TDVI: Inteligencia Artificial

Word2vec

UTDT - LTD

Word2vec

Siguiendo a Jurafsky:

"*The idea of vector semantics is thus to represent a word as a point in some multidimensional semantic space*"

"*Vectors for representing words are generally called embeddings, because the word is embedded in a particular vector space*"

palabra → vector

Word2vec

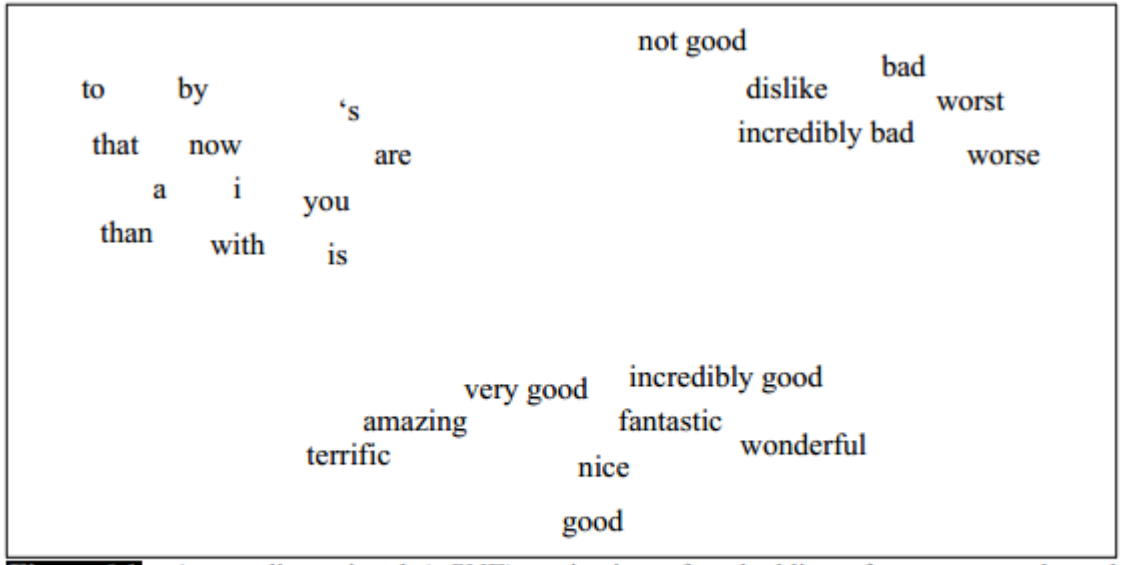


Figure 6.1 A two-dimensional (t-SNE) projection of embeddings for some words and phrases, showing that words with similar meanings are nearby in space. The original 60-dimensional embeddings were trained for a sentiment analysis task. Simplified from Li et al. (2015).

Noten que palabras que podríamos considerar similares se encuentran cerca en el espacio generado

Word2vec

Hipótesis distribucional (*distributional hypothesis*): palabras que se utilizan y aparecen en los mismos contextos tienden a transmitir significados cercanos/relacionados

¿"veterinaria" aparece en los mismos contextos que "gato"?

¿En los mismos que "perro"?

¿En los mismos de "álgebra"?

Word2vec

Para cada palabra en una oración uno puede ver su contexto cómo las *n* palabras anteriores y posteriores (esta es una definición de trabajo)

their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**.	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

Luego, para cada palabra de un corpus dado, uno puede tratar de cuantificar qué tanto ocurre otra palabra en su contexto

¿Veterinaria aparece en los contextos de gato? ¿Y en los de perro?
¿Aparecerá en los contextos de álgebra?

