Análisis de Textos

Grupo PLN – InCo
2024

Clasificación aplicando aprendizaje automático supervisado

Clasificación con aprendizaje automático

Tenemos ejemplos anotados (conjunto de entrenamiento)

- $D = \{ \langle d,c \rangle / \langle d,c \rangle \in X \times C \}$
  - X es el espacio de instancias
  - C es el conjunto de clases
- Una función de clasificación mapea documentos a clases:

$$Y: X \rightarrow C$$

- Un método de aprendizaje supervisado recibe los ejemplos anotados y devuelve la función de clasificación

Clasificación con aprendizaje automático

- Qué es el conjunto X de instancias?
  - Tweets
  - Documentos
  - ...
- Transformar las instancias en una representación que sirva para el procesamiento:
  - Modelo X como vector de atributos (*features*):

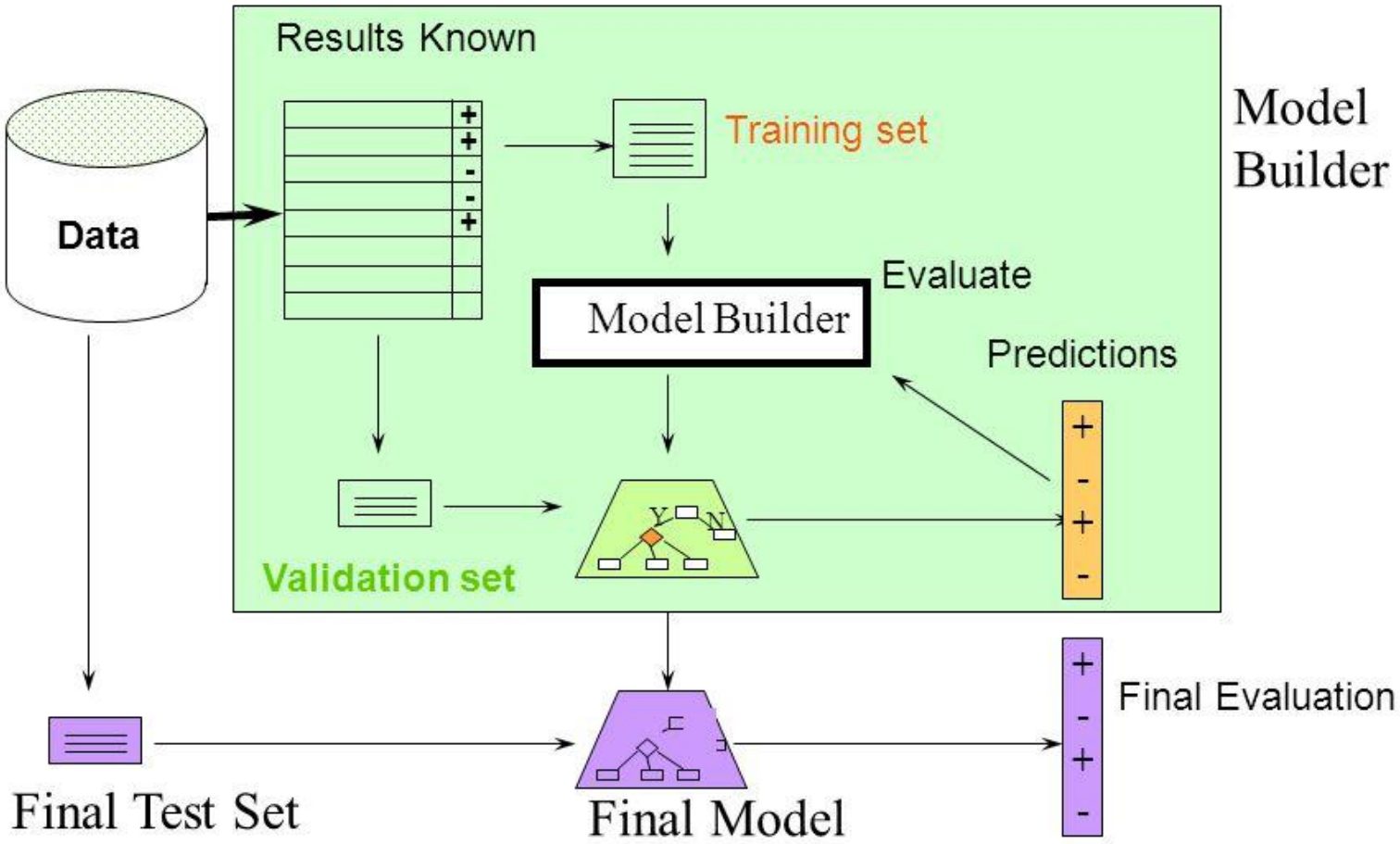
    $X = [f1,f2,...,fn]$

  - Habitualmente las herramientas de aprendizaje automático requieren $X \in Rn$

Clasificación con aprendizaje automático

- Existen varios métodos para “aprender” la función:
  - Naïve Bayes, SVM, Regresión Logística, Árboles de Decisión...
- Una vez que tengo la función, quiero estimar qué tan buena es:
  - Debo estimar sobre datos no vistos: conjunto de evaluación nuevo,
    - pero debería tener la misma distribución que el de entrenamiento
  - Partir el corpus en entrenamiento (~80%) y test (~20%).
    - O en tres: entrenamiento (~70%), desarrollo (~15%) y test (~15%).
  - Nunca evalúo sobre el corpus de entrenamiento: sobreajuste (*overfitting*)
  - Si tengo pocos datos: *cross validation*

Classification: Train, Validation, Test split



Fuente: **Evaluation and Credibility How much should we believe in what was learned?**
Published by[Monica Dorsey]

Clasificación con aprendizaje automático: atributos

Atributos usuales en PLN:

- Atributos para palabras: forma de superficie, lema, terminación, POS, pertenencia a una lista, cantidad de letras, etc, etc.
- Atributos para oraciones: incluye palabra de una lista, número de veces que aparece una palabra, incluye expresión, largo, etc, etc.
- Atributos para documentos: largo, ocurrencias de cada palabra, etc, etc.
- Uno de los atributos es la clase objetivo (target class).

