


# **Semântica e similaridade de palavras: Parte IV**

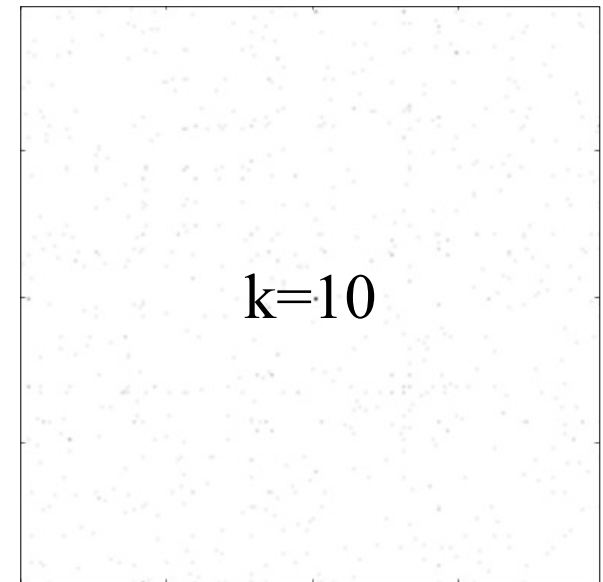
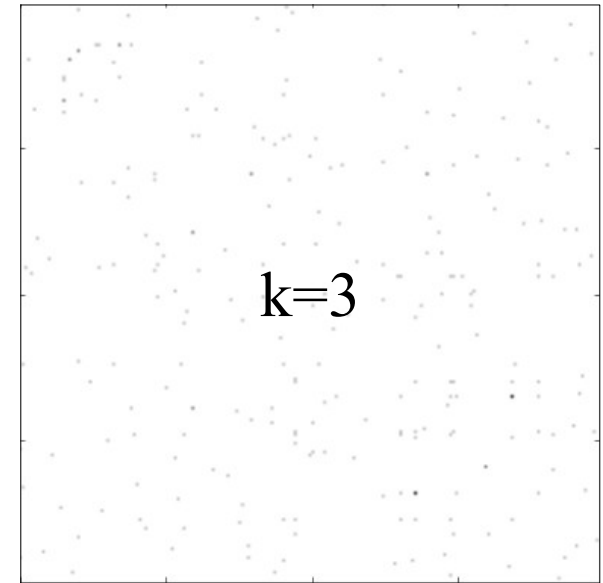
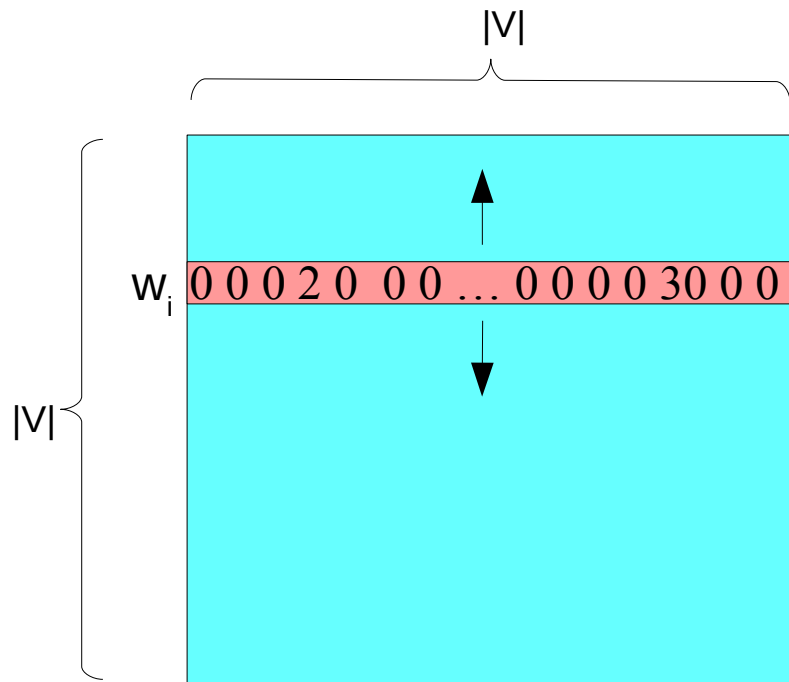
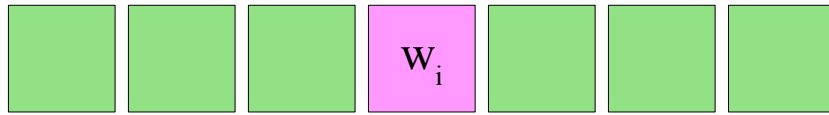
Prof. Jesús P. Mena-Chalco  
[jesus.mena@ufabc.edu.br](mailto:jesus.mena@ufabc.edu.br)

2Q-2019



# Matriz (esparsa): termo-contexto

# Matriz (esparsa): termo-contexto



Cada palavra é representado por um vetor cumprido mas com muitos valores nulos.

# Matriz (esparsa): termo-contexto

Tamanho da matriz (sem stopwords):

Supondo 4 bytes para armazenar um inteiro

Nome	V	Tamanho
Ufabcc-bcc	96	0.035MB
notícias	1 580	~9.5 MB
Machado-db (4 obras mais conhecidas)	22 784	~1.9 GB
Machado-db (todas as obras)	62 933	~15 GB
Português (incluindo as formas verbais)	~400 000	~596 GB



# **Semântica com vetores densos**

# Vetor esparsos -> Vetor denso

$w_i$  00020 00000001000001000...000030000000000000000000 ~60mil



$w_i$  1 0 3 4 2 1 4 2 8 5 6 2 9 13 ~1mil

Estrutura mais "fácil" de trabalhar nos algoritmos de aprendizado de máquina.

# PLN: Vetor esparsos -> Vetor denso

Existem vários métodos (abordagens) para gerar vetores de dimensão pequena mas densos:

- **SVD:**

Redução de dimensionalidade via Singular Value Decomposition.

- **Skip-gram** ou **CBOW:**

Usando redes neurais.