Word2vec

Existen diversas formas de generar embeddings de palabras (e.g., cada columna de una *document-term-matrix* puede considerarse un embedding)

Ahora vamos a ver una forma de generar vectores pequeños (50-500 elementos) y densos (sin ceros)

Word2vec

Estrategia: vamos a entrenar un modelo que intente predecir si dos palabras tienden a aparecer juntas o no

No nos va a interesar el problema de predicción en sí. Nos van a interesar los parámetros que aprenda este modelo (los embeddings)

Los parámetros a aprender van a estar contenidos en dos matrices $T \in \mathbb{R}^{Vxd}$ y $C \in \mathbb{R}^{dxV}$. En donde *V* es el tamaño del vocabulario y *d* la dimensión de los embeddings

Word2vec

El modelo a usar va a ser casi una regresión logística

Concretamente será un clasificador que deberá predecir:

- *P(+|t, c)* altos para palabras que comparten contextos
- *P(+|t, c)* bajos para palabras que no lo hacen

Word2vec

Definición de similitud en word2vec: en w2v dos palabras (*i* y *j*) serán consideradas similares si el producto interno de sus vectores t_i y c_j es alto

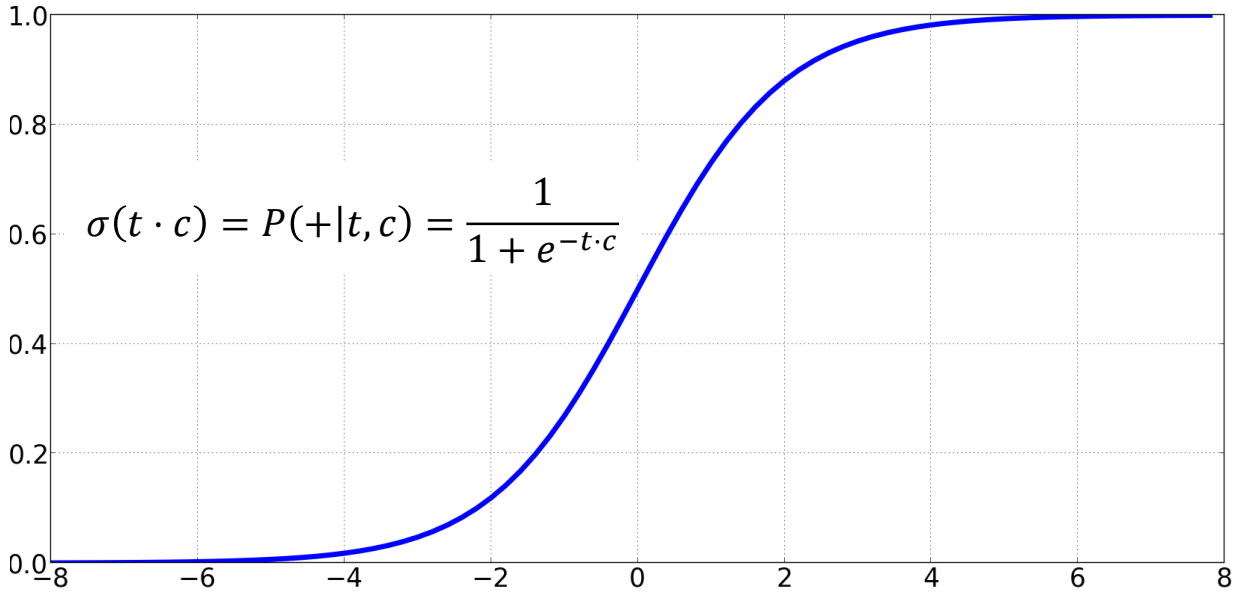
$$Similarity(t, c) \approx t \cdot c$$

- Queremos que sea alto para *t* = perro y *c* = veterinaria

- Queremos que sea bajo para *t* = perro y *c* = álgebra

Word2vec

Transformamos el producto interno en una probabilidad mediante la función sigmoidea

