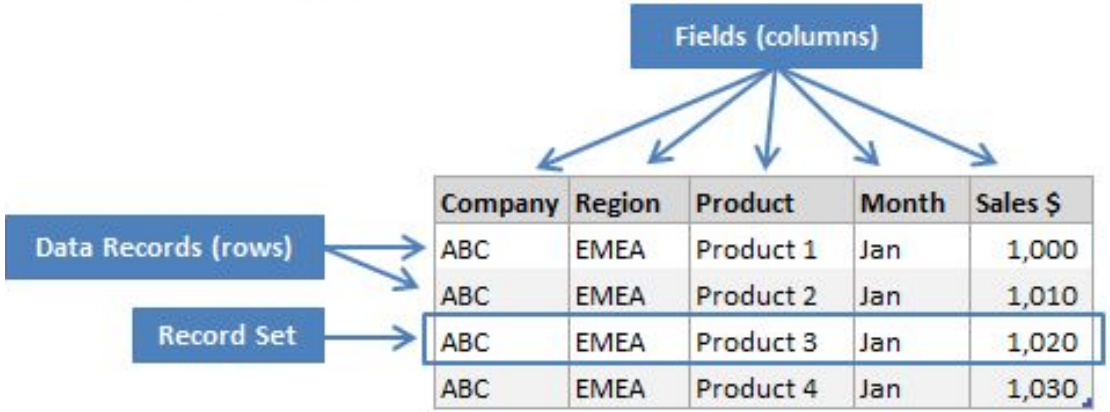
TEMA DEL DÍA

Preprocesamiento del lenguaje natural

Relativamente fácil para los humanos, no tanto para las computadoras.

Pero aún antes de entrenar modelos, hay muchísima información a la que podemos acceder usando las técnicas de NLP.

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

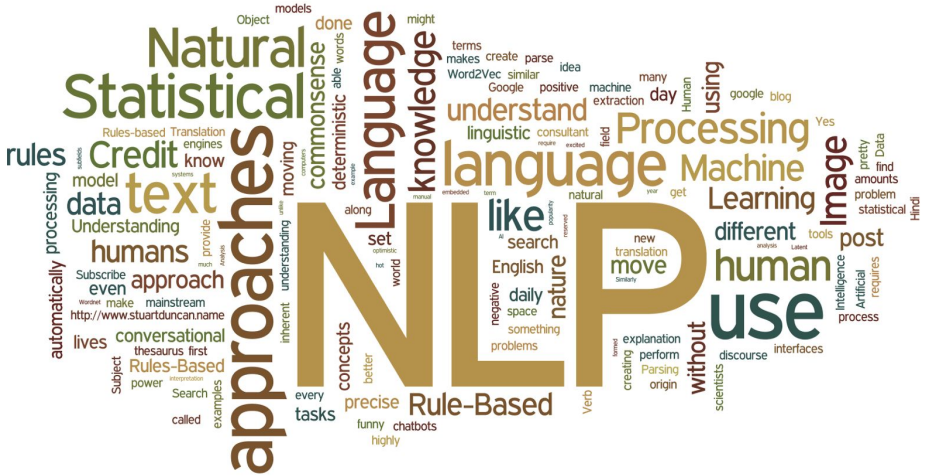


Sabemos trabajar con datos **estructurados** (tablas y números).