La ingeniería de atributos es precisamente obtener esos atributos a partir de nuestra forma original.

Clasificación con aprendizaje automático: atributos

Estrategia usual para convertir categorías a números: *one-hot-encoding*

idioma: {español, inglés, francés} →

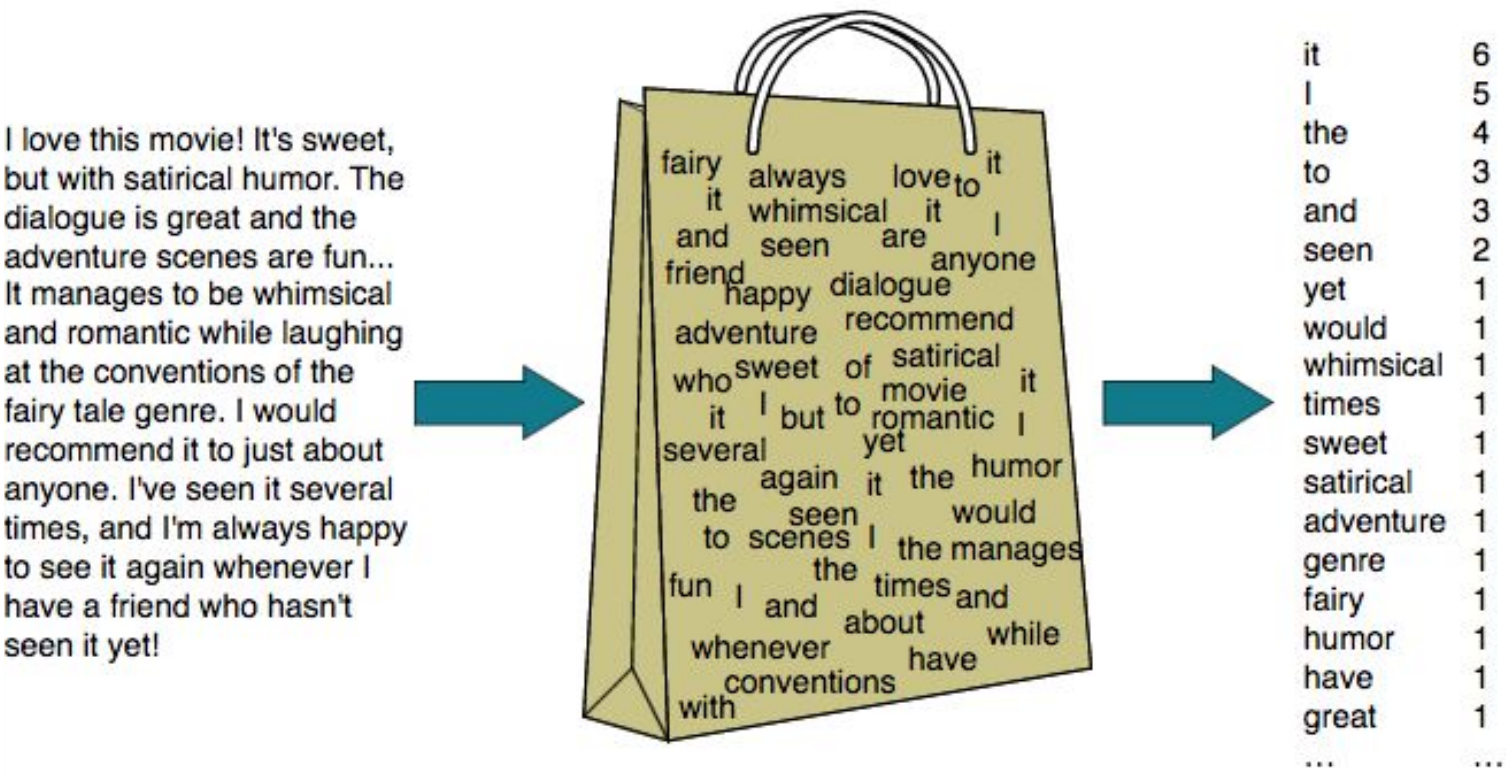
Es_español:{0,1}

Es_inglés:{0,1}

Es_francés:{0,1}

Clasificación con aprendizaje automático: atributos

En PLN ha sido usual trabajar con atributos de tipo Bag Of Words (BOW)



(Tomado de Jurafsky et al.)

Clasificación con aprendizaje automático: atributos

En PLN es usual trabajar con atributos de tipo Bag Of Words (BOW):

- La cantidad de atributos es el tamaño del vocabulario.
- Para cada atributo pongo **0** o **1** dependiendo si esa palabra ocurre en el texto (one hot encoding).
- Variantes:
  - cantidad de ocurrencias
  - lemas en vez de palabras
  - ignorar stop-words
  - n-gramas

Problema: ¡se pierde información sobre el orden de las palabras!

Métodos de Clasificación

Naïve Bayes

Árboles de Decisión

Regresión Logística

SVM

Redes Neuronales

Clasificación con aprendizaje automático: métodos

Naïve Bayes

Idea general:

Estimo si es más probable que el tweet sea positivo (o negativo) en base a las probabilidades del corpus.

Regla de Bayes:

$P(A|B) = P(B|A) * P(A) / P(B)$

Si el tweet dice...	P(positivo)	P(negativo)
buenísimo	0.85	0.15
horrible	0.1	0.9
excelente	0.95	0.05
...		

