slides

December 22, 2020

#

Métodos de aprendizaje automático para análisis de textos

Víctor Muñiz Sánchez

Diciembre 2020

1 Sobre el curso

1.1 Objetivos:

- Mostrar los conceptos básicos de Procesamiento de Lenguaje Natural (NLP) orientado a textos.
- Mostrar repersentaciones vectoriales útiles de textos a partir de modelos probabilísticos de lenguaje y modelos neuronales de lenguaje.
- Abordar modelos de aprendizaje supervisado y no supervisado utilizando métodos de Machine Learning y Deep Learning, dando especial énfasis en arquitecturas de aprendizaje profundo y diversas aplicaciones.

2 Temario

2. Conceptos básicos de python para el curso
3. Introducción y conceptos básicos de NLP
4. Representaciones básicos de textos: one-hot encoding, modelo n-gram, bolsa de palabras y TF-IDF.
5. Embeddings para palabras y documentos basados en modelos neuronales de lenguaje
6. Modelos de ML y aplicaciones en textos
7. Modelos de DL para textos: redes convolucionales, recurrentes y aplicaciones
8. Arquitecturas avanzadas de DL: sequence to sequence y mecanismos de atención (si da tiempo…)

3 Introducción

NLP (Jurafsky & Martin, Speech and Language Processing, 2nd. Ed. Es un campo de estudio enfocado en la interacción entre **lenguaje humano** y computadoras. Se encuentra en la intersecciónn de ciencias de la computación, inteligencia artificial y linguistica computacional.

El objetivo es que las computadoras, realicen tareas útiles que involucren lenguaje humano, como comunicación máquina-humano, mejorar la comunicación humano-humano o simplemente, realizar procesamiento útil de texto o discurso.

Concepto clave: **lenguaje humano**:

- Signos linguísticos
- Signos gráficos (textual)
- Secuencias sonoras
- Gestos y señales

Nosotros hablaremos sobre textos

NLP es un área bastante compleja. Esto se debe principalmente, a que el lenguaje natural es complejo en sí:

- Altamente ambiguo
- Utiliza procesos mentales complejos para obtener un significado (uso del entorno)
- Considera diferentes tipos de "entradas": texto, audio, imágenes, expresiones faciales y corporales, otras representaciones pictóricas
- Resultados en tiempo real (machine translation, automatic answering, etc...).
- En constante evolución

También es un área bastante amplia.

¿Cuántas tareas/aplicaciones de NLP para textos conoces?

Veamos lo que dice Wikipedia

[20]:
```
import wikipedia
wikipedia.set_lang("en")
print(wikipedia.search("natural language processing"))
```

```
['Natural language processing', 'History of natural language processing',
'Natural language', 'Natural-language understanding', 'Outline of natural
language processing', 'Natural-language user interface', 'Natural Language
Toolkit', 'Process', 'List of artificial intelligence projects', 'GPT-3']
```

[ ]:
```
nlp = wikipedia.page('Natural language processing')
print(nlp.content)
```

Independientemente de la tarea o aplicación, necesitamos una **representación eficiente** del texto.

...it is not enough to simply provide a computer with a large amount of data and expect it to learn to speak –the data has to be prepared in such a way that the computer can more easily find patterns and inferences."

- James Pustejovsky et al. *Natural Language Annotation.*

Aquí es donde empezamos con la parte técnica...

4 Conceptos básicos de programación para el curso

4.1 Lo que necesitamos saber sobre `python` para éste curso

4.2 ¿Porqué `python`?

- Es un lenguaje interpretado (no compilado, como C, C++) y de alto nivel, como Matlab, R o Java.
- Es un lenguaje orientado a objetos
- Es el lenguaje de programación de mayor crecimiento en la última década
- Es el lenguaje científico de mayor uso para aplicaciones de análisis y ciencia de datos, desplazando a R y Matlab
- Es el lenguaje orientado a Deep Learning
- Es "*fácil*" de aprender
- Es `wrapper` de muchas librerias de bajo nivel (C, CUDA-NVIDIA) implementadas en paralelo, lo que mejora el rendimiento

4.2.1 Python y la infraestructura para cómputo científico y ciencia de datos

Veremos algunas cosas básicas y la mayoría, las veremos sobre la marcha.

4.3 Instalación local.

- La opción más óptima: instalar python (https://www.python.org/) y los editores que sean de su preferencia (`vi`, `emacs`, `pycharm`, `spyder`, `jupyter-notebook`, etc).
- La opción más rápida (recomendada para iniciar), instalar Anaconda: https://www.anaconda.com/
- En cualquier caso, recomiendo ampliamente crear **virtual environments** para proyectos específicos que requieran a su vez, librerías específicas

4.4 Instalar librerías

En Anaconda, se puede hacer directamente en el framework. También puedes hacerlo en consola con los comandos `pip` y `conda` (si tienes Anaconda).

4.4.1 Ejecución de código python

- The Python interpreter
- The IPython interpreter
- Self-contained Python scripts
- The Jupyter notebook

Jupyter Notebook *Jupyter Project*:

- A useful hybrid of the interactive terminal and the self-contained script
- A document format that allows executable code, formatted text, graphics, and even interactive features to be combined into a single document.
- Useful both as a development environment, and as a means of sharing work via rich computational and data-driven narratives that mix together code, figures, data, and text.

4.4.2 Jupyter + Google Colab

¿Qué es Colaboratory? (del sitio oficial de Google Colab)

Colaboratory, también llamado Colab, te permite escribir y ejecutar código de Python en un navegador, con las siguientes particularidades: - Sin configuración requerida - Acceso gratuito a GPU y TPU - Facilidad para compartir

Seas estudiante, científico de datos o investigador de IA, Colab facilita tu trabajo. Mira este video introductorio sobre Colab para obtener más información.

