

Procesamiento del Lenguaje Natural

---

# Tema 7. Modelado estadístico del lenguaje

# Índice

## Esquema

## Ideas clave

7.1. Introducción y objetivos

7.2. Introducción al modelo del lenguaje

7.3. Modelos estadísticos del lenguaje basados en N-gramas

7.4. Técnicas de suavizado

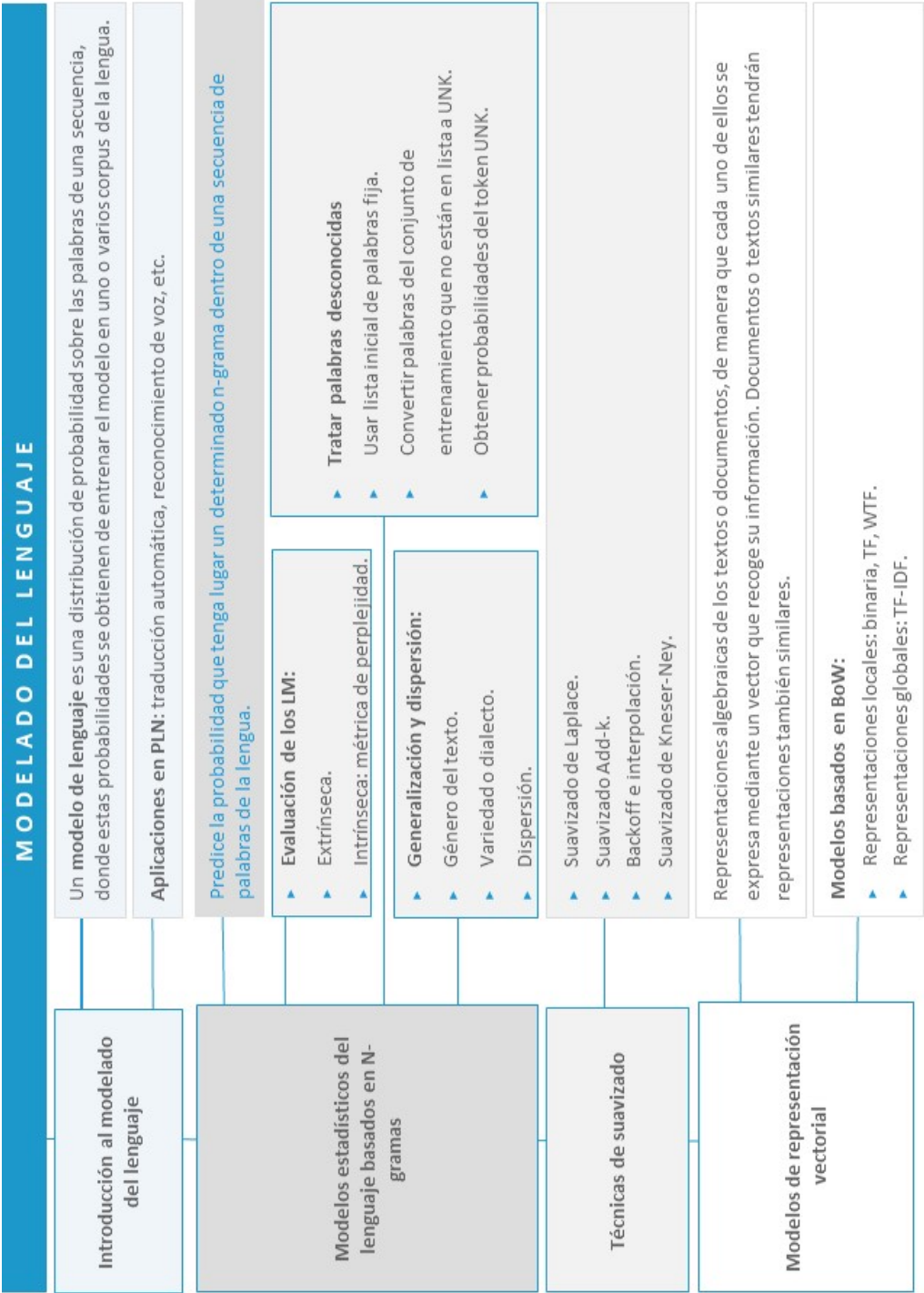
7.5. Modelos de representación vectorial

7.6. Referencias bibliográficas

## A fondo

Modelos de Lenguaje

## Test



## 7.1. Introducción y objetivos

A continuación, profundizaremos en qué es el modelado del lenguaje, y cómo se puede llevar a cabo con técnicas estadísticas como en el caso del uso de n-gramas. Además, veremos algunas limitaciones de esta aproximación y cómo se pueden solucionar con distintas técnicas de suavizado. Finalmente, presentaremos el concepto de modelado vectorial para representar el contenido de los textos de cara a poder trabajar con ellos para algunas tareas específicas de PLN usando como caso concreto el modelado mediante bolsas de palabras.

### Objetivos

- ▶ Definir qué es un modelo del lenguaje y para qué tareas de PLN puede ser útil.
- ▶ Explicar cómo poder construir un modelo de lenguaje basado en n-gramas, y cómo poder evaluar que funciona correctamente.
- ▶ Identificar las limitaciones del modelado del lenguaje en base a n-gramas, y cómo se pueden solucionar algunas de ellas con técnicas de suavizado.
- ▶ Entender qué son los modelos de representación vectorial de los textos, para qué sirven, y cómo se pueden construir mediante el modelado con bolsas de palabras.

## 7.2. Introducción al modelo del lenguaje

Si se tiene una frase como:

- El alumno entregó al profesor...

Donde el final de ella no es conocido, se puede intentar predecir que podría seguir en base al conocimiento que se tiene *a priori* del lenguaje. Así, una persona que leyese esto podría pensar que esa frase podría estar seguida de cosas como «el examen» o «el ejercicio» más que de otras como «las llaves del coche» o «la ropa de deporte». Esto no quiere decir que estas últimas opciones no puedan aparecer en un determinado texto, sino que cuando no se tiene más información, en muchas ocasiones se acude a lo que es más probable.

