiões que possibilitem a definição de classes.

Fim {Algoritmo Kohonen – Fase de Treinamento}

Início {Algoritmo Kohonen – Fase de Operação}

<1> Apresentar a amostra $\{\boldsymbol{x}\}$ a ser classificada e normalizar;

<2> Assumir os vetores de pesos $\{\boldsymbol{w}^{(j)}\}$ já ajustados durante a fase de treinamento;

<3> Executar as seguintes instruções:

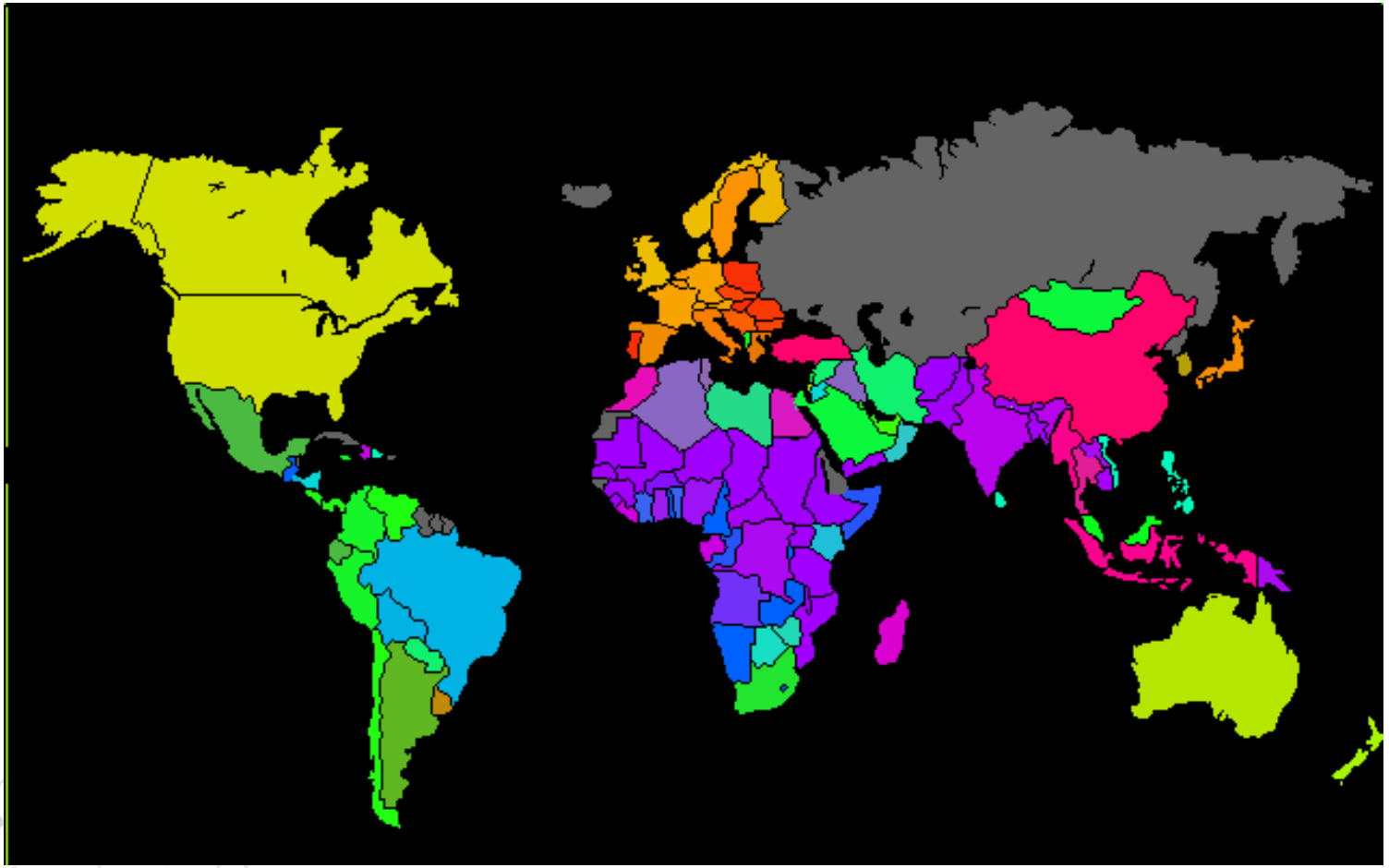
- <3.1> Calcular as distâncias euclidianas entre $\boldsymbol{x}$ e $\boldsymbol{w}^{(j)}$, conforme a expressão (8.1);
- <3.2> Declarar como vencedor o neurônio *j* que contenha a menor distância euclidiana;
- <3.3> Localizar o neurônio vencedor dentro do mapa auto-organizável;
- <3.4> Associar a amostra à classe que foi identificada a partir da confecção do mapa de contexto;

<4> Disponibilizar a eventual classe em que a amostra foi associada.

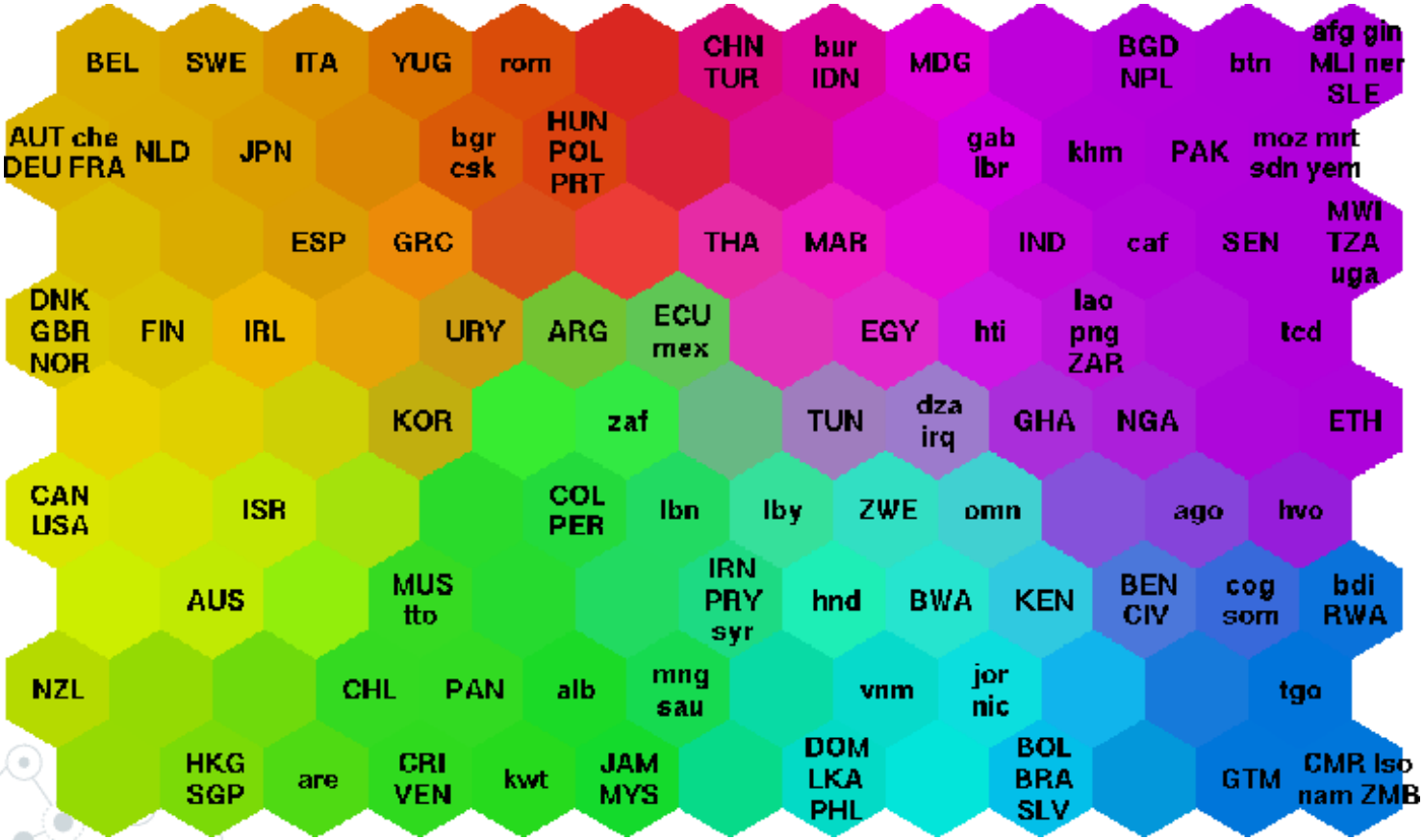
Fim {Algoritmo Kohonen – Fase de Operação}