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

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

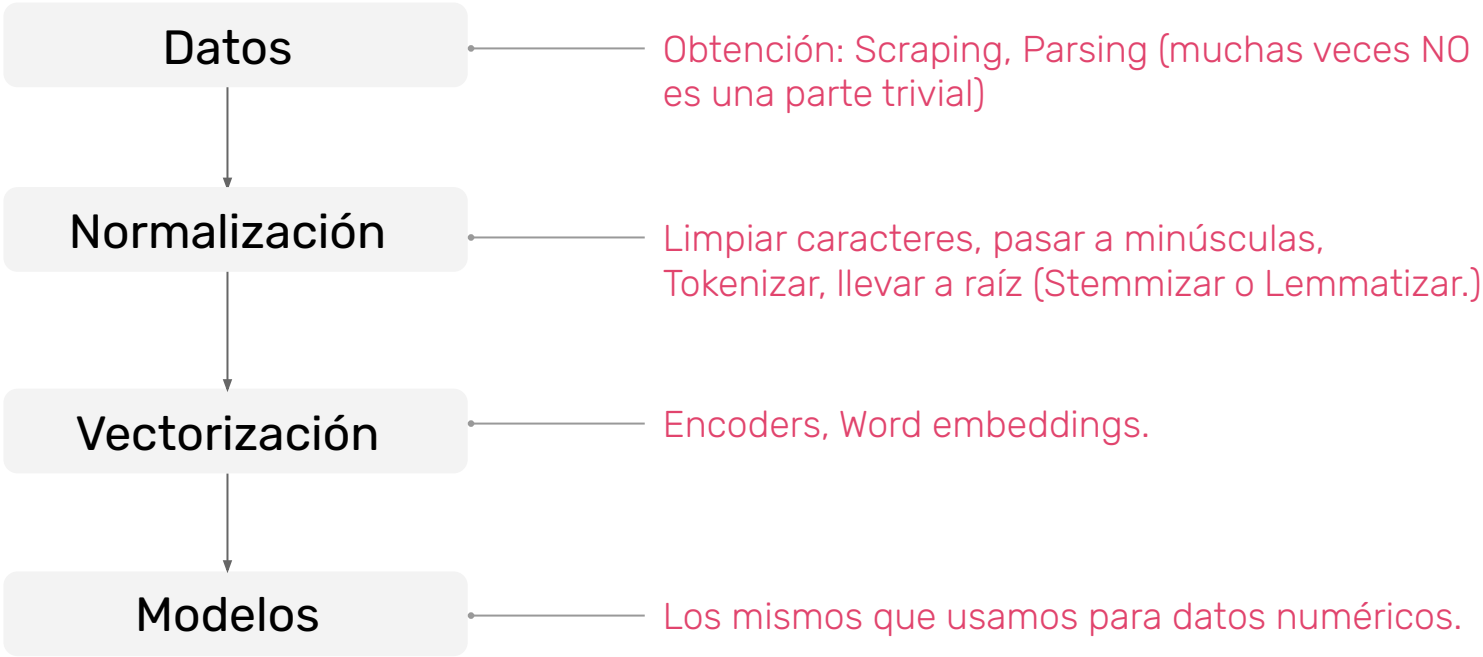
NLP • Flujo de trabajo

Datos

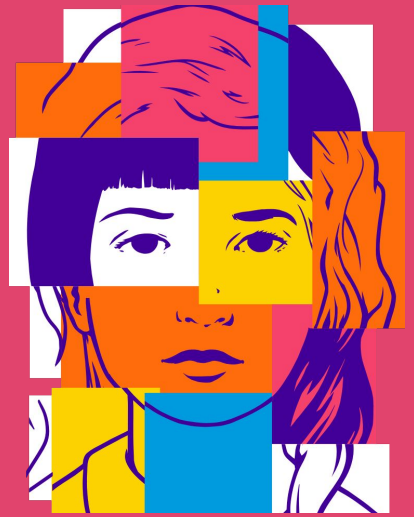
Modelos

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.

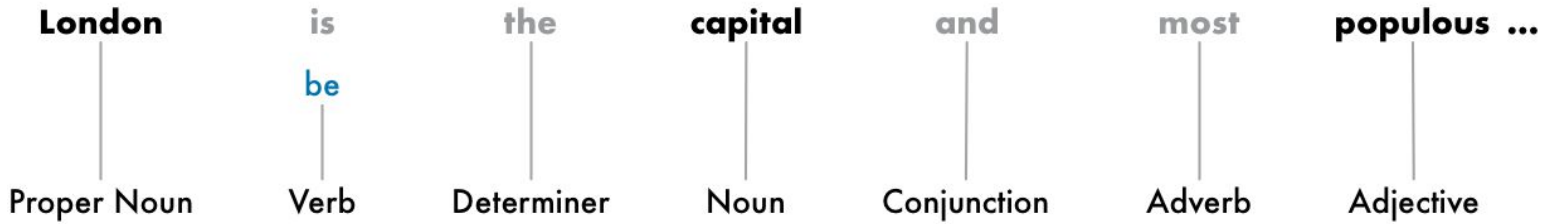
["starting", "wants", "repartitions", "america's"] → ["start", "want", "repar", "america"]

