ACÁMICA

¡Bienvenidos/as a Data Science!

Agenda

¿Cómo anduvieron?

Repaso de Redes Neuronales con Kahoot

Explicación: Procesamiento de Lenguaje Natural

Break

Hands-On

Lanzamiento Entrega 05

Cierre

¿Dónde estamos?

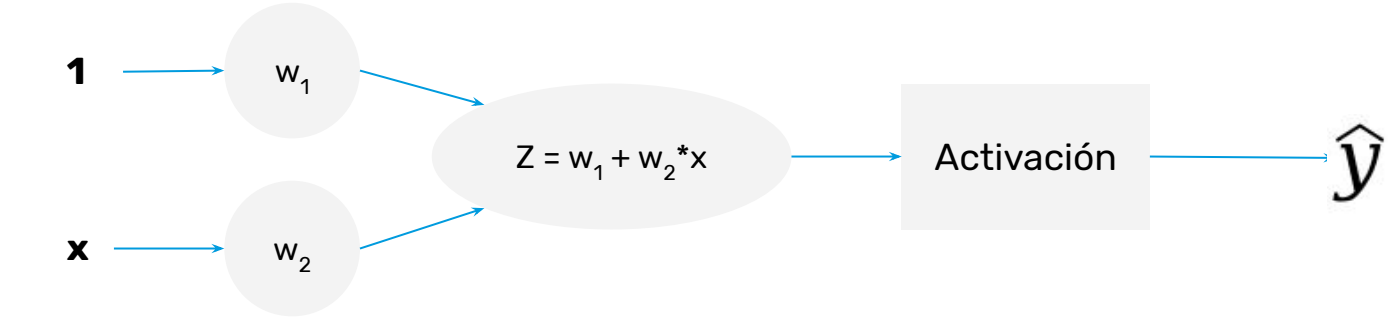
¿Cómo anduvieron?

Repaso:
Redes Neuronales

Preguntas 1 a 5

Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1



Entradas: Features + Bias

Pesos: Lo que buscamos en el entrenamiento

Unión sumadora: combina los features con los pesos

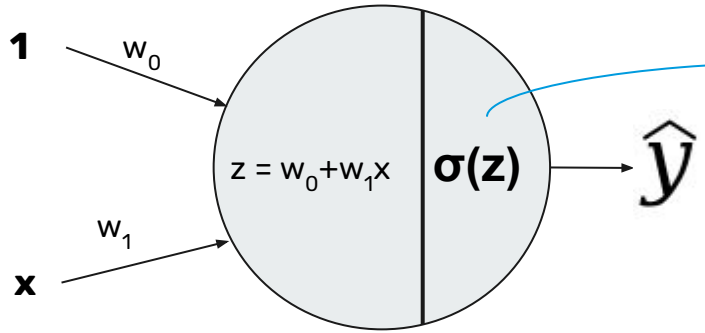
Activación: si no está, ¡es una función lineal!

Salida: entre 0 y 1, probabilidad del evento

Perceptrón con una variable

Necesitamos algo que, dado los features, devuelva probabilidades.
Las probabilidades deben estar entre 0 y 1

Otra representación



Activación:

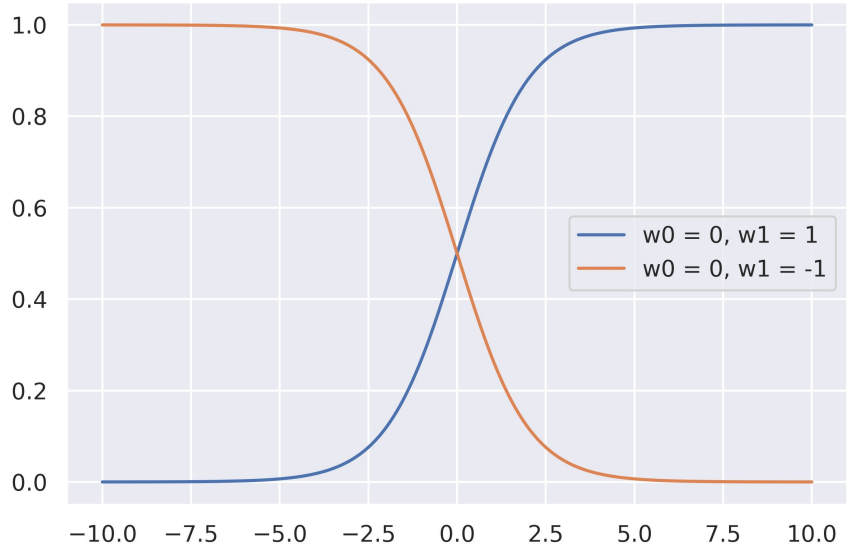
- Sin la activación, es una función lineal
- Necesitamos introducir algo que *sature* la entrada en 0 o en 1 dependiendo del resultado de la unión sumadora

Función Logística / Sigmoide

$$y(z) = \frac{1}{1 + e^{-z}}$$

$$z = w_0 + w_1 x$$

$$y(x) = \frac{1}{1 + e^{-(w_0 + w_1 x)}}$$



Preguntas 5 a 9

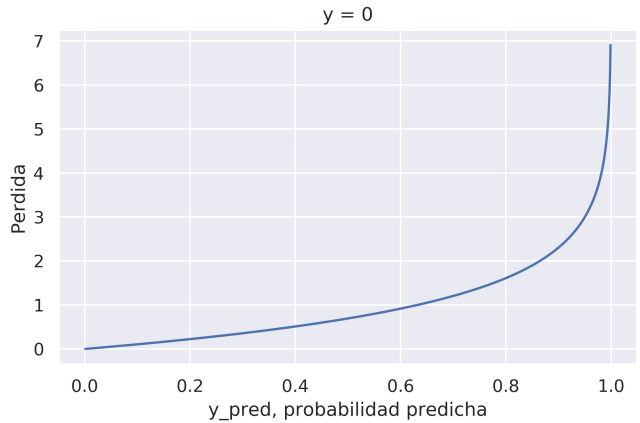
Entropía cruzada (cross-entropy)