Aplicações

Classificação de países por condição socioeconômica

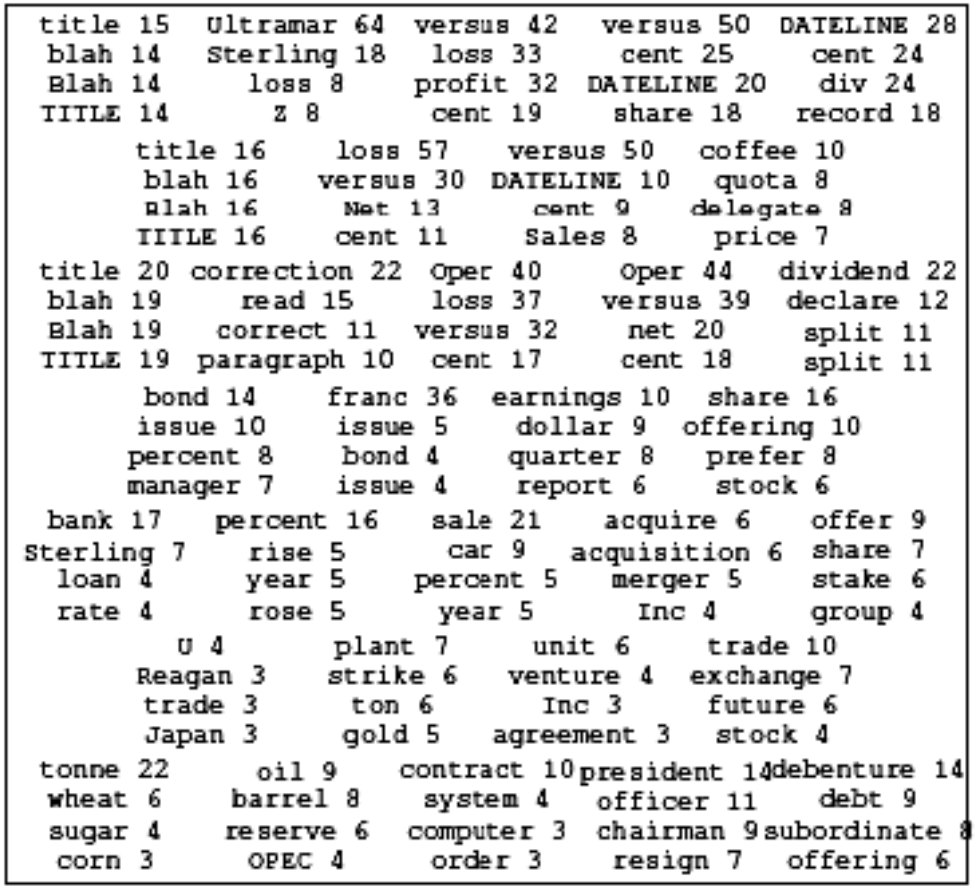


Classificação de países por condição socioeconômica



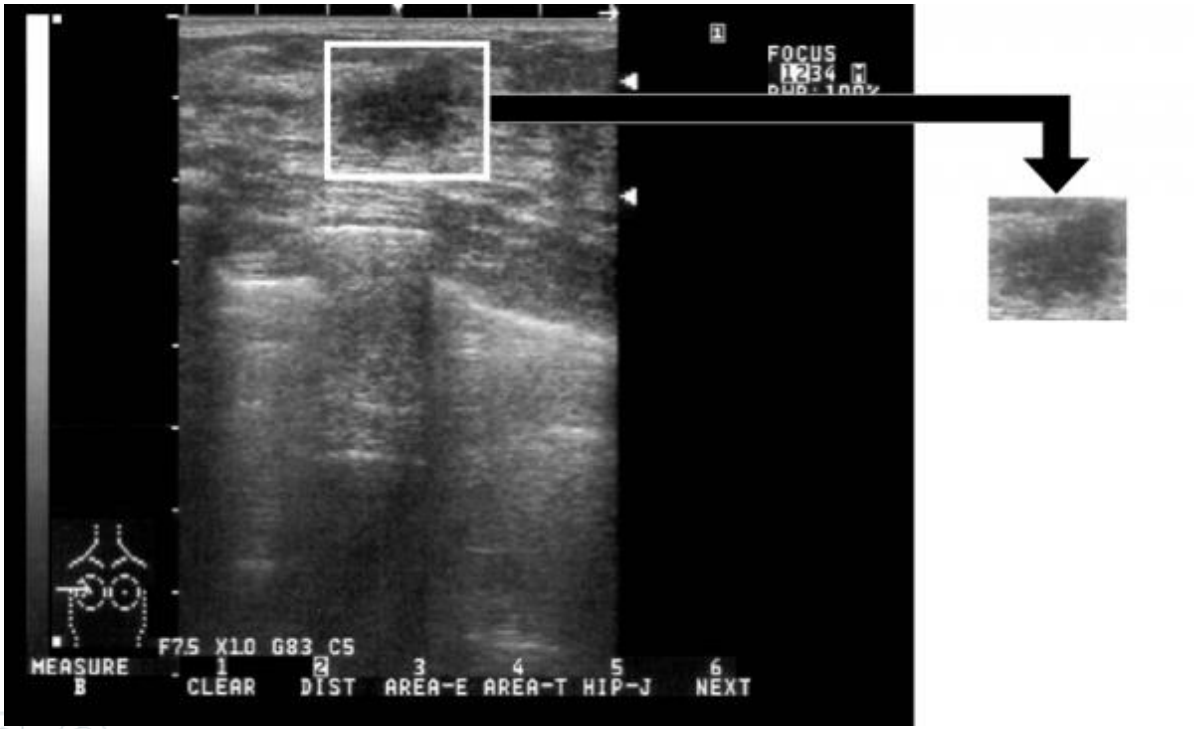
Similaridade em Textos

- 80% dos textos obtidos em datamining não são estruturados ou catalogados
- Combinação de PNL com SOM's para sintetização e classificação de textos



Diagnóstico de Cancer de Mama

- A intensidade e textura de uma anomalia pode ser usada para classificar a mesma como maligna ou benigna



Implementação

SOMTF

somtf.py

```python
import tensorflow as tf
import numpy as np

class SOM(object):
  '''
  2-D Self-Organizing Map with Gaussian Neighbourhood function and linearly decreasing learning rate.
  '''
  # To check if the SOM has been trained
  _trained = False

  def __init__(self, m, n, dim, n_iterations = 100, alpha=.1, sigma=None, decay_in_time = True):
    '''
    Initializes all necessary components of the TensorFlow Graph

    Args:
      m: map width
      n: map height
      dim: dimension of the input vector
      n_iterations: number of iterations (epochs) - Default value is 100
      alpha: initial value of the learning rate - Default value is 0.1
      sigma: initial value of neighborhood radius - Default value is max(m, n)/2
      decay_in_time: decay the learning rate value through time
    '''
    # Assign input variables
    self._m = m
    self._n = n
    self._n_iterations = int(n_iterations)

    alpha = float(alpha)
    if sigma is None:
      sigma = max(m,n)/2.0
    else:
      sigma = float(sigma)

    if decay_in_time is not True:
      decay_in_time = False
```

