Vetorização de texto

Tópicos de Inteligência Artificial

Prof. Dr. Dieval Guizelini

Março 2023

Word2vec

- Tomas Mikolov, et al. no Google em 2013, amplamente utilizada para tarefas de NLP

- Considere o texto: “O cachorro corre atrás do gato. O gato corre atrás do rato.”

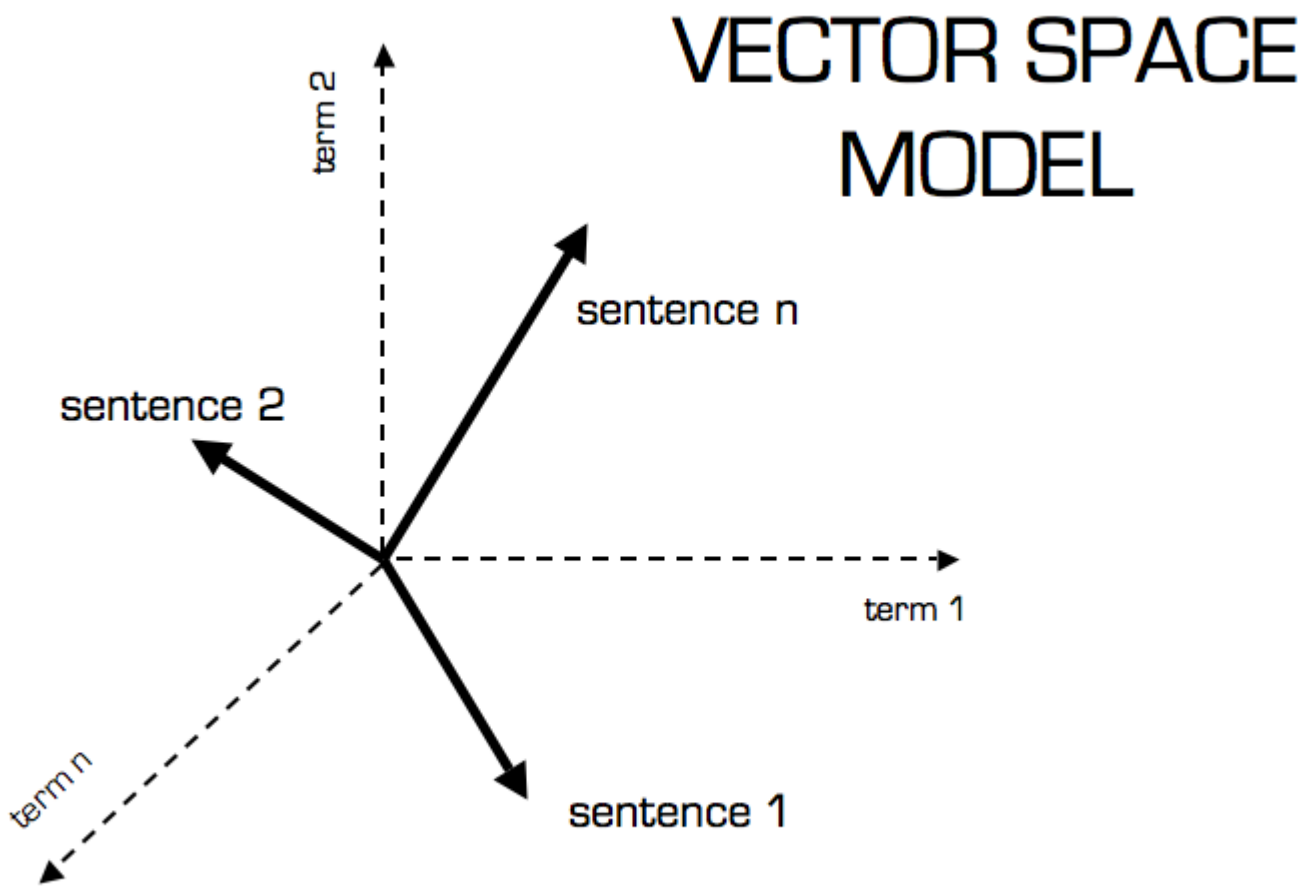
	O	cachorro	corre	atrás	do	gato	.	O	gato	corre	atrás	do	rato	.
O	1	0	0	0	0	0	0	1	0	0	0	0	0	0
cachorro	0	1	0	0	0	0	0	0	0	0	0	0	0	0
corre	0	0	1	0	0	0	0	0	0	1	0	0	0	0
atrás	0	0	0	1	0	0	0	0	0	0	1	0	0	0
do	0	0	0	0	1	0	0	0	0	0	0	1	0	0
gato	0	0	0	0	0	1	0	0	1	0	0	0	0	0
.	0	0	0	0	0	0	1	0	0	0	0	0	0	1
rato	0	0	0	0	0	0	0	0	0	0	0	0	1	0

[1, 2, 3, 4, 5, 6, 1, 7, 3, 4, 5, 8]

	O	cachorro	corre	atrás	do	gato	.	O	gato	corre	atrás	do	rato	.
O	1	0	0	0	0	0	0	1	0	0	0	0	0	0
cachorro	0	1	0	0	0	0	0	0	0	0	0	0	0	0
corre	0	0	1	0	0	0	0	0	0	1	0	0	0	0
atrás	0	0	0	1	0	0	0	0	0	0	1	0	0	0
do	0	0	0	0	1	0	0	0	0	0	0	1	0	0
gato	0	0	0	0	0	1	0	0	1	0	0	0	0	0
.	0	0	0	0	0	0	1	0	0	0	0	0	0	1
rato	0	0	0	0	0	0	0	0	0	0	0	0	1	0

- Nessa estratégia, é possível recuperar a frequência das palavras e eventualmente, a co-ocorrência, mas nada de semântica.
- Com Word Embedding, conseguimos aprender o mapa de características das palavras, através de seus valores numéricos, que são passados ao modelo de treinamento.