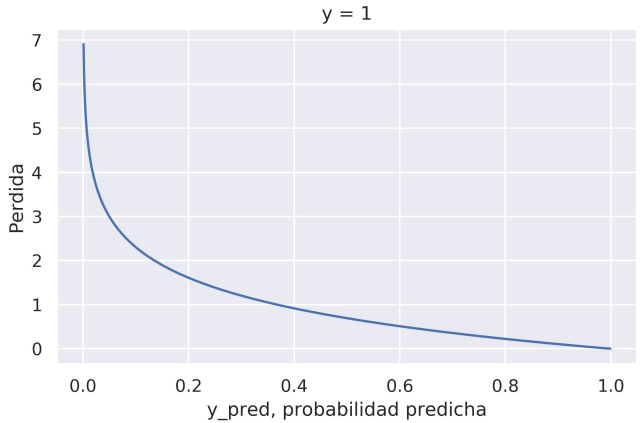
Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. $\hat{y}$

Caso binario: etiquetas y = 0 y 1.

$$L(\hat{y}, y) = -y * log(\hat{y}) - (1 - y) * log(1 - \hat{y})$$

Pérdida para una instancia





Entropía cruzada (cross-entropy)

Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. $\hat{y}$

Caso binario: etiquetas y = 0 y 1.

$$L(\hat{y}, y) = -y * log(\hat{y}) - (1 - y) * log(1 - \hat{y})$$ **Pérdida para una instancia**

Entropía cruzada (cross-entropy)

Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. $\hat{y}$

Caso binario: etiquetas y = 0 y 1.

$$L(\hat{y}, y) = -y * log(\hat{y}) - (1 - y) * log(1 - \hat{y})$$

Pérdida para una instancia

$$J(\overline{W}) = \frac{1}{n} \sum_{i=0}^{n-1} L(\widehat{y^{(i)}}, y^{(i)})$$

Costo para todas las instancias

Entropía cruzada (cross-entropy)

Necesitamos una función de pérdida entre una etiqueta (y) y la probabilidad de pertenecer o no a esa etiqueta. $\hat{y}$

Caso binario: etiquetas y = 0 y 1.

$$L(\hat{y}, y) = -y * log(\hat{y}) - (1 - y) * log(1 - \hat{y})$$

Pérdida para una instancia

$$J(\overline{W}) = \frac{1}{n} \sum_{i=0}^{n-1} L(\widehat{y^{(i)}}, y^{(i)})$$

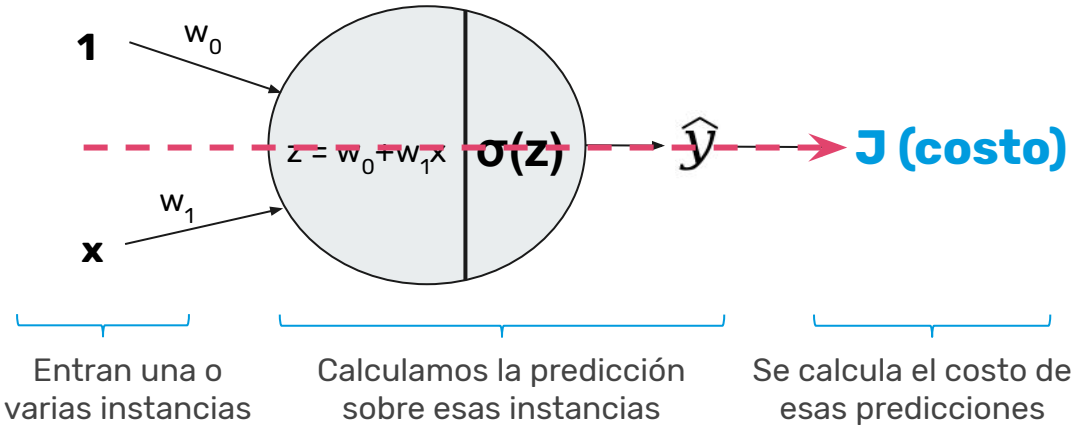
Costo para todas las instancias

$$J(w_0,\ w_1) = \frac{1}{n} \sum_{i=0}^{n-1} L(\widehat{y^{(i)}}, y^{(i)})$$

Costo para todas las instancias, caso 1D

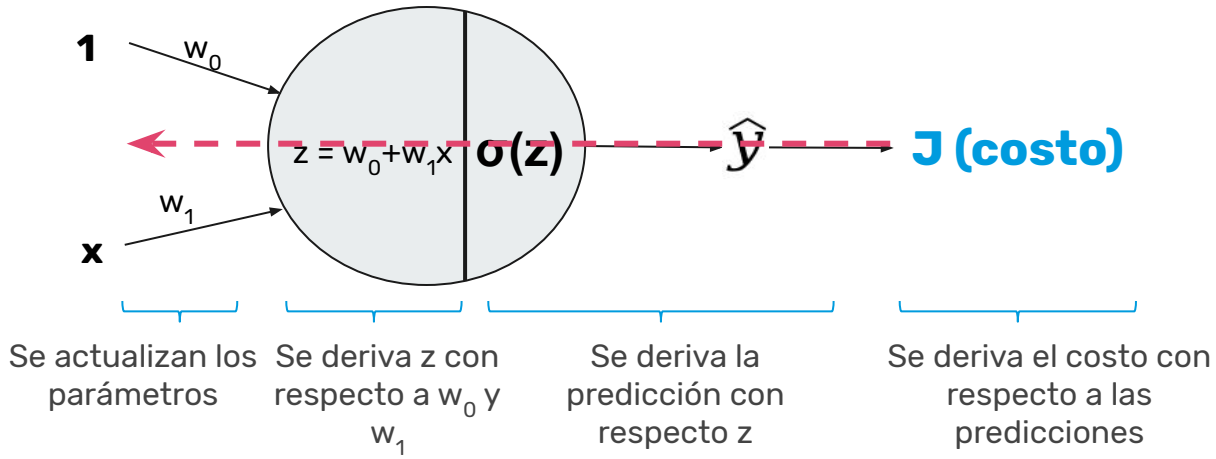
1. Descenso por gradiente calcula la derivada/gradiente del costo y con eso actualiza los parámetros. Este proceso lo va a hacer muchas veces hasta llegar al mínimo.
2. En cada una de esas iteraciones, tiene que calcular el costo. El costo depende de las instancias de entrenamiento y de los parámetros que tengamos hasta ese momento.