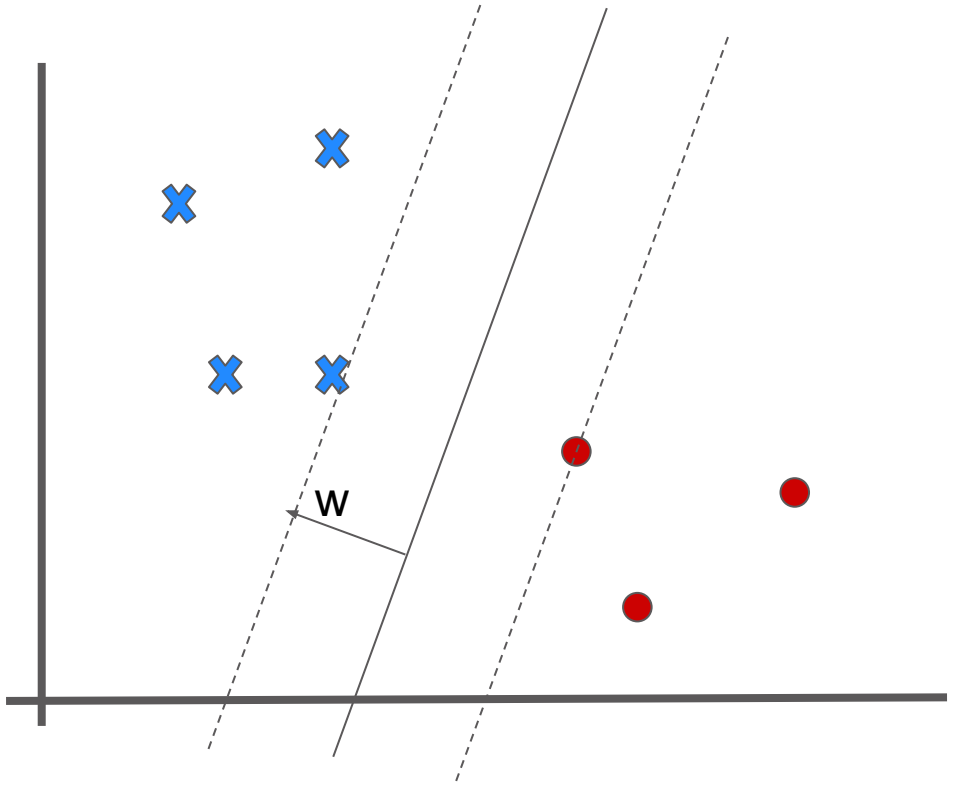
Clasificación con aprendizaje automático: métodos

SVM

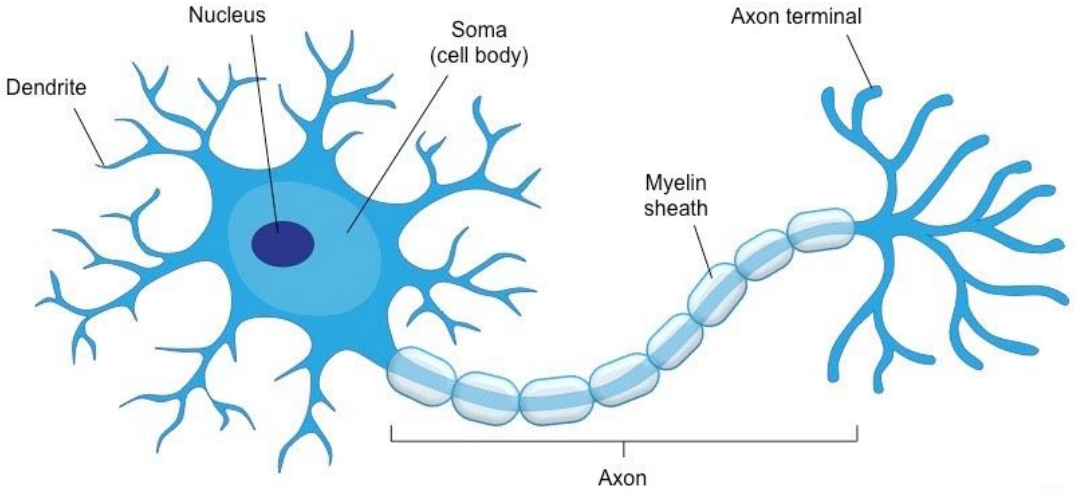
Idea general:

Los ejemplos son puntos en cierto espacio.

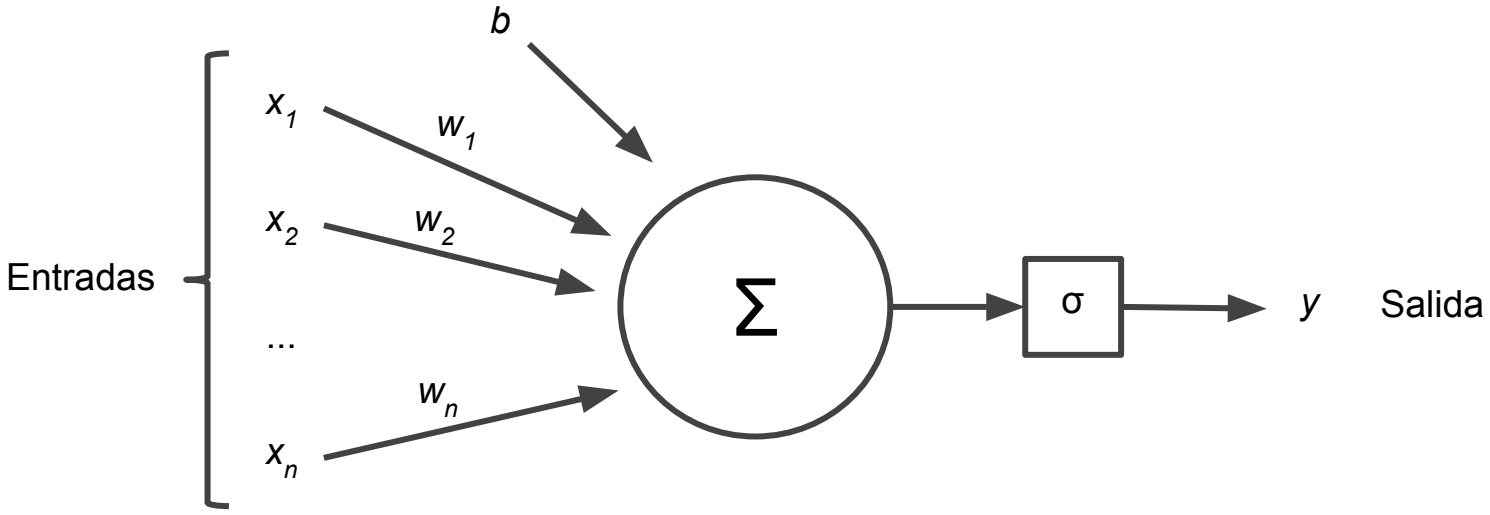
Aprender el mejor hiperplano que separa los ejemplos negativos de los positivos.