- **Brown clustering:**

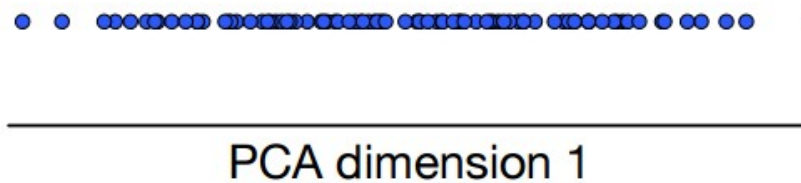
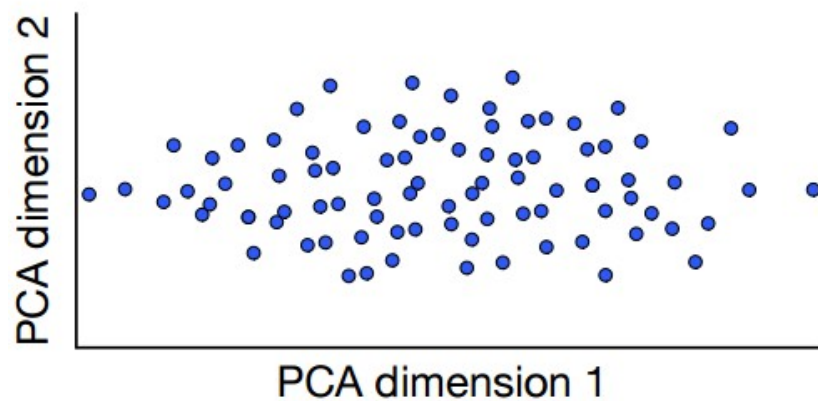
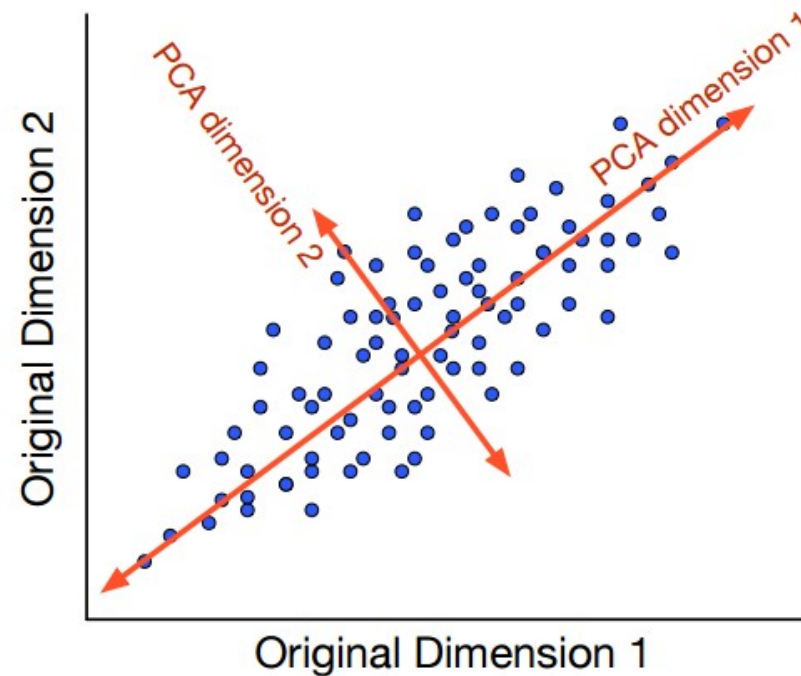
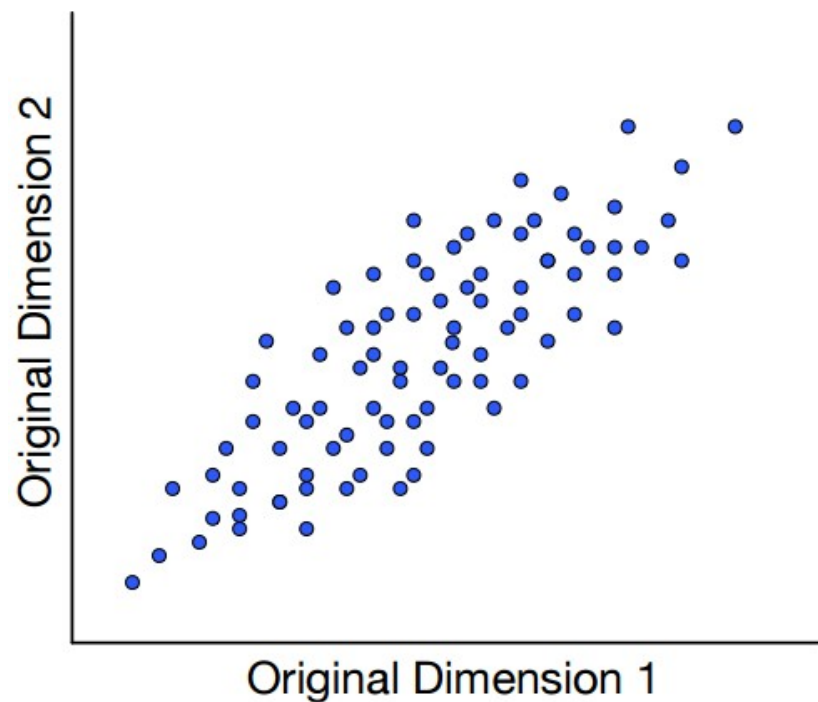
Agrupamento baseado na vizinhança de palavras.

# Diminuindo a dimensão?

- A ideia é utilizar um vetor de **menor dimensão**, mas que represente o vetor maior.
- Como identificar um conjunto menor de dimensões?
  - PCA – Análise de componentes principais
  - FA – Análise de fatores
  - SVD



# PCA



EDUCATION

# Ten quick tips for effective dimensionality reduction

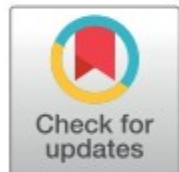
Lan Huong Nguyen<sup>1</sup>, Susan Holmes<sup>2\*</sup>

**1** Institute for Mathematical and Computational Engineering, Stanford University, Stanford, California, United States of America, **2** Department of Statistics, Stanford University, Stanford, California, United States of America

\* [susan@stat.stanford.edu](mailto:susan@stat.stanford.edu)

## Introduction

Dimensionality reduction (DR) is frequently applied during the analysis of high-dimensional data. Both a means of denoising and simplification, it can be beneficial for the majority of modern biological datasets, in which it's not uncommon to have hundreds or even millions of simultaneous measurements collected for a single sample. Because of "the curse of dimensionality," many statistical methods lack power when applied to high-dimensional data. Even if the number of collected data points is large, they remain sparsely submerged in a voluminous high-dimensional space that is practically impossible to explore exhaustively (see chapter 12 [1]). By reducing the dimensionality of the data, you can often alleviate this challenging and troublesome phenomenon. Low-dimensional data representations that remove noise but retain the signal of interest can be instrumental in understanding hidden structures and patterns. Original high-dimensional data often contain measurements on uninformative or redundant variables. DR can be viewed as a method for latent feature extraction. It is also frequently used for data compression, exploration, and visualization. Although many DR techniques have been developed and implemented in standard data analytic pipelines, they are easy to misuse, and their results are often misinterpreted in practice. This article presents a set of useful guidelines for practitioners specifying how to correctly perform DR, interpret its output, and communicate results. Note that this is not a review article, and we recommend some important reviews in the references.



## OPEN ACCESS

**Citation:** Nguyen LH, Holmes S (2019) Ten quick tips for effective dimensionality reduction. PLoS Comput Biol 15(6): e1006907. <https://doi.org/10.1371/journal.pcbi.1006907>

**Editor:** Francis Ouellette, University of Toronto, CANADA

**Published:** June 20, 2019


**Copyright:** © 2019 Nguyen, Holmes. This is an

**Table 1. Dimensionality reduction methods.**

Method	Input Data	Method Class	Nonlinear	Complexity
PCA	continuous data	unsupervised		$\mathcal{O}(\max(n^2p, np^2))$
CA	categorical data	unsupervised		$\mathcal{O}(\max(n^2p, np^2))$
MCA	categorical data	unsupervised		$\mathcal{O}(\max(n^2p, np^2))$
PCoA (cMDS)	distance matrix	unsupervised		$\mathcal{O}(n^2p)$
NMDS	distance matrix	unsupervised		$\mathcal{O}(n^2h)$
Isomap	continuous*	unsupervised	✓	$\mathcal{O}(n^2(p + \log n))$
Diffusion Map	continuous*	unsupervised	✓	$\mathcal{O}(n^2p)$
Kernel PCA	continuous*	unsupervised	✓	$\mathcal{O}(n^2p)$
t-SNE	continuous/distance	unsupervised	✓	$\mathcal{O}(n^2p + n^2h)$
Barnes–Hut t-SNE	continuous/distance	unsupervised	✓	$\mathcal{O}(nh \log n)$
LDA	continuous (X and Y)	supervised		$\mathcal{O}(np^2 + p^3)$
PLS (NIPALS)	continuous (X and Y)	supervised		$\mathcal{O}(npd)$
NCA	distance matrix	supervised	✓	$\mathcal{O}(n^2h)$
Bottleneck NN	continuous/categorical	supervised	✓	$\mathcal{O}(nph)$
STATIS	continuous	multidomain		$\mathcal{O}(n^2P, nP^2)$
DiSTATIS	distance matrix	multidomain		$\mathcal{O}(n^2P, nP^2)$

**Table 2. Example implementations.**

<b>Method</b>	<b>R function</b>	<b>Python function</b>
PCA	<code>stats::prcomp</code>	<code>sklearn.decomposition.PCA</code>
CATPCA	<code>gifi::princals</code>	
CA	<code>FactoMineR::CA</code>	
MCA	<code>FactoMineR::MCA</code>	
PCoA (cMDS)	<code>stats::cmdscale</code>	<code>sklearn.manifold.MDS</code>
NMDS	<code>ecodist::nmms</code>	<code>sklearn.manifold.MDS</code>
Isomap	<code>vegan::isomap</code>	<code>sklearn.manifold.Isomap</code>
Diffusion Map	<code>diffusionMap::diffuse</code>	
(Barnes-Hut) t-SNE	<code>Rtsne::Rtsne</code>	<code>sklearn.manifold.TSNE</code>
LDA	<code>MASS::lda</code>	<code>sklearn.discriminant_analysis.LinearDiscriminantAnalysis</code>
PLS (NIPALS)	<code>mixOmics::pls</code>	<code>sklearn.cross_decomposition.PLSRegression</code>
DiSTATIS	<code>DistatisR::distatis</code>	
Procrustes	<code>vegan::procrustes</code>	<code>scipy.spatial.procrustes</code>