Calcular el costo con las instancias de entrenamiento es lo que se conoce como **Forward Propagation.**



1. Con el costo calculado, queremos actualizar los valores de los parámetros según la regla vista en la clase anterior.
2. Para eso, tenemos que derivar el costo y propagar esa derivada hacia atrás, hasta llegar a los parámetros w_0 y w_1.

Calcular las derivadas y actualizar los parámetros "hacia atrás" se conoce como **Backpropagation.**



Preguntas 10 a 12

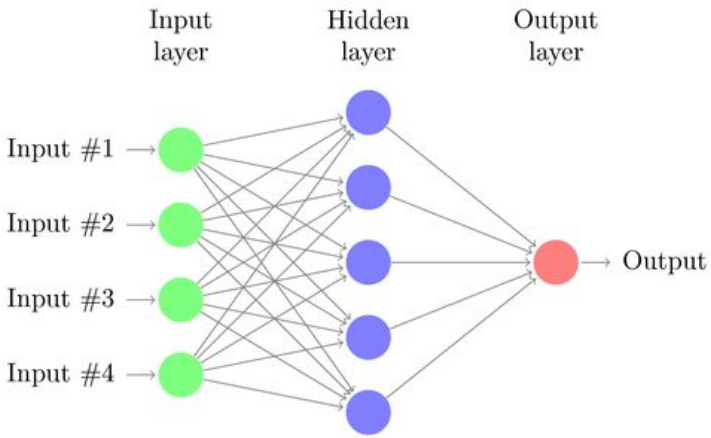
Perceptrón Multicapa

Ampliando el Perceptrón

Problema con el Perceptrón:
solo encuentra fronteras lineales.

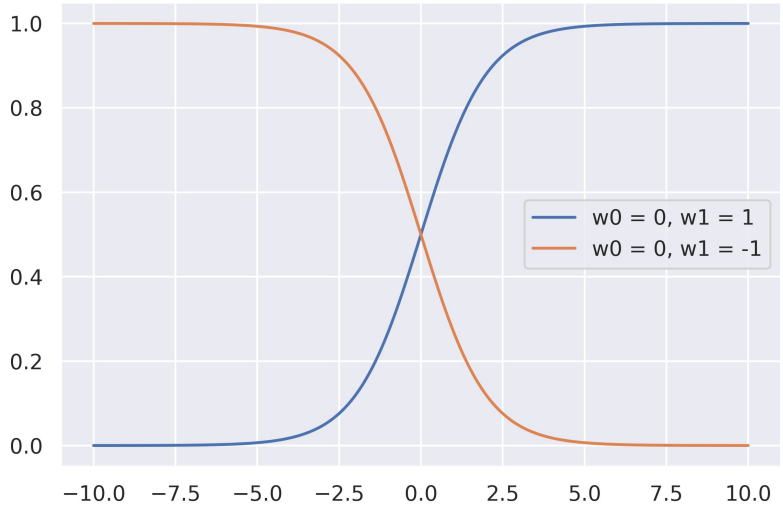
Ampliando el Perceptrón

Solución: Perceptrón Multicapa



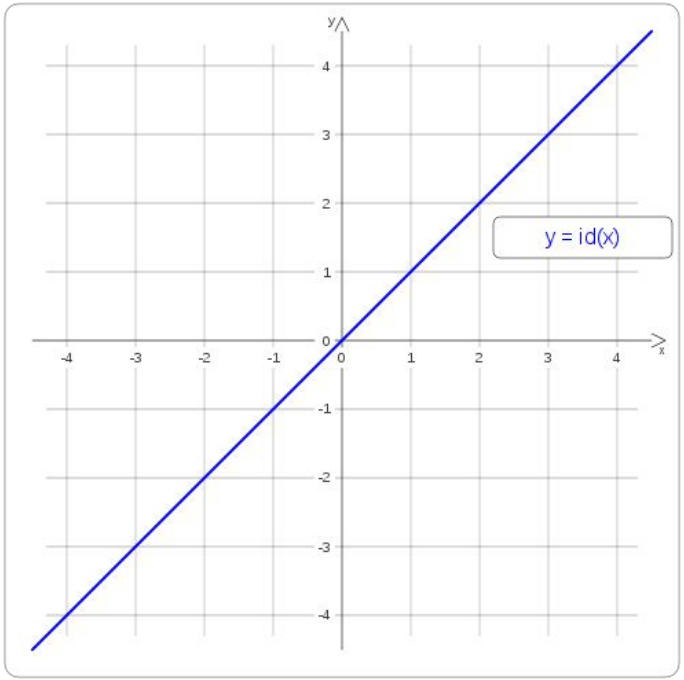
- Cada neurona tiene sus propios pesos/parámetros. En aplicaciones comunes suelen ser desde miles a millones de parámetros para toda la red.
- **Deep Learning es** encontrar esos pesos de manera eficiente, bajo la condición de realizar correctamente una tarea objetivo.