Redes Neuronales



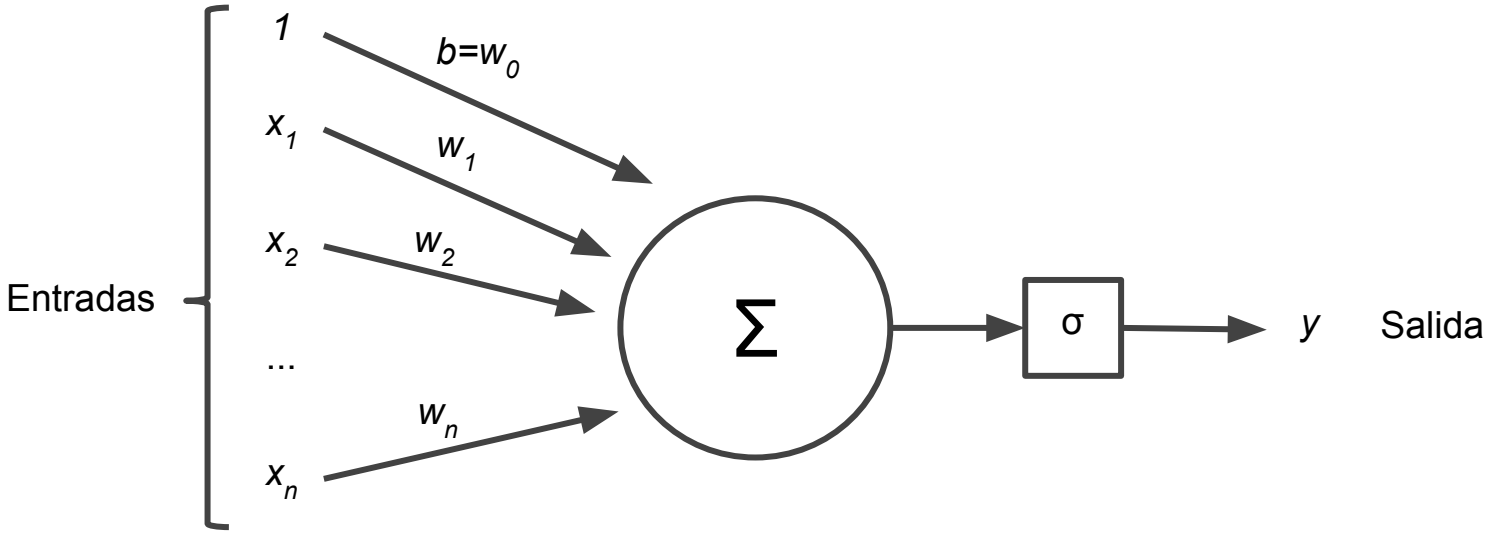
Redes Neuronales



$$y = \sigma(\sum_i x_i w_i + b)$$

Neurona de McCulloch-Pitts, 1943

Redes Neuronales



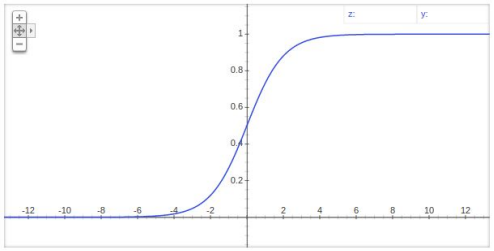
$$\hat{x} = [1, x_1, x_2, ..., x_n]$$
$$\hat{w} = [w_0, w_1, w_2, ..., w_n]$$

$$y = \sigma(\hat{x}.\hat{w})$$

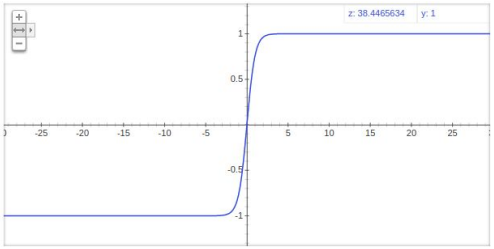
Redes Neuronales

Funciones de activación

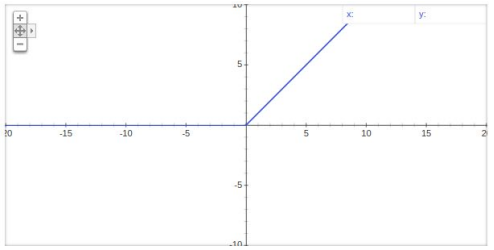
- Función sigmoide o logística: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$



- Tangente hiperbólica: $\tanh(z) = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$