Modelo do espaço vetorial



“embed words”

Chamamos de “incorporar palavras” (embeddings, embed word) o processo pela qual criamos uma representação vetorial de alta dimensionalidade no qual as palavras semelhantes estão próximas.

Em pacotes como o Tensorflow, existem três mecanismos para isso:

- Método nativo: que é uma distribuição aleatória uniforme.
- Keras layer:
- Tensorflow Layers:

O word2vec é um modelo “embeddings” já treinado com milhares de palavras: https://arxiv.org/abs/1301.3781, outro é o glove https://nlp.stanford.edu/pubs/glove.pdf

Sobre a “incorporação” de palavras

- Tf-idf
  - Cavalo de batalha da recuperação de informações!
  - Um modelo de linha de base comum
  - vetores esparsos
  - As palavras são representadas por (uma função simples de) as contagens de palavras próximas
- Word2vec
  - vetores densos
  - A representação é criada treinando um classificador para prever se uma palavra provavelmente aparecerá nas proximidades (CBOW,skip-gram)

Os resultados (artigo)

Table 2: *Accuracy on subset of the Semantic-Syntactic Word Relationship test set, using word vectors from the CBOW architecture with limited vocabulary. Only questions containing words from the most frequent 30k words are used.*

Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

Table 3: *Comparison of architectures using models trained on the same data, with 640-dimensional word vectors. The accuracies are reported on our Semantic-Syntactic Word Relationship test set, and on the syntactic relationship test set of [20]*

Model Architecture	Semantic-Syntactic Word Relationship test set		MSR Word Relatedness Test Set [20]
	Semantic Accuracy [%]	Syntactic Accuracy [%]	
RNNLM	9	36	35
NNLM	23	53	47
CBOW	24	64	61
Skip-gram	55	59	56

modelos

Table 4: *Comparison of publicly available word vectors on the Semantic-Syntactic Word Relationship test set, and word vectors from our models. Full vocabularies are used.*

Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	**64.5**	50.8
CBOW	300	783M	15.5	53.1	36.1
Skip-gram	300	783M	**50.0**	55.9	**53.3**

Mais resultado

Table 5: *Comparison of models trained for three epochs on the same data and models trained for one epoch. Accuracy is reported on the full Semantic-Syntactic data set.*

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days]
			Semantic	Syntactic	Total	
3 epoch CBOW	300	783M	15.5	53.1	36.1	1
3 epoch Skip-gram	300	783M	50.0	55.9	53.3	3
1 epoch CBOW	300	783M	13.8	49.9	33.6	0.3
1 epoch CBOW	300	1.6B	16.1	52.6	36.1	0.6
1 epoch CBOW	600	783M	15.4	53.3	36.2	0.7
1 epoch Skip-gram	300	783M	45.6	52.2	49.2	1
1 epoch Skip-gram	300	1.6B	52.2	55.1	53.8	2
1 epoch Skip-gram	600	783M	56.7	54.5	55.5	2.5