Este mismo concepto es el que tratan de resolver los **modelos de lenguaje** (LM, Language Model): teniendo un **conocimiento estadístico de la lengua**, ven qué probabilidades hay en una secuencia de que continúe con un determinado elemento u otro. En general, con secuencia se hace referencia a secuencias de palabras, y con elementos a las palabras que la forman, de manera que se obtiene la probabilidad de que una determinada palabra siga a la secuencia previa de palabras

Un modelo de lenguaje es una distribución de probabilidad sobre las palabras de una secuencia, donde estas probabilidades se obtienen de entrenar el modelo en uno o varios corpus de la lengua.

Los **LM** son útiles para distintas tareas de **PLN**. Un ejemplo de ello es en el caso del **reconocimiento de voz**. Por ejemplo, supongamos el caso de un sistema de reconocimiento de voz que recibe un mensaje que puede corresponder a las siguientes frases:

- ▶ Pedro tiene mucho pelo y se quiere depilar-
- ▶ Pedro tiene mucho pelo y se quiere de Pilar-

En estos casos aparece **una ambigüedad fonética**, de manera que el sistema de reconocimiento de voz puede convertir ese audio en dos construcciones de texto distintas. Ahora bien, gracias a los LM, es más probable que «depilar» siga a la secuencia de texto previa que de «Pilar».

Los LM también son útiles para tareas como la **traducción automática**. Supongamos que queremos traducir la siguiente frase de inglés a castellano:

- ▶ *He is my friend Stephen, let me introduce him to you.*

El verbo «introduce» puede hacer referencia a presentar o a «introducir», de manera que dos posibles traducciones serían:

- ▶ Él es mi amigo Stephen, déjame que te lo presente.
- ▶ Él es mi amigo Stephen, déjame que te lo introduzca.

Con el conocimiento que se tiene en el LM, se puede detectar que dada la secuencia previa a la palabra introduce, es más probable que se tenga la primera secuencia que la segunda. No sólo eso, sino que como se puede ver, la secuencia de palabras traducida al castellano no sigue el orden textual de la secuencia de palabras en inglés. De esta manera, es más probable tener una construcción en castellano como «Él es mi amigo Stephen, déjame que te lo presente» que una más textual como «Él es mi amigo Stephen, déjame que lo presente a ti».

### 7.3. Modelos estadísticos del lenguaje basados en N-gramas

Supongamos que se tiene una frase como:

- La playa está tan tranquila.

Y se quiere obtener la probabilidad de que la palabra «que» siga a esa secuencia. Esto, a nivel formal, se expresaría como la probabilidad de tener «que» **condicionada** a tener la secuencia «la playa está tan tranquila»:

$$P(\text{que} \mid \text{la playa está tan tranquila})$$

El cálculo de esta probabilidad se puede obtener de manera similar a cómo se obtenía la **probabilidad de transición** para el caso del POS tagging:

$$P(\text{que} \mid \text{la playa está tan tranquila}) = \frac{C(\text{la playa está tan tranquila que})}{C(\text{la playa está tan tranquila})}$$

Es decir, que la probabilidad se obtiene como el número de veces que aparece que después de esa secuencia dividido entre el número de veces que se tiene esa secuencia. Estos valores se obtienen en base a un análisis previo de uno o varios corpus de la lengua.

No obstante, si se consideran secuencias muy largas no será habitual encontrarlas dentro del corpus. Las lenguas son a menudo muy flexibles, y pueden expresar un mismo tema con palabras distintas o con órdenes de la secuencia distintos. La misma frase de antes, si fuese «la playa estuvo tan tranquila que» sería una secuencia distinta a todos los efectos bajo esta aproximación.

Podríamos también calcular la probabilidad de tener esa secuencia mediante el uso de la **regla de la cadena**, donde la probabilidad de cada palabra de la secuencia se va obteniendo de manera recursiva en función de las probabilidades condicionadas previas:

$$P(w_i:n) = P(w_1) P(w_2|w_1) P(w_3|w_{1:2}) \dots P(w_n|w_{1:n-1}) = \prod_{k=1}^n P(w_k \vee w_{1:k-1})$$

La cuestión es que a medida que vaya creciendo la secuencia, esta expresión irá también creciendo. Sin embargo, en muchas ocasiones es suficiente con definir un tamaño de histórico previo concreto y calcular las probabilidades sólo dentro de esa subsecuencia. Esto es lo que se hace con el **modelo de n-gramas**, donde la probabilidad condicionada de una determinada palabra se calcula sólo con respecto a las **n palabras previas**. Un caso concreto es el modelo de bigramas, donde la probabilidad de una palabra dada la secuencia previa se aproxima a la probabilidad de tener esa palabra dada únicamente la palabra anterior:

$$P(w_k|w_{1:k-1}) \approx P(w_k \vee w_{k-1})$$

El caso de los bigramas está relacionado con la **hipótesis de Markov**, ya vista en el tema de POS tagging, donde se trabaja con modelos probabilísticos de Markov que asumen que la probabilidad de obtener un elemento concreto de una secuencia sólo depende del elemento inmediatamente anterior. Por tanto, la probabilidad de tener toda una secuencia concreta usando bigramas sería:

$$P(w_i:n) \approx \prod_{k=1}^n P(w_k \vee w_{k-1})$$

Este concepto de bigramas se puede generalizar a otros n-gramas, como los trigramas, con los que ya no se considera sólo la palabra previa, sino que se amplía para considerar las dos palabras anteriores:

$$P(w_k | w_{1:k-1}) \quad P(w_k \vee w_{k-2} : w_{k-1})$$

Así, esta expresión se generalizaría para un n-grama de tamaño N de la siguiente manera:

$$P(w_k | w_{1:k-1}) \quad P(w_k \vee w_{k-N+1} : w_{k-1})$$

Las probabilidades condicionadas se calculan utilizando el **método de máxima verosimilitud**, igual que se hizo en el tema de *POS tagging*, dividiendo el número de veces que aparece la secuencia previa considerada más la palabra en concreto entre el número de veces totales que aparece esa secuencia. A modo de ejemplo, para el caso de bigramas sería:

$$P(w_k | w_{k-1}) = \frac{C(w_{k-1}, w_k)}{C(w_{k-1})}$$

Y generalizándola para un caso de un n-grama cualquiera:

$$P(w_k | w_{k-N+1:k-1}) = \frac{C(w_{k-N+1:k-1}, w_k)}{C(w_{k-N+1:k-1})}$$

Un modelo de lenguaje basado en n-gramas predice la probabilidad de que tenga lugar un determinado n-grama dentro de una secuencia de palabras de la lengua.