# **Semântica com vetores densos - via SVD**

# SVD

- Singular Value Decomposition (SVD) - Decomposição em valores singulares
- É uma fatora  o de uma matriz (m×n):  $M = U\Sigma V^*$ 
  - **U**:    uma matriz unit  ria m×m
  - **Σ**:    uma matriz retangular diagonal m×n com n  meros reais n  o-negativos na diagonal
    - $\Sigma_{i,i}$  s  o os valores singulares de M.
  - **V\*** (a conjugada transposta de U)    uma matriz unit  ria n×n

# SVD

- Muito utilizada para analisar sistemas multivariadas.

$$M = U\Sigma V^*$$

$$g = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$g = \begin{bmatrix} 0.319 & 0.447 & 0.835 \\ 0.934 & 0 & -0.357 \\ 0.160 & -0.894 & 0.418 \end{bmatrix} \begin{bmatrix} 6.854 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.146 \end{bmatrix}^{\frac{1}{2}} \begin{bmatrix} 0.835 & 0.447 & 0.319 \\ 0.357 & 0 & -0.934 \\ 0.418 & -0.894 & 0.160 \end{bmatrix}^T$$



```
import numpy

g = [[1,0,0],[2,1,1],[0,0,1]]

numpy.linalg.svd(g)

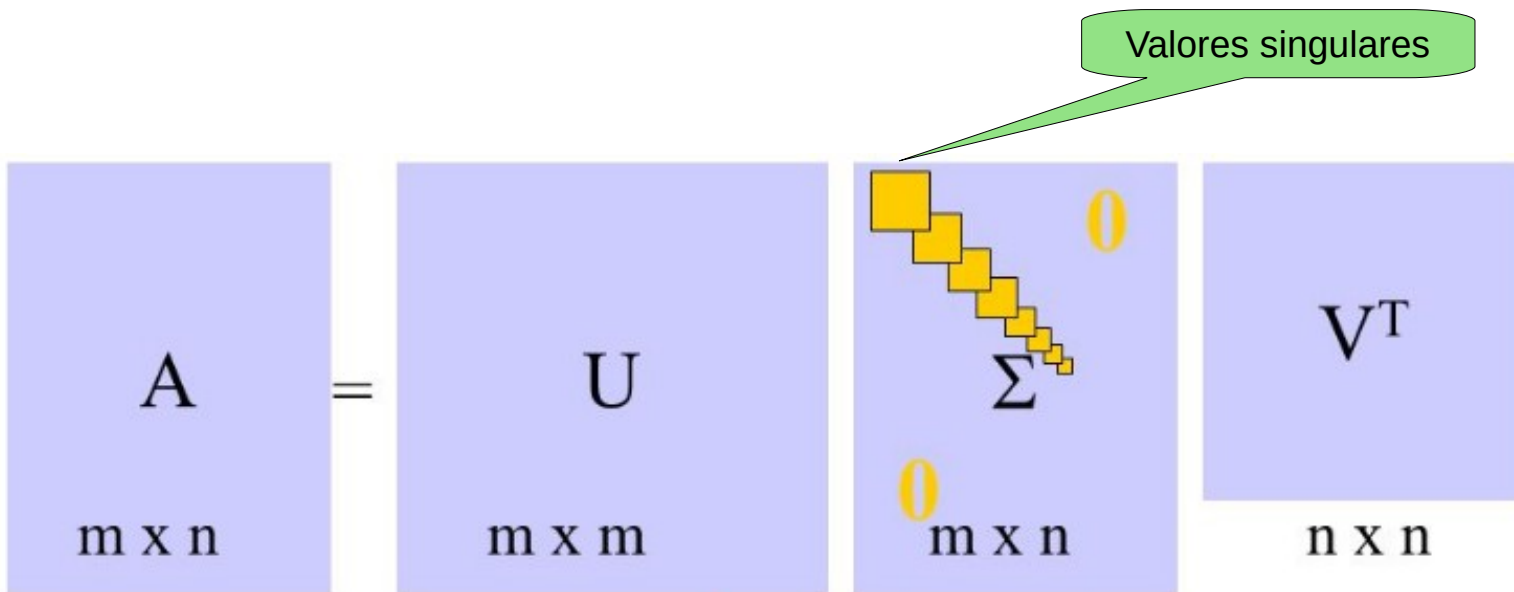
array([[ -3.19151379e-01,  4.47213595e-01,  8.35549159e-01],
       [ -9.34172359e-01, -1.66533454e-16, -3.56822090e-01],
       [ -1.59575690e-01, -8.94427191e-01,  4.17774579e-01]])

array([2.61803399, 1.          , 0.38196601])

array([[ -8.35549159e-01, -3.56822090e-01, -4.17774579e-01],
       [  4.47213595e-01,  1.11022302e-16, -8.94427191e-01],
       [  3.19151379e-01, -9.34172359e-01,  1.59575690e-01]])
```

