Clase 1: Del Texto al Vector

En Machine Learning generamos funciones...

Cuando programas de forma tradicional, tú escribes la función (las reglas). En Machine Learning, el algoritmo intenta "descubrir" esa función a partir de datos.

$$y = f(x)$$

$$y = f(x)$$

Lo que sea que quiera predecir.

Función que voy a entrenar (regresión logistica, arbol de desición,regresión lineal, red neuronal etc...)

Características de las instancias.

Un ejemplo...

$$y = f(x)$$

{pera,manzana}

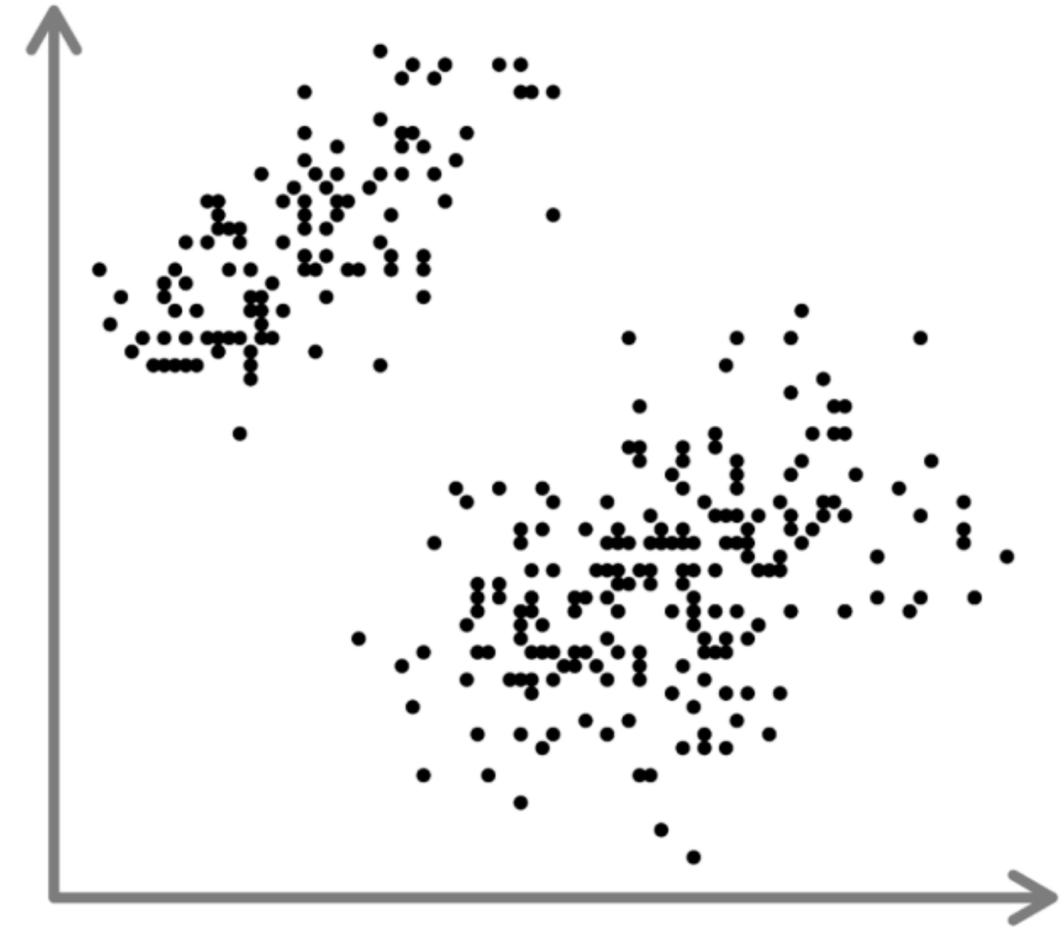
Función

Características de la fruta

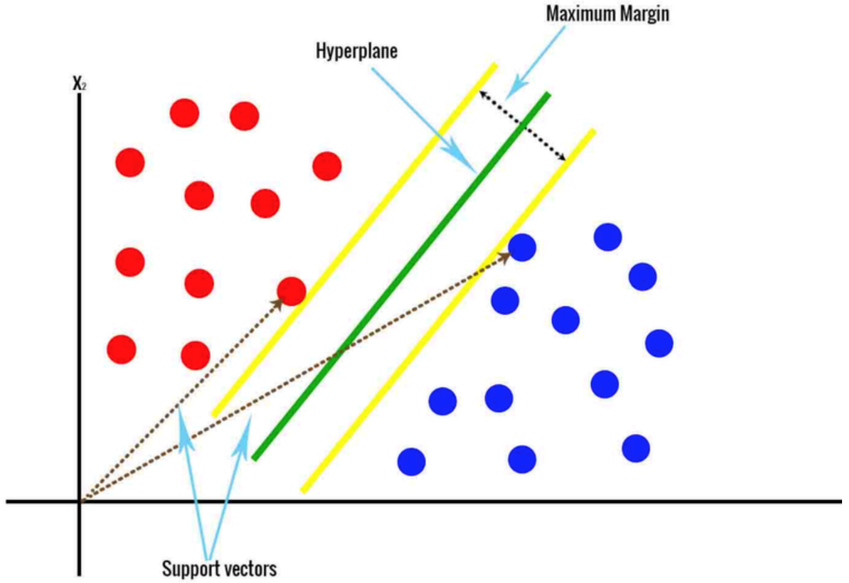
Las caracteristicas se transforman en vectores que viven en un espacio euclediano

$$\{x\}$$

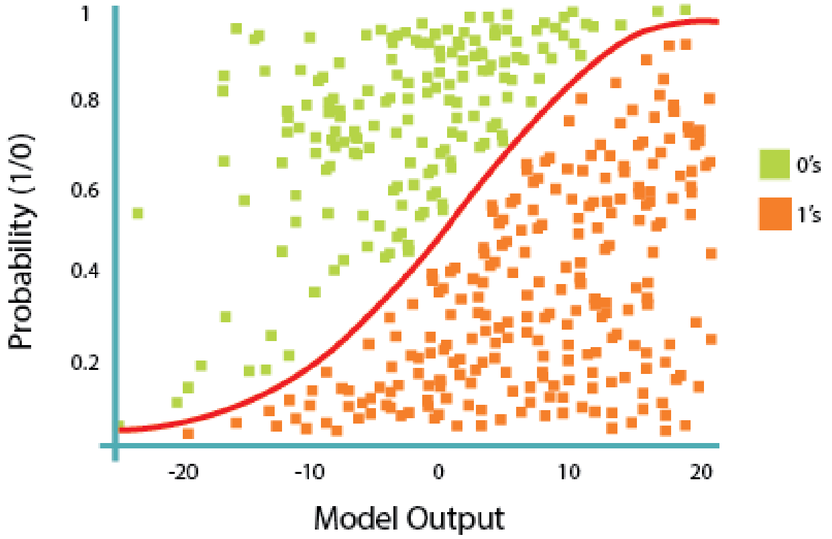
$$\{x\}$$



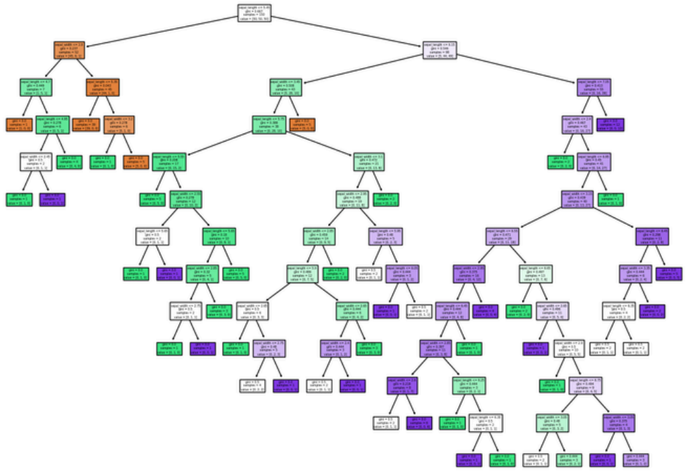
Al final lo que tenemos son puntos etiquetados, diferentes, con los que podemos trabajar



SVM



Regresión logistica.