## Evaluación de los modelos: perplejidad

De cara a la evaluación de los LM se plantean dos opciones. En primer lugar, se puede evaluar cómo funcionan a nivel de la aplicación de PLN para la que se estén utilizando. Por ejemplo, si se usa el LM dentro de un sistema de reconocimiento de voz, se podría evaluar si el rendimiento mejora al utilizar dicho LM. Este tipo de evaluaciones se denomina **evaluación extrínseca**. La parte negativa es que el coste de ejecutar todo un sistema de PLN para hacer evaluaciones puede ser muy costoso.

Una evaluación extrínseca de un modelo de lenguaje consiste en evaluarlo desde el análisis de la salida que tiene la aplicación de PLN que lo usa.

Por este motivo, existen también otro tipo de evaluaciones denominadas **intrínsecas**, en las que se evalúan los resultados del LM con respecto a un conjunto de datos de referencia. Este conjunto de datos de referencia se puede obtener directamente del corpus que se fuese a usar para entrenar el LM, haciendo una separación en datos de **entrenamiento/test** como se hace con los modelos de aprendizaje automático.

El conjunto de datos de entrenamiento se usaría para ajustar las probabilidades de las secuencias en el LM, y el conjunto de datos de test para su evaluación. Comparando dos LM, darían mejores resultados el que diese una **mayor probabilidad** a las secuencias correctas. El conjunto de datos original también se podría separar en **entrenamiento/validación/test** si se quiere tener un subconjunto para hacer validaciones iniciales y ajustar parámetros (por ejemplo, decidir si usar bigramas o trigramas para construir el LM).

Una evaluación intrínseca de un modelo de lenguaje consiste en evaluar sus resultados con respecto a un conjunto de datos de referencia (test).

La evaluación intrínseca en base a la máxima probabilidad se hace mediante el cálculo de una métrica denominada **perplejidad** (*perplexity*, PP). Esta métrica calcula la probabilidad inversa sobre los datos de test, normalizada en función del número de palabras. Por ejemplo, para un conjunto de palabras  $W = w_1, w_2, \dots, w_N$  sería:

$$PP(W) = P(w_1, w_2, \dots, w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1, w_2, \dots, w_N)}}$$

Aplicando la regla de la cadena para expandir el denominador de las probabilidades se tiene:

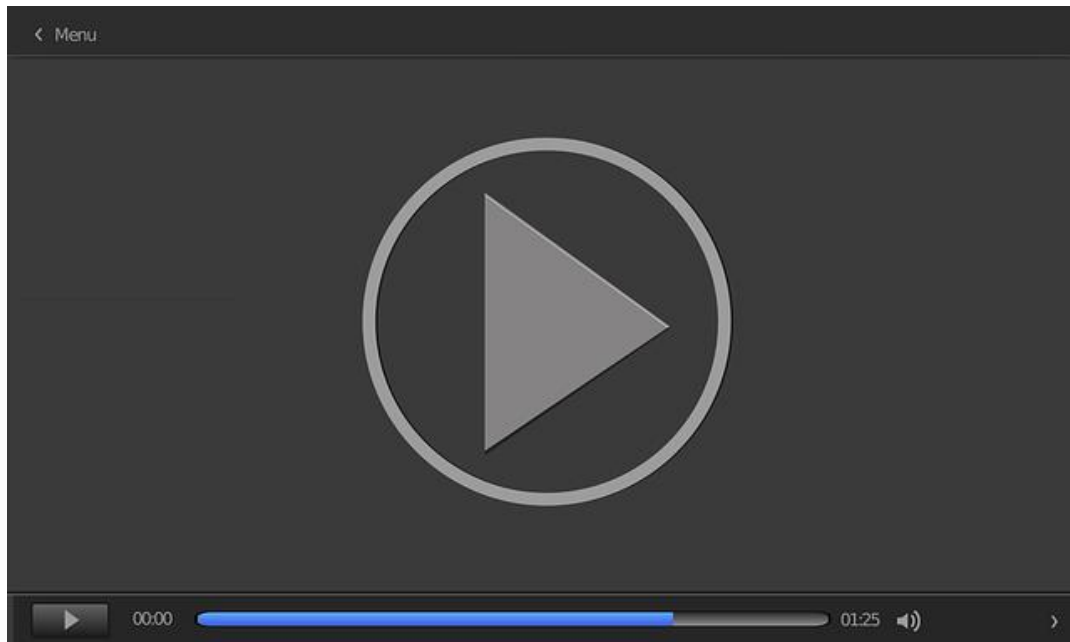
$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1, \dots, w_{i-1})}}$$

A modo de ejemplo cuando se trabaja con bigramas, la fórmula sería:

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}}$$

Ya que la métrica PP se calcula en base al valor inverso de las probabilidades, cuanto mayor probabilidad tenga la secuencia, menor será el valor de PP. Por este motivo, el LM será mejor cuanto **más bajo** sea el valor de PP.

En el vídeo *Modelos estadísticos del lenguaje* se verá qué es un modelo de lenguaje y cómo se pueden construir utilizando técnicas estadísticas.