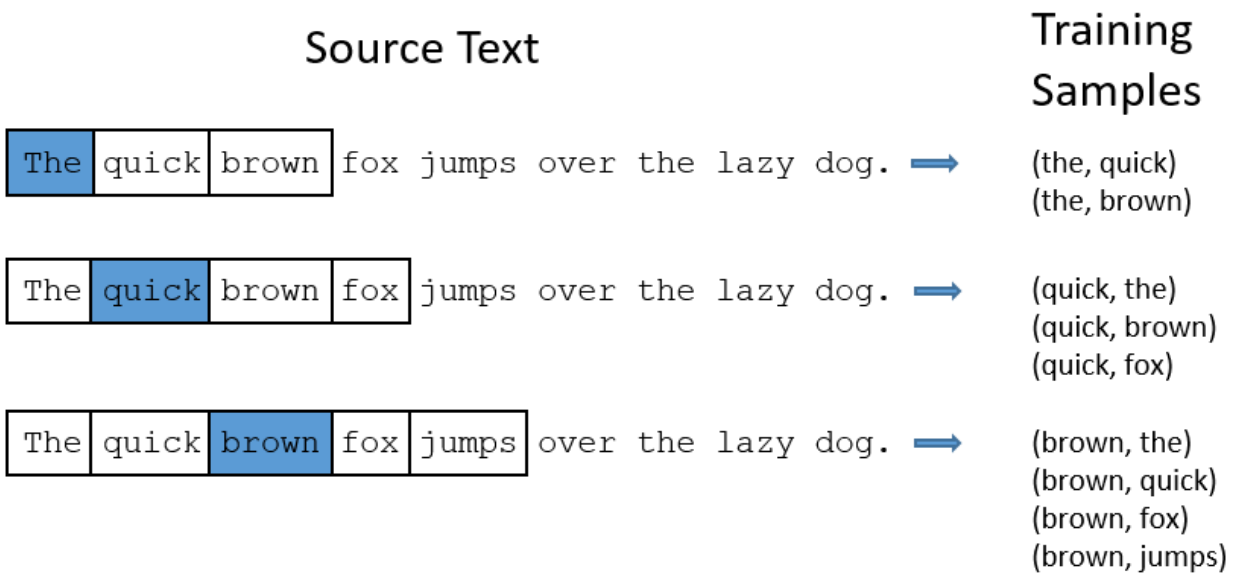


- *t* = perro y *c* = veterinaria debería dar una valor cercano a 1
- *t* = perro y *c* = álgebra debería dar una valor cercano a 0

Word2vec

Para entrenar un clasificador se necesitan datos, con casos + y casos -. Estos se pueden construir fácilmente a partir de un corpus (*self-supervised learning*)

Casos positivos (noten que importa el tamaño de ventana)



Word2vec

Casos negativos

Para cada caso positivo se muestrean *k* casos negativos (que no sean el target), a estas se les llama *noise words*

```
... lemon,  a  [tablespoon  of  apricot  jam,  a]  pinch ...
                    c1      c2     t      c3     c4
```

positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

w	c_{neg}	w	c_{neg}
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Word2vec

Lo que se busca es encontrar los valores de *T* y *C* (que en conjunto se llaman θ) que hacen que la *P(+)* sea alta para los casos positivos y baja para los casos negativos

En concreto, se busca maximizar la siguiente función:

$$L(\theta) = \sum_{(t,c)\in+} \log P(+|t,c) + \sum_{(t,c)\in-} \log P(-|t,c)$$

Para una observación positiva y sus negativas muestreadas se tiene:

$$L(\theta) = \log P(+|t,c) + \sum_{i=1}^{k} \log P(-|t,n_i)$$

Word2vec

Veámoslo matricialmente. Recordemos que los word embeddings se encuentran en dos matrices *T* y *C*