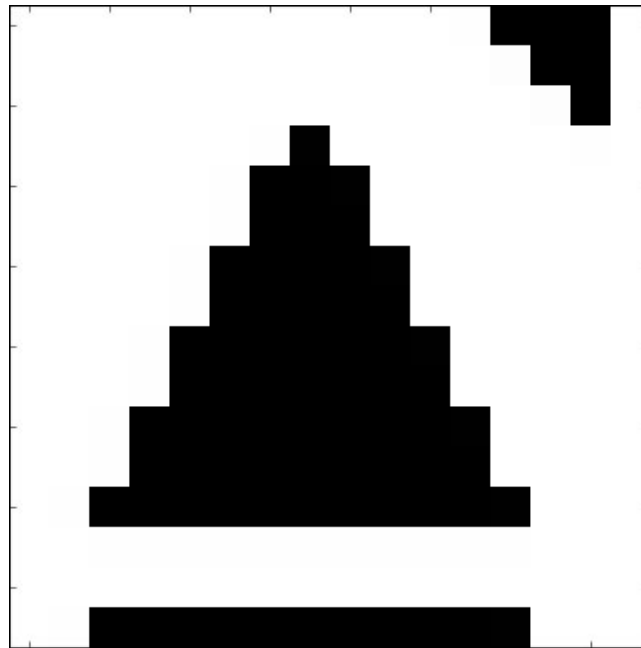


# SVD



# Exemplo

IM

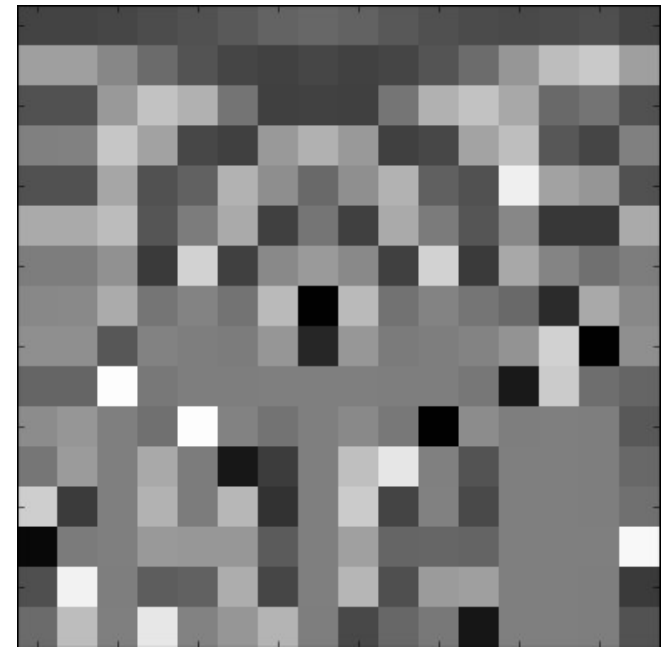
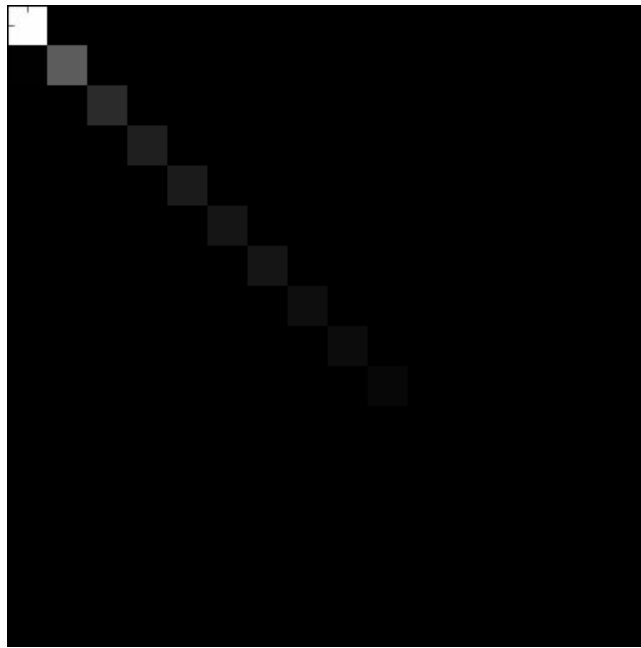
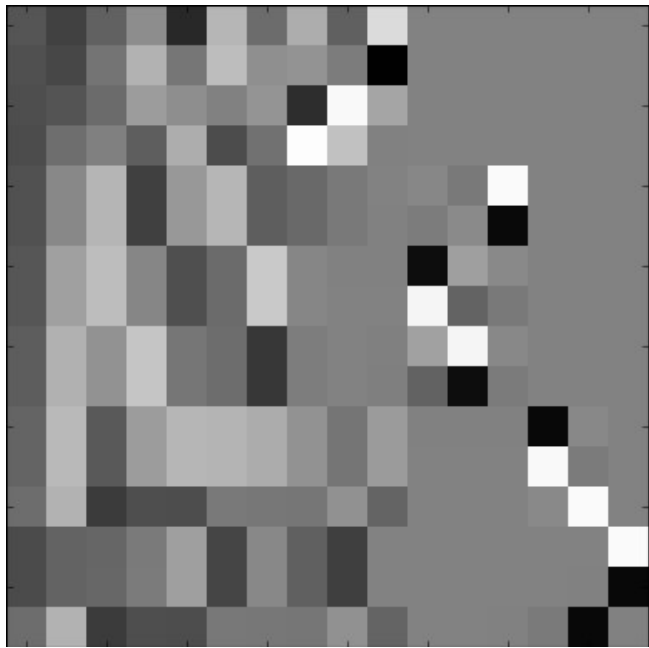
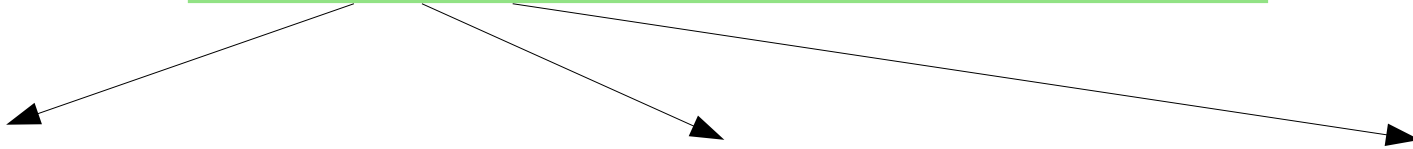


0

1

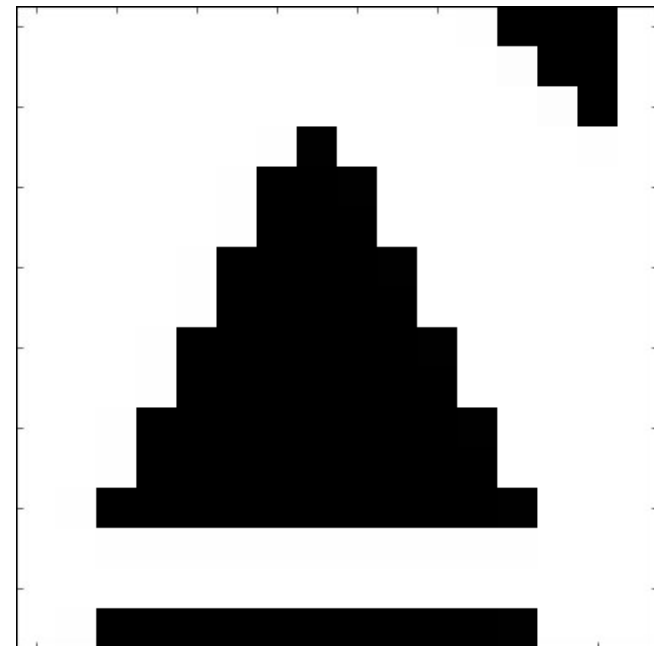
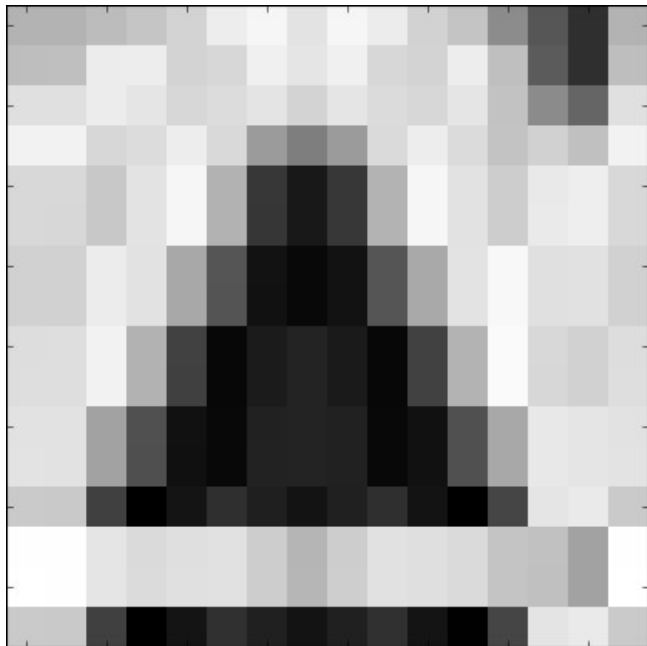
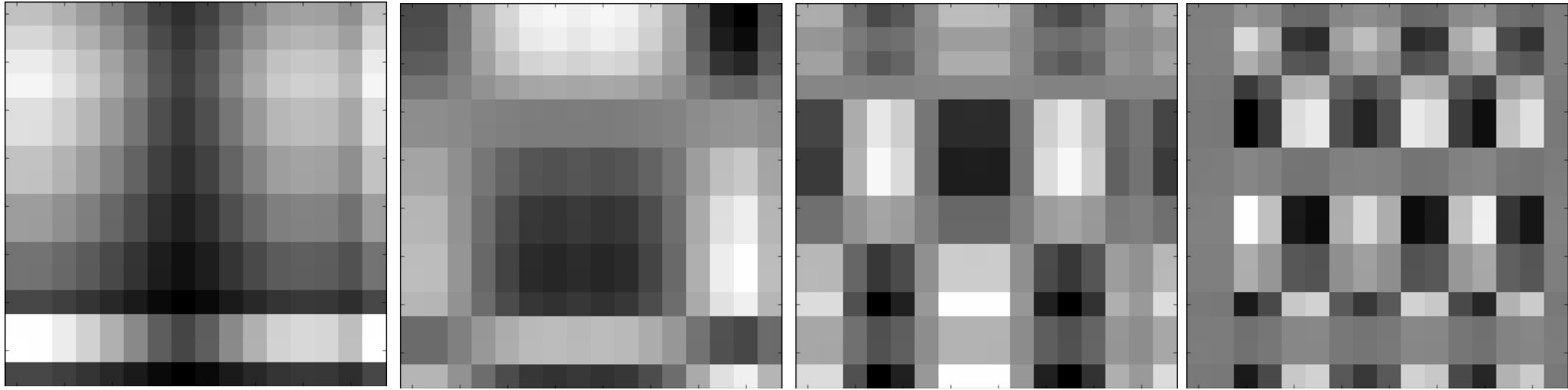
# Exemplo

```
(W, S, C) = numpy.linalg.svd(IM)
```