07.01. Modelos estadísticos del lenguaje

---

Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=9876d22c-ae59-4073-b380-af5b0149ce20>

---

### Generalización y dispersión en los modelos

Como cualquier modelo estadístico, los LM basados en n-gramas dependen en gran medida del conjunto de datos usado para el entrenamiento, ya que las probabilidades de las subsecuencias estarán calculadas en base a lo que se ha visto en ese subconjunto de datos. Por este motivo, es importante tener en cuenta el tipo de texto (o textos) que se han usado como corpus de entrenamiento del modelo.

Si, por ejemplo, el LM está generado sobre textos de Shakespeare, las secuencias que se podrán generar con ese LM serán distintas de las que se podrían generar con un LM entrenado sobre textos de noticias, a pesar de que haya n-gramas que puedan ocurrir en ambos conjuntos de textos.

Así, el género de los textos es importante y se debe tener en cuenta cuando se crean LM o se usa un LM ya existente.

No sólo eso, sino que también ocurrirá que los n-gramas de noticias obtenidos a través de *tweets* serán también distintos respecto a los que se podrían obtener de las noticias de un periódico.

Por lo que, además del género, hay que tener en cuenta la variedad o el dialecto de los textos utilizados.

No obstante, aun considerando un mismo género y variedad/dialecto, el LM estará **siempre limitado al conjunto de datos de entrenamiento**, de manera que puede que haya secuencias válidas en el lenguaje que nunca se generarían por no aparecer en esos datos de entrenamiento. Este es el problema de la dispersión, y se acrecienta cuanto más grandes son los n-gramas, ya que se limitarán más las posibles secuencias que se pueden generar con el LM.

Cuanto mayor sea el n-grama, las secuencias que se pueden generar serán más fieles al texto original, pero precisamente por esto, será también más fácil que haya secuencias válidas que no estén contempladas en el LM.

1 gram	-To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have -Hill he late speaks; or! a more to leg less first you enter
2 gram	-Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. -What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	-Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. -This shall forbid it should be branded, if renown made it empty.
4 gram	-King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; -It cannot be but so.

Figura 1. Ejemplo de secuencias generadas con un LM entrenado en textos de Shakespeare. Fuente: Jurafsky y Martin, 2020.

Cuanto mayor es el n-grama, los textos son más coherentes y fieles a la fuente original. Sin embargo, también será más fácil que haya secuencias válidas en el lenguaje que no estén recogidas en el LM.

El problema de las secuencias válidas no contempladas en los datos de entrenamiento tiene también impacto en el cálculo de la PP, ya que, si hay secuencias en los datos de test que no están en los de entrenamiento, no se podría calcular como tal la PP con la fórmula vista previamente ya que se estaría dividiendo por 0.

## Palabras desconocidas

Además de las problemáticas anteriores, un LM va a estar limitado a las palabras que haya en el conjunto de datos de entrenamiento, de manera que se tiene que ver cómo lidiar con palabras que no estén en él.

Estas palabras se denominan palabras desconocidas (*unknown words*) o palabras fuera del vocabulario (*out of vocabulary, OOV*). En relación a ellas, una métrica de evaluación adicional consiste en calcular el ratio de OOV, que se obtiene calculando el porcentaje de palabras del conjunto de test que son OOV.

La forma de trabajar con ellas en un LM es mediante el uso de un token genérico denominado **UNK** (*unknown*), que sirve para considerar a través de él todas las palabras OOV. Los pasos para utilizarlo son:

- ▶ Usar una lista inicial de palabras fija.
- ▶ Antes de entrenar el LM, convertir todas las palabras del conjunto de datos de entrenamiento que no están en esa lista fija en el token UNK.
- ▶ Obtener las probabilidades del token UNK como si fuese cualquier otro token, de manera que con ello se modelaría la aparición de palabras desconocidas.

Si no se tuviese una lista inicial de palabras, esta se podría obtener directamente desde el conjunto de datos de entrenamiento, reemplazando algunas de las palabras por ese token UNK (por ejemplo, reemplazando las que tengan una baja frecuencia en el corpus por aparecer poco; por ejemplo, las que aparezcan menos de  $n$  veces).

## 7.4. Técnicas de suavizado

Además de palabras que sean OOV, ocurre también que pueden aparecer palabras en el conjunto de datos de test que, aunque existen en el de entrenamiento, aparecen en **contextos distintos**. Es decir, que aparecen, por ejemplo, tenemos «gran perro» en el conjunto de datos de test cuando en el de entrenamiento sólo se tenía «pequeño perro». Para evitar que se dé directamente un valor de probabilidad 0 a estos eventos (que es lo que ocurriría con las técnicas vistas hasta ahora), **se aplican técnicas denominadas de suavizado** (*smoothing*), con las que se busca **asignar un pequeño valor de probabilidad** para esos eventos no vistos, de manera que sus probabilidades no sean 0 y, así, sea posible generar esas secuencias con el LM.

### Suavizado de Laplace

Una de las técnicas más sencillas para hacer suavizado es el **suavizado de Laplace**, donde se analizan todas las combinaciones posibles de n-gramas que se pueden dar con los datos de entrenamiento (no sólo las que ocurren como tal). Con ello, se aumentan en 1 todas las ocurrencias de n-gramas, de manera que los que no ocurren en los datos de entrenamiento (y que tendrían, por tanto, una probabilidad de 0 al dar 0 la cuenta de cuantas veces ocurre) tendrían una probabilidad por defecto distinta de 0, aunque muy baja.

Esto se ve a modo de ejemplo en la Figura 2 y 3, donde se tiene una matriz con los bigramas posibles que se obtendrían sobre un corpus de entrenamiento de ejemplo, con un valor 0 en los bigramas que no ocurren. Con el suavizado de Laplace se incrementan todos los valores en 1, con lo que los valores que eran 0 pasan a valer 1.

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figura 2. Ejemplo de combinaciones de n-gramas (bigramas) antes (arriba) y después (abajo) de aplicar el suavizado de Laplace. Fuente: Jurafsky y Martin, 2020.

