

DATA SCIENCE

Machine Learning

DS-BUE-8/ Clase 20



Hoja de ruta de hoy

19:10	Repaso
19:30	Puesta en común NLP
20:20	Break
20:35	NLP (continuación...!)
21:50	Cierre y actividad para próximo encuentro

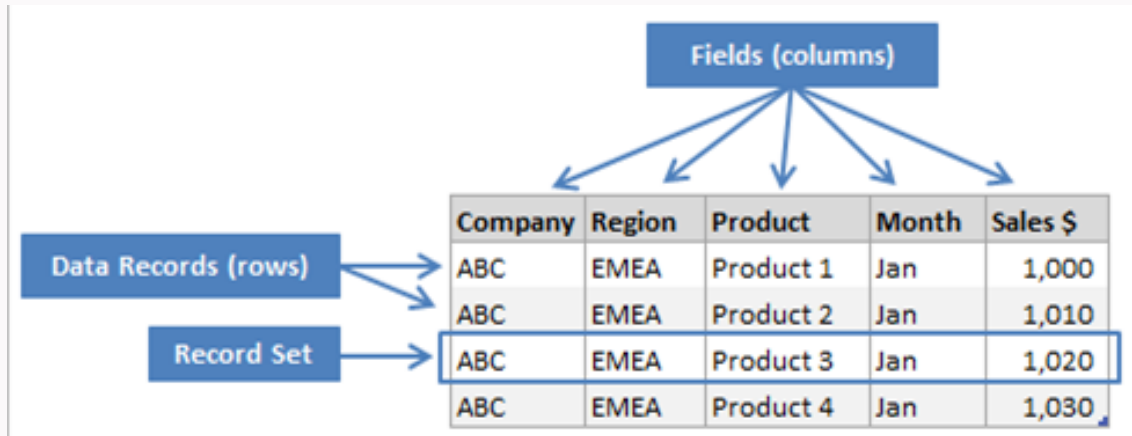


Procesamiento de Lenguaje Natural



NLP

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

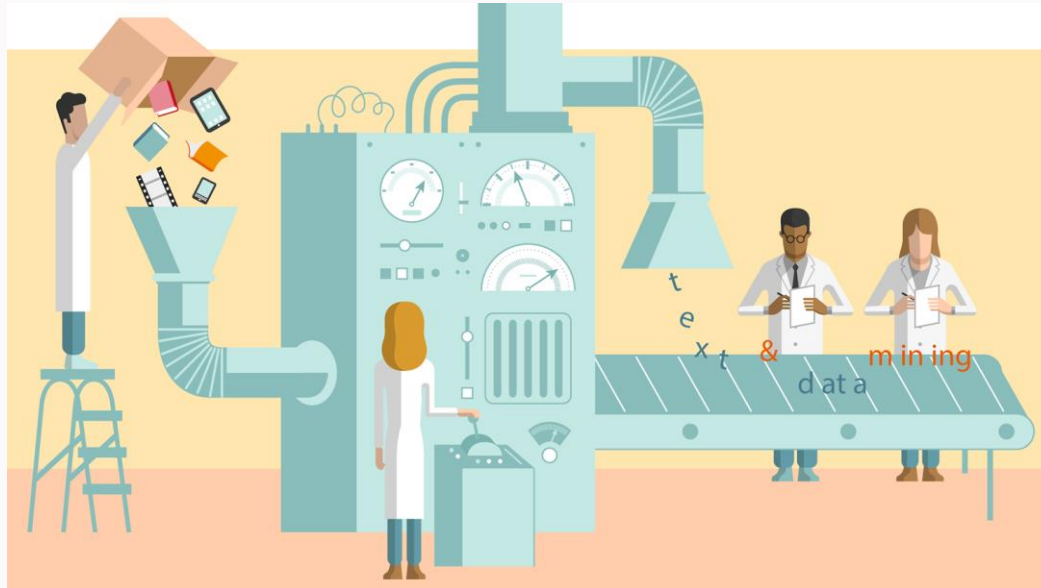


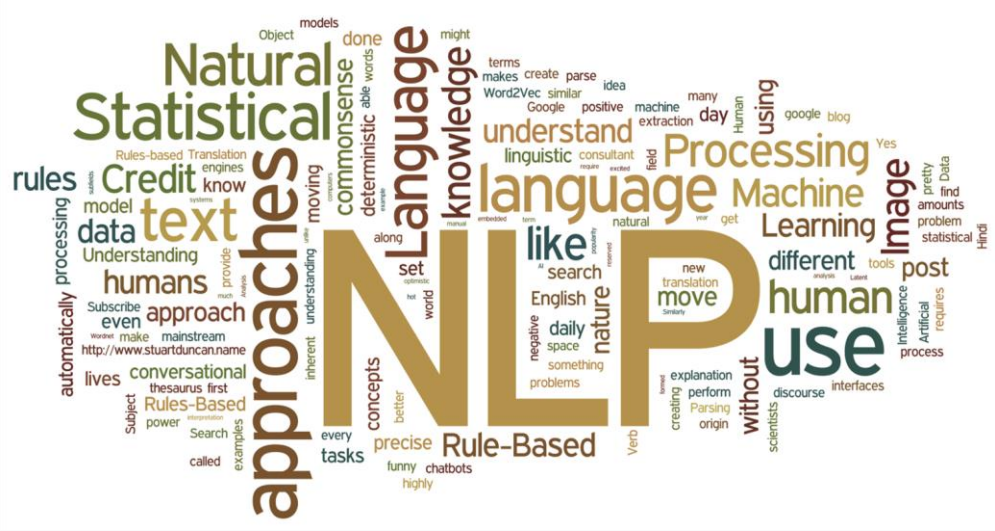
Problema: Hay muchísimos datos disponibles en forma de lenguaje natural (texto, no estructurado) que contienen información relevante.