master* 0 0 Ln 54, Col 28 Spaces: 2 UTF-8 LF Python

https://github.com/hsneto/redes_neurais/blob/master/neural_networks/somtf.py

SOMPY

sompy.py

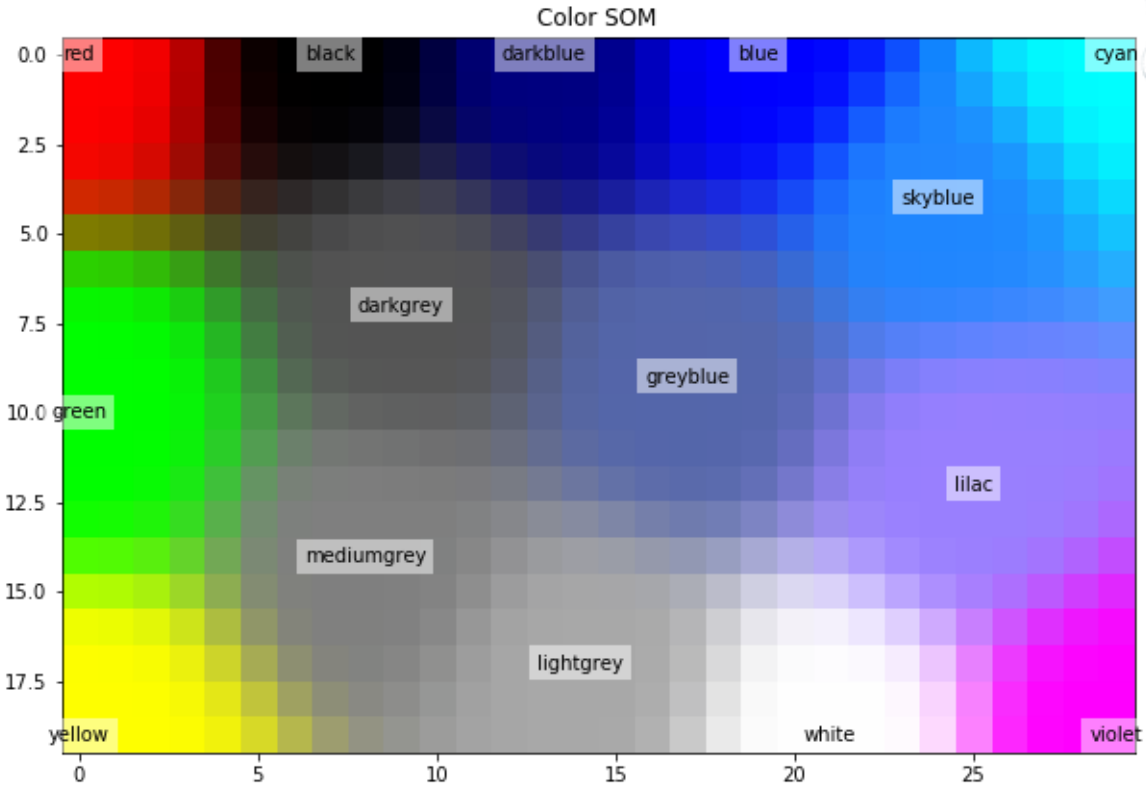
```python
import numpy as np

class SOM(object):
  '''
  2-D Self-Organizing Map with Gaussian Neighbourhood function and linearly decreasing learning rate.
  '''
  # To check if the SOM has been trained
  _trained = False

  def __init__(self, m, n, dim, n_iterations = 100, alpha=.1, sigma=None, decay_in_time = True):
    '''
    Initializes all necessary components of the TensorFlow Graph

    Args:
      m: map width
      n: map height
      dim: dimension of the input vector
      n_iterations: number of iterations (epochs) - Default value is 100
      alpha: initial value of the learning rate - Default value is 0.1
      sigma: initial value of neighborhood radius - Default value is max(m, n)/2
      decay_in_time: decay the learning rate value through time
    '''
    # Assign input variables
    self._m = m
    self._n = n
    self._dim = dim
    self._n_iterations = int(n_iterations)

    self._alpha = float(alpha)
    if sigma is None:
      self._sigma = max(m,n)/2.0
    else:
      self._sigma = float(sigma)

    self._decay_in_time = True
    if decay_in_time is not True:
      self._decay_in_time = False
```

master* 0 0 Ln 52, Col 8 Spaces: 2 UTF-8 LF Python

https://github.com/hsneto/redes_neurais/blob/master/neural_networks/sompy.py

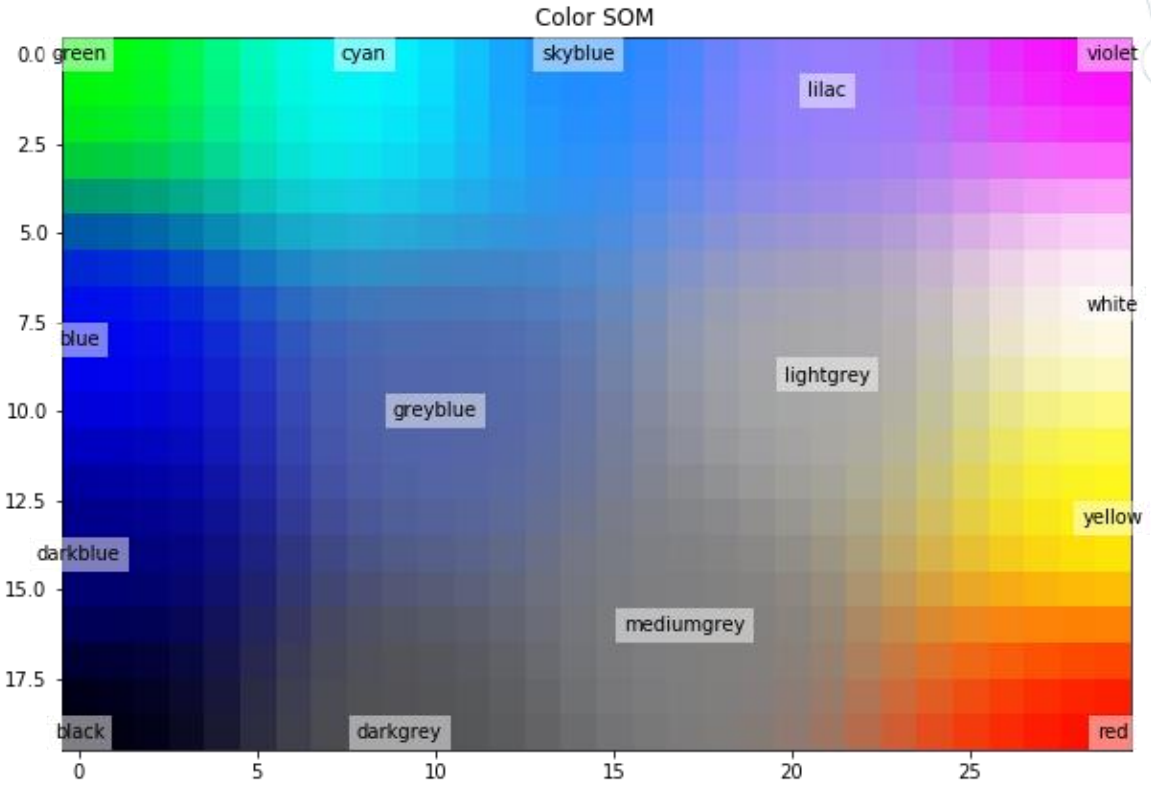
Testes

- m = 20
- n = 30
- dim = 3
- n_interations = 400
- alpha = 0.5
- sigma = None
- decay_in_time = True