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

Normalizar • Formas de hacerlo

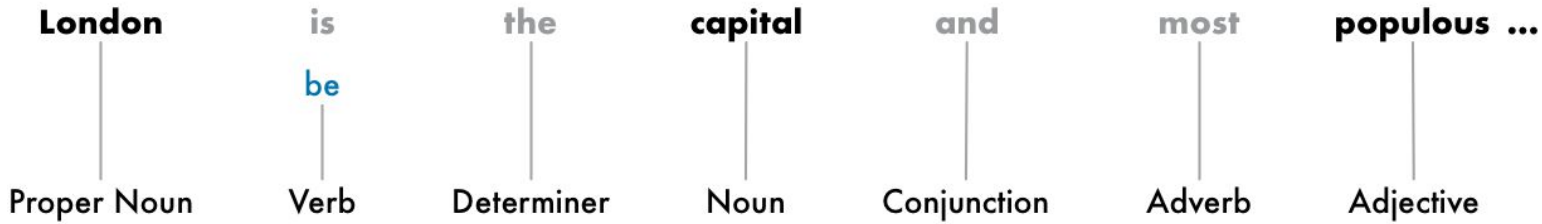
- **Llevar a raíz:** buscamos llevar palabras distintas con significados similares a una forma común.

- **Opción 2: 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 2: 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)

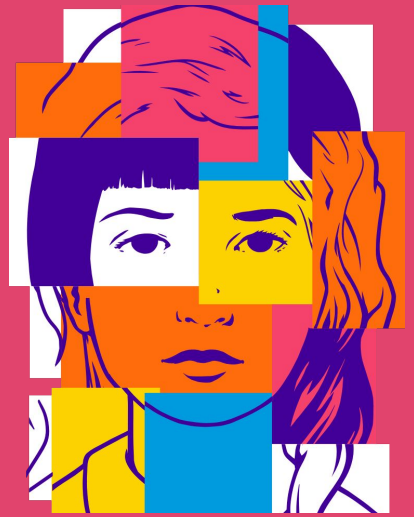
Normalizar • Formas de hacerlo

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

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

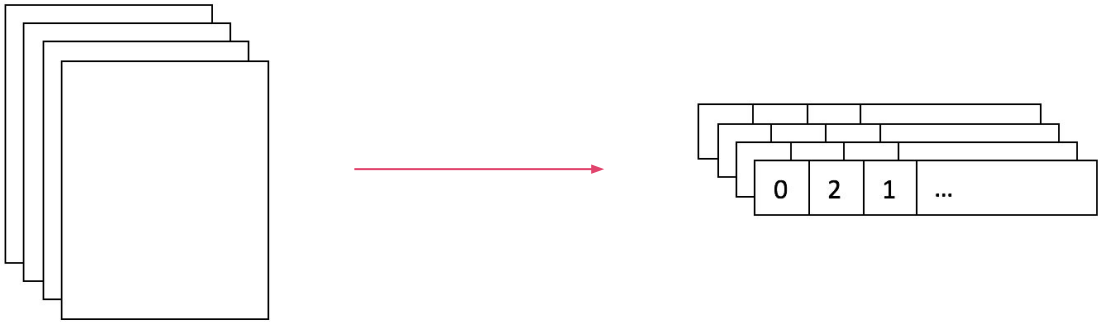
Es más preciso que el Stemmizer, pero lleva más tiempo y su performance depende de la precisión con la que le pasemos los POS.