Árbol

¿Qué pasa con el texto?

¡El texto es particularmente complicado de representar!

Existen características del texto que hacen que sea difícil de representar:

- El contexto y la Polisemia: Una misma unidad de texto puede significar cosas opuestas según lo que tenga al lado. Ejemplo: Me senté en el banco a esperar.", "Fui al banco a cobrar un cheque."
- La explosión de la dimensionalidad: Podemos tener corpus como miles-millones de palabras.

A pesar de eso, hemos creado tecnologías útiles

Conteo de palabras

Similitud semántica

Secuencia de palabras

Relaciones globales y contexto

Bag of Words (BoW)

Es el método más sencillo. Imagina que tomas un texto, lo metes en una bolsa y solo te importa qué palabras hay y cuántas veces aparecen, ignorando el orden.

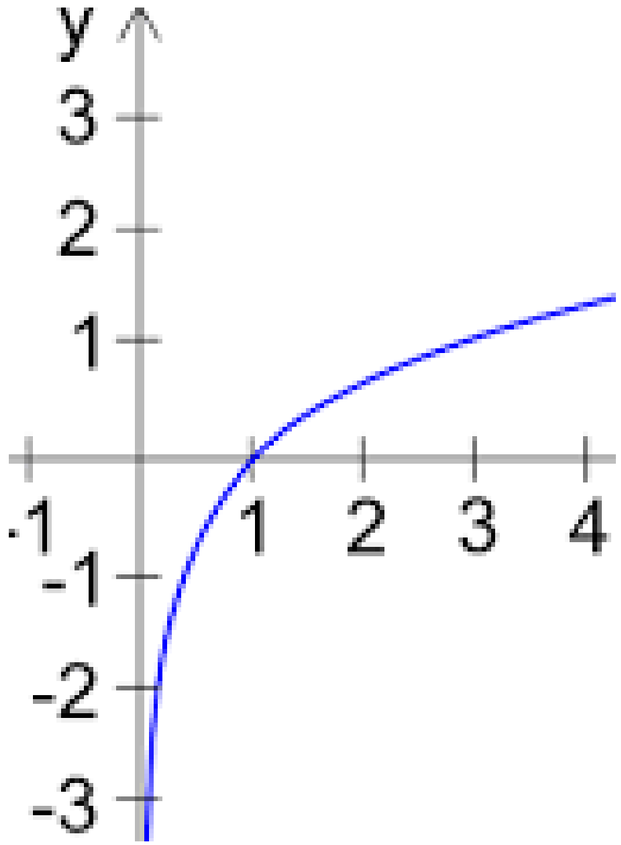
- Vocabulario: Se crea una lista con todas las palabras únicas de todos tus documentos.
- Vectorización: Para cada documento, creas un vector donde cada posición corresponde a una palabra del vocabulario.
- Conteo: Escribes cuántas veces aparece cada palabra.

Ejemplo:
Frase A: "El gato caza al ratón".
Frase B: "El ratón escapa del gato".

Documento	el	gato	caza	al	ratón	escapa	del
Frase A	1	1	1	1	1	0	0
Frase B	1	1	0	0	1	1	1

TF-IDF

El problema de BoW es que palabras como "el", "de" o "que" aparecen mucho pero no dicen nada. TF-IDF (Term Frequency - Inverse Document Frequency) soluciona esto castigando a las palabras comunes y premiando a las raras o específicas.