# Exemplo

Considerando os 4 primeiros valores singulares





```
import matplotlib.image as img
import matplotlib.pyplot as plt
import numpy

IM = img.imread("imagem.tiff")
IM = numpy.matrix(IM)/255
plt.imshow(IM, cmap=plt.cm.gray, interpolation='none')
plt.show()
```

```
# decomposicao SVD
(W,S,C)= numpy.linalg.svd(IM)
```

```
plt.imshow(W, cmap=plt.cm.gray, interpolation='none')
plt.show()
```

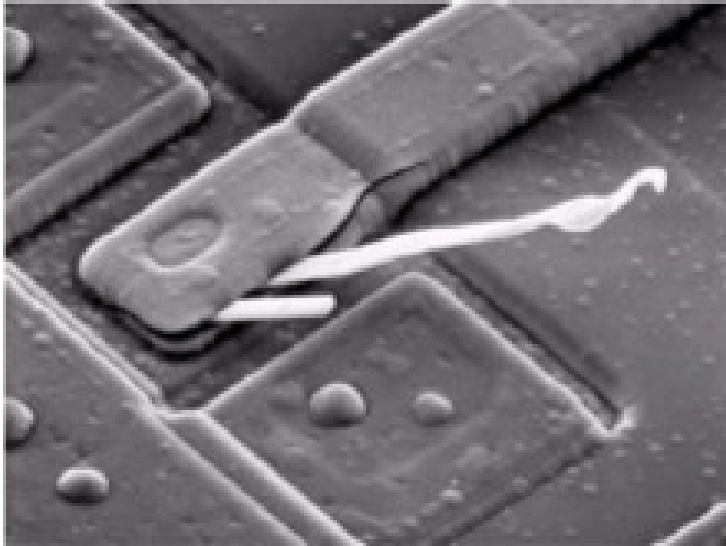
```
plt.imshow(numpy.diag(S), cmap=plt.cm.gray, interpolation='none')
plt.show()
```

```
plt.imshow(C, cmap=plt.cm.gray, interpolation='none')
plt.show()
```

```
IM2 = W*numpy.diag(S)*C
plt.imshow(IM2, cmap=plt.cm.gray, interpolation='none')
plt.show()
```

# Exemplo

Imagem Original



Valores Singulares

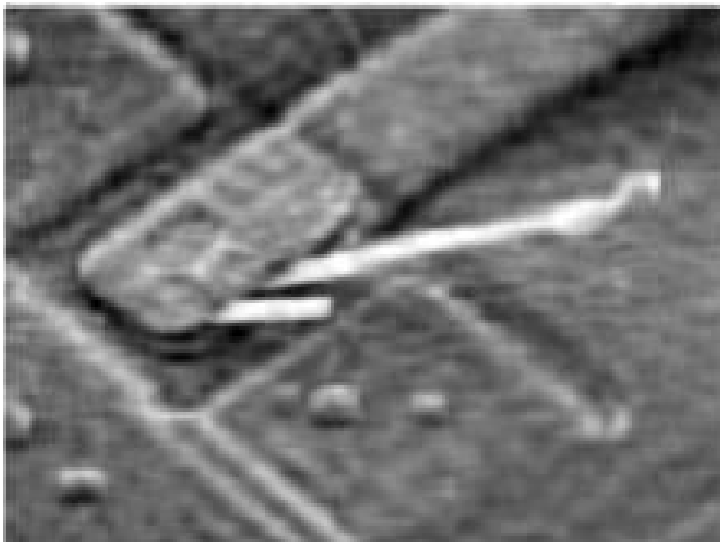
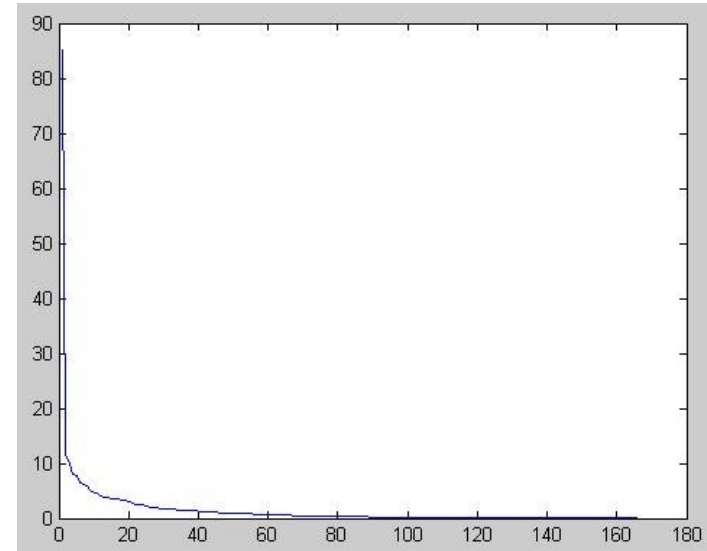


Imagem reconstruída com  
apenas 10% dos Valores singulares

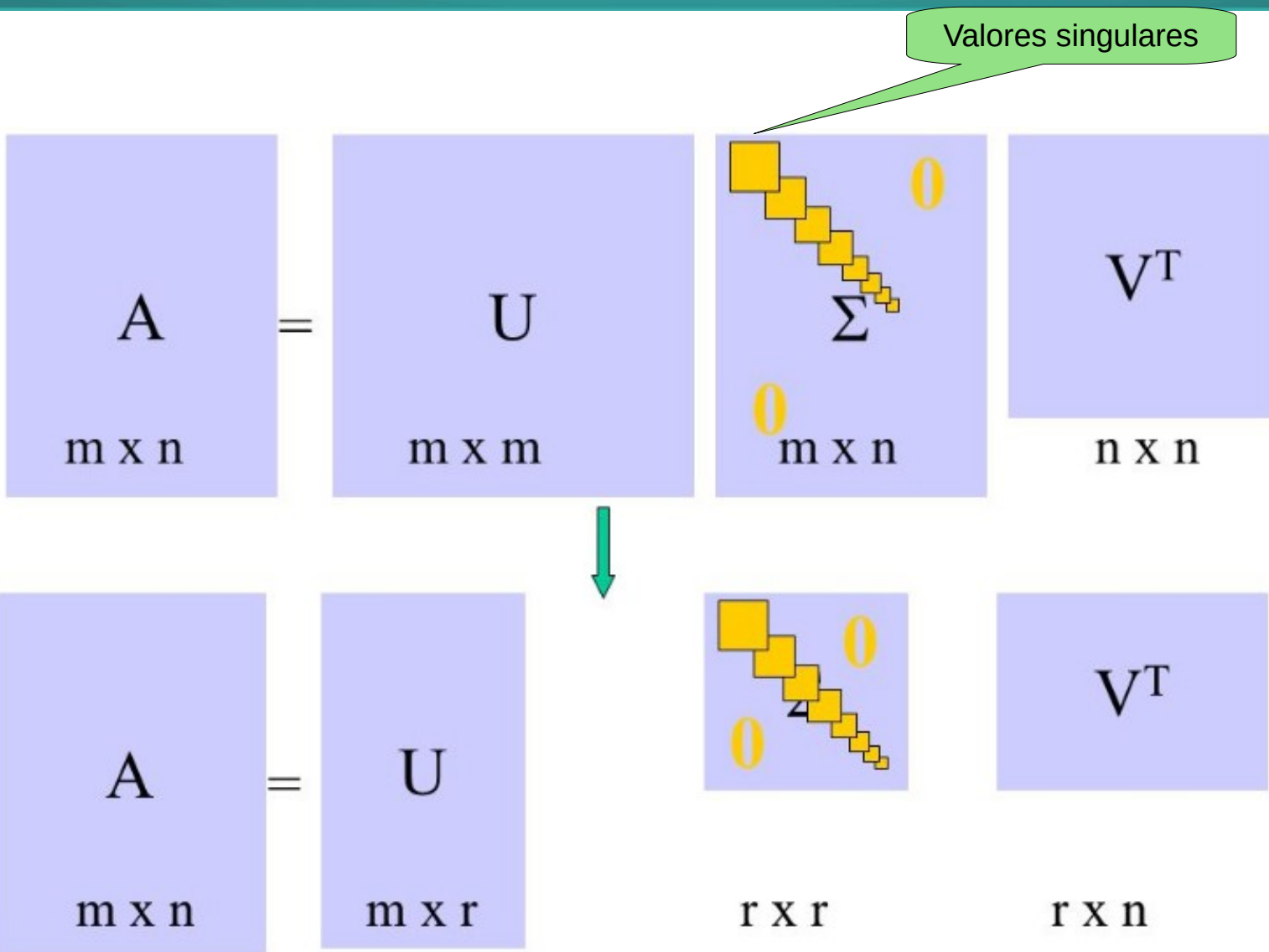
# SVD na matriz termo-contexto

$m$  é o rank de  $X$  (ie, número de linhas linearmente independentes)