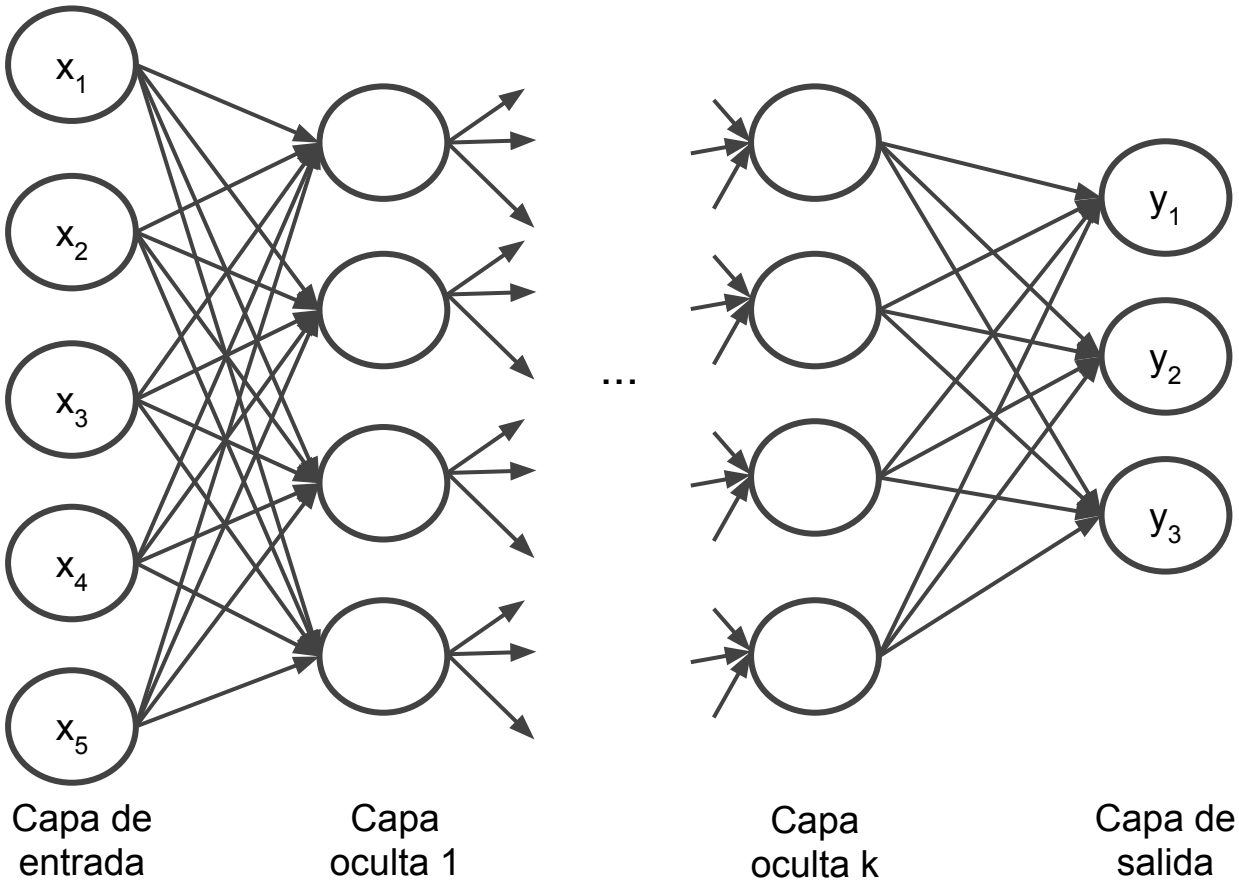


- ReLU: $\mathrm{relu}(z) = max(0, z)$

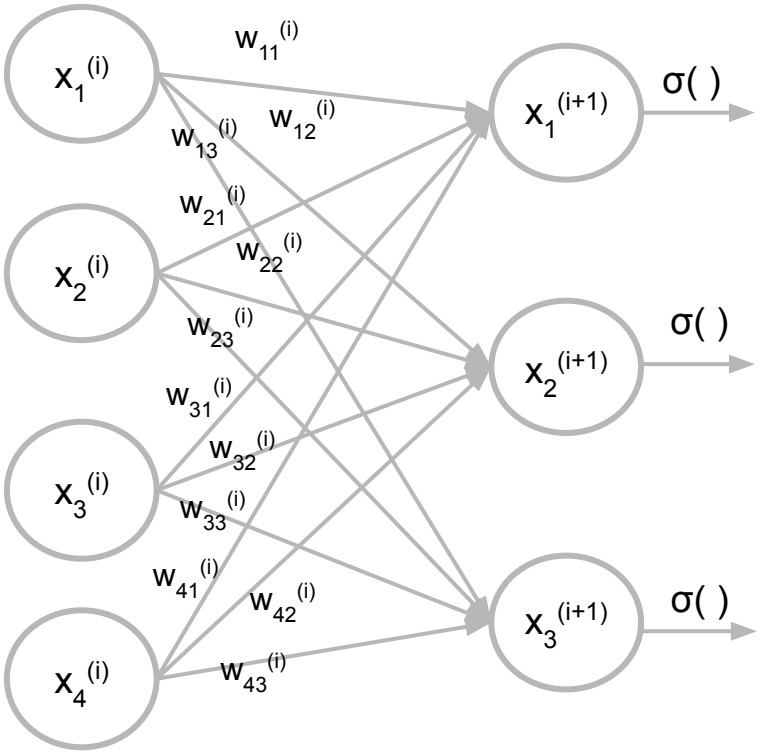


(Y hay otras...)

Redes Neuronales Multicapa



Redes Neuronales



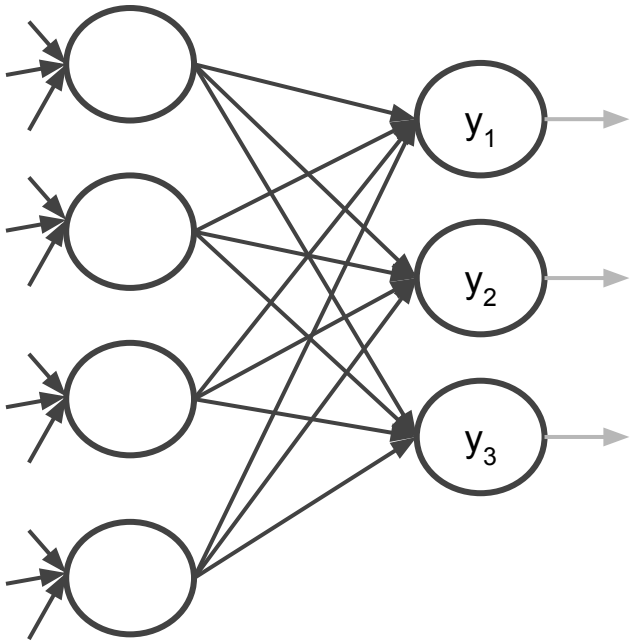
Entrada: $x^{(i)} = [\, x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, x_4^{(i)} \,]$