Perceptrón Multicapa • Funciones de activación



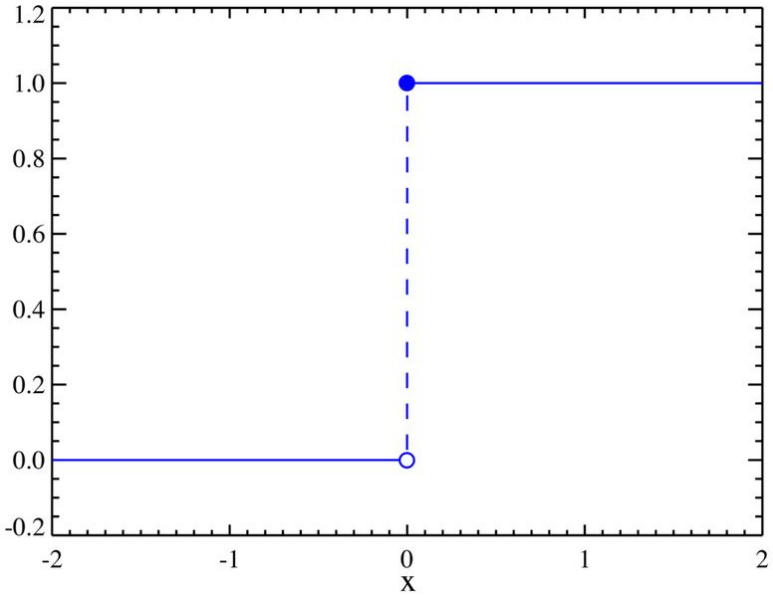
1. **Sigmoide/logística**
2. Identidad: f(x) = x
3. Escalón: f(x)=0 si x<0, 1 si x≥0
4. Tangente Hiperbólica: f(x)=tanh(x)
5. ReLU: f(x)=0 si x<0, x si x≥0

Perceptrón Multicapa • Funciones de activación



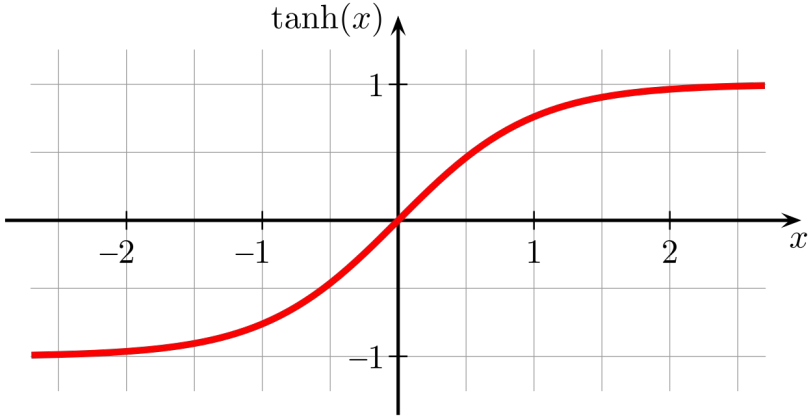
1. Sigmoide/logística
2. **Identidad: f(x) = x**
3. Escalón: f(x)=0 si x<0, 1 si x≥0
4. Tangente Hiperbólica: f(x)=tanh(x)
5. ReLU: f(x)=0 si x<0, x si x≥0

Perceptrón Multicapa • Funciones de activación



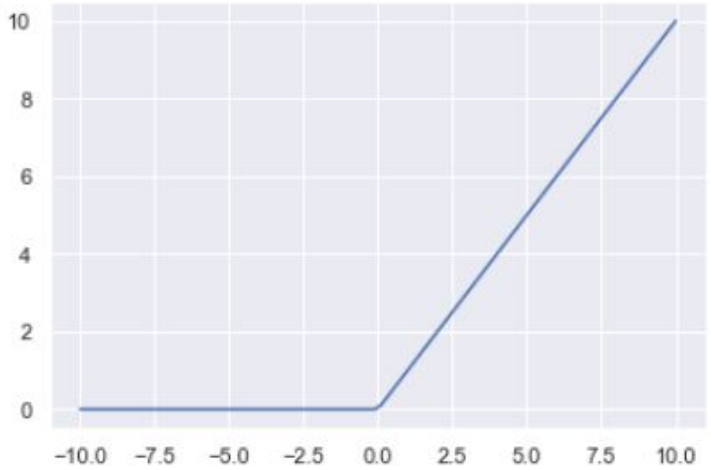
1. Sigmoide/logística
2. Identidad: f(x) = x
3. **Escalón: f(x)=0 si x<0, 1 si x≥0**
4. Tangente Hiperbólica: f(x)=tanh(x)
5. ReLU: f(x)=0 si x<0, x si x≥0

Perceptrón Multicapa • Funciones de activación



1. Sigmoide/logística
2. Identidad: f(x) = x
3. Escalón: f(x)=0 si x<0, 1 si x≥0
4. **Tangente Hiperbólica: f(x)=tanh(x)**
5. ReLU: f(x)=0 si x<0, x si x≥0

Perceptrón Multicapa • Funciones de activación



1. Sigmoide/logística
2. Identidad: f(x) = x
3. Escalón: f(x)=0 si x<0, 1 si x≥0
4. Tangente Hiperbólica: f(x)=tanh(x)
5. **ReLU: f(x)=0 si x<0, x si x≥0**

Perceptrón Multicapa • Funciones de activación

Clasificación:
lo más común es encontrar ReLU en las capas interiores y Sigmoide en la salida

1. **Sigmoide/logística**
2. Identidad: f(x) = x
3. Escalón: f(x)=0 si x<0, 1 si x≥0
4. Tangente Hiperbólica: f(x)=tanh(x)
5. **ReLU: f(x)=0 si x<0, x si x≥0**

Perceptrón Multicapa

	Funciones de activación	**Costos (Keras)**
Multiclase	1. Sigmoide/logística 2. Softmax	Categorical_crossentropy
Regresión	Identidad	1. mean_squared_error 2. mean_absolute_error 3. Otras