Large Scale Parallel Training of Models

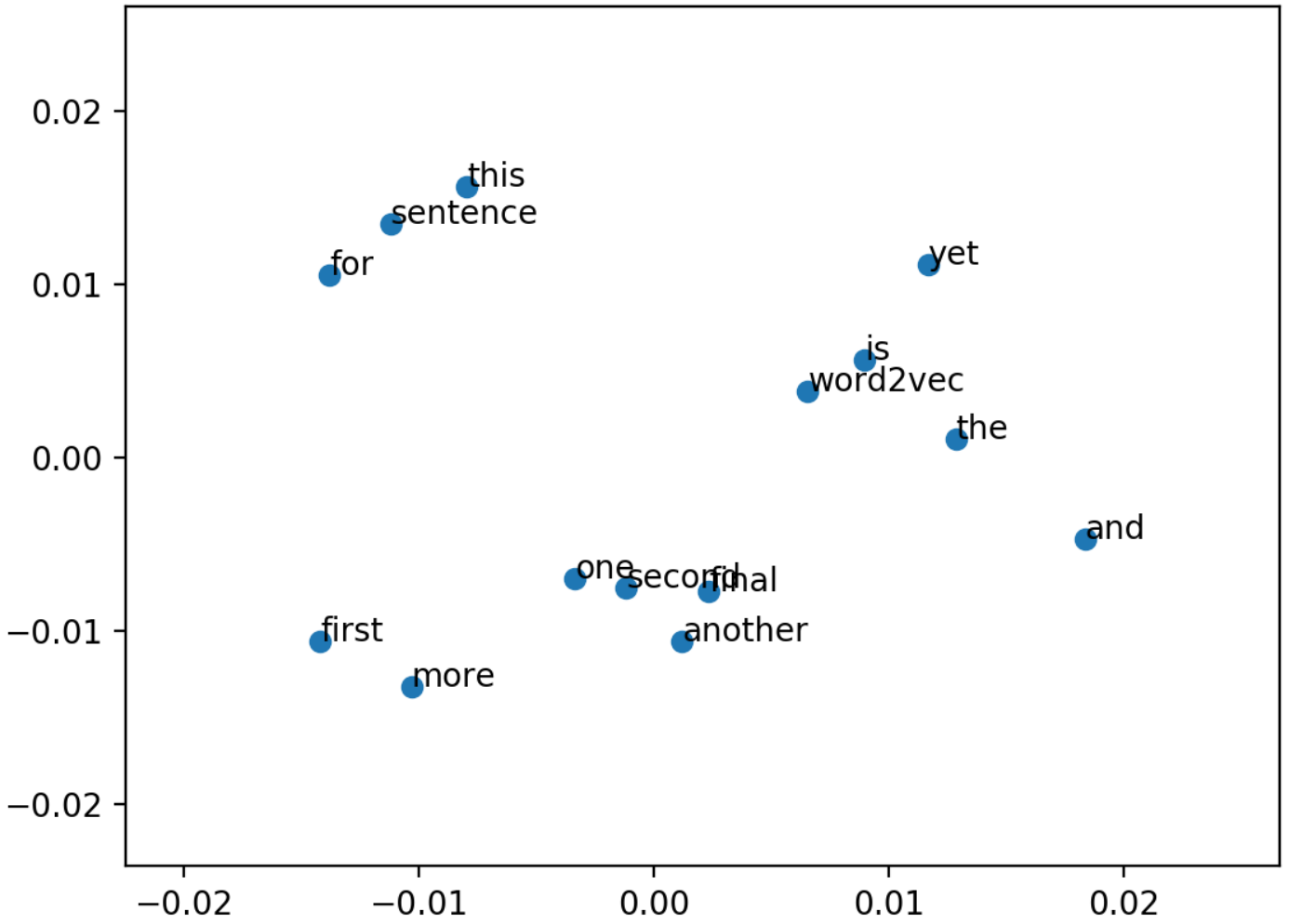
Table 6: *Comparison of models trained using the DistBelief distributed framework. Note that training of NNLM with 1000-dimensional vectors would take too long to complete.*

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125

Table 7: *Comparison and combination of models on the Microsoft Sentence Completion Challenge.*

Architecture	Accuracy [%]
4-gram [32]	39
Average LSA similarity [32]	49
Log-bilinear model [24]	54.8
RNNLMs [19]	55.4
Skip-gram	48.0
Skip-gram + RNNLMs	**58.9**

PCA


0.02
0.01
0.00
−0.01
−0.02
−0.02
−0.01
0.00
0.01
0.02
this
sentence
for
yet
is
word2vec
the
and
one
second
final
first
another
more