Salida: $x^{(i+1)} = [\, x_1^{(i+1)}, x_2^{(i+1)}, x_3^{(i+1)} \,]$

$$W^{(i)} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}$$

$$x^{(i+1)} = \sigma\left(x^{(i)} W^{(i)}\right)$$

Redes Neuronales



Problemas de clasificación discretos, queremos que la salida sea una distribución de probabilidad

Se suele utilizar una capa softmax $P(j|x) = \frac{e^{y_j}}{\sum_k e^{y_k}}$

Redes Neuronales

Entrenamiento:

- Se define una **función de pérdida** que mide la diferencia entre los valores predichos y los valores esperados, por ejemplo Error Cuadrático Medio:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

- Se busca encontrar los pesos que minimicen la función de pérdida (métodos usuales: Descenso por gradiente, Estocástico, Backpropagation, …).

- Problema importante a tener en cuenta: Sobreajuste

Vectores de palabras

- En PLN trabajamos principalmente con **texto.**
- Las RN y la mayoría de los clasificadores utilizan valores numéricos como entrada, por lo que necesitamos una **representación numérica** de textos:
  - palabras
  - oraciones
  - documentos
- Es deseable que esta representación numérica tenga propiedades explotables (e.g. un resultado de *distancia* interpretable).

Hipótesis distribucional

En los 1950s surge la hipótesis distribucional (Firth):

Palabras que aparecen en contextos similares tienden a tener significados similares

La milanesa con queso más rica es la uruguaya.

Sí, es re rica la hamburguesa con queso de ese lugar.

A la milanesa con queso mozzarella y salsa le decimos napolitana.

El otoño es una de las estaciones del año.

¡El verano es una de mis estaciones favoritas!

En invierno hace pila de frío.

En verano nunca hace frío.

Matriz término-término

Representa las palabras contando las palabras que las rodean, según un **contexto**.
El **contexto** puede ser el documento entero (archivo, tweet, página web o lo que sea)
pero lo más común es tomar **N palabras de ventana**.

O sea, si *X* es la palabra a modelar: $\text{palabra}_{-N} \ldots \text{palabra}_{-2}\text{palabra}_{-1}\ X\ \text{palabra}_{1}\text{palabra}_{2} \ldots \text{palabra}_{N}$

¿Cómo quedaría la matriz con el ejemplo anterior y usando N=5?

La milanesa con queso más rica es la uruguaya.

Sí, es re rica la hamburguesa con queso de ese lugar.

A la milanesa con queso mozzarella y salsa le decimos napolitana.

El otoño es una de las estaciones del año.

¡El verano es una de mis estaciones favoritas!

En invierno hace pila de frío.

En verano nunca hace frío.

	...	rica	queso	frío	estaciones	...
...						
milanesa		1	2	0	0	
hamburguesa		1	1	0	0	
otoño		0	0	0	1	
verano		0	0	1	1	
invierno		0	0	1	0	
...						

PROBLEMA → los vectores son enormes y con muchos ceros (dispersos, *sparse*)

Word2Vec

En 2013 Mikolov et al. propusieron **word2vec**, un par de algoritmos para crear colecciones de vectores de palabras **densos** (con pocos 0s) y de baja dimensionalidad (típicamente entre 150 o 300).

Idea: en vez de contar las palabras en una ventana de contexto, entrenemos un clasificador que prediga qué tan probable es que la palabra **c** aparezca en el contexto de **w**.

Como queremos que las palabras más relacionadas tengan vectores cercanos y las menos relacionadas tengan vectores alejados necesitamos **ejemplos negativos**.

Técnica de ***negative sampling***: elegir palabras que no compartan contexto con **w**. Por cada ejemplo positivo (**w**,$\mathbf{c}_{pos}$) tomamos *k* ejemplos negativos (**w**,$\mathbf{c}_{neg}$).

Word2Vec

Word2Vec

- El objetivo no es usar el clasificador entrenado, sino las representaciones intermedias que se generan dentro de la red neuronal.
- Los pesos aprendidos en la capa oculta de la red son los valores que forman el embedding de la palabra w.
- El entrenamiento es **autosupervisado** porque los valores esperados de salida del clasificador quedan determinados por las palabras que aparecen cerca de w en el texto original (sin anotaciones de ningún tipo).