Gráfica log(x)

TF (Term Frequency): ¿Qué tan frecuente es la palabra en el documento actual?

$$TF(t,d) = \frac{\text{frecuencia de la palabra } t \text{ en documento } d}{\text{total de palabras en } d}$$

IDF (Inverse Document Frequency): ¿Qué tan rara es la palabra en todo el conjunto de documentos?

$$IDF(t,D) = \log\left(\frac{\text{Total de documentos}}{\text{Documentos con la palabra } t}\right)$$

$$TF-IDF\,(t,d,D) = TF\,(t,d) \times IDF\,(t,D)$$

Explosión de la dimensionalidad

Cuando usas Bag of Words o TF-IDF, cada palabra única en tu vocabulario se convierte en una columna (una dimensión) de tu matriz. Esto genera una seria de problemas:

- Si tenemos 50,000 palabras únicas, la función de Machine Learning ahora tiene que procesar 50,000 dimensiones.
- En una frase de 10 palabras, el vector tendrá 10 números y 49,990 ceros.
- Para BoW, "médico" y "doctor" son dimensiones totalmente distintas y no guardan relación alguna.

Reducción de la dimensionalidad. LSA.

Para reducir la dimensión de los vectores que representan al texto se utiliza SVD(.

$$A = U\Sigma V^T$$

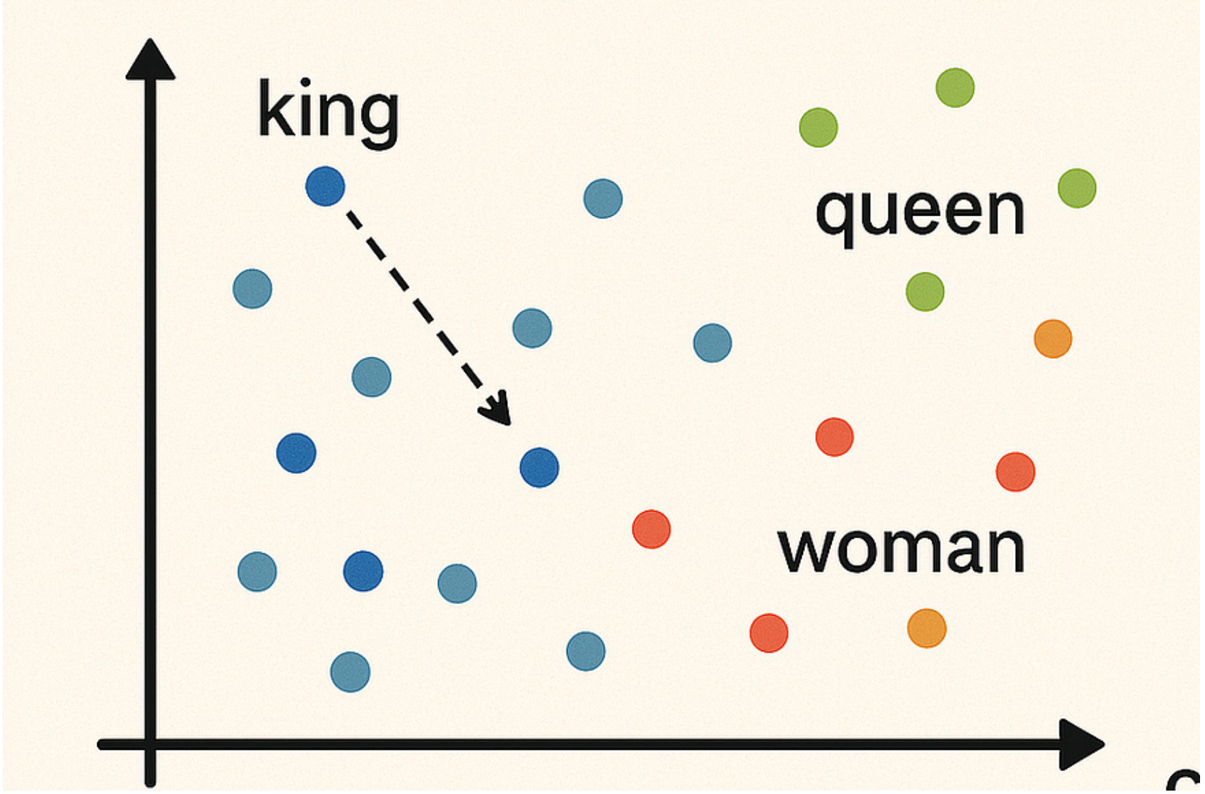
Una vez descompuesta la matriz, la truncamos utilizando las k dimensiones mas importantes.

$$A_k \approx U_k \Sigma_k V_k^T$$

Asi obtenemos una matriz densa, cuyos vectores codifican la información de los documentos.