$$\begin{bmatrix} X \\ |V| \times c \end{bmatrix} = \begin{bmatrix} W \\ |V| \times m \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_m \end{bmatrix} \begin{bmatrix} C \\ m \times c \end{bmatrix}$$

As colunas são ortogonais

# SVD





# SVD na matriz termo-contexto

$$\begin{bmatrix} X \\ |V| \times c \end{bmatrix} = \begin{bmatrix} W_k \\ |V| \times k \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_k \end{bmatrix} \begin{bmatrix} C \\ k \times c \end{bmatrix}$$

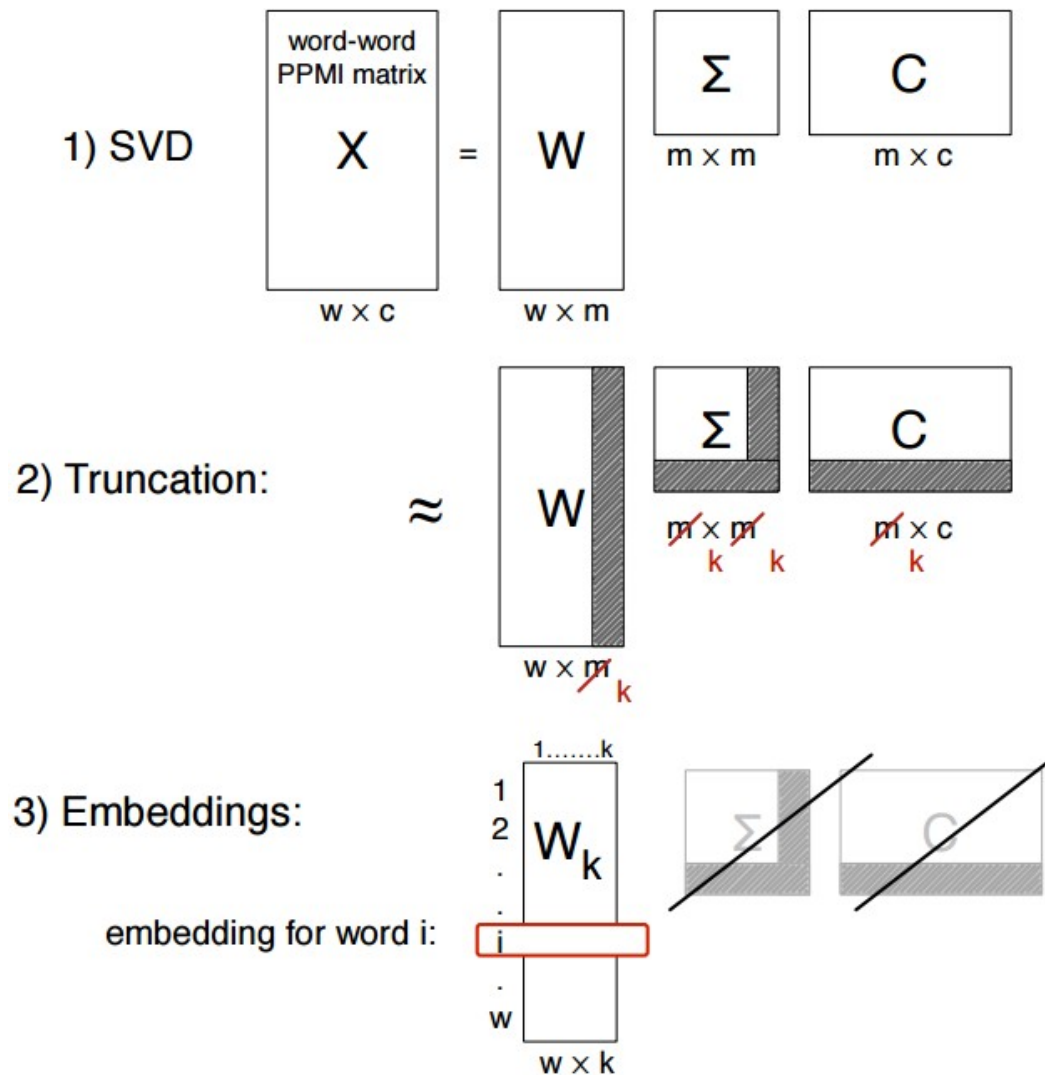
A ideia é considerar  $k < m$  (os  $k$  primeiros valores singulares)

# SVD na matriz PPMI

- No lugar de considerar os  $m$  valores singulares podemos usar os  $k$  ( $< m$ ) primeiros valores singulares.
- O resultado **seria bem próximo à matriz original** (ver exemplo de aproximação usando imagens).

# SVD na matriz PPMI

A mesma ideia pode ser considerada na matriz PPMI



Comumente  $k \approx 300$

# SVD na matriz PPMI

- Usar apenas  $W$ ?
- Usar  $W^*S$ ?

Nas abordagens iniciais foi considerado  $W^*S$ , mas foi observado que tem um desempenho pior, para o caso de PLN, quando comparado para o uso de apenas  $W$ .

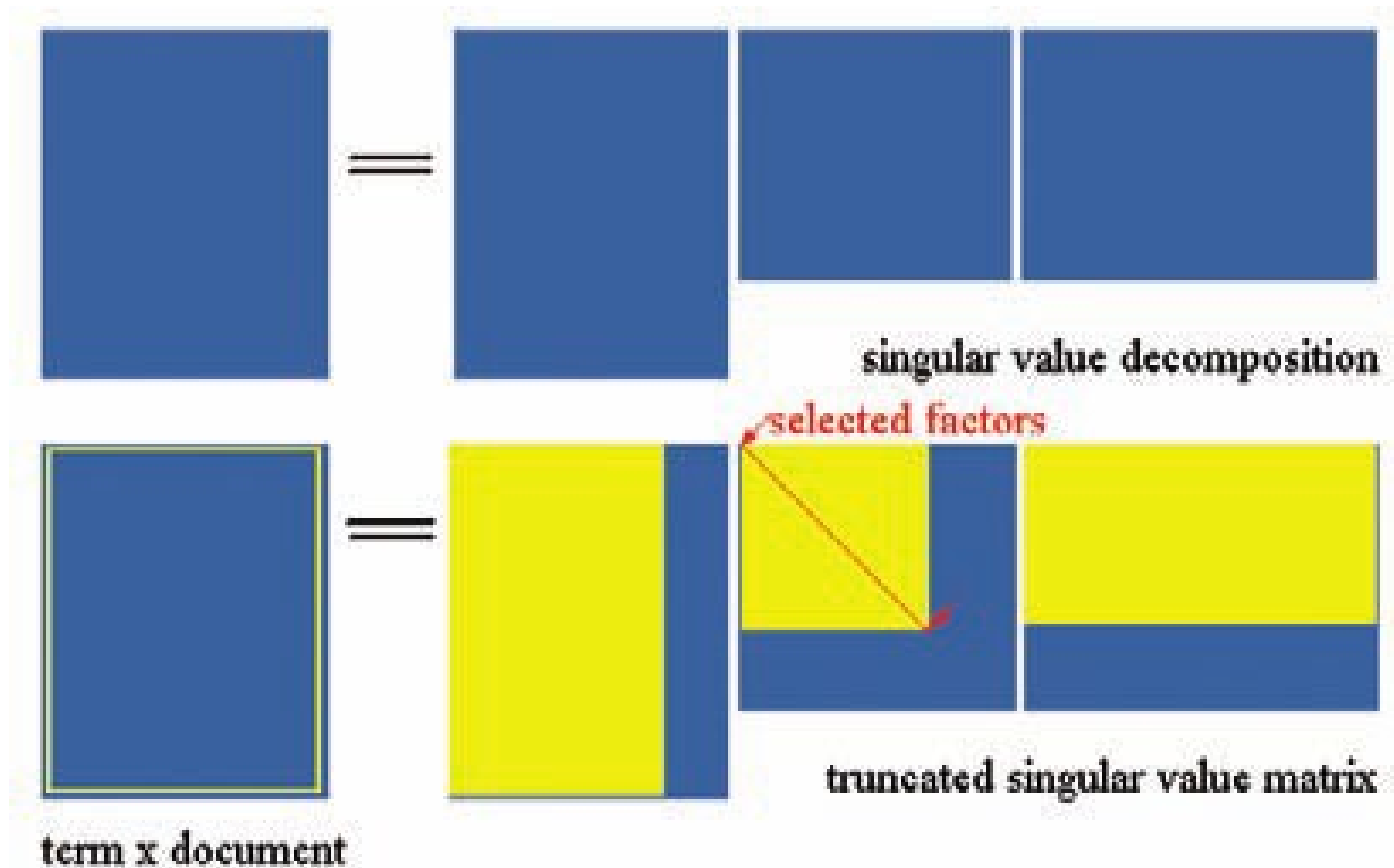
# E nas matrizes termo-termo?