Perceptrón Multicapa

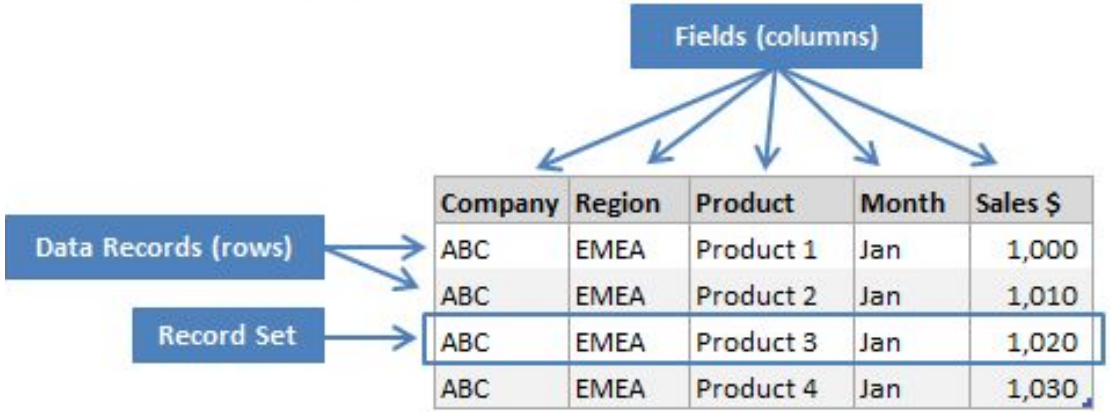
	Funciones de activación	Costos (Keras)
Multiclase	1. Sigmoide/logística **2. Softmax**	Categorical_crossentropy
Regresión	Identidad	1. mean_squared_error 2. mean_absolute_error 3. Otras

Generalización de la sigmoide, útil cuando las clases son excluyentes.

Kahoot!

Procesamiento de Lenguaje Natural (NLP)

Sabemos trabajar con datos **estructurados** (tablas y números).



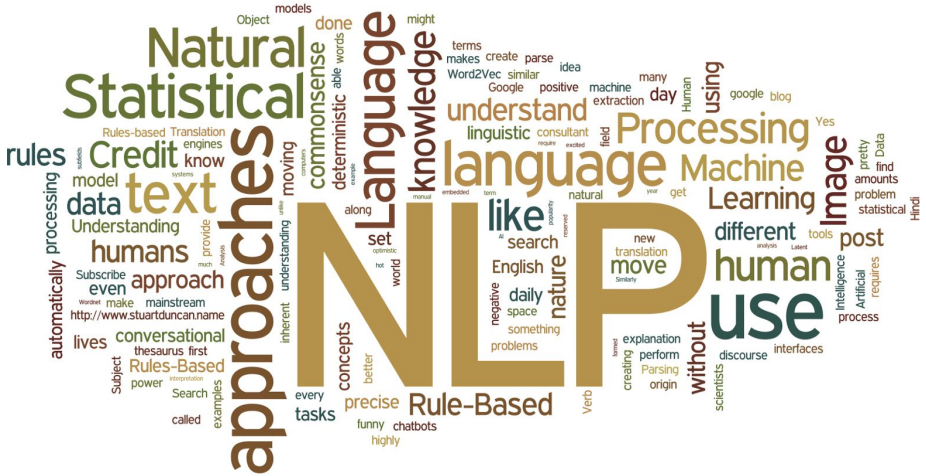
Sabemos trabajar con datos **estructurados** (tablas y números).

Problema: Hay muchísimos datos disponibles en forma de lenguaje natural (texto, no estructurado) que contienen información relevante.

¿Cómo hacemos para darle sentido a estos datos y trabajar con ellos en el marco de Ciencia de Datos?

NLP es la solución

El procesamiento de lenguaje natural es una rama de la inteligencia artificial que se enfoca en permitirle a las computadoras entender y procesar lenguaje natural.