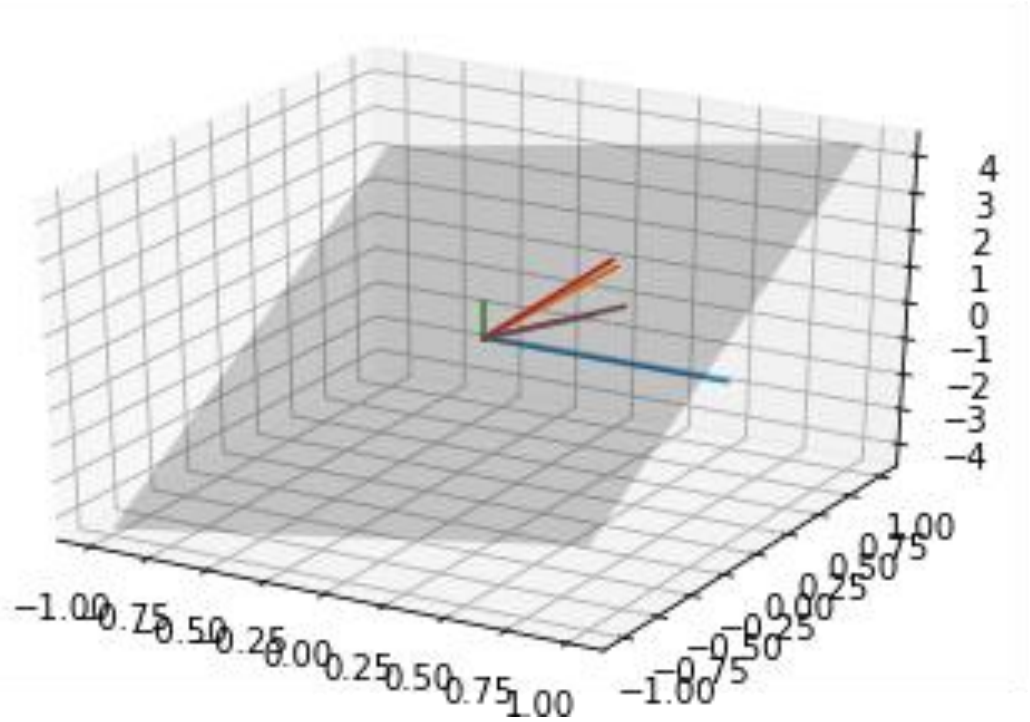
No Python

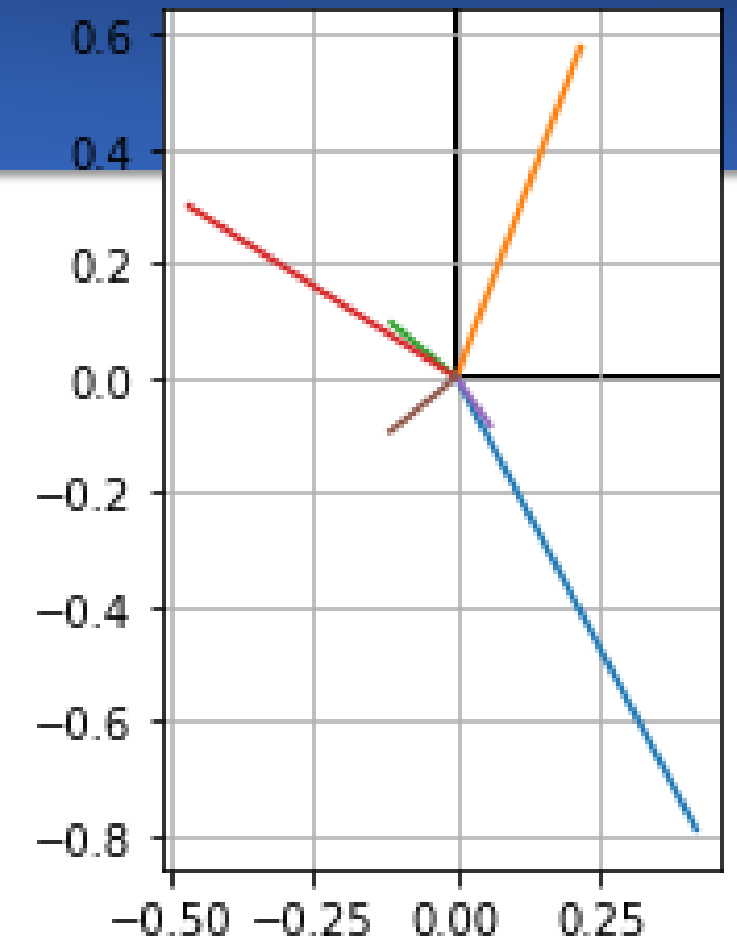
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.cluster.hierarchy import dendrogram, linkage, ClusterWarning
from warnings import simplefilter

aux = list( [ (1,0,0), (0,1,0), (0,0,1)] )
aux.append( tuple(np.random.random_sample(3,)) )
aux.append( tuple(np.random.random_sample(3,)) )
aux.append( tuple(np.random.random_sample(3,)) )
vetores = np.array(aux)
print(vetores)
centro = vetores.mean(axis=0)
v_centro = vetores - centro

print('centro','\n',centro)
print( np.allclose(vetores, v_centro+centro))
```

```
# obter o plano para projeção ortogonal
U,s,Vh = np.linalg.svd(vetores)

# print('Variancia: ', np.square(s) / np.sq
uare(s).sum() )
W2 = Vh.T[:, :2]
v1_plano = Vh.T[:, 0]
v2_plano = Vh.T[:, 1]
projetados2d = v_centro.dot(W2)
print(projetados2d)
```

```
#
# calculando o plano de projeção
menor = np.min(vetores,axis=0)-1
maior = np.min(vetores,axis=0)+1
x1s = np.linspace(menor[0], maior[0], 10)
y1s = np.linspace(menor[1], maior[1], 10)
z1s = np.linspace(menor[2], maior[2], 10)
C = Vh
R = C.T.dot(C)
x1, x2 = np.meshgrid(x1s, y1s)
z = (R[0, 2] * x1 + R[1, 2] * x2) / (1 - R[2, 2])
```

```
#
# tentando visualizar as projeções no espaço 3d
fig = plt.figure(figsize=(6, 3.8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x1, x2, z, alpha=0.2, color="k")

# pontos originais
for v in vetores:
  xs = np.linspace(0, v[0], 10)
  ys = np.linspace(0, v[1], 10)
  zs = np.linspace(0, v[2], 10)
  plt.plot(xs,ys,zs=zs)