$$\begin{bmatrix} X \\ |V| \times |V| \end{bmatrix} = \begin{bmatrix} W \\ |V| \times |V| \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_V \end{bmatrix} \begin{bmatrix} C \\ |V| \times |V| \end{bmatrix}$$

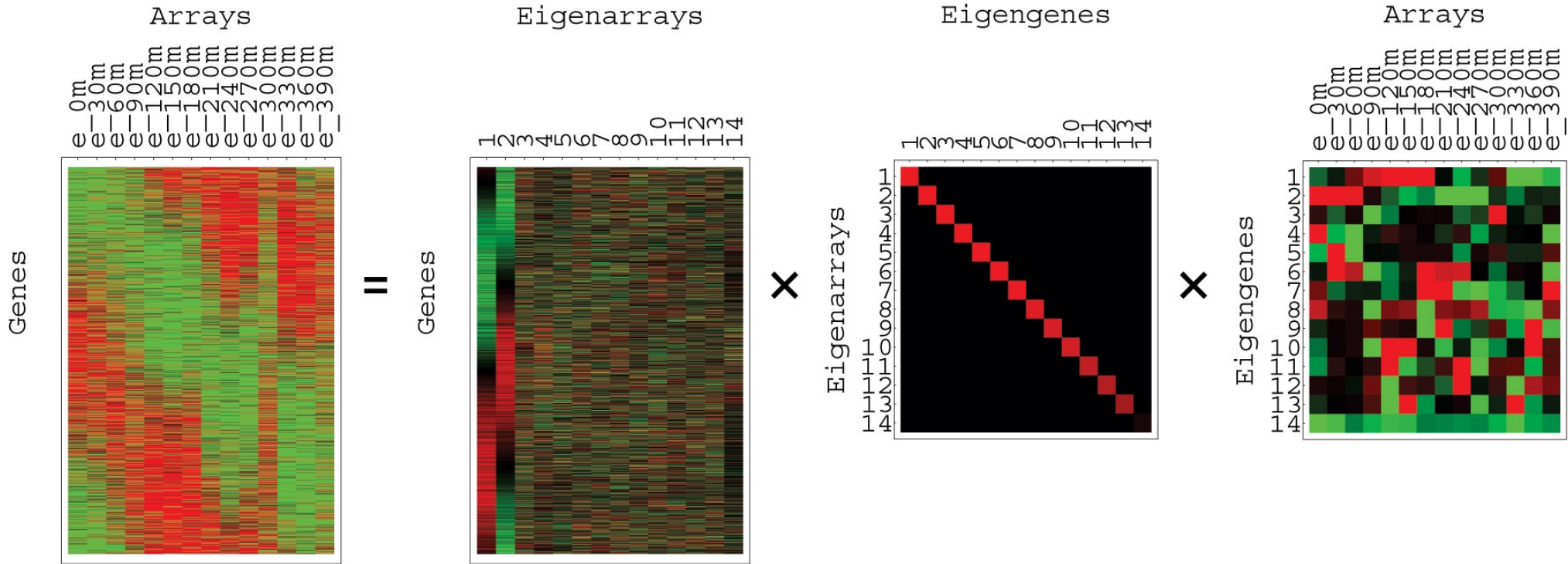
# E nas matrizes termo-termo?

$$\begin{bmatrix} X \\ |V| \times |V| \end{bmatrix} = \begin{bmatrix} W \\ |V| \times k \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_k \end{bmatrix} \begin{bmatrix} C \\ k \times |V| \end{bmatrix}$$