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Figura 3. Ejemplo de combinaciones de n-gramas (bigramas) antes (arriba) y después (abajo) de aplicar el suavizado de Laplace (continuación). Fuente: Jurafsky y Martin, 2020.

Como se puede ver, todas las probabilidades que eran 0 porque no se daban esos bigramas, ahora tienen un valor de 1. Las demás probabilidades también aumentan en 1.

Una vez se tiene esto se recalcularían las probabilidades de los n-gramas. A modo de ejemplo, para el caso de unigramas sería:

$$P(w_i) = \frac{c_i}{N} \rightarrow P_{\mathcal{L}}(w_i) = \frac{c_i + 1}{N + V}$$

De esta manera, para cada palabra se incrementaría en 1 el valor del número de veces que aparece, y se dividiría entre la suma del número de veces que aparecen todas las palabras originalmente (N) más el número de palabras que hay en el vocabulario (V), ya que se está incrementando en 1 el número de veces que aparece cada una de ellas.

De manera análoga, para el caso de bigramas sería:

$$P(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n)}{C(w_{n-1})} \rightarrow$$

$$P_{\mathcal{L}}(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n) + 1}{\sum_w (C(w_{n-1}, w) + 1)} = \frac{C(w_{n-1}, w_n) + 1}{C(w_{n-1}) + V}$$

A pesar de que con este algoritmo simplemente, se incrementan en 1 el número de veces que aparecen todas las combinaciones de n-gramas en el corpus, la consecuencia es que debido a que esto afecta a todas las probabilidades (no sólo a las de los n-gramas que no aparecen) la contribución de los n-gramas que sí existen en el corpus para el cálculo de las probabilidades finales sea menor. Por ejemplo, para el caso de un unigrama:

$$P_{\mathcal{L}}(w_i) = \frac{c_i + 1}{N + V} = \frac{c_i}{N} \rightarrow c_i = (c_i + 1) \frac{N}{N + V}$$

De esta manera se puede ver cómo va a cambiar el peso que tiene un determinado n-grama en el cálculo de las probabilidades finales. Así, al aplicar la técnica de suavizado, es como si cambiase el número de veces

que aparecía el n-grama originalmente ( $c_i$ ) para pasar tener otro valor ( $\hat{c}_i$ ).

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Figura 4. Ejemplo del valor reconstruido de  $\hat{c}_i$  ( $c_i$ ) tras haber usado el suavizado de Laplace. Fuente: Jurafsky y Martin, 2020.

Propiamente, con las técnicas de suavizado va a tener lugar un descuento en el valor que se va a usar en el cálculo de las probabilidades de un determinado n-grama.

Este descuento se expresa en la siguiente fórmula.

$$d_c = \frac{c}{c}$$

Esto se puede extender para el caso de otros n-gramas. A modo de ejemplo y para un bigrama sería:

$$c(w_{n-1}, w_n) = \frac{C(w_{n-1}, w_n) + 1 \vee \times C(w_{n-1})}{C(w_{n-1}) + V}$$

## Suavizado Add-k

La idea del suavizado de Laplace se puede generalizar para que, en lugar de incrementar el valor del número de veces que aparece una palabra en 1, se incremente en un valor  $k$  que se especifique. Esto es lo que se hace con el algoritmo **suavizado add-k** (*add-k smoothing*). Para el caso de bigramas, sería de la siguiente manera:

$$P_{Add-k}(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n) + k}{C(w_{n-1}) + kV}$$

En este algoritmo se tiene que especificar a priori ese valor de  $k$ . Esto es algo que también se puede considerar como un **hiperparámetro** a definir tras seguir un proceso de optimización de los resultados sobre los **datos de validación**.

## Backoff e interpolación

Los algoritmos vistos hasta ahora ayudan a resolver el problema de las combinaciones de n-gramas con frecuencia 0. Ahora bien, otra consideración que se puede hacer es que cuando no se tengan casos de un

determinado n-grama, se estime esa probabilidad con un n-grama más pequeño. Esto es lo que se lleva a cabo con el **algoritmo de backoff**.

Por ejemplo, trabajando con trigramas, si se quiere calcular  $P(w_n | w_{n-2} w_{n-1})$  y no se tiene el trigramo  $w_{n-2} w_{n-1} w_n$ , se puede estimar esa probabilidad usando el bigrama  $P(w_n | w_{n-1})$ . De igual manera, si no se tienen casos para obtener  $P(w_n | w_{n-1})$ , se puede estimar con la del unigrama  $P(w_n)$ . Así, este algoritmo se va aplicando de manera recursiva sobre los n-gramas que no aparecen hasta encontrar uno de menor orden que sí esté.

El algoritmo de *backoff* se usa para estimar las probabilidades de los n-gramas que no aparecen en los datos de entrenamiento con las probabilidades de los n-gramas de menor orden.

Frente al algoritmo de *backoff*, existe la alternativa del **algoritmo de interpolación lineal**, con el que la probabilidad para un determinado n-grama se obtiene de hacer una suma ponderada de la probabilidad de ese n-grama junto con las de sus n-gramas de menor orden. Para un ejemplo de un trigramo sería:

$$P_{inter}(w_n | w_{n-2} w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n | w_{n-1}) + \lambda_3 P(w_n | w_{n-2} w_{n-1})$$

Con  $\sum_i \lambda_i = 1$

Estos valores  $\lambda_i$  se pueden considerar también como **hiperparámetros** a estimar frente a los **datos de validación**.