NLP • Flujo de trabajo

Datos

Modelos

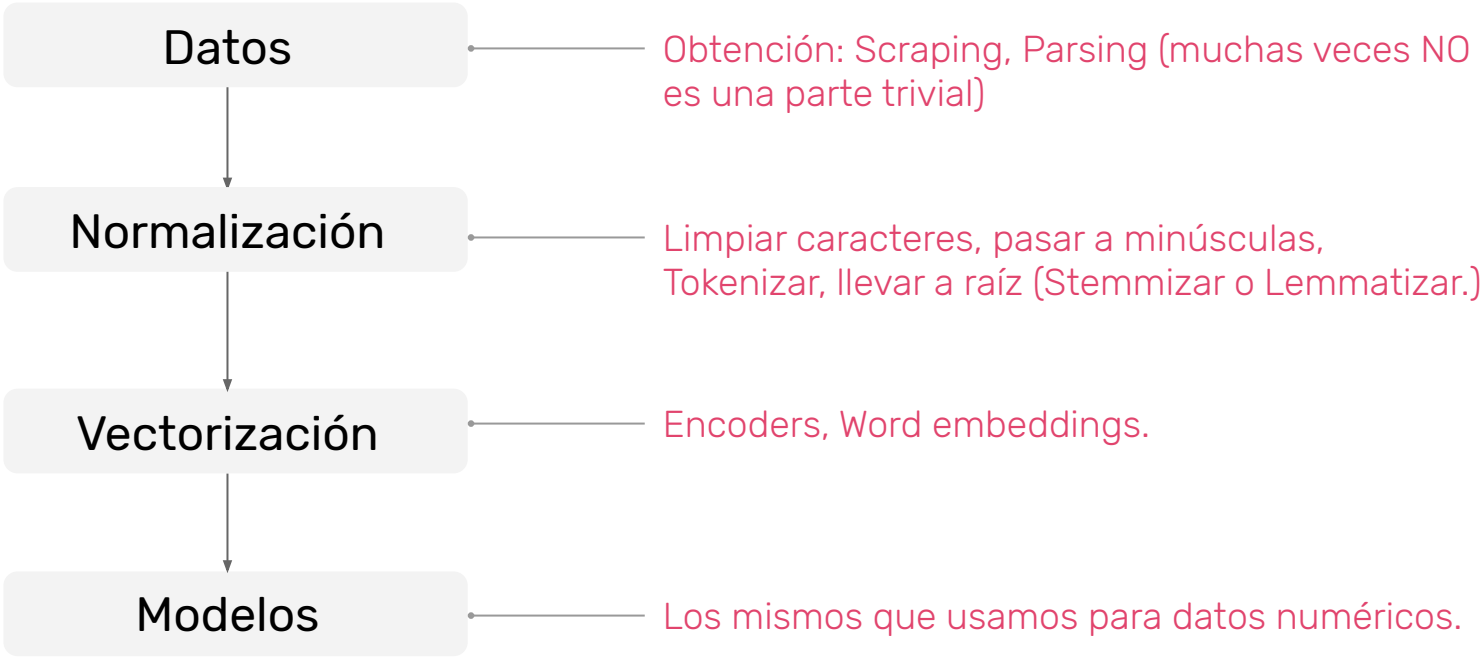
NLP • Flujo de trabajo

Datos

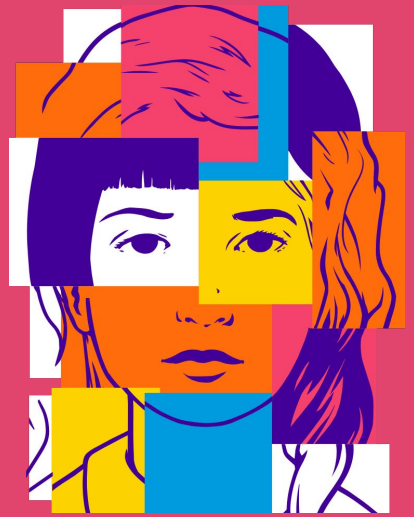
Modelos

Para esta vamos a usar la librería NLTK de Python

NLP • Flujo de trabajo



Normalización



Normalizar

Idea: llevar todo el texto a un formato común donde palabras escrita de manera distinta o con significados similares se representen de la misma manera.

Normalizar

Quiero Pasear a mi perro por #Palermo

Quisiera pasear a mis perros por Palermo

Normalizar

Quiero Pasear a mi perro por #Palermo

≠

Quisiera pasear a mis perros por Palermo

quiero pasear mi perro Palermo

Buscamos llevarlo a una forma común

Normalizar • Formas de hacerlo

- Pasar a minúsculas
- Tokenizar
- Limpiar caracteres
- Llevar a raíz

Normalizar • Formas de hacerlo

- **Pasar a minúsculas:** pasar todas los caracteres de un texto a su forma minúscula para homogeneizar.

“Esto es un texto. Tiene varias oraciones. Todas son distintas, ninguna es igual.”

texto.lower()

“esto es un texto. tiene varias oraciones. todas son distintas, ninguna es igual.”

Normalizar • Formas de hacerlo

- **Tokenizar:** pasar de un único string de texto a una lista de strings de oraciones.

“esto es un texto. tiene varias oraciones. todas son distintas, ninguna es igual.”

[“esto es un texto.”, “tiene varias oraciones.”, “todas son distintas, ninguna es igual.”]

nltk.tokenize.sent_tokenize(texto)

Normalizar • Formas de hacerlo

- **Tokenizar palabras:** pasar de un único string de una oración a una lista de strings de Tokens (palabras, puntuaciones, símbolos).

“esto es un #hastag.” → [“esto”, “es”, “un”, “#”, “hashtag”, “.”]

nltk.tokenize.word_tokenize(texto)

Normalizar • Formas de hacerlo

- **Limpiar caracteres:** nos quedamos sólo con los caracteres de interés. Esto dependerá de nuestro problema en particular. En nuestro caso vamos a utilizar la librería ‘re’, que nos permite modificar texto.

[“esto es un #hashtag.”] → [“esto es un hashtag”]

```
import re
re.sub("[^a-zA-Z]"," ",str(texto))
```

Normalizar • Formas de hacerlo

- **Llevar a raíz:** buscamos llevar palabras distintas con significados similares a una forma común.

- **Opción 1: Stemmizer:** Logra esto recortando las palabras mediante un proceso heurístico. Es rápido y fácil de usar, pero a veces no es certero.

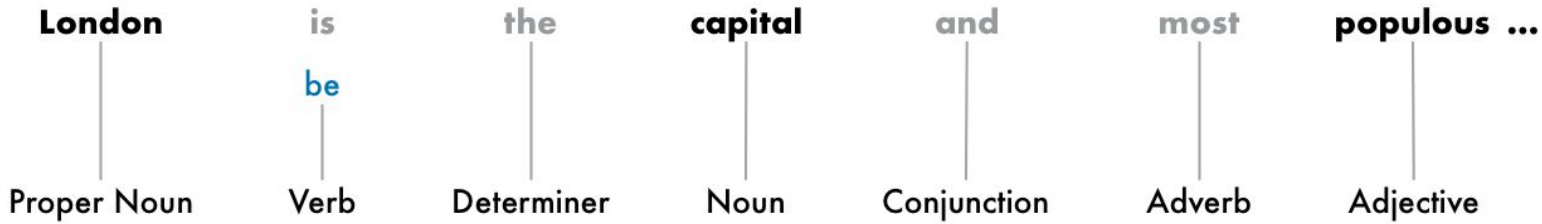
[“starting”, “wants”, “repartitions”, “america’s”] → [“start”, “want”, “repar”, “america”]

```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
stemmer.stem(palabra)
```

Normalizar • Formas de hacerlo

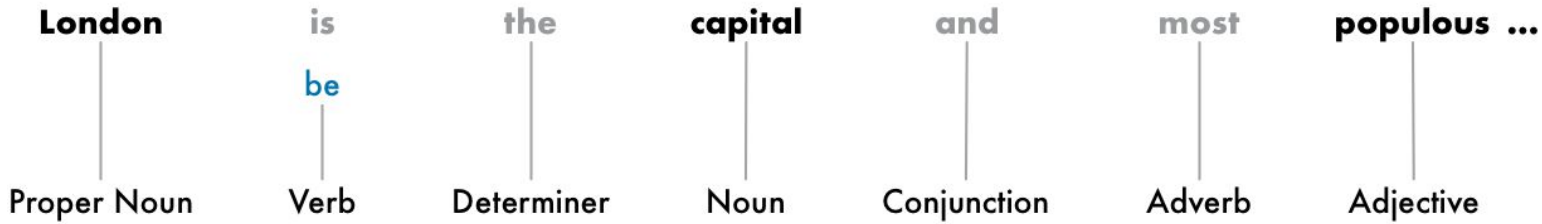
- **Llevar a raíz:** buscamos llevar palabras distintas con significados similares a una forma común.