Vectorización



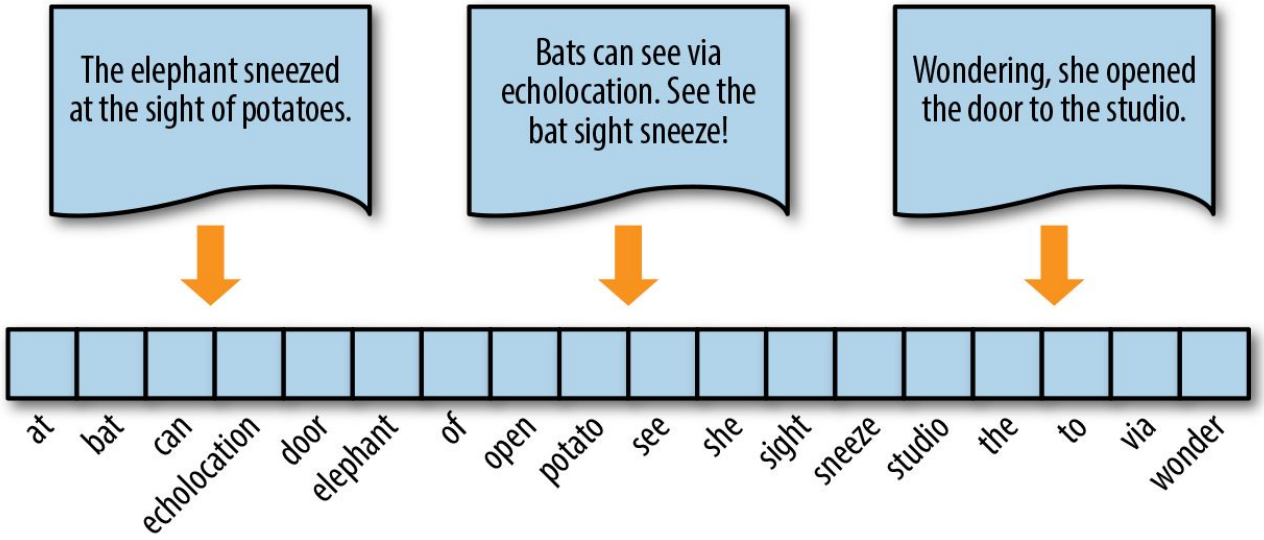
Vectorizar

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



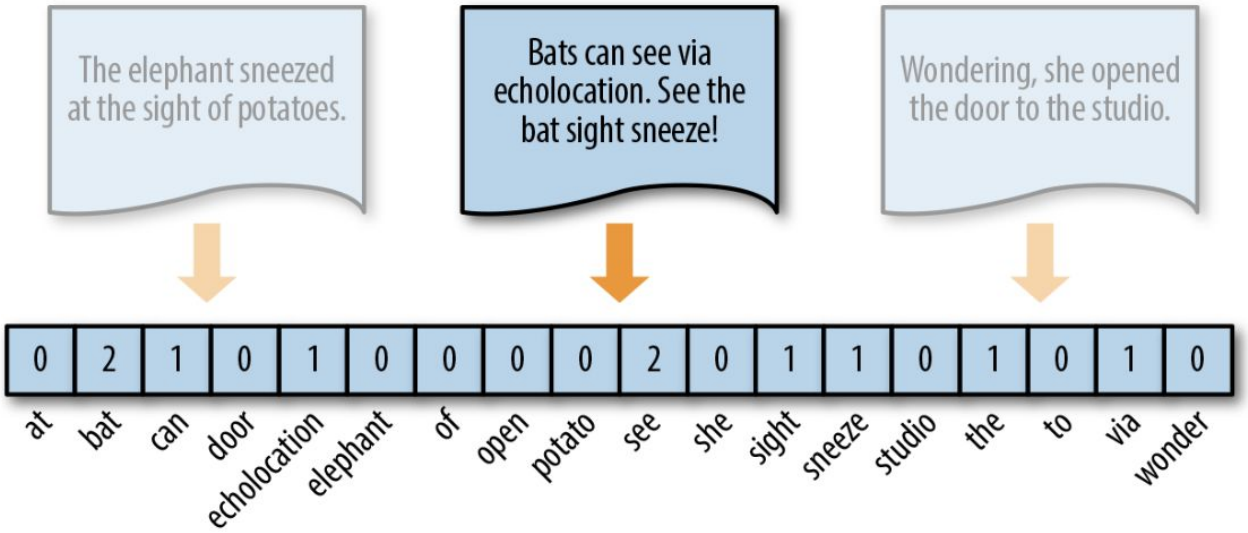
Vectorizar • Bag of words

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 skl 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.

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

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

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*).

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

Vectorizar • TF - IDF

Term Frequency

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

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

Ejemplo en un documento:

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

Vectorizar • TF - IDF

Document Frequency

Fracción de todos los documentos en nuestro corpus que contienen el término.

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

Ejemplo en un documento:

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

Vectorizar • TF - IDF

Inverse Document Frequency

Logaritmo inversa de DF.

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

Ejemplo: si está en todos los docs log(N/N) = log(1) = 0

Vectorizar • TF - IDF

Inverse Document Frequency

Producto del valor de TF por el de IDF.

TF-IDF(*term*,*corpus*,*doc*) = **TF**(*term*,*doc*) X **IDF**(*term*,*corpus*)

Vectorizar • TF - IDF

Inverse Document Frequency

Producto del valor de TF por el de IDF.

TF-IDF(*term*,*corpus*,*doc*) = **TF**(*term*,*doc*) X **IDF**(*term*,*corpus*)

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