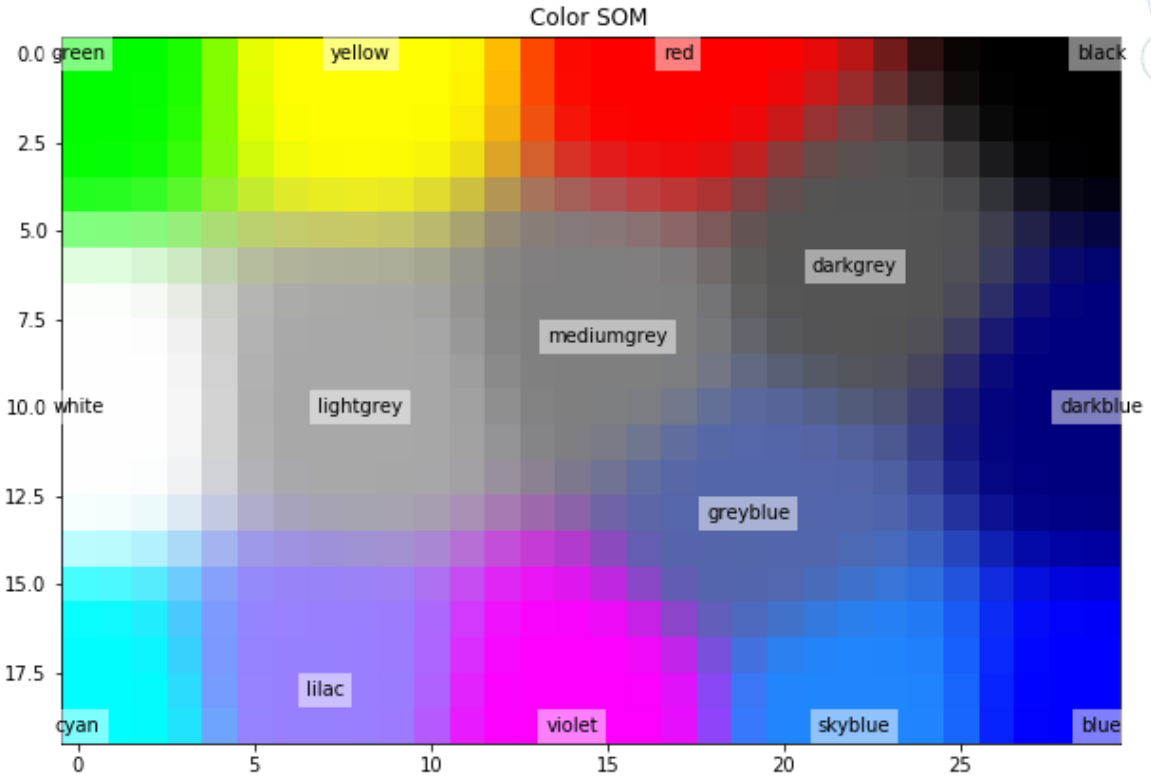
Testes

- m = 20
- n = 30
- dim = 3
- n_interations = 400
- **alpha = 0.1**
- sigma = None
- decay_in_time = True



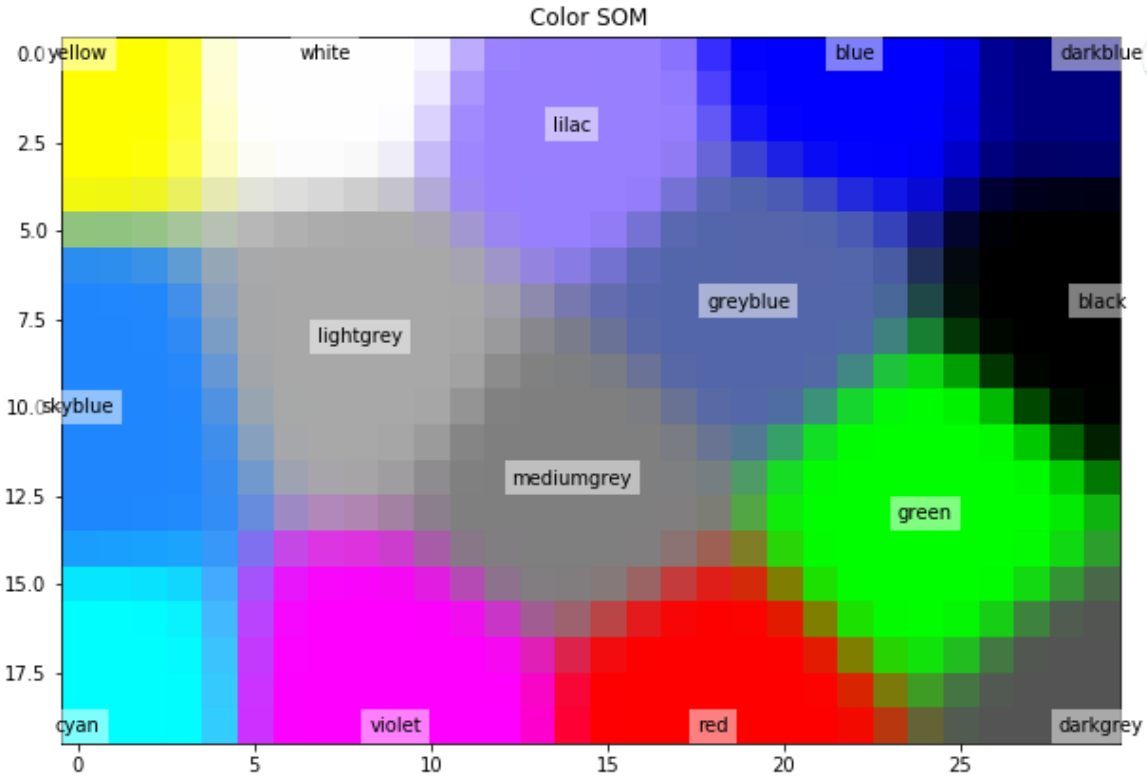
Testes

- m = 20
- n = 30
- dim = 3
- n_interations = 400
- **alpha = 1.0**
- sigma = None
- decay_in_time = True



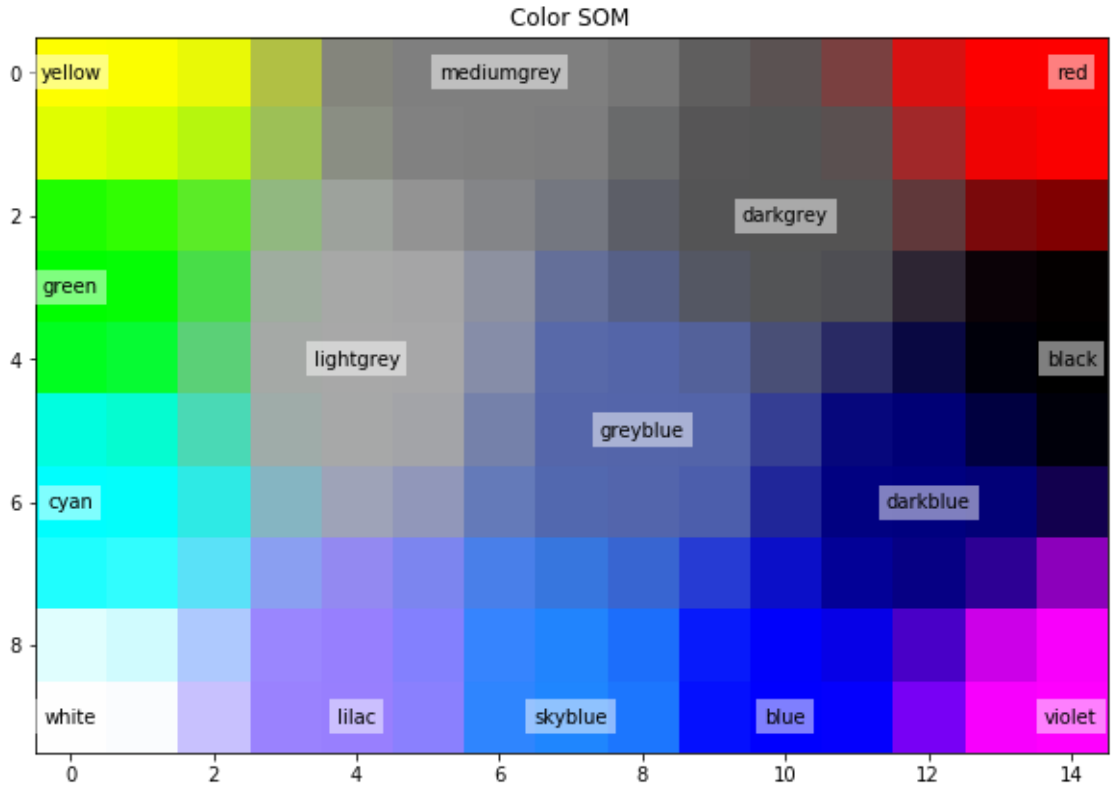
Testes

- m = 20
- n = 30
- dim = 3
- n_interations = 400
- alpha = 0.5
- **sigma = 5**
- decay_in_time = True