- **Opción 1: Lemmatizer:** Logra esto utilizando un vocabulario y realizando un análisis morfológico de las palabras. Precisa que además de la palabra se le informe cual es la función de la palabra en el texto



Normalizar • Formas de hacerlo

- **Opción 1: Lemmatizer:** Logra esto utilizando un vocabulario y realizando un análisis morfológico de las palabras. Precisa que además de la palabra se le informe cual es la función de la palabra en el texto



Para determinar la función de la palabra automáticamente nos ayudamos con la **función 'nltk.pos_tag'.** A esta función se le llama POS (Part of Speech)

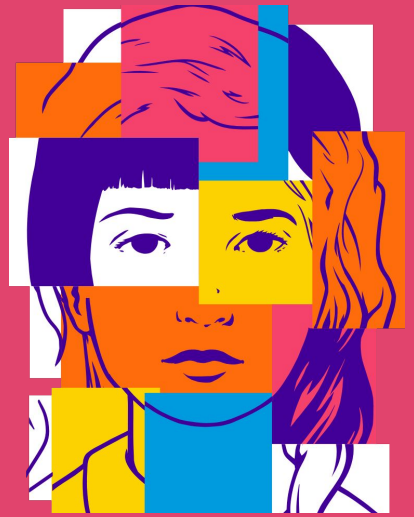
Normalizar • Formas de hacerlo

[“was”, “running”, “hours”] ⟶ [“be”, “run”, “hour”]

```
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
wordnet_lemmatizer.lemmatize(palabra,
get_wordnet_pos(palabra)
```

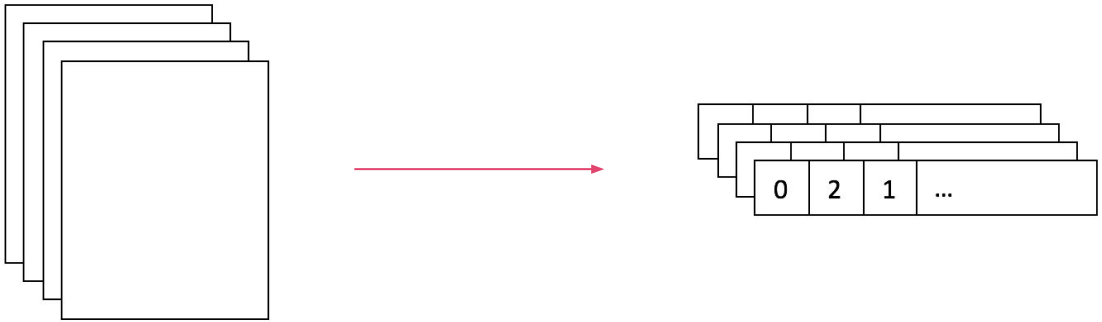
Es más preciso que el Stemmizer, pero lleva más tiempo y su performance depende de la precisión con la que le pasemos los POS.

Vectorización



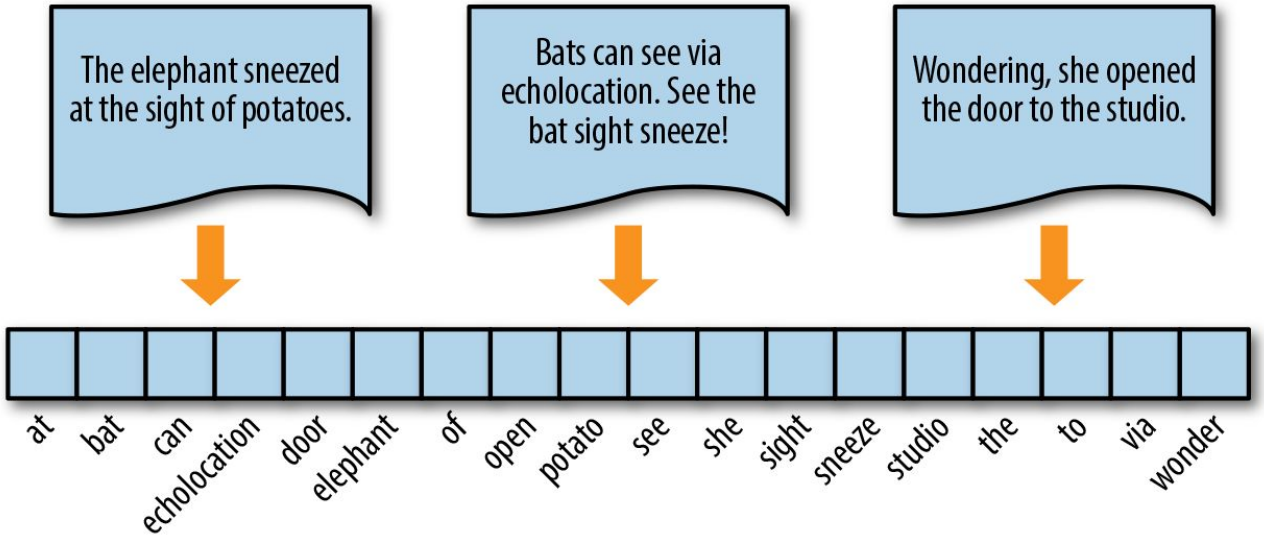
Vectorizar

Objetivo: Representar cada texto (instancia de la base de datos) como un vector que podamos usar como vector de features para entrenar una de los modelos