plt.show()
```

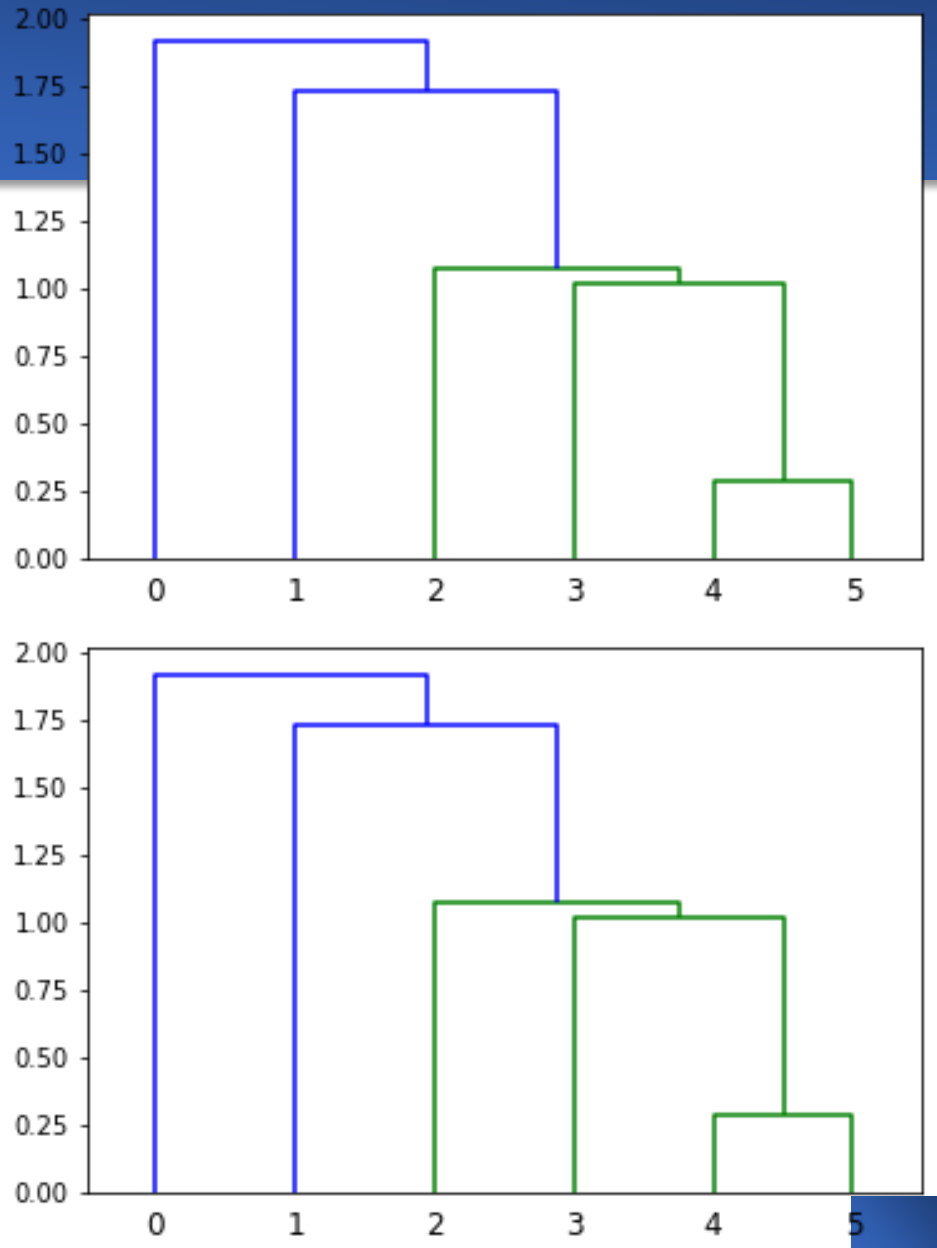





```
fig = plt.figure(figsize=(6, 3.8))
ax = fig.add_subplot(111, aspect='equal')
# pontos projetados
for v in projetados2d:
  xs = np.linspace(0, v[0], 10)
  ys = np.linspace(0, v[1], 10)
  ax.plot(xs,ys)

ax.arrow(0, 0, 0, 1, head_width=0.05, \
       length_includes_head=True, head_length=0.1, fc='k', ec='k')
ax.arrow(0, 0, 1, 0, head_width=0.05, \
       length_includes_head=True, head_length=0.1, fc='k', ec='k')
ax.grid(True)
plt.show()
```

```
original
[[0.         1.41421356 1.41421356 1.45594982 0.89398269 0.95453964]
 [1.41421356 0.         1.41421356 1.02841177 0.99977719 1.02510946]
 [1.41421356 1.41421356 0.         0.70014822 0.60543884 0.61646016]
 [1.45594982 1.02841177 0.70014822 0.         0.64838817 0.52664191]
 [0.89398269 0.99977719 0.60543884 0.64838817 0.         0.17752304]
 [0.95453964 1.02510946 0.61646016 0.52664191 0.17752304 0.        ]]
projetada
[[0.         1.37747273 1.03029247 1.40022819 0.791225   0.87470855]
 [1.37747273 0.         0.58173509 0.73510591 0.67621874 0.74664486]
 [1.03029247 0.58173509 0.         0.40682754 0.24730662 0.18947039]
 [1.40022819 0.73510591 0.40682754 0.         0.64816006 0.52637396]
 [0.791225   0.67621874 0.24730662 0.64816006 0.         0.17423772]
 [0.87470855 0.74664486 0.18947039 0.52637396 0.17423772 0.        ]]
 ---
```

```
simplefilter("ignore", ClusterWarning)
Z1 = linkage(d, 'single')
fig = plt.figure()
dn = dendrogram(Z1)
plt.show()

Z2 = linkage(d1, 'single')
fig = plt.figure()
dn = dendrogram(Z1)
plt.show()
```



```python
def vectorDistance(v1,v2):
  soma = 0
  for i in range(0,len(v1)):
    d = v1[i]-v2[i]
    soma = soma + d*d
  return math.sqrt(soma)
```

1