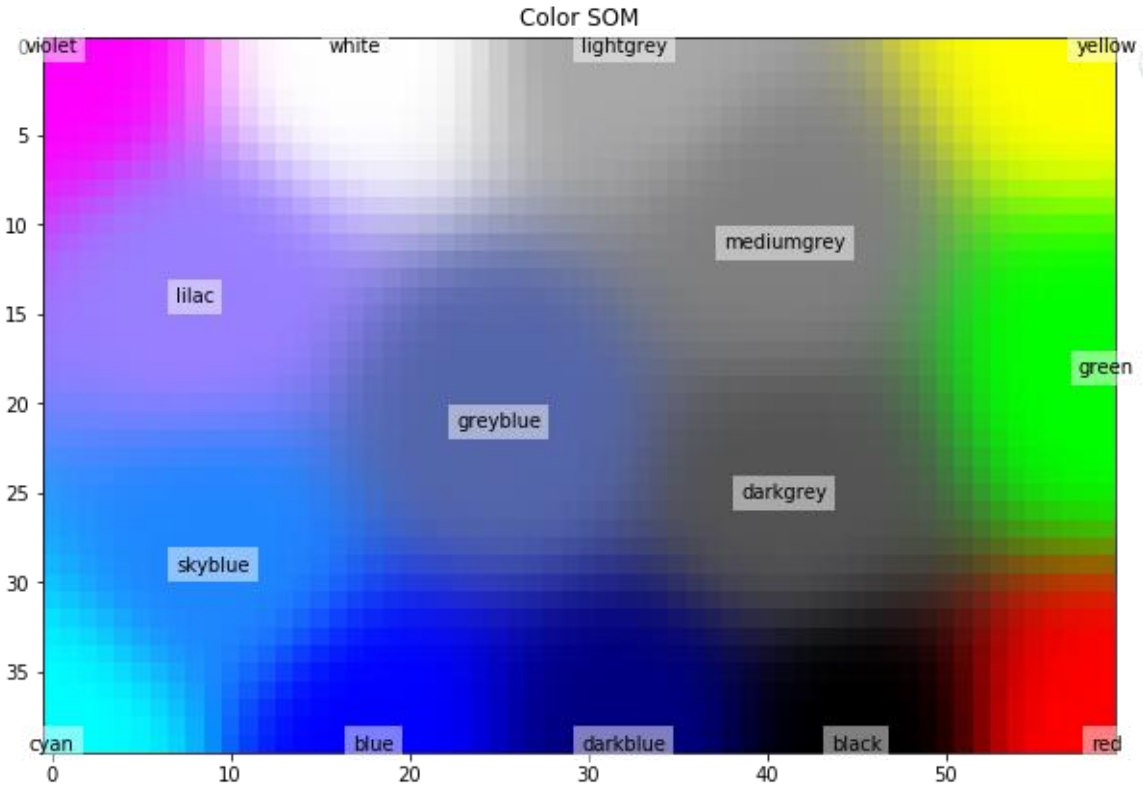
Testes

- **m = 10**
- **n = 15**
- dim = 3
- n_interations = 400
- alpha = 0.5
- sigma = None
- decay_in_time = True



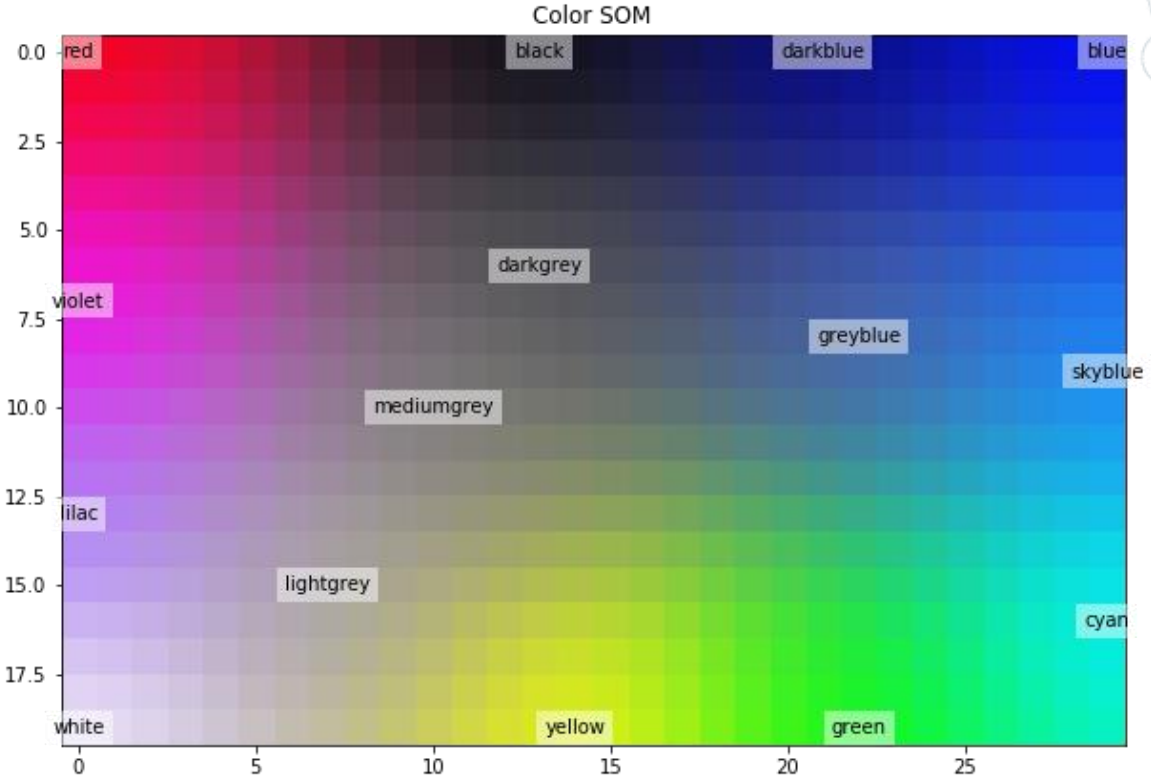
Testes

- **m = 40**
- **n = 60**
- dim = 3
- n_interations = 400
- alpha = 0.5
- sigma = None
- decay_in_time = True



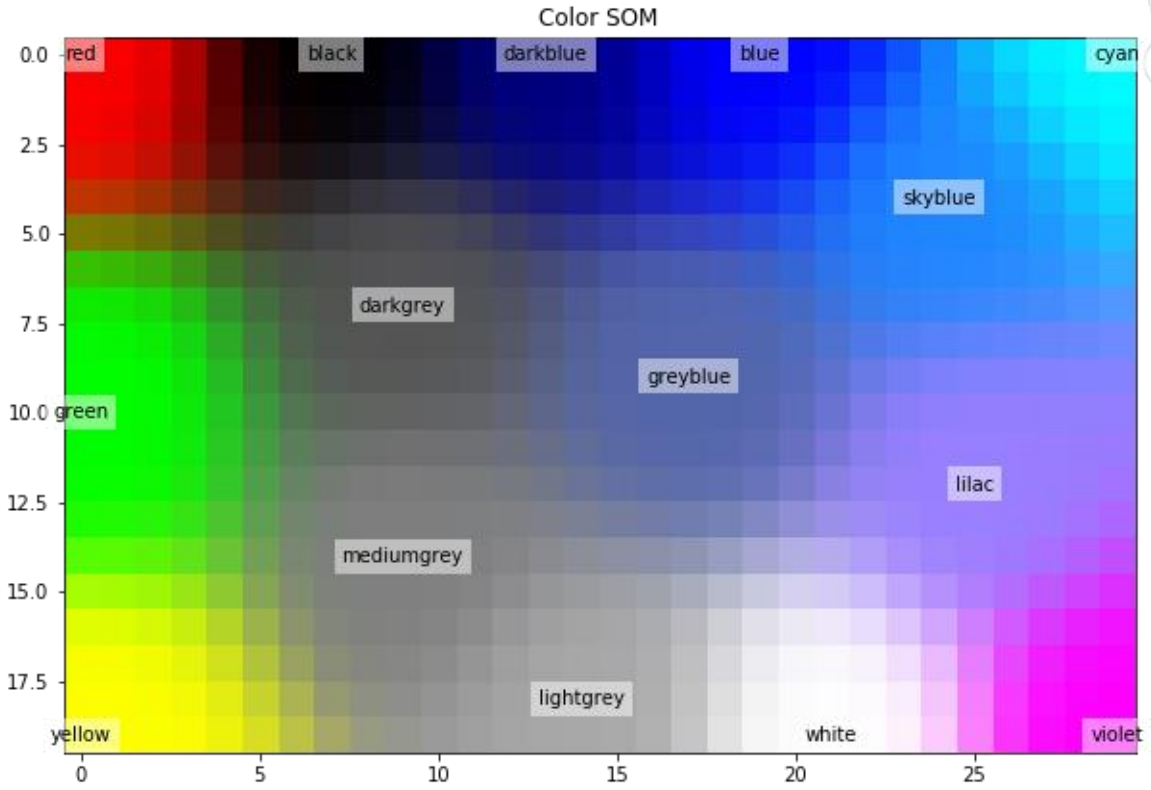
Testes

- m = 20
- n = 30
- dim = 3
- **n_interations = 100**
- **alpha = 0.1**
- sigma = None
- decay_in_time = True