NLP

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

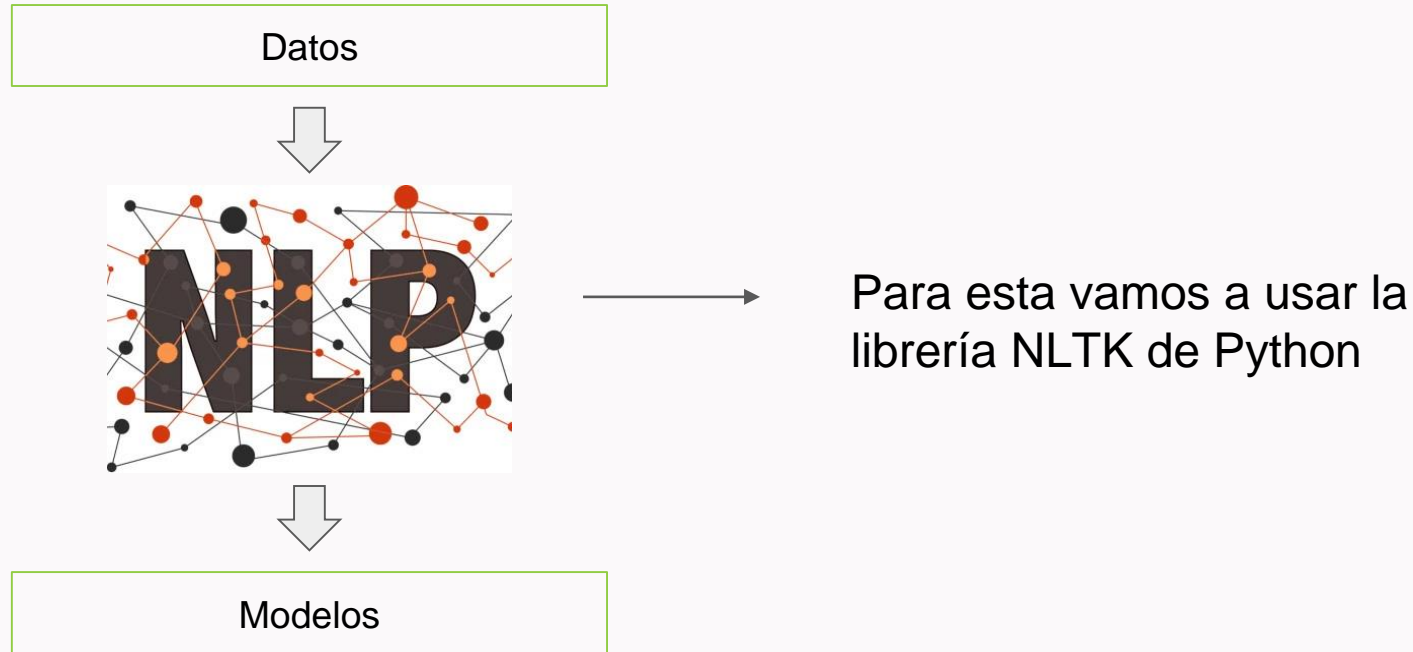




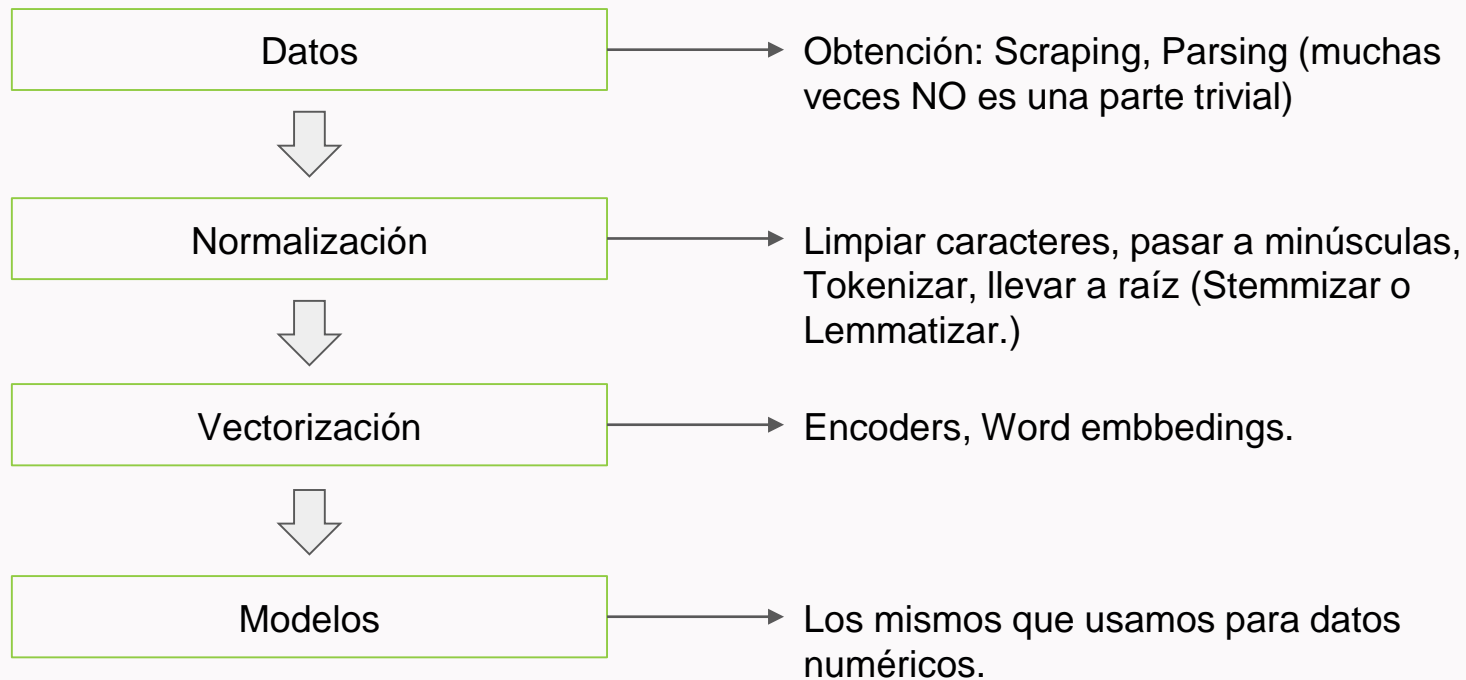
Solución: El procesamiento de lenguaje natural es una rama de la inteligencia artificial que se enfoca en permitirle a las computadoras entender y procesar lenguaje natural.