Word2Vec: Algoritmo skip-gram

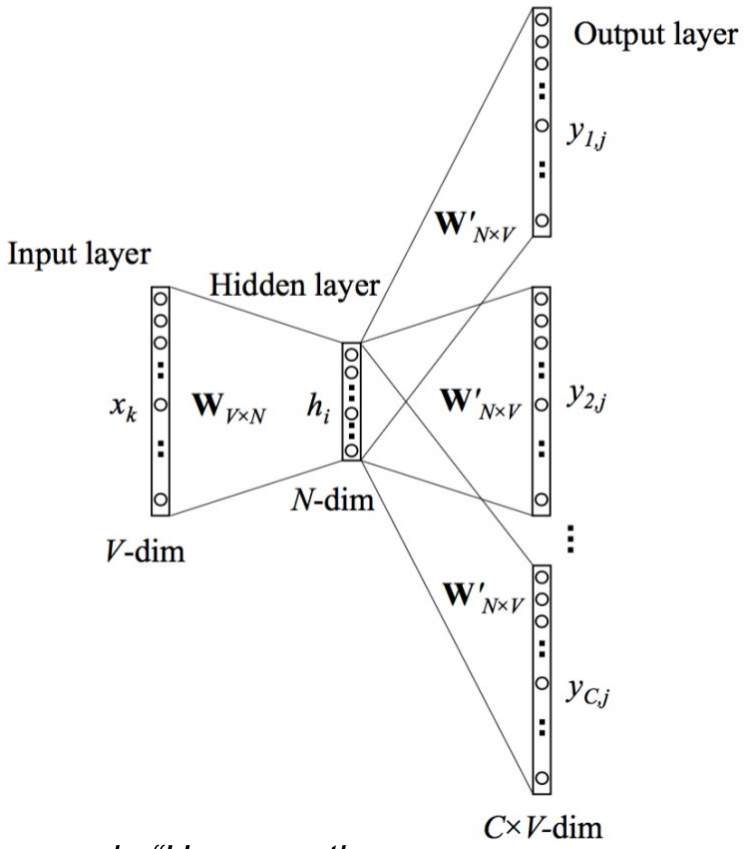


Imagen de "How exactly does word2vec work?" (Meyer, 2016)

skip-gram intenta modelar las palabras más probables que aparecerán alrededor de una palabra

- **Entrada:** Codificación 1-hot de la palabra k
- **Salidas:** Probabilidad de que la palabra j esté en el contexto C alrededor de la palabra k

Los *word embeddings* son el estado de la capa oculta luego del entrenamiento

Word2Vec

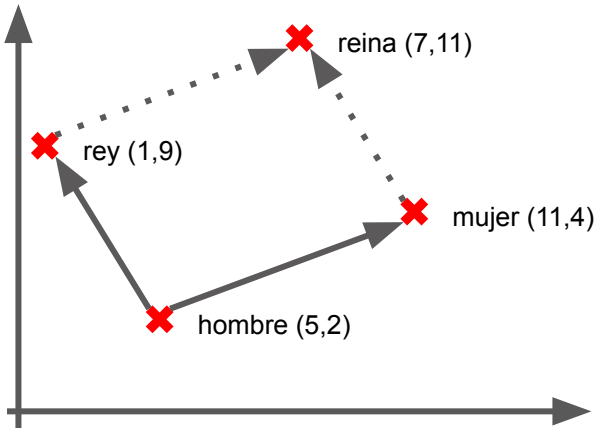
Se asocia una palabra (string) a un vector de reales

Vectores más cercanos tienden a ser semánticamente similares (similaridad coseno)

"Descubre" relaciones entre palabras

rey – hombre + mujer ≅ reina

uruguay – montevideo + francia ≅ parís

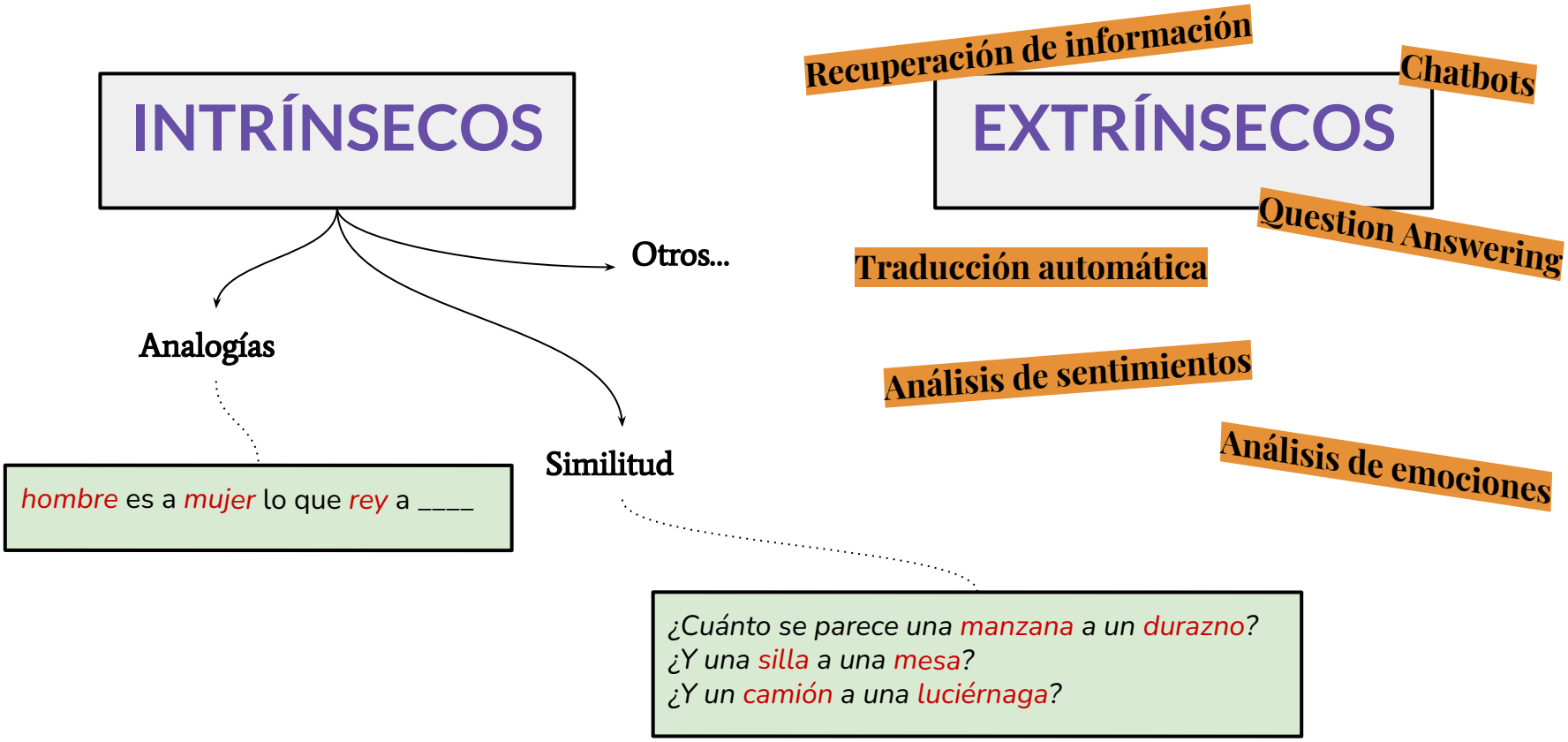


Se considera palabra a nivel de *string*, por lo que "vela" 🕯️ y "vela" ⛵ van a estar representadas por el mismo vector