```python
def calcDistance(mat):
  nrows = len(mat)
  ncols = len(mat[0])
  res = np.zeros( (nrows,nrows) )
  for i in range(0,nrows):
    m1 = mat[i]
    for j in range(0,nrows):
      if i!=j:
        m2 = mat[j]
        res[i][j] = vectorDistance(m1,m2)
      else:
        res[i][j] = 0
  return res
```

2

```python
d = calcDistance(vetores)
d1 = calcDistance(projetados2d)

print('original')
print(d)

print('projetada')
print(d1)
```

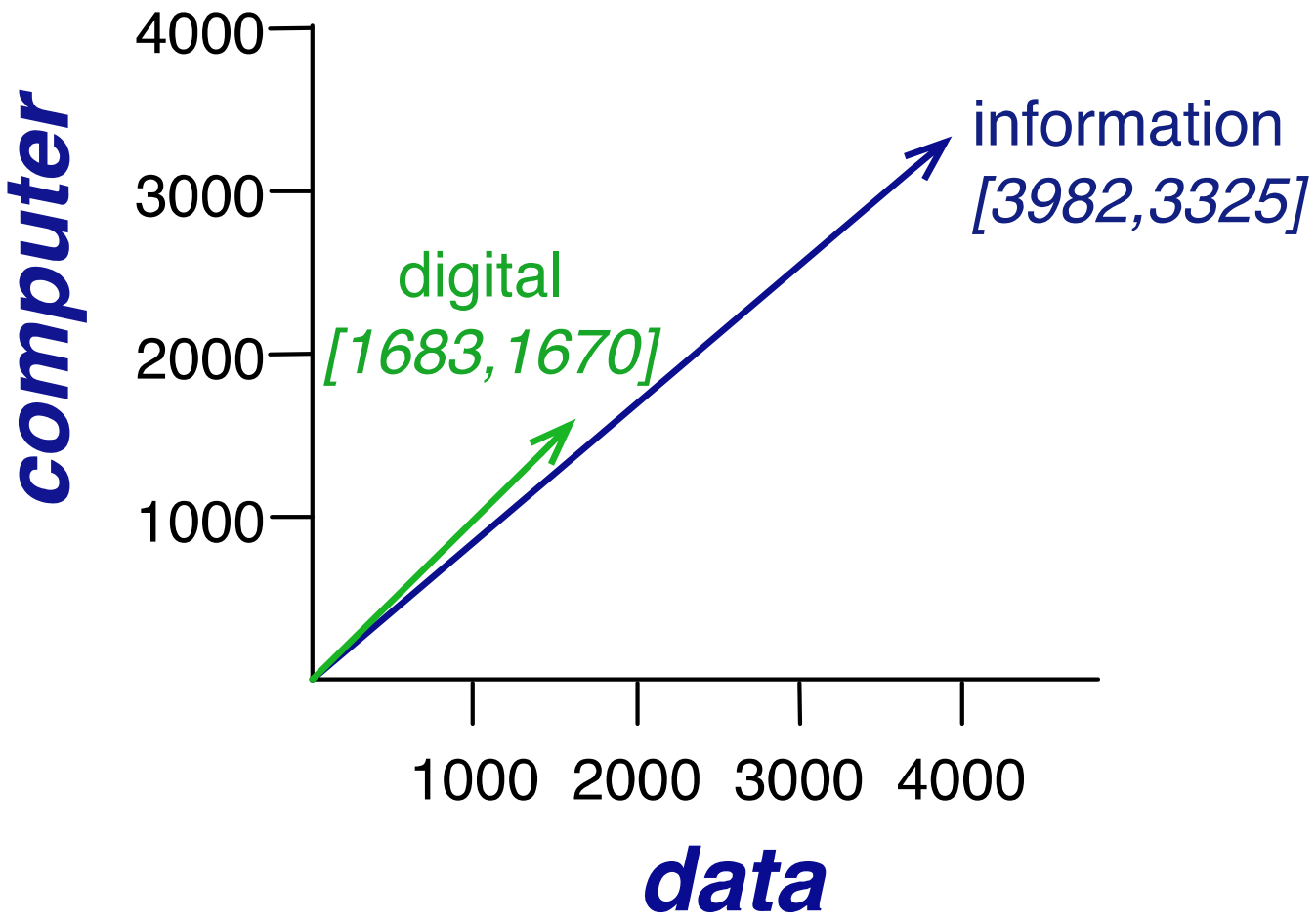
3

- Duas palavras são semelhantes em significado se seus vetores de contexto forem semelhantes

is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet

	aardvark	...	**computer**	**data**	**result**	**pie**	**sugar**	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...



Computando a similaridade de palavras: produto escalar e cosseno

- O produto escalar entre dois vetores é um escalar:

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

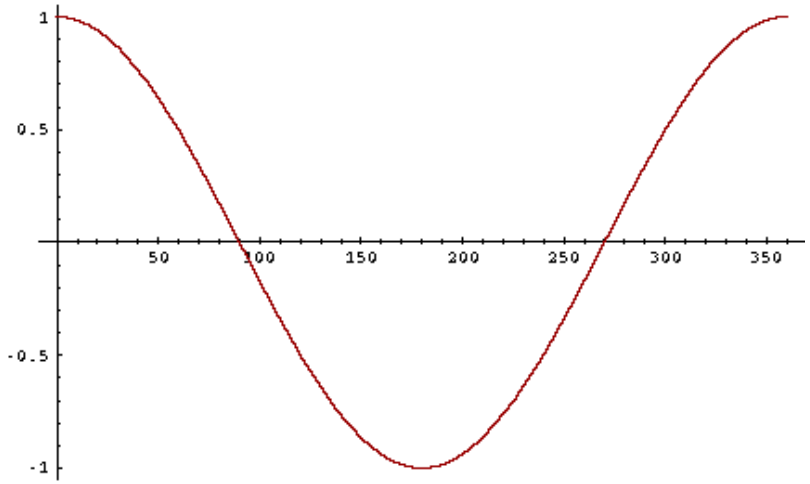
- O produto escalar tende a ser alto quando os dois vetores têm valores grandes nas mesmas dimensões
- O produto escalar pode, portanto, ser uma métrica de similaridade útil entre vetores

- Problema com produto escalar bruto
  - O produto escalar favorece vetores longos
  - O produto escalar é maior, se um vetor for mais longo (tem valores maiores em muitas dimensões)
  - Palavras frequentes (de, o, você) têm vetores longos (já que ocorrem muitas vezes com outras palavras).
  - Portanto, o produto escalar favorece excessivamente palavras frequentes
- Comprimento do vetor:

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

alternativa:
cosseno para calcular similaridade entre palavras

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



Com base na definição do produto escalar entre dois vetores a e b

$$\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\theta$$

$$\frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} = \cos\theta$$

+1 os vetores apontam para a mesma direção
-1 os vetores apontam para direções opostas
0 os vetores são ortogonais

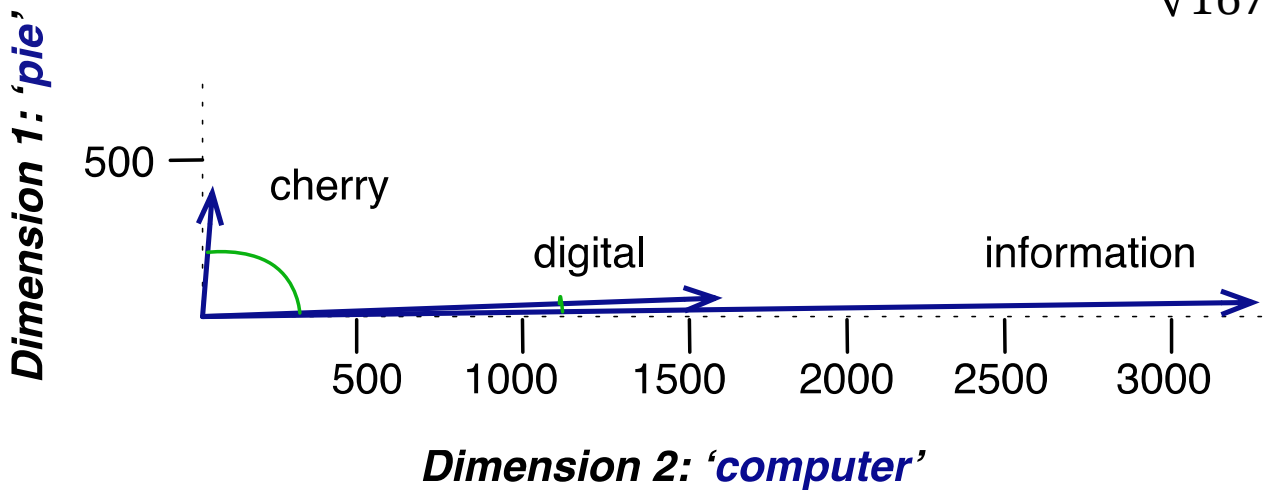
Mas, como os valores brutos de frequência não são negativos, o cosseno para vetores de matriz de termo-termo varia de 0 a 1

Exemplo com cosseno

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

$$\cos(cherry, information) = \frac{2 * 3325 + 8 * 3982 + 442 * 5}{\sqrt{442^2 + 8^2 + 2^2}\sqrt{5^2 + 3982^2 + 3325^2}} = 0.17$$

$$\cos(digital, information) = \frac{1670 * 3325 + 1683 * 3982 + 5 * 5}{\sqrt{1670^2 + 1683^2 + 5^2}\sqrt{5^2 + 3982^2 + 3325^2}} = 0.996$$



Referencias

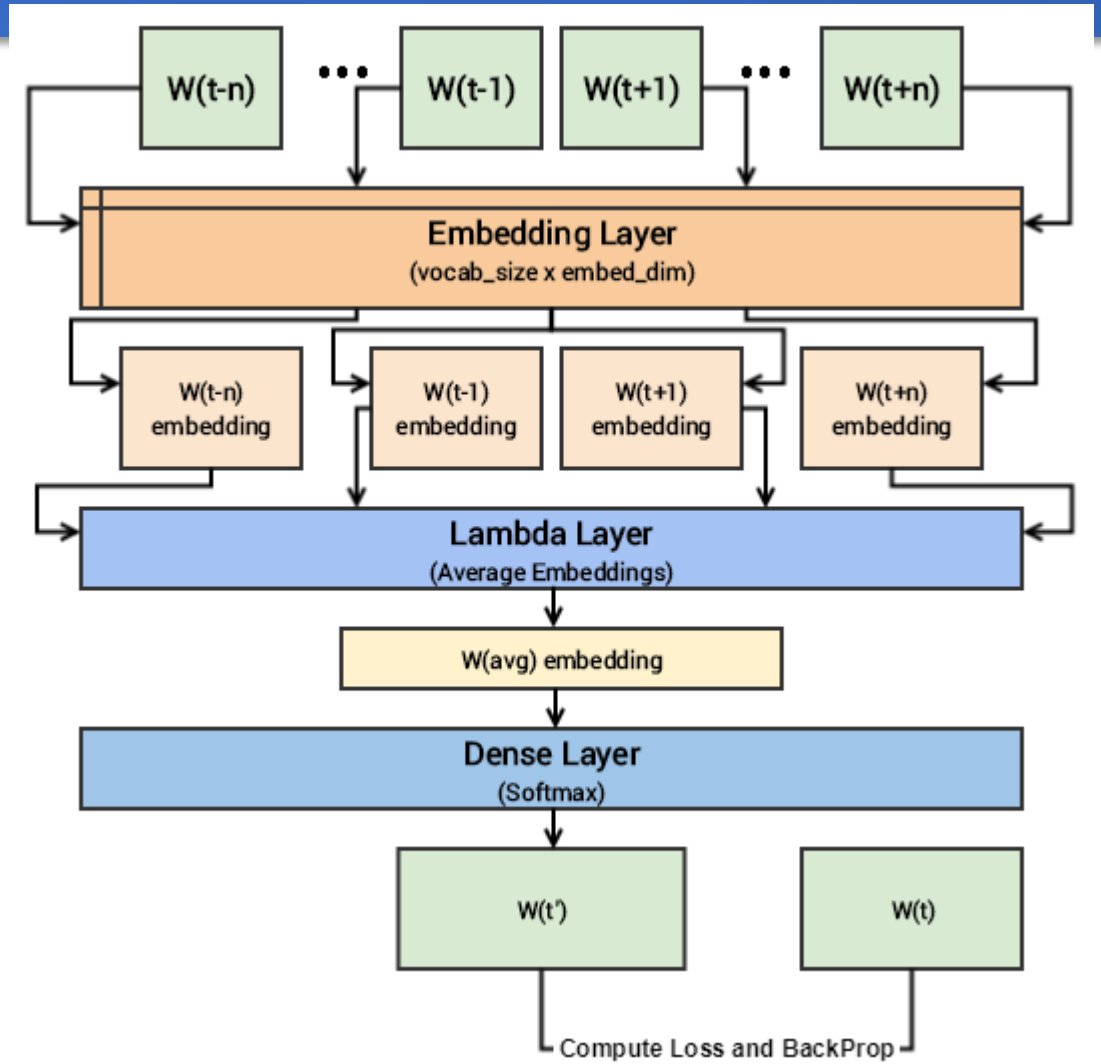
- Mikolov et al, Efficient Estimation of Word Representations in Vector Space, 2013 https://arxiv.org/pdf/1301.3781.pdf

- Gensim word2vec https://radimrehurek.com/gensim/

- Dois de cinco conjuntos disponíveis com o software SENNA (2011) http://ronan.collobert.com/senna/

- Tripadvisor and Edmunds https://github.com/kavgan/OpinRank/tree/master

- Alternativa a visualização do PCA, use o t-sne https://observablehq.com/@robstelling/abrindo-a-caixa-preta-do-t-sne/2