Los valores  $\lambda_i$  se pueden calcular teniendo en cuenta la información de contexto. Si para el cálculo de las probabilidades de un determinado n-grama se tienen muchos casos de n-gramas de menor orden en el corpus, esto quede reflejado en sus valores  $\lambda_i$  para que la probabilidad final sea mayor respecto a otros casos en los que se tengan menos casos de n-gramas de menor orden. Formalizando esto para el caso de los trigramas se tendría:

$$P_{inter}(w_n | w_{n-2} w_{n-1}) = \lambda_1(w_{n-2:n-1}) P(w_n) + \lambda_2(w_{n-2:n-1}) P(w_n | w_{n-1}) + \lambda_3(w_{n-2:n-1}) P(w_n | w_{n-2} w_{n-1})$$

Una última variación del algoritmo de *backoff* es el algoritmo **Katz backoff**, donde se usan las probabilidades originales cuando ese n-grama existe en el corpus de entrenamiento, y en caso de que no exista, se busca el primer n-grama de menor orden para el que sí se tengan datos, ponderando ese valor en función de un parámetro (como en el caso de la interpolación) que dependerá de la información de contexto, como en el caso de la interpolación ya mencionado.

Suavizado de Kneser-Ney

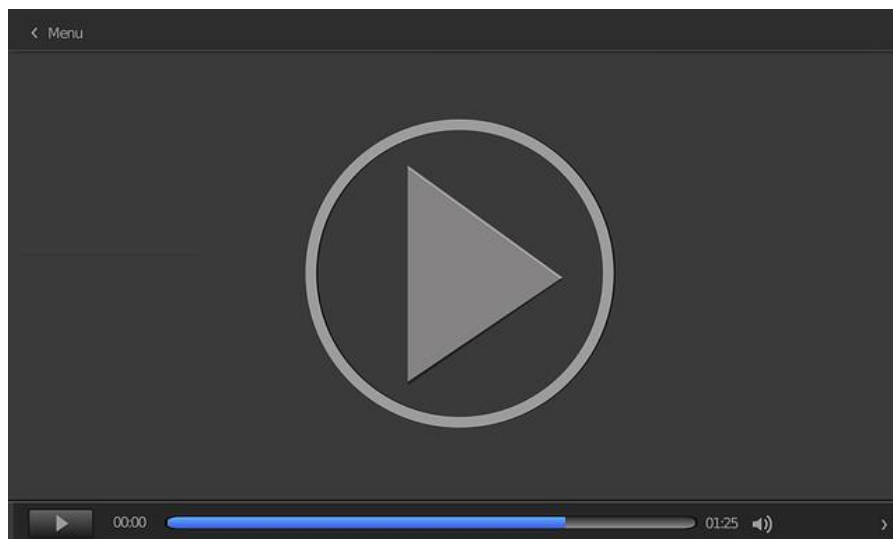
Además de los algoritmos vistos previamente, existen soluciones más avanzadas y efectivas para hacer el suavizado, como es el caso del suavizado de Kneser-Key.

---

Para completar el estudio de esta sección puedes leer las **páginas 17-20** del Capítulo 3 del libro: Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall. Disponible en: <https://web.stanford.edu/~jurafsky/slp3/3.pdf>

---

En el vídeo *Técnicas de suavizado* se verá qué son y cómo aplicar técnicas de suavizado para la construcción de modelos del lenguaje.



07.02. Técnicas de suavizado

---

Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=ec29c5b6-ac76-4bf1-a588-af5b0149cde2>

---

## 7.5. Modelos de representación vectorial

Como se ha visto previamente, con los modelos de lenguaje se modela la probabilidad de que se tengan secuencias concretas de palabras en base a un corpus previo de entrenamiento, y esto luego es útil para distintas tareas de PLN. De esta manera, partiendo de los textos de un corpus, se hace un modelado estadístico para poder trabajar con esos textos a nivel computacional y usarlos dentro de esas aplicaciones de PLN.

Ahora bien, existen otras maneras de representar el contenido de un texto para poder trabajar con él a nivel computacional para llevar a cabo distintas tareas de NLP. Este es el caso de los **modelos de representación vectorial**, donde los textos se representan como vectores dentro de un espacio vectorial en base a la información que se extrae de las palabras que aparecen en ellos. Con esto se busca que dos textos similares tengan representaciones vectoriales también similares, por lo que esta aproximación es útil para tareas como la **recuperación de información**, donde se puede realizar una consulta en formato texto y se busca encontrar documentos similares a ella:

$$d_j = (w_{1j}, w_{2j}, \dots, w_{tj})$$

$$q = (w_{1q}, w_{2q}, \dots, w_{nq})$$

Como podemos ver en las ecuaciones anteriores, cada documento  $d_j$  se representaría por un vector obtenido en base a las palabras que aparecen en el texto, y análogamente se representaría la consulta  $q$ . Al ser vectores dentro de un mismo espacio vectorial, los documentos con un contenido similar al de la consulta deberían tener vectores similares.

Los modelos de representación vectorial son representaciones algebraicas de los textos o documentos, de manera que cada uno de ellos se expresa mediante un vector que recoge su información. Con ello, documentos o textos similares tendrán representaciones también similares.

La cuestión es cómo construir esos vectores que modelen el contenido de los textos. Existen muchas aproximaciones para hacerlo, y en este tema presentaremos algunas de las más sencillas. Ahora bien, hoy en día existen representaciones vectoriales más complejas basadas en redes neuronales, que se verán en detalle en el siguiente capítulo.

Una de las aproximaciones más sencillas para representar vectorialmente el contenido de texto es mediante lo que se denomina **bolsa de palabras (Bag of Words, BoW)**. Con esta aproximación, se consideran las palabras del texto de manera aislada (unigramas), sin considerar información en relación con el orden de estas en el texto. De esta manera, partiendo de un corpus de varios textos, esta técnica daría como salida una matriz donde cada fila representa uno de esos textos, y cada columna representa cada una de las palabras que aparecen en el corpus.