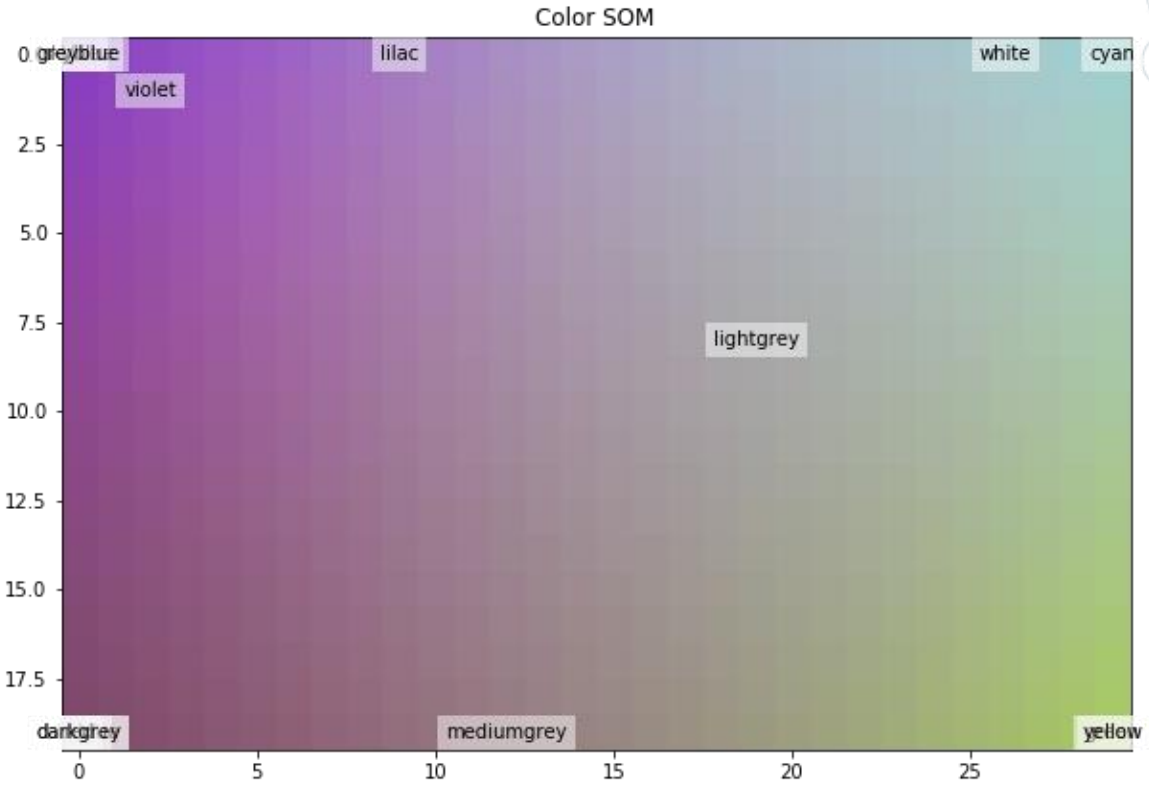
Testes

- m = 20
- n = 30
- dim = 3
- **n_interations = 1000**
- **alpha = 0.1**
- sigma = None
- decay_in_time = True



Testes

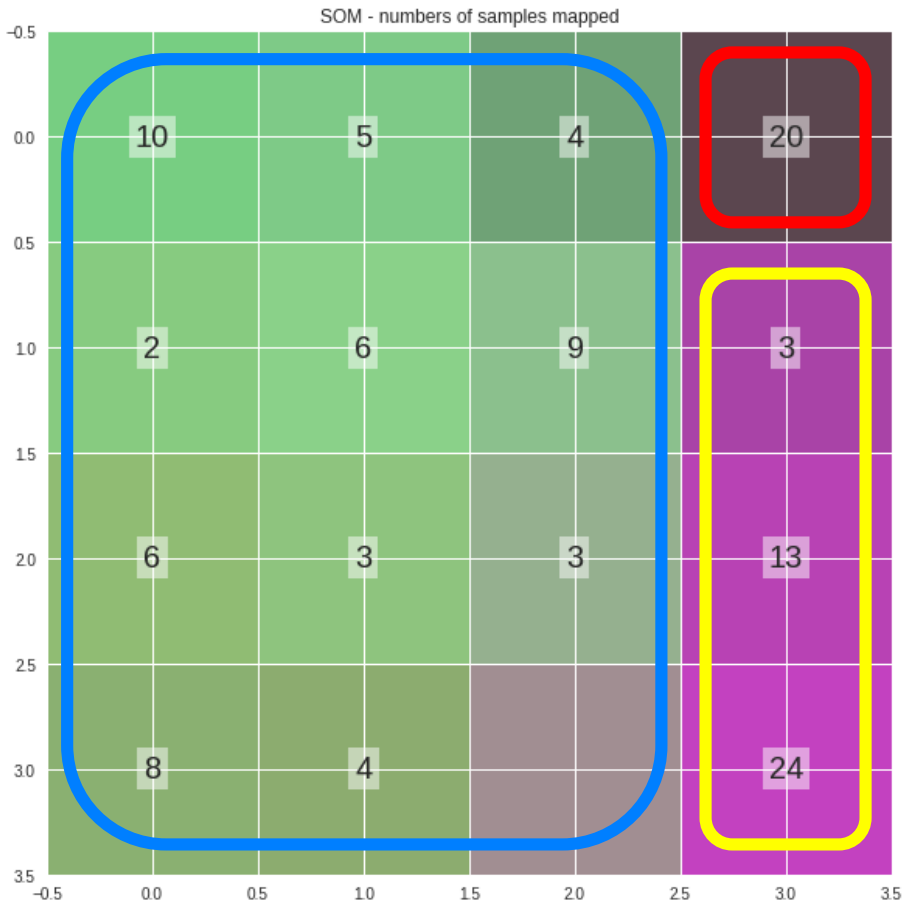
- m = 20
- n = 30
- dim = 3
- n_interations = 400
- alpha = 0.5
- sigma = None
- **decay_in_time = False**