# E nas matrizes termo-documento?



# SVD - exemplo em outros contextos





# Sobre SVD...

- SVD geralmente é preferível no lugar das matrizes PPMI ou matriz termo-termo, ou matriz termo-documento.
- O “ruído” de similaridade entre palavras (ou documentos) **pode ser eliminado quando considerarmos apenas os principais valores singulares** (assim, a versão que trunca a matriz pode ser útil).
- Dimensões menores nas matrizes podem tornar outros algoritmos (e.g. aprendizado de máquina) mais “simples”.
- Porém, **o custo computacional para decompor uma matriz é caro**. Avalie se o custo vale a pena.



```
# Calculando para cada palavra do vocabulário: PPMI
N = numpy.sum(Mcontext)
PPMI = numpy.zeros((V, V))

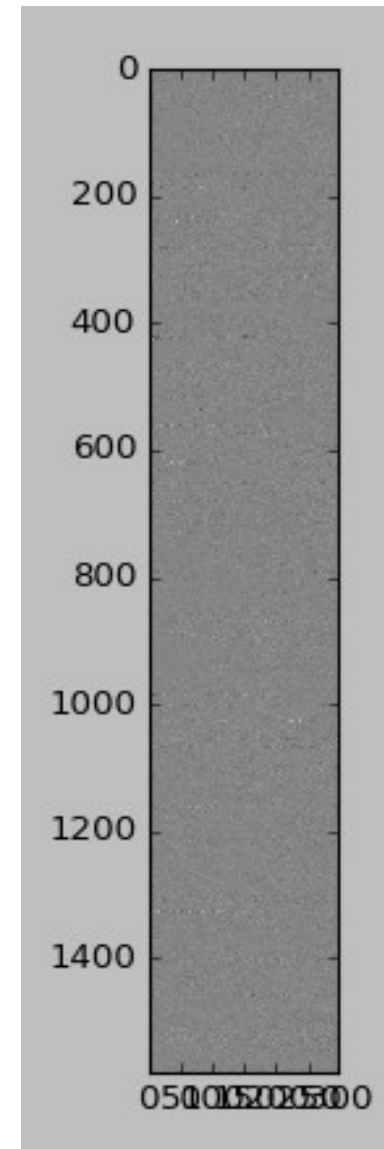
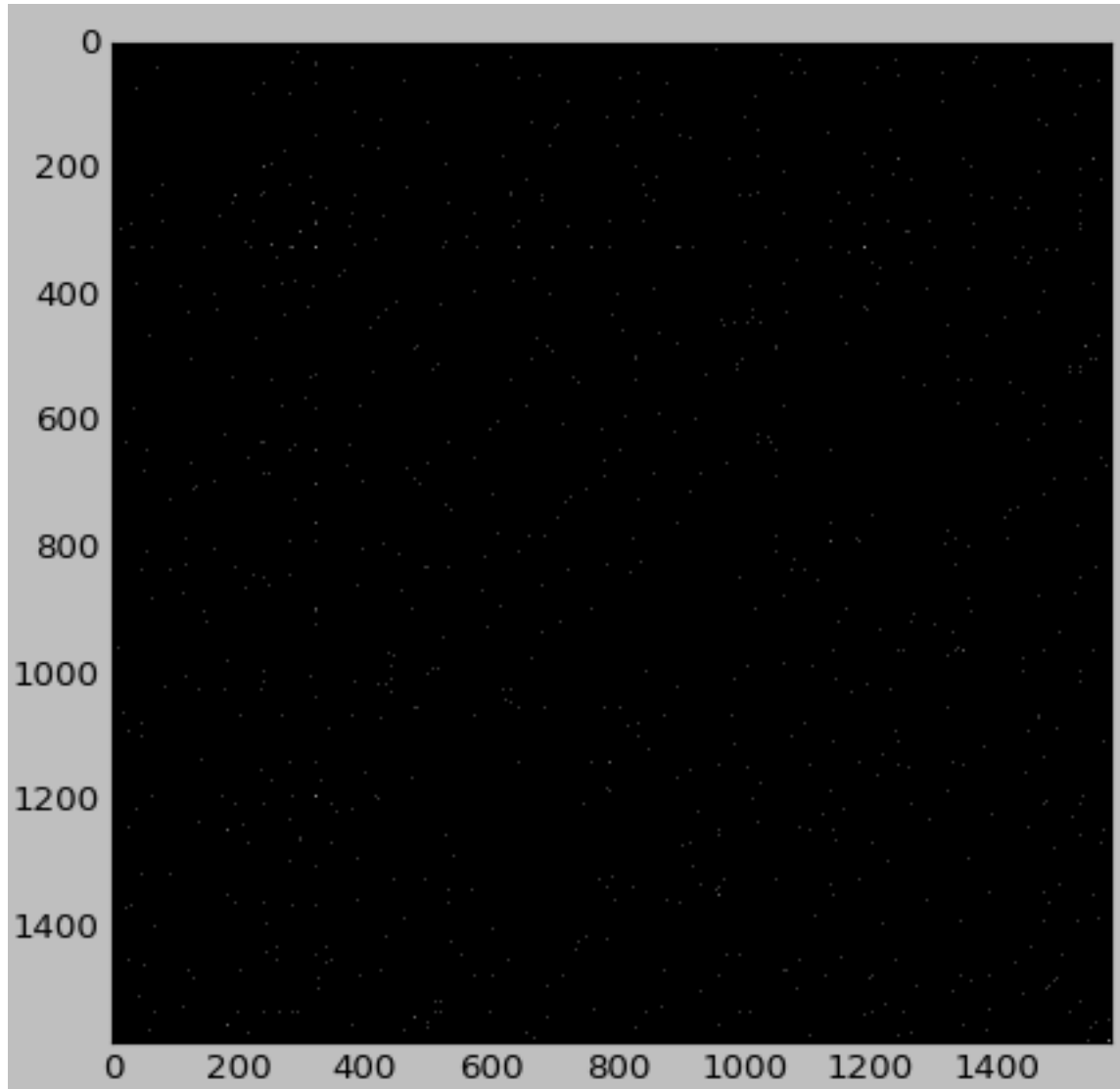
Fw = numpy.sum(Mcontext, axis=1) # somatoria de cada linha
Fc = numpy.sum(Mcontext, axis=0) # somatoria de cada coluna

for i in range(0, V):
    for j in range(0, V):
        PPMI[i,j] = max(0, math.log2((Mcontext[i,j]/N)/(Fw[i]/N*Fc[j]/N)) )

(W,S,C)= numpy.linalg.svd(PPMI)
Wt = W[:,0:299] # Apenas as 300 primeiras dimensoes

import matplotlib.image as img
import matplotlib.pyplot as plt
plt.imshow(Wt[0:300,:], cmap=plt.cm.gray, interpolation='none')
plt.show()
```

# Exemplo





**Mensurando similaridade por distância  
Cosseno usando essa nova representação**

# Produto interno entre dois vetores?

$$\text{dot-product}(\vec{v}, \vec{w}) = \vec{v} \cdot \vec{w} = \sum_{i=1}^N v_i w_i = v_1 w_1 + v_2 w_2 + \dots + v_N w_N$$

Length of vector  $\mathbf{u}, \mathbf{v}$

Symbol for inner product

Angle between  $\mathbf{u}$  and  $\mathbf{v}$

$$\mathbf{u} \bullet \mathbf{v} = |\mathbf{u}| |\mathbf{v}| \cos(\theta) \quad \text{--- 1}$$
$$= x_1 \times x_2 + y_1 \times y_2 \quad \text{--- 2}$$

Esta medida pode ser considerado como medida de similaridade:

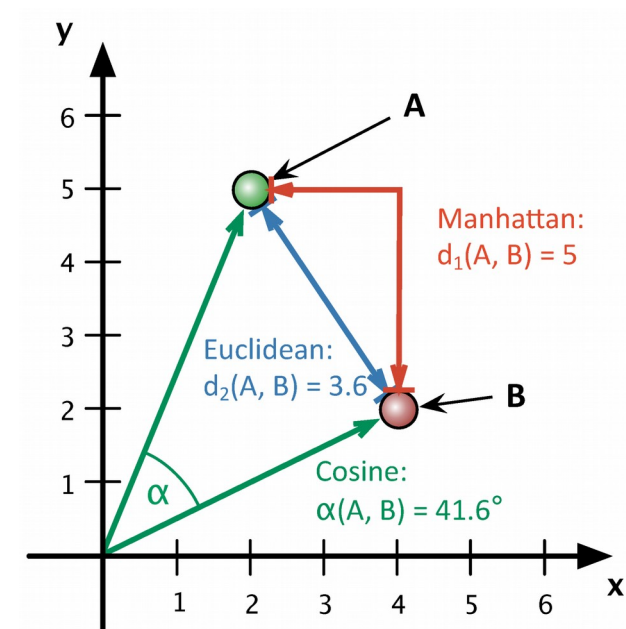
- Um valor **alto** representaria maior proximidade entre palavras
- Um valor **baixo** representaria menor proximidade entre palavras

# Distância cosseno

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\vec{v}}{|\vec{v}|} \cdot \frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

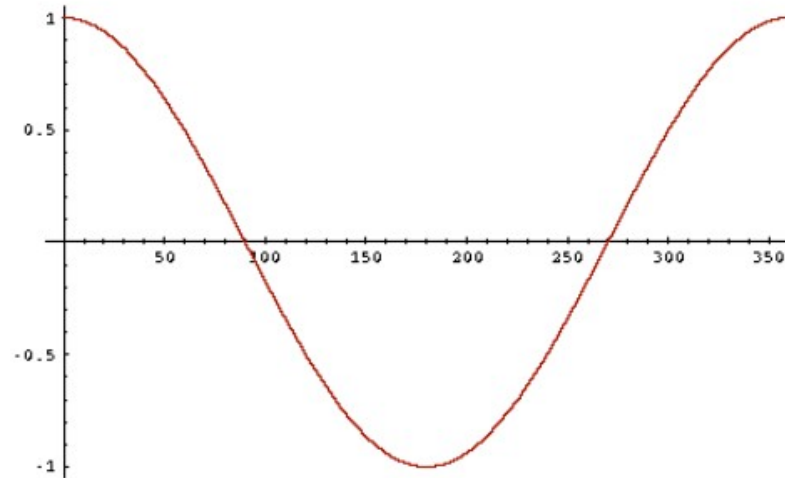
Dot product

Unit vectors



- $v_i$  é o valor de **PPMI** da palavra  $v$  no contexto  $i$ .
- $w_i$  é o valor de **PPMI** da palavra  $w$  no contexto  $i$ .

# Distância cosseno como medida de similaridade



- -1: Vetores apontam em direções opostas.
- +1: Vetores **apontam à mesma direção**.
- 0: Vetores ortogonais

PPMI terá valores entre **0 e +1**

# Exemplo - ângulos

	large	data
apricot	2	0
digital	0	1
information	1	6