A modo de ejemplo, si se tiene un corpus con los siguientes textos:

$$\begin{cases} d_1 = \textit{La ingeniera ha viajado en verano} \\ d_2 = \textit{El abogado ha trabajado en verano} \end{cases}$$

Tras aplicar las técnicas de normalización (tokenización, eliminado de *stopwords*, eliminado de mayúsculas y lematizado) quedaría:

$$\begin{bmatrix} d_1 & \textit{ingeniero} & \textit{viajar} & \textit{verano} & \textit{abogado} & \textit{trabajar} \\ d_2 & x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ & x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \end{bmatrix}$$

Cada documento, representado por un vector con varias componentes, se puede ver también desde la perspectiva de las matrices de rasgos que se usan en aprendizaje automático para entrenar los modelos. Así, cada componente sería un rasgo o variable, pudiéndose usar directamente estas matrices para entrenar los modelos. Esto puede utilizarse para, por ejemplo, tareas de análisis de sentimiento, donde las variables de entrada de un modelo supervisado son precisamente los elementos de la matriz BOW, y la variable de salida sería la categoría del sentimiento (identificada previamente) asociada a cada uno de esos documentos. Con estas representaciones se tiene, por tanto, un vector por texto de una dimensionalidad  $N$  donde  $N$  es el número de palabras del vocabulario del corpus. Por este motivo es importante también normalizar antes los textos de cara a reducir la dimensionalidad de los vectores.

Dentro del esquema de BoW, los valores que aparecen en la matriz se pueden calcular de distinta manera. Existen dos aproximaciones, las **locales** y las **globales**. En el caso de las locales, los valores de las componentes del vector que representa cada texto se calculan en base a sólo la información de ese texto. En el caso de las representaciones globales, las componentes del vector se construyen usando información tanto del texto en sí como del resto de textos del corpus.

### Representaciones locales

Una de las representaciones más sencillas dentro de las representaciones locales es la **binaria**, donde cada componente toma un valor binario (1 o 0) dependiendo de si esa palabra aparece o no en el texto en cuestión.

$$Bin(t_i, d_j) = \begin{cases} 1 & \text{si el token } t_i \text{ aparece en el documento } d_j \\ 0 & \text{en otro caso} \end{cases}$$

Uno de los problemas de las representaciones binarias es que se ponderan por igual todas las palabras, independientemente de que aparezcan mucho o poco en un determinado texto. Por este motivo, existen otras representaciones, como las de **frecuencia de términos** (Term Frequency, TF) donde en lugar de usar valores binarios, se cuenta el número de veces que aparece la palabra en el texto en cuestión.

$$TF(t_i, d_j) = f_{ij}, \text{ siendo la frecuencia del token } t_i \text{ en } d_j$$

Ahora bien, la representación en base a frecuencias de términos también tiene sus problemas, ya que puede que no todos los textos tengan el mismo número de palabras, y por tanto, se dé más importancia a los textos más largos frente a los cortos. Para solucionar eso, se usa también la técnica **frecuencia de términos ponderada** (Weighted Term Frequency, WTF) donde se normalizan los valores de las frecuencias en base a la suma de las frecuencias dentro del documento.

$$WTF(t_i, d_j) = \frac{f_{ij}}{\sum_{t_p \in d_j} f_{pj}}$$

### Representaciones globales

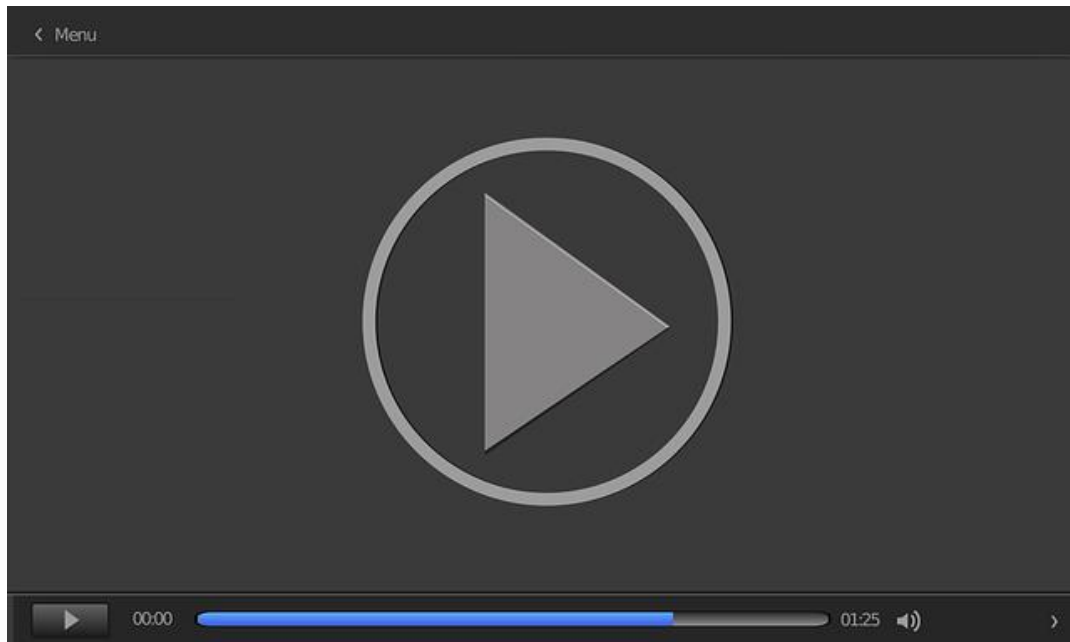
Junto a las representaciones locales aparecen también las **globales**, donde la información de los componentes del vector **usa sólo información del propio texto**, se tienen también las globales donde se usa información de todo el corpus. Un ejemplo es la **Frecuencia de términos-Frecuencia inversa del documento** (Term Frequency–Inverse Document Frequency, TF-IDF).

Una de las utilidades de construir las componentes de los vectores con TF-IDF es la siguiente. Con las técnicas vistas previamente, como TF, se puede acabar dando importancia a palabras poco representativas que aparecen mucho en cualquier tipo de texto. Es verdad que esto ocurre principalmente con palabras como determinantes, conjunciones que se eliminarían al quitar las *stopwords*. Sin embargo, puede haber otras palabras que no sean *stopwords* y que no sean representativas por aparecer mucho. Por ejemplo, teniendo textos de sinopsis de películas, la palabra «película» podría aparecer mucho en todos ellos, pero no es representativa para modelar el contenido de esos textos. Esto precisamente se soluciona con TF-IDF, donde el valor de la frecuencia (TF) se pondera en función de la rareza de las palabras dentro del corpus de la siguiente manera:

$$TFIDF(t_i, d_j) = f_{ij} \times \log \left( \frac{N}{df(t_i)} \right)$$

Donde  $f_{ij}$  es la frecuencia del token  $t_i$  en el documento (o texto)  $d_j$ ,  $N$  el número de documentos en el corpus, y  $df(t_i)$  el número de documentos en el corpus donde aparece el token  $t_i$ .