El salto a los Word Embeddings

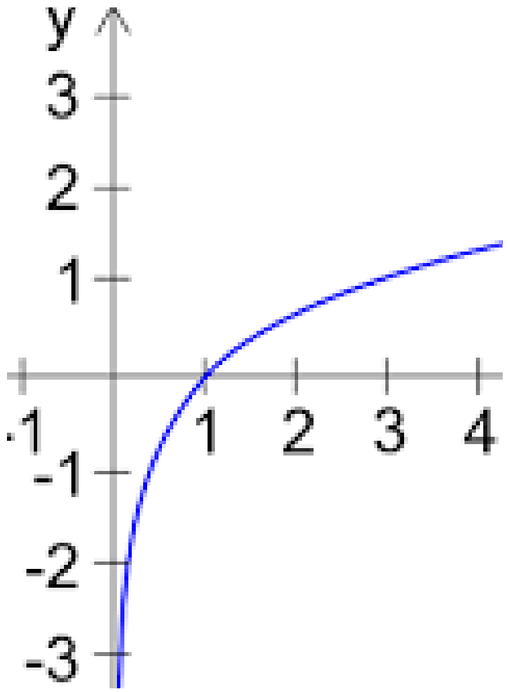
Los métodos tradicionales como LSA son incapaces de capturar la sutileza del lenguaje, los embeddings utilizan redes neuronales para predecir el contexto inmediato de una palabra, esto permite que el modelo no solo entienda la temática global, sino que capture relaciones entre palabras y contexto.



¿Cómo funciona?

Podemos vizualizar el proceso de entrenamiento como una tarea de predicción, el la que intentamos predecir las palabras del contexto, a partir de una palabra central.

El objetivo es maximizar la probabilidad de observar las palabras del contexto dada la palabra central.



Gráfica log(x)

$$J(\theta) = \prod_{t=1}^{T} \prod_{-k \leq j \leq k, j \neq 0} P(w_{t+j}|w_t)$$

En la práctica, minimizamos el log-loss negativo

$$L = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-k \leq j \leq k, j \neq 0} \log P(w_{t+j}|w_t)$$

Dos matrices de pesos

Se utilizan realmente dos representaciones (vectores) para cada palabra del vocabulario.

La matriz de entrada: Cuando la palabra es la central.

$$W \in \mathbb{R}^{d \times V}$$

La matriz de salida: Cuando la palabra es la contexto..

$$W' \in \mathbb{R}^{V \times d}$$

Calculo de probabilidad

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

La probabilidad de que una palabra de contexto (o) aparezca dada una palabra central (c) se calcula mediante la función Softmax.

En la parte superior del Softmax tenemos:

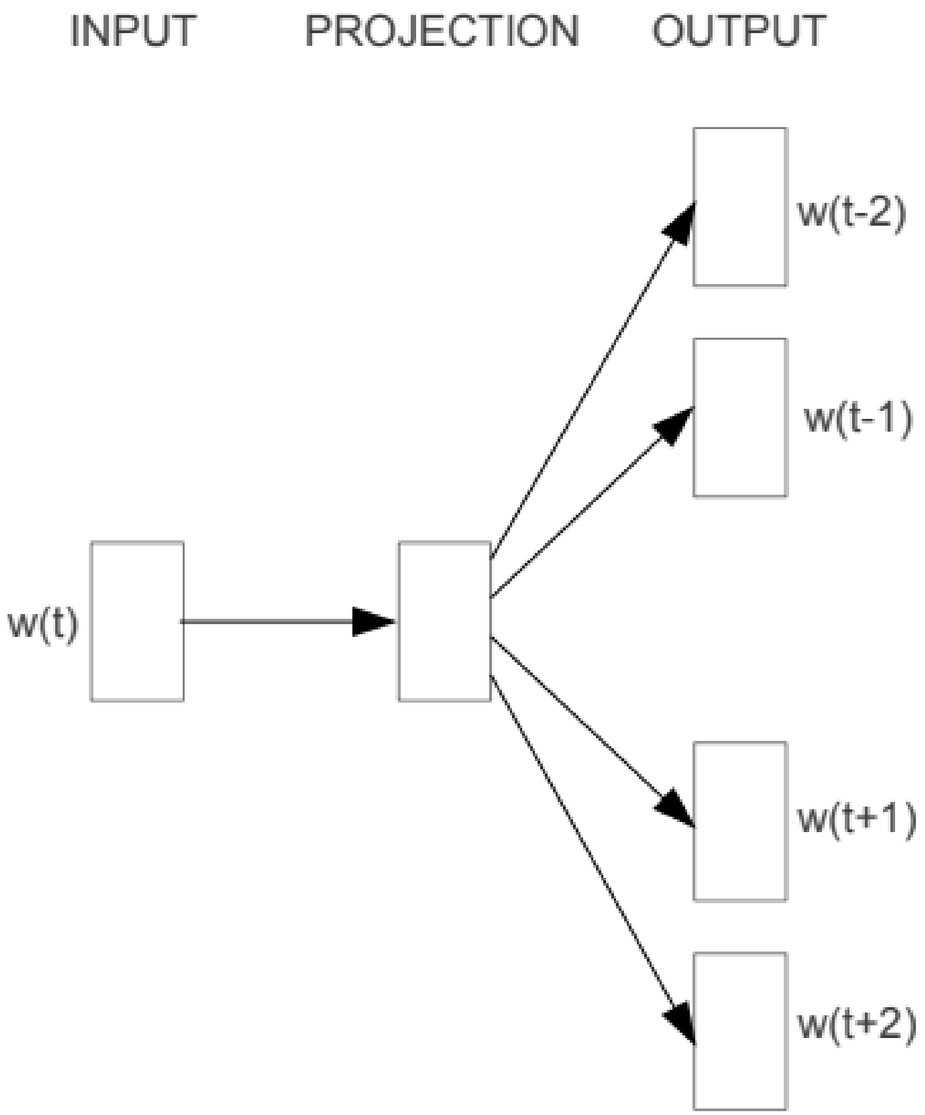
$$\exp(u_o^T v_c)$$

El producto punto se define matemáticamente como:

$$u_o \cdot v_c = \|u_o\| \|v_c\| \cos(\theta)$$

Si el ángulo entre la palabra central y la de contexto es pequeño, el coseno es cercano a 1, esto maximiza el producto punto y minimiza la función de costo. El algoritmo empuja los vectores de palabras que aparecen juntas para que su ángulo sea menor (aumentando su similitud coseno) ,y mediante el denominador ,aleja los vectores de palabras que no tienen relación.

Arquitectura



La palabra entra como un vector binario gigante donde solo su posición es 1.

$$[0,1,0,0,0] \times \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ \mathbf{v_{gato1}} & \mathbf{v_{gato2}} & \mathbf{v_{gato3}} \\ \vdots & \vdots & \vdots \end{pmatrix} = [v_{gato1}, v_{gato2}, v_{gato3}]$$

Ahora tomamos ese vector y lo multiplicamos por la matriz de salida

$$[v_{g1}, v_{g2}, v_{g3}] \times \begin{pmatrix} u_{1,1} & u_{1,2} & \dots \\ u_{2,1} & u_{2,2} & \dots \\ u_{3,1} & u_{3,2} & \dots \end{pmatrix} = [z_1, z_2, z_3, z_4, z_5]$$

Finalmente, convertimos esos productos punto en una distribución de probabilidad.

$$\hat{y}_i = \frac{e^{z_i}}{\sum e^{z_j}}$$