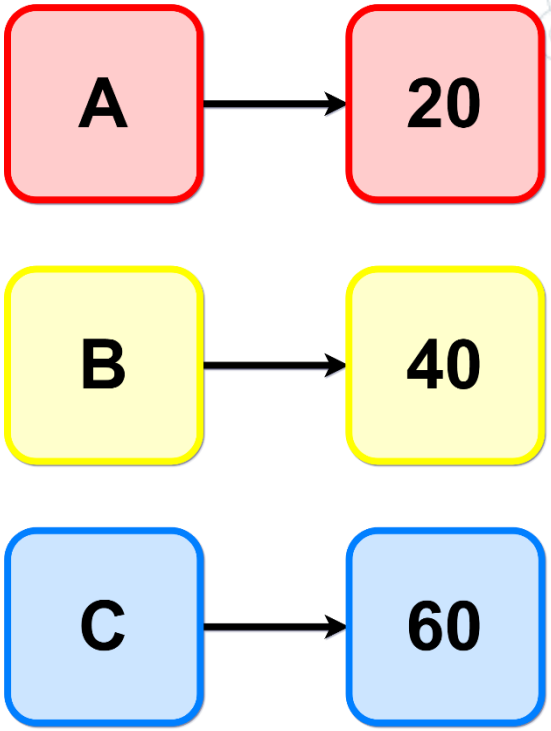


Problema proposto

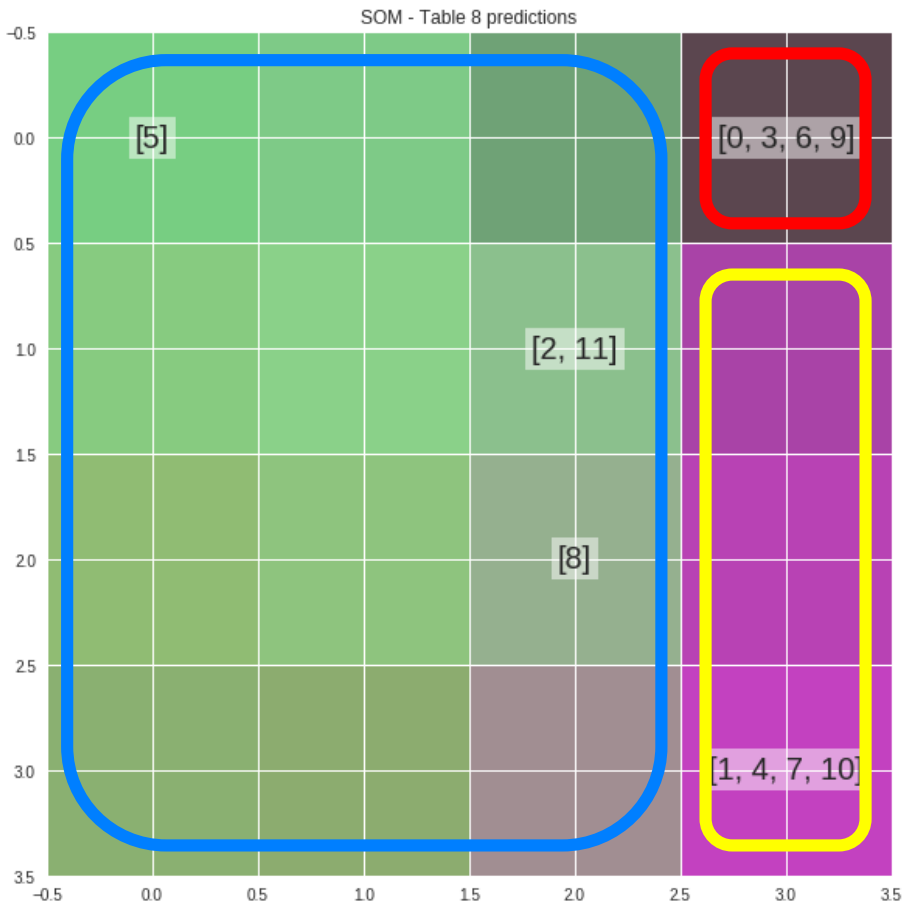
- No processo de identificação de pneus:
  - Identificar eventuais similaridades e correlações entre as variáveis que fazem parte do processo de fabricação de pneus
- Mapa de Kohonen:
  - Com 16 neurônios
  - $\eta = 0{,}001$
  - Vizinhança unitária
  - 3 classes distintas
    - A – 20 amostras
    - B – 40 amostras
    - C – 60 amostras