En el vídeo *Modelos de representación vectorial* se revisarán las distintas técnicas de representación de textos basadas en modelos vectoriales.



07.03. Modelos de representación vectorial

---

Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=d6aec96f-2d71-4787-a9b9-af5b0149cdb5>

---

## 7.6. Referencias bibliográficas

Aggarwal, C. (2018). *Machine Learning for Text*. Springer Cham.

Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. New Jersey (Estados Unidos): Prentice-Hall.

Manning, C. y Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

## Modelos de Lenguaje

Jurafsky, D. y Martin, J. H. (2021) . *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall. <https://web.stanford.edu/~jurafsky/slp3/14.pdf>

El capítulo 3 describe en detalle todo lo relacionado con el modelado de lenguaje mediante n-gramas, junto con las técnicas de suavizado que se han visto en este tema además de alguna otra adicional.

1. Indica cuál de estas afirmaciones es verdadera sobre los modelos de lenguaje:
  - A. Se pueden usar en aplicaciones de PLN como por ejemplo para reconocimiento de voz o traducción automática.
  - B. No necesitan ningún dato de partida, sino que generan las secuencias más probables sin ningún conocimiento previo de la lengua.
  - C. Permiten modelar sólo secuencias de texto cortas, pero no largas.
  - D. Todas las anteriores.
  
2. Si el número de veces que aparece «el gato es» en un corpus es 100, y el número de veces que aparece «el gato es grande» es 20, la probabilidad condicionada para la palabra «grande» dada la secuencia previa «el gato es» es
  - A. 0.1.
  - B. 0.2.
  - C. 0.3.
  - D. 0.4.
  
3. Indica cuál de estas afirmaciones es verdadera sobre la evaluación de los modelos del lenguaje:
  - A. Un ejemplo de evaluación extrínseca es la métrica de perplejidad.
  - B. Un ejemplo de evaluación intrínseca es el análisis del modelo del lenguaje desde los resultados que tiene al usarse dentro de una aplicación de PLN, como por ejemplo una de traducción automática.
  - C. Cuanto más alto sea el valor de perplejidad, mejor será el modelo de lenguaje.
  - D. Ninguna de las anteriores.

4. Indica cuál de estas afirmaciones es verdadera sobre los modelos de lenguaje:
- A. Si se usan n-gramas de un orden alto (ej.: trigramas o superior), es más difícil que el modelo genere secuencias de texto fieles a las del corpus original.
  - B. Si se usan n-gramas de bajo orden (ej.: unigramas), es más fácil que el modelo genere secuencias de texto fieles a las del corpus original.
  - C. Si se usan n-gramas de bajo orden (ej. unigramas), es más difícil que el modelo genere secuencias de texto fieles a las del corpus original.
  - D. Ninguna de las anteriores.
5. ¿Qué pasos se pueden seguir para tener un modelo de lenguaje que sea capaz de trabajar con palabras desconocidas?
- A. 1) Usar una lista de palabras abierta que pueda cambiar. 2) Convertir las palabras del conjunto de test que no estén en la lista a UNK. 3) Obtener las probabilidades de esos tokens UNK.
  - B. 1) Usar una lista de palabras cerrada. 2) Convertir las palabras del conjunto de entrenamiento que no estén en la lista a UNK. 3) Obtener las probabilidades de esos tokens UNK.
  - C. 1) Usar una lista de palabras cerrada. 2) Convertir las palabras del conjunto de test que no estén en la lista a UNK. 3) Obtener las probabilidades de esos tokens UNK.
  - D. Ninguna de las anteriores.

6. Dado un vocabulario de 250 palabras, si la palabra «buen» aparece cien veces, y si la secuencia «buen amigo» aparece veinte veces, ¿cuál es la probabilidad  $P(\text{amigo} \mid \text{buen})$  considerando el suavizado de Laplace?

- A. 0.04.
- B. 0.05.
- C. 0.06.
- D. 0.08.

7. Dado un vocabulario de 250 palabras, si la palabra «buen» aparece 100 veces, y si la secuencia «buen amigo» aparece veinte veces, ¿cuál es la probabilidad  $P(\text{amigo} \mid \text{buen})$  considerando el suavizado add-k con  $k=5$ ?

- A. 0.001.
- B. 0.019.
- C. 0.051.
- D. 0.133.

8. Dado un vocabulario de 250 palabras, con una suma de frecuencias totales de 10 000, si la palabra «buen» aparece cien veces, y si no se tiene la secuencia «buen amigo», ¿qué probabilidad tendría asignado ese bigrama usando el suavizado de *backoff*?

- A. 0.01.
- B. 0.02.
- C. 0.3.
- D. 0.4.

9. Indica cuál de estas afirmaciones es verdadera sobre los modelos de representación vectorial:

- A. Son representaciones algebraicas de textos o documentos.
- B. Representan los textos o documentos en un espacio vectorial, de manera que dos textos similares tengan representaciones similares.
- C. A y B son correctas.
- D. Ninguna de las anteriores.

10. Indica cuál de estas afirmaciones es verdadera respecto a los modelos de representación vectorial basados en BoW:

- A. Se caracterizan por conservar la información relativa al orden de las palabras de los textos.
- B. Se caracterizan por conservar la información relativa a la sintaxis de los textos
- C. Se caracterizan por usar sólo representaciones binarias para indicar las palabras que aparecen en un determinado texto.
- D. Ninguna de las anteriores.