Supongamos: *t* es la palabra *2*, *c*+ es *V-1*, y *c*- son 1 y la *V*

$$[0 \quad 1 \quad \dots \quad 0 \quad 0] * \begin{bmatrix} - & t_1 & - \\ - & t_2 & - \\ - & t_3 & - \\ & \vdots & \\ - & t_{V-2} & - \\ - & t_{V-1} & - \\ - & t_V & - \end{bmatrix} * \begin{bmatrix} | & | & & | & | \\ c_1 & c_2 & \dots & c_{V-1} & c_V \\ | & | & & | & | \end{bmatrix}$$

$$[- \quad t_2 \quad -] * \begin{bmatrix} | & | & & | & | \\ c_1 & c_2 & \dots & c_{V-1} & c_V \\ | & | & & | & | \end{bmatrix}$$

$$[t_2 \cdot c_1 \quad t_2 \cdot c_2 \quad t_2 \cdot c_3 \quad \dots \quad t_2 \cdot c_{V-2} \quad t_2 \cdot c_{V-1} \quad t_2 \cdot c_V]$$

Word2vec

A cada elemento del vector de similitudes se lo pasa por la función sigmoidea (ahora se tiene probabilidades estimadas)

$$P = [\sigma(t_2 \cdot c_1) \quad \sigma(t_2 \cdot c_2) \quad \sigma(t_2 \cdot c_3) \quad \dots \quad \sigma(t_2 \cdot c_{V-2}) \quad \sigma(t_2 \cdot c_{V-1}) \quad \sigma(t_2 \cdot c_V)]$$

Se compara con el vector de contexto positivo y negativos (recuerden: *c*+ es *V*-1, y *c*- son 1 y la *V*):

$$\mathrm{R} = [0 \quad * \quad * \quad \dots \quad * \quad 1 \quad 0]$$

Se modifican levemente t_2, c_1, c_{V-1}, c_V para que *P* se acerque a *R* (descenso gradiente estocástico)

Word2vec

Habiendo minimizado la función de costo importan únicamente los pesos aprendidos (*T* y *C*, o sea los word embeddings)

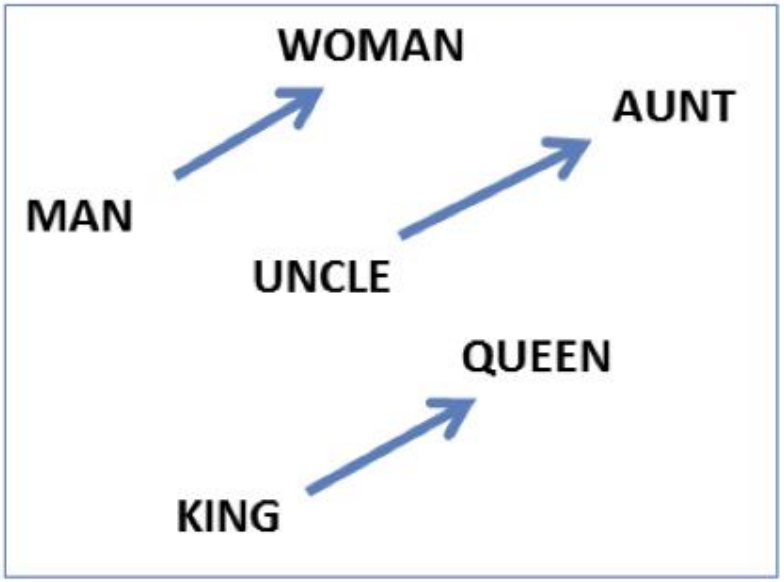
Para cada palabra *i* se aprenden dos vectores (t_i y c_i), generalmente se usan sólo los t_i (aunque existen otras variantes; e.g., sumarlos)

Word2vec

Los word embeddings de word2vec capturan analogías:

vector(*"king"*) - vector(*"man"*) + vector(*"woman"*) ≈ vector(*"queen"*)

vector(*"paris"*) - vector(*"france"*) + vector(*"italy"*) ≈ vector(*"rome"*)



Word2vec

La similitud depende del tamaño de la ventana. Según Jurafsky:

- Ventanas cortas (palabras semánticamente similares)
- Ventanas grandes (palabras relacionadas por tópicos)

Ejemplo:

- C = ±2 palabras más cercanas a Hogwarts:
  - Sunnydale
  - Evernight
- C = ±5 palabras más cercanas a Hogwarts:
  - Dumbledore
  - Malfoy
  - halfblood

Word2vec

Word2vec no genera embedding contextuales

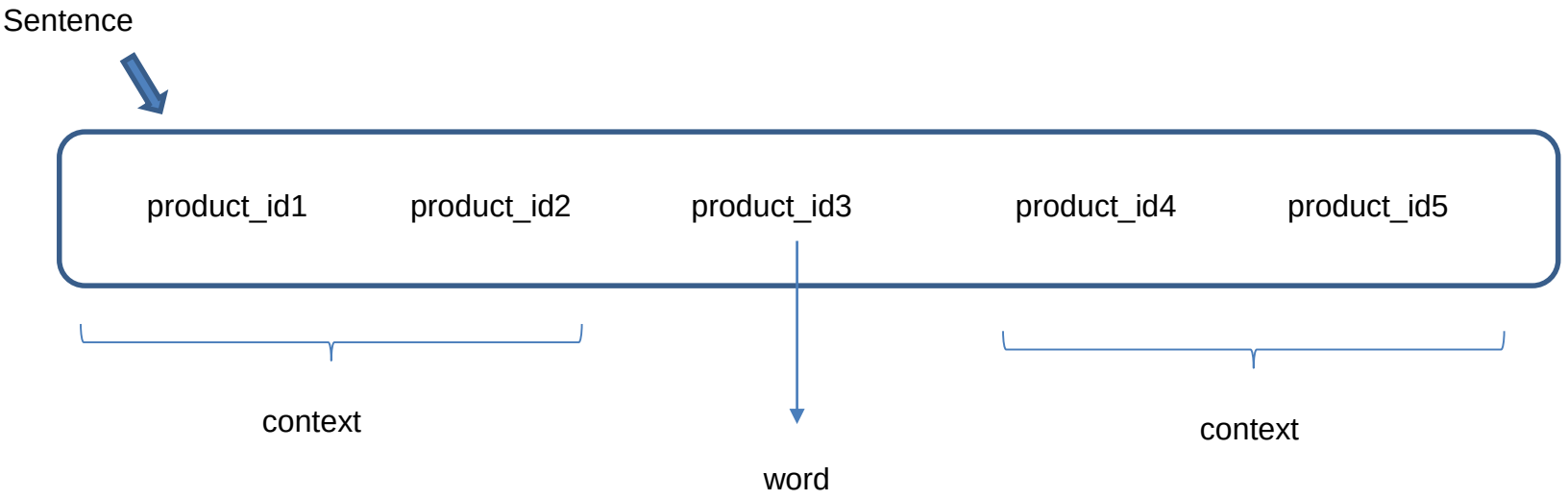
Existen otras alternativas para generar embeddings densos (por ej.: CBOW, GloVe, Fasttext, ELMO, BERT, roBERTA, BETO, GPT-3, GPT-4)

A su vez, se pueden descargar embeddings entrenados en corpus masivos de datos

- Word2vec (Mikolov et al.) *https://code.google.com/archive/p/word2vec/*
- Fasttext (Bojanowski et. al, Joulin et. al) *https://research.fb.com/downloads/fasttext/*
- Glove (Pennington, Socher, Manning) *http://nlp.stanford.edu/projects/glove/*
- Bert (Devlin et. al) *https://github.com/google-research/bert/*

Word2vec

La técnicas es muy versátil. Por ejemplo: Prod2vec



Usando la misma lógica de word2vec podemos aprender embeddings de productos,
tal que, si dos productos son “similares”, sus embeddings estén “cerca” en el espacio vectorial.

Fuente: https://www.analyticsvidhya.com/blog/2019/07/how-to-build-recommendation-system-word2vec-python/

Word2vec

Probémoslo en Python

Bibliografía

- Jurafsky & Martin, "*Speech and Language Processing*". Secciones 6.8, 6.9, 6.10, 6.11 y 6.12