NLP - Flujo de Trabajo



NLP - Flujo de Trabajo



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

Buscamos llevarlo a una forma común



querer pasear mi perro palermo



Normalizar - Pasar a minúsculas

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 - Tokenizar

Tokenizar oraciones: Pasar de un único string de texto a una lista de strings de oraciones.

```
nltk.tokenize.sent_tokenize(texto)
```

**“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.”]**



Normalizar - Tokenizar

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

```
nltk.tokenize.word_tokenize(texto)
```

“esto es un #hastag.”



**[“esto”, “es”, “un”,
“#”, “hashtag”, “.”]**



Normalizar - Limpiar caracteres

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.

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

[“esto es un #hashtag.”]



[“esto es un hashtag”]



Normalizar - Llevar a raíz

Idea: Buscamos llevar palabras distintas con significados similares a una forma común.

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

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

["starting", "wants",
"repartitions", "america's"]

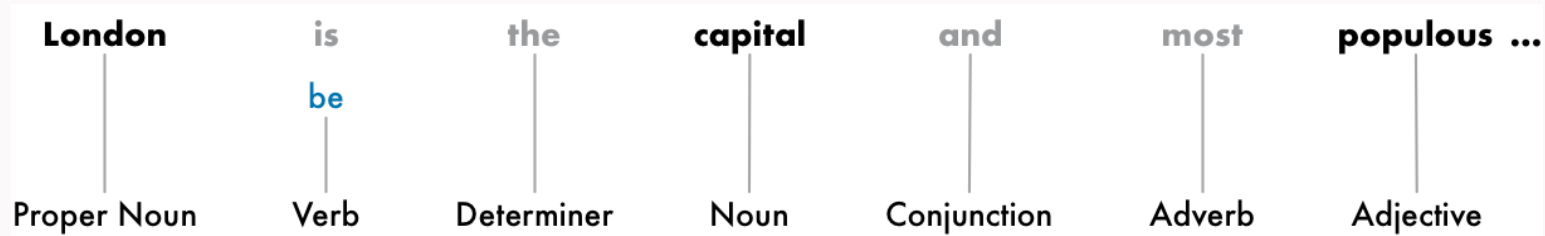


["start", "want", "repar",
"america"]



Normalizar - Llevar a raíz

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 - Llevar a raíz

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

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

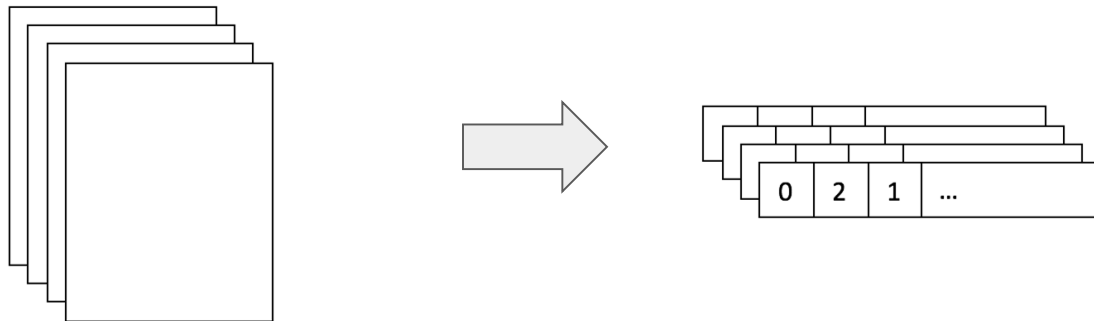
[“was”, “running”, “hours”]  [“be”, “run”, “hour”]