PROBLEMA → no hay distinción entre diferentes significados de una palabra

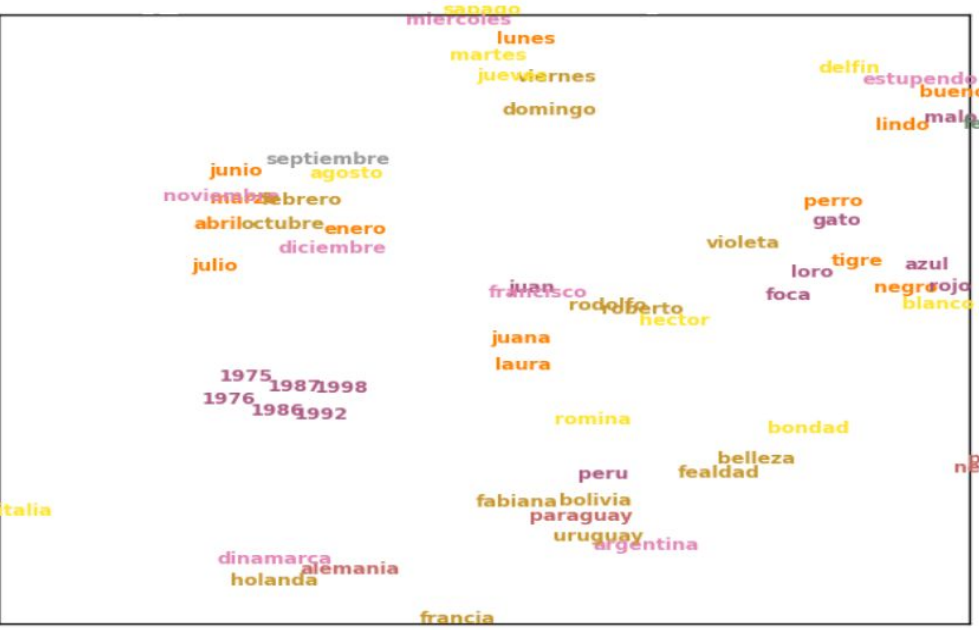
Evaluación

¿Cómo sabemos si una colección de embeddings está bien?

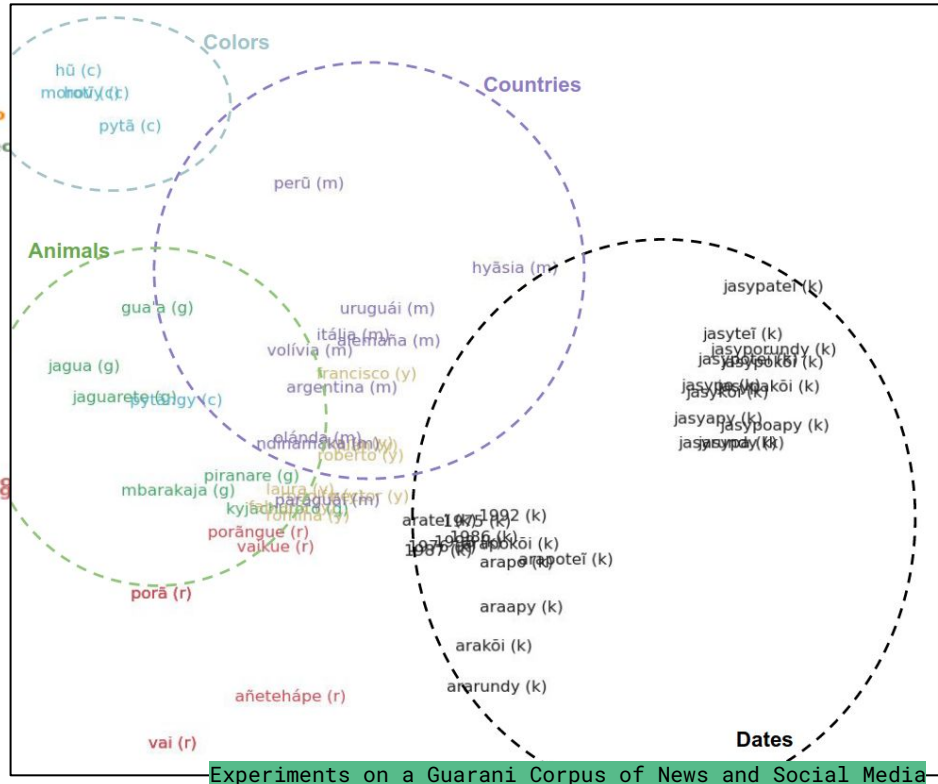


Evaluación

Visualización: reducir la dimensionalidad del espacio vectorial y graficar algunas palabras para ver cómo se agrupan



Spanish Word Vectors from Wikipedia (Etcheverry & Wonsever, LREC 2016)



Experiments on a Guarani Corpus of News and Social Media (Góngora, Giossa & Chiruzzo, AmericasNLP 2021)

Referencias

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics*, 3rd edition draft. Stanford. 2024. **[https://web.stanford.edu/~jurafsky/slp3/ed3bookfeb3_2024.pdf** Acceso: junio 2024]:

Capítulo 4: Naive Bayes, Text Classification, and Sentiment

Capítulo 7: Neural Networks and Neural Language Models

R. Garreta, G. Moncecchi, Learning Scikit-learn: Machine Learning in Python. 2013. Packt Publishing Ltd., Birmingham.