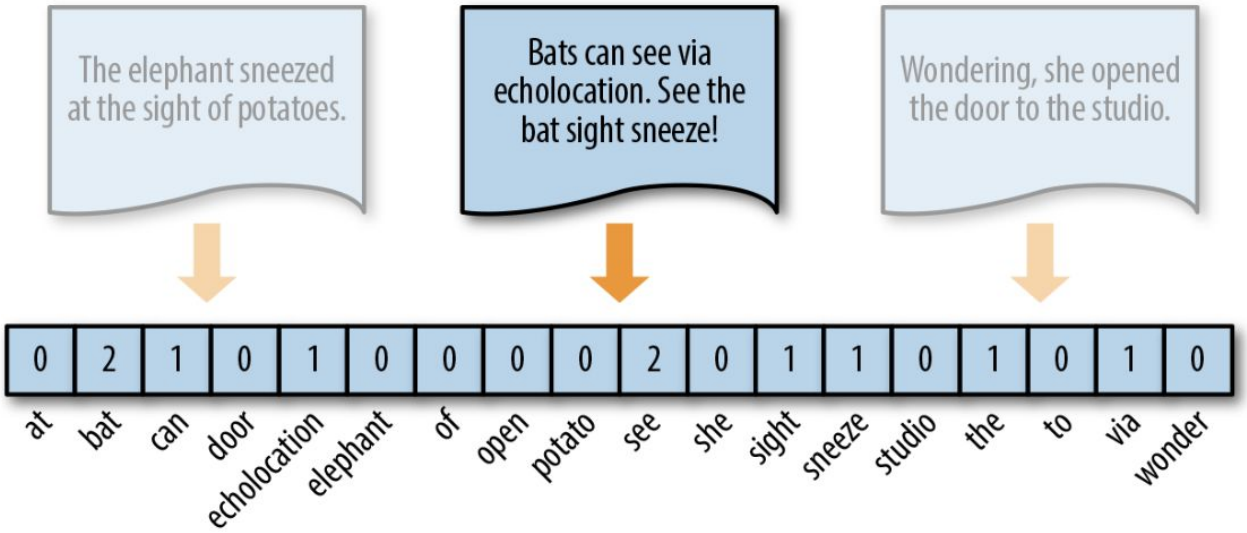
Vectorizar • Bag of words

Idea: Generar un vector que represente todas las palabras del corpus. Representar cada instancia como un vector con la cantidad de veces que aparecen las palabras.



Vectorizar • Bag of words

Idea: Generar un vector que represente todas las palabras del corpus. Representar cada instancia como un vector con la cantidad de veces que aparecen las palabras.



Vectorizar • Bag of words

Para implementar esto utilizamos una función de sklearn llamada CountVectorizer:

```
from sklearn.feature_extraction.text import CountVectorizer
```

Vectorizar • Bag of words

Para implementar esto utilizamos una función de sklearn llamada CountVectorizer:

from skle orizer

Problema: la cantidad de palabras en la base de datos suele ser muy grande. No conviene tener tantos features.

Solución (por ahora)

Utilizamos sólo las palabras que aparecen una mayor cantidad de veces en el texto, o que aparecen en un mayor número de instancias.

¡BREAK!

Ph. Credit: Drew Coffmann

Hands-on training

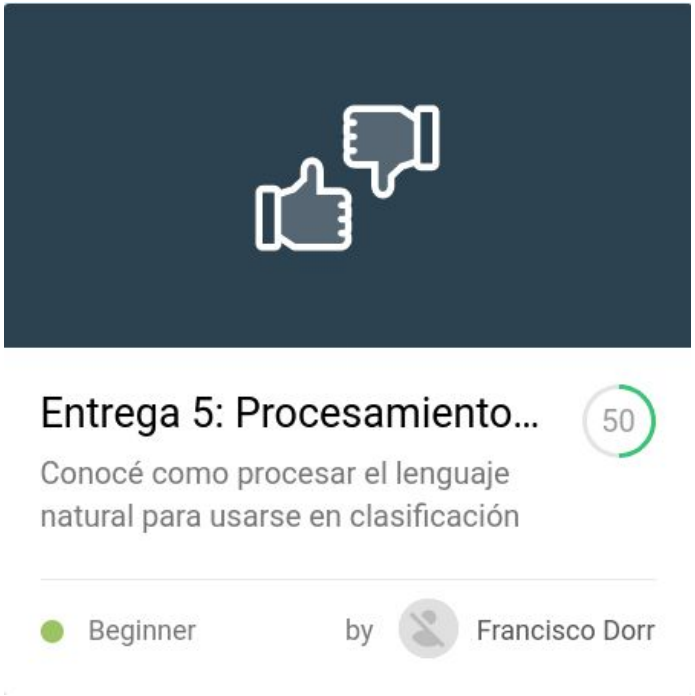
Hands-on training

DS_Encuentro_34_NLTK.ipynb

Parte 1 a 3

Proyecto 2:
Lanzamiento
Entrega 05

Proyecto 2: Modelado (Entrega 05)



1. Bajar los materiales.
2. Leer la Checklist
3. ¡Empezar a trabajar en la entrega!

Recursos

Recursos

NLP

Introducción corta a NLP, accesible y gráfica (con muy poca matemática):
https://towardsdatascience.com/introduction-to-natural-language-processing-for-text-df845750fb63

Otra parecida, pero más enfocada en la parte lingüística:
https://medium.com/@gon.esbuyo/get-started-with-nlp-part-i-d67ca26cc828

Bibliografía extendida sobre NLP con NLTK (libro open access):
Natural Language Processing with Python - https://www.nltk.org/book/

Recordatorio sobre condiciones de aprobación

Proyectos • Condiciones de aprobación

Los/as evaluadores/as considerarán una entrega como **Aprobada** cuando el/la estudiante haya cumplido satisfactoriamente el 100% de los puntos que pide el checklist (aunque no los haya hecho a todos perfectos).

Caso contrario, el/la evaluador/a considerará la entrega como **Para rehacer.**

Nota: no hay un límite de iteraciones (el/la estudiante puede tener que rehacer su trabajo todas las veces que sea necesario hasta aprobar).

Para la próxima

1. Terminar de ver los videos de la plataforma “Procesamiento del Lenguaje Natural”

2. Trabajar en la entrega 05.

3. Completar Notebook de hoy y atrasados.

ACÁMICA