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



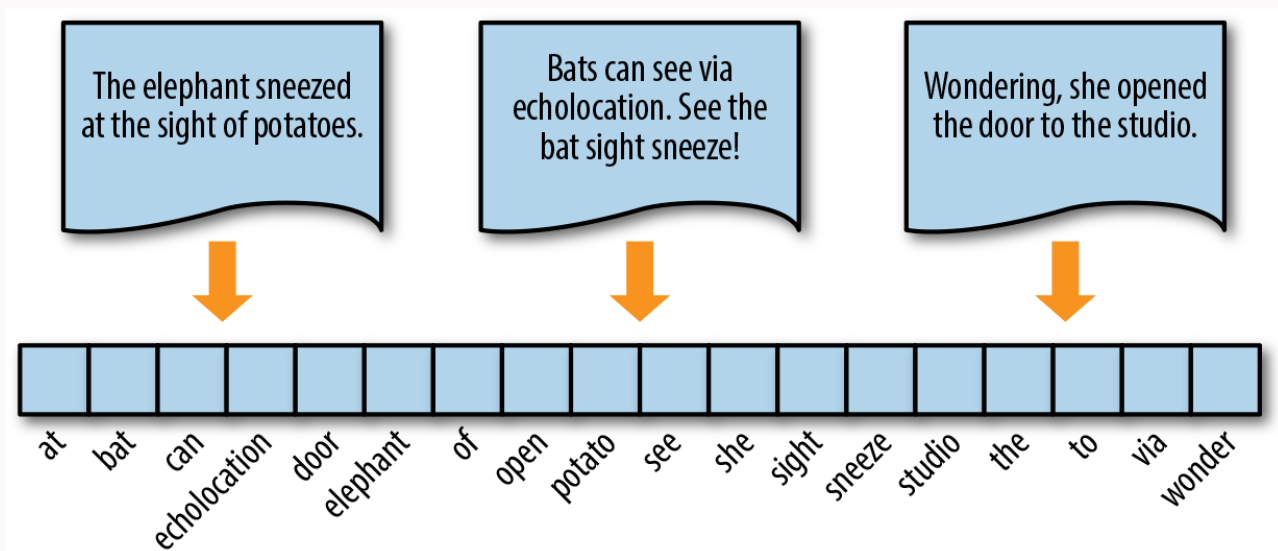
Vectorizar

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



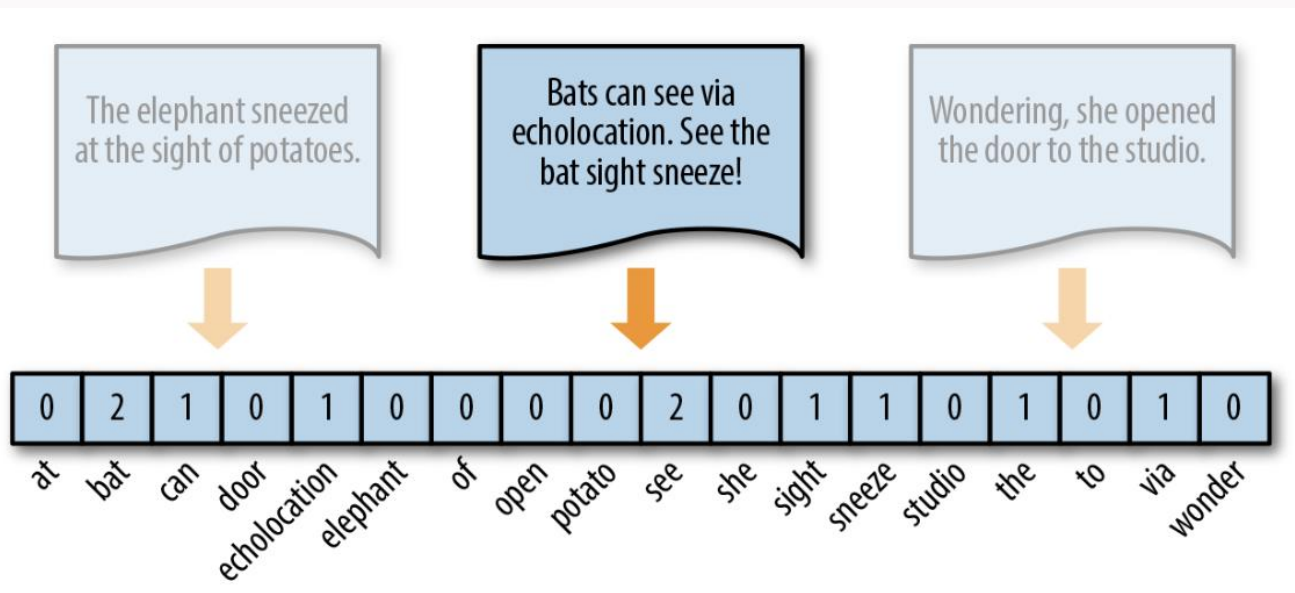
Vectorizar - Bag of Words

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
```

Problema: La cantidad de palabras en a 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.



Vectorizar - Bag of Words con N-gramas

Problema: Hay palabras que cobran sentido cuando se las agrupa con otras, ejemplos: “Plaza Italia” y “Control Remoto” .



Vectorizar - Bag of Words con N-gramas

Problema: Hay palabras que cobran sentido cuando se las agrupa con otras, ejemplos: “Plaza Italia” y “Control Remoto” .

Solución: Además de cada palabra por separado, agregamos los grupos de 2 (ó N) palabras contiguas a nuestro vector de Features.

Para implementar esto usando CountVectorizer:

```
CountVectorizer(max_features=max_features, stop_words="english"  
                , ngram_range=(1, 2))
```



Vectorizar - Bag of Words con N-gramas

Problema: Hay palabras que cobran sentido cuando se las agrupa con otras, ejemplos: “Plaza Italia” y “Control Remoto” .

Solución: Además de cada palabra por separado, agregamos los grupos de 2 (ó N) palabras contiguas a nuestro vector de Features.

Para implementar esto usando CountVectorizer:

```
CountVectorizer(max_features=max_features, stop_words="english"  
                , ngram_range=(1, 2))
```



Ojo con la cantidad
de Features



Vectorizar - TF-IDF

Observación: Si buscamos diferenciar cada documento por las palabras que lo componen, las palabras que están en todos ellos no aportan información.



Vectorizar - TF-IDF

Observación: Si buscamos diferenciar cada documento por las palabras que lo componen, las palabras que están en todos ellos no aportan información.

Idea: Hay que medir no sólo cuanto aparece una palabra en una instancia (documento), sino también qué tan frecuente es esa palabra en todo el corpus.



Term Frequency - Inverse Document Frequency
TF - IDF



Vectorizar - TF-IDF

Term Frequency: Frecuencia de una palabra (*term*) en una instancia o documento (*doc*).

$$\mathbf{TF}(term, doc) = \frac{\# \text{ de veces que el } term \text{ aparece en el } doc}{\# \text{ de } terms \text{ diferentes en el } doc}$$



Vectorizar - TF-IDF

Term Frequency: Frecuencia de una palabra (*term*) en una instancia o documento (*doc*).

$$\mathbf{TF}(term, doc) = \frac{\# \text{ de veces que el } term \text{ aparece en el } doc}{\# \text{ de } terms \text{ diferentes en el } doc}$$

Ejemplo en un documento:

$\overbrace{0.125}$
Hello, my $\overbrace{0.125}$ name is $\overbrace{0.375}$ Brandon. Brandon Brandon. The $\overbrace{0.125}$ elephant $\overbrace{0.125}$ jumps over the $\overbrace{0.125}$ moon.



Vectorizar - TF-IDF

Document Frequency (DF): Fracción de todos los documentos en nuestro corpus que contienen el término.

$$\text{DF}(term, corpus) = \frac{\# \text{ de docs que contienen } term}{\# \text{ total de docs}}$$

Inverse Document Frequency (IDF): Logaritmo de la inversa de DF.

$$\text{IDF}(term, corpus) = \log\left(\frac{\# \text{ total de docs}}{\# \text{ de docs que contienen } term}\right)$$

Ejemplo: Si está en todos los docs $\log(N/N) = \log(1) = 0$



Vectorizar - TF-IDF

Inverse Document Frequency (TF-IDF): Producto del valor de TF por el de IDF.

$$\mathbf{TF-IDF}(term, corpus, doc) = \mathbf{TF}(term, doc) \times \mathbf{IDF}(term, corpus)$$



Vectorizar - TF-IDF

Inverse Document Frequency (TF-IDF): Producto del valor de TF por el de IDF.

$$\text{TF-IDF}(term, corpus, doc) = \text{TF}(term, doc) \times \text{IDF}(term, corpus)$$



Cada palabra tiene un valor asociado en cada documento, con esto formamos nuestro vector (no necesariamente serán valores enteros):

0	0.2	0.5	0	0.3	0	0	0	0	2	0	0.1	1	0	1	0	1	0
at	bat	can	door	echolocation	elephant	of	open	potato	see	she	sight	sneeze	studio	the	to	via	wonder