Resultado: treino



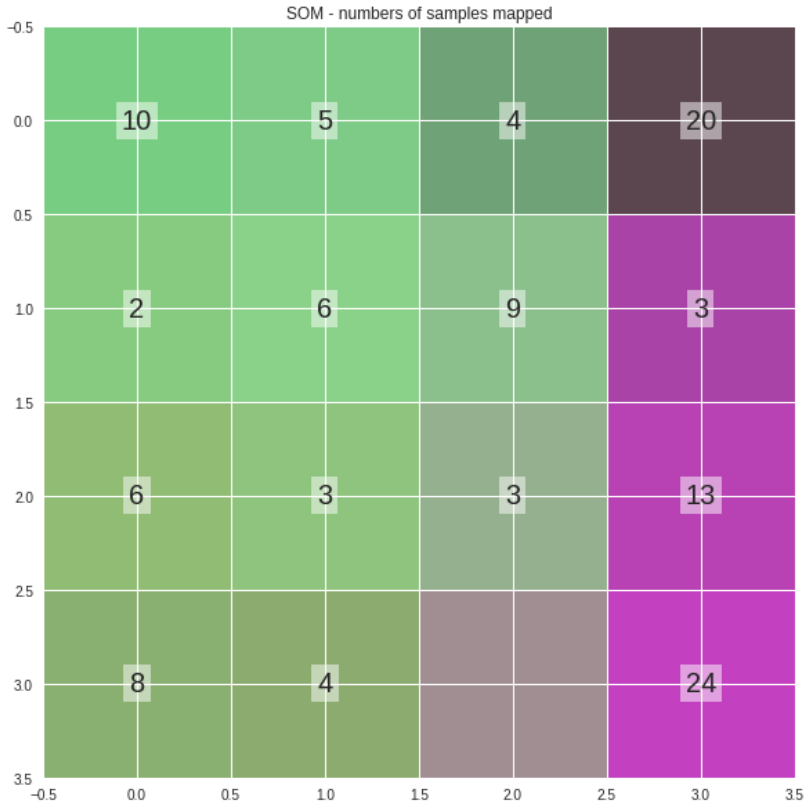


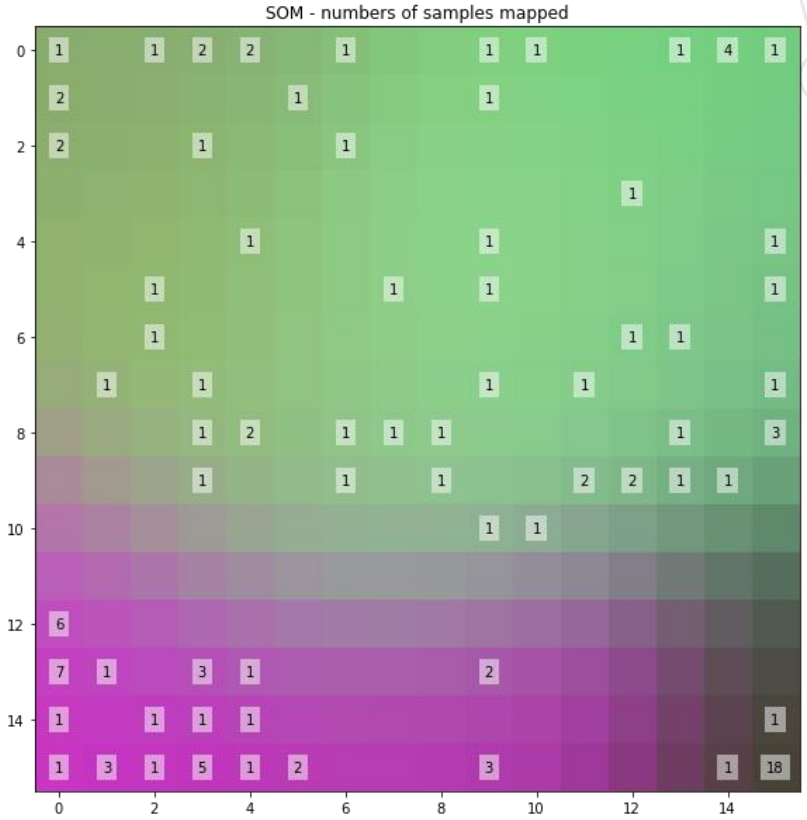
Resultado: teste



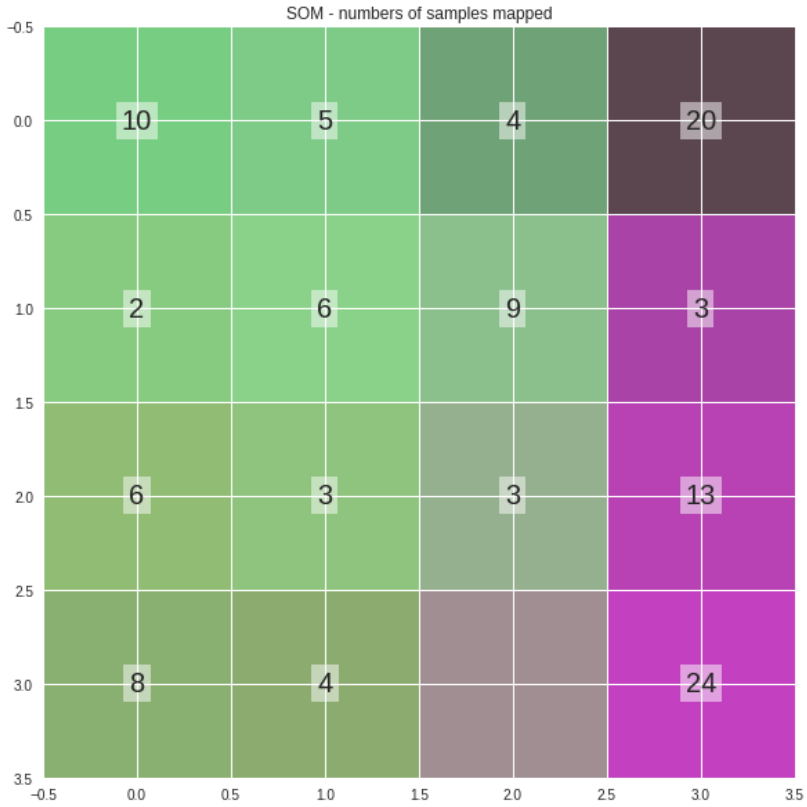
	x1	x2	x3
0	0.2471	0.1778	0.2905
1	0.8240	0.2223	0.7041
2	0.4960	0.7231	0.5866
3	0.2923	0.2041	0.2234
4	0.8118	0.2668	0.7484
5	0.4837	0.8200	0.4792
6	0.3248	0.2629	0.2375
7	0.7209	0.2116	0.7821
8	0.5259	0.6522	0.5957
9	0.2075	0.1669	0.1745
10	0.7830	0.3171	0.7888
11	0.5393	0.7510	0.5682

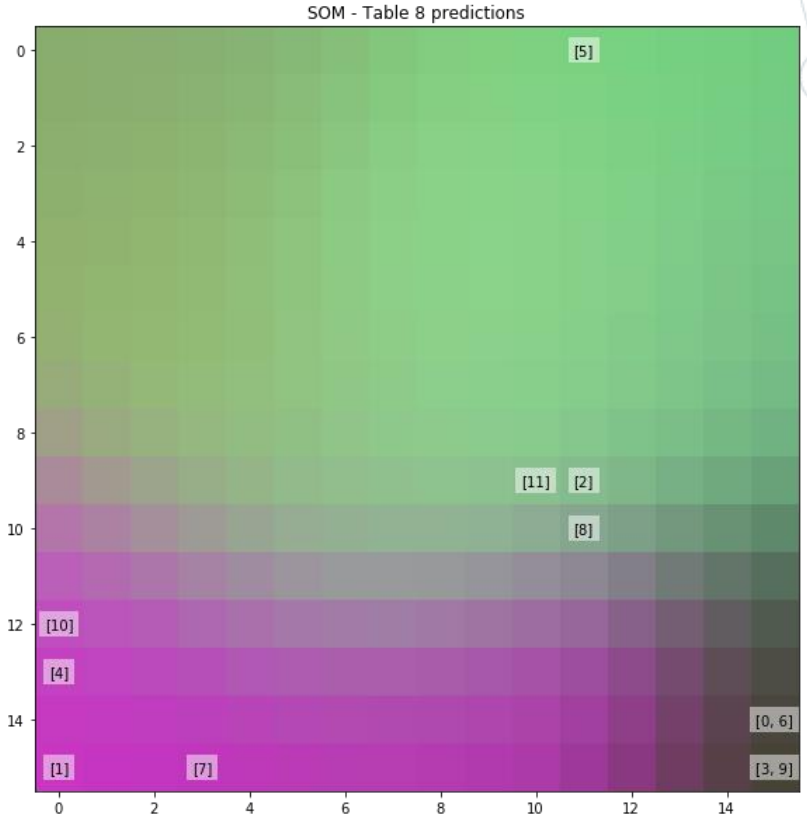
Resultado: comparações





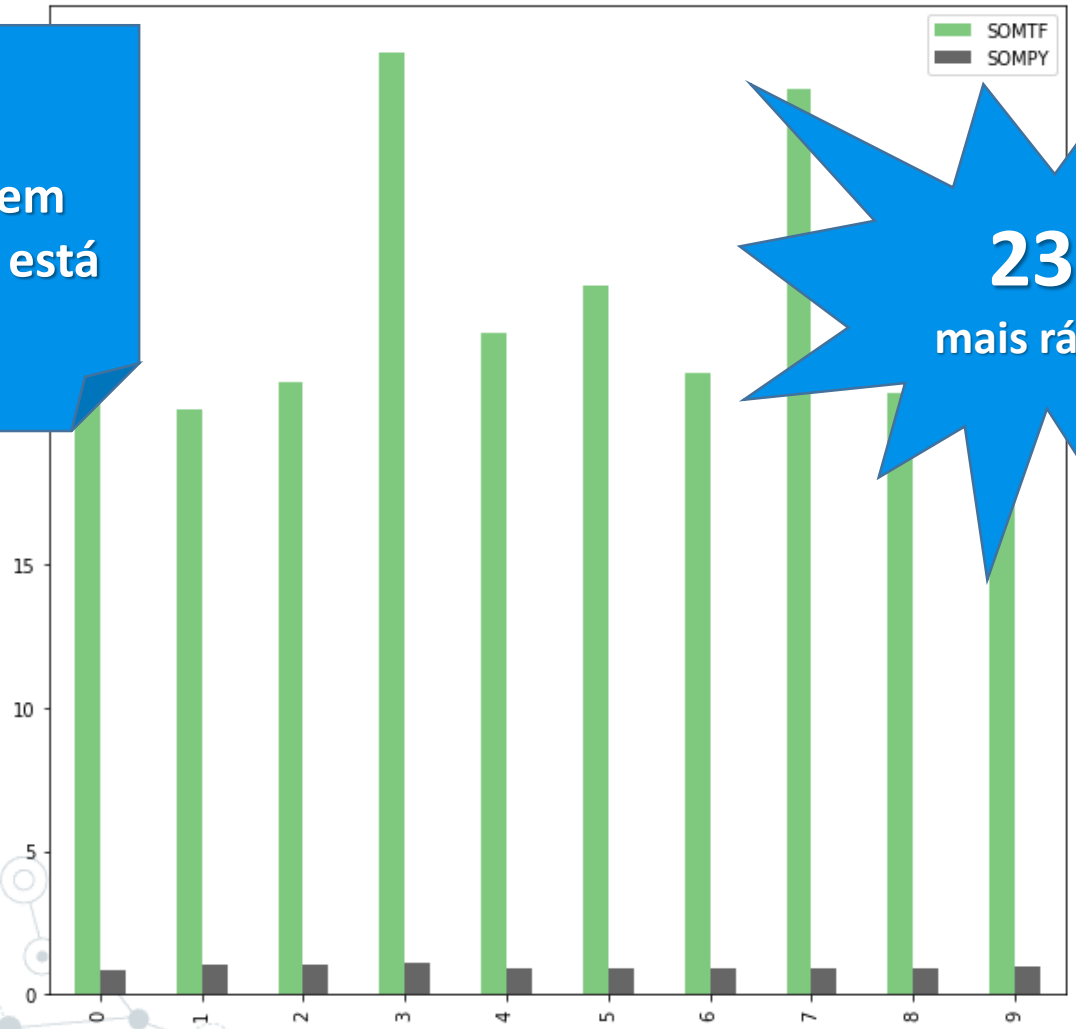
Resultado: comparações





Resultado: comparações

Conclusão:
Nosso código em Tensorflow não está otimizado!



23x
mais rápido

Referências

- Ivan Silva, Danilo Spatti and Rogério Fluzino "Redes Neurais Artificiais para engenharia e ciências aplicadas", Artliber, 2010
- http://blog.yhat.com/posts/self-organizing-maps-1.html
- http://www.ai-junkie.com/ann/som/som1.html
- https://vahidmoosavi.com/2014/02/18/a-self-organizing-map-som-package-in-python-sompy/
- https://codesachin.wordpress.com/2015/11/28/self-organizing-maps-with-googles-tensorflow/
- http://www.decom.ufop.br/imobilis/self-organizing-maps/
- https://bop.unibe.ch/linguistik-online/article/view/788/1355