```
embeddings = tf.keras.layers.Embedding(VOCAB_LEN, EMBED_SIZE)
embed = embeddings(words_ids)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print(sess.run(embed))
```

```
[[0.9134803  0.36847484 0.51816785 0.19543898 0.07610226 0.8685185
  0.7445053  0.5340642  0.5453609  0.72966635 0.06846464 0.19424069
  0.2804587  0.77481234 0.7343868  0.16347027 0.56002617 0.76706755
  0.16558647 0.6719606  0.05563295 0.22389805 0.47797906 0.98075724
  0.47506428 0.7846818  0.65209556 0.89036727 0.14960134 0.8801923
  0.23688185 0.70695686 0.59664845 0.6206044  0.69665396 0.60709286
  0.42249918 0.7317171  0.03822994 0.37915635 0.60433483 0.4168439
  0.5516542  0.84362316 0.27857065 0.33540523 0.8601098  0.47720838
  0.9827635  0.09320438]
 [0.27832222 0.8259096  0.5726856  0.96932447 0.21936393 0.26346993
  0.38576245 0.60339177 0.03083277 0.665465   0.9077859  0.6219367
  0.5185654  0.5444832  0.16380131 0.6688931  0.82876015 0.9705752
  0.40097427 0.28450823 0.9425919  0.50802815 0.02394092 0.24661314
  0.45858765 0.7080616  0.8434526  0.46829247 0.0329994  0.10844195
  0.6812979  0.3505745  0.67980576 0.71404254 0.8574227  0.40939808
  0.8668809  0.58524954 0.52820635 0.31366992 0.05352783 0.8875419
  0.04600751 0.27407455 0.6398467  0.74402344 0.9710648  0.5717342
  0.78711486 0.9209585 ]]
```

Exemplos: https://github.com/FrancescoSaverioZuppichini/How-To-Embed-in-TensorFlow/blob/master/README.md