En este curso, usaremos Jupyter Notebooks y Colab como nuestra plataforma de programación y procesamiento.

¿Qué necesitamos?

- Una cuenta de Google
- Una carpeta dentro de Google Drive para el curso (te ayudará a tenerlo mejor organizado)

El uso de Colab es muy intuitivo, y también hay varios tutoriales para aprender a usarlo. Por ejemplo: * An Absolute Beginner's Guide To Google Colaboratory. * Getting Started With Google Colab

El Github para el curso es https://github.com/victorm0202/Curso_banxico/tree/Python-basics

4.5 Python basics

Lo veremos directamente en Colab.

Algo que quizá pueda ayudar: - Python for R users - Python for SAS users

5 Procesamiento de textos

Volviendo al tema...

Los textos se obtienen de distintas fuentes, tienen diferentes características y su análisis depende en gran parte de tales características.

El procedimiento es como siempre:

- Obtención del texto
- Preprocesamiento
- Representación
- Modelación

En `python` hay varios módulos especializados. Nosotros usaremos varios: - NLTK - Scikit Learn - Keras

Preprocesamiento y análisis básicos de textos.

Lo veremos en el notebook

El preproceso y normalización (canonicalización) de texto es una parte muy importante para su posterior modelación y análisis.

Este puede incluir alguno de los siguientes operaciones:

- Convertir letras a minúsculas o mayúsculas
- Remover números o convertirlos a palabras
- Remover signos de puntuación, acentos y otros signos diacríticos
- Remover caracteres repetidos, incluídos espacios en blanco
- Remover caracteres con poca frecuencia
- Remover palabras funcionales o stop words
- Convertir símbolos a palabras (emojis y otros)
- Stemming, lematización y POS-tagging

Sin embargo, el preproceso y en general, la **extracción** del texto puede ser mucho más complejo.

Formalidad de la escritura. The formality continuum (Choudhyrt M. *NLP for social media*)

Datos de la web:

Digitalización y reconocimiento de caracteres:

Textos generados por usuarios:

Textos generados por usuarios:

Una gran variedad de módulos para preproceso y normalización están disponibles en `python`. Ver por ejemplo:

Módulos para procesamiento de textos

Nosotros usaremos preprocesamientos básicos. Hay dos que son relevantes para ciertas aplicaciones:

- Stemming
- Lematización

5.1 Stemming

Proceso heurístico para, secuencialmente, remover las partes finales de una palabra (affixes) para obtener su "raíz". Es una forma básica para hacer análisis morfológico. El algoritmo mas usado es el de Porter (Porter, M. F. (1980). *An algorithm for suffix stripping.* Program, 14(3), 130–127).

Por ejemplo, considera las palabras

token	stemming
operate	oper
operating	oper
operates	oper
operation	oper
operative	oper
operatives	oper
operational	oper

5.2 Lematización

Análisis morfológico para obtener el *"lema"* de cada palabra, es decir, una forma normalizada de un conjunto de términos morfológicamente relacionados (como aparece generalmente en un diccionario).

token	stemming	lemma	POS
operate	oper	operate	VB
operating	oper	operate	VBG
operates	oper	operate	VBZ
operation	oper	operation	NN
operative	oper	operative	NN
operatives	oper	operative	NNS
operational	oper	operational	JJ

Hay diferentes formas de realizarlo. El más común es obtener su etiqueta gramatical (POS-tag) y luego obtener su lema en un diccionario. Dependiente del idioma.

5.3 Ejemplos de preprocesamiento

[6]:
```
# revisar las funciones, si tienes dudas, me preguntas...
from my_functions import *

preprocesador = preprocesaTexto(idioma='es', _tokeniza=False,␣
 ↪_muestraCambios=False, _quitarAcentos=True,
                                 _remueveStop=False, _stemming=False,␣
 ↪_lematiza=True, _removerPuntuacion=True)

txt = 'Sobre la mesa está el sobre'
txt_prep = preprocesador.preprocesa(txt)
print(txt_prep)
```

```
sobrar lo mesar este el sobrar
```

[10]:
```
preprocesador = preprocesaTexto(idioma='es', _tokeniza=False,␣
 ↪_muestraCambios=False, _quitarAcentos=True,
                                 _remueveStop=False, _stemming=False,␣
 ↪_lematiza=True, _removerPuntuacion=True)

corpus = ['Hola, me llamo Víctor, vivo en Monterrey, ¿y tú?', 'El perro se␣
 ↪comió mi tarea.',
          'Mi vecina se pelea con otra vecina.', 'El gato toca el piano.']
corpus_prep = []
for txt in corpus:
    txt_prep = preprocesador.preprocesa(txt)
    corpus_prep.append(txt_prep)
```

```
corpus_prep
```

[10]:
```
['hola me llamar victor vivir en monterrey y tu ',
 'el perro se comio mi tarea ',
 'mi vecino se pelear con otro vecino ',
 'el gato tocar el piano ']
```

6 Representaciones de textos

6.1 One-hot encoding

Dado un vocabulario, usamos variables *dummy* para indicar la presencia o ausencia de las palabras del vocabulario en algún documento.

El orden NO es importante

[68]:
```
import keras
from keras.preprocessing.text import Tokenizer
import pandas as pd

samples = ['El perro se comió mi tarea.', 'Mi vecina se pelea con otra vecina.
 ↪', 'El gato toca el piano.']

max_words = 15
tokenizer = Tokenizer(num_words=max_words)
# word index
tokenizer.fit_on_texts(samples)
# vectorizacion (one-hot u otro)
one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary') #"count",␣
 ↪"tfidf", "freq"

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
pos = list(word_index.values())
cols = ['0' for i in range(15)]
cols[pos[0]:pos[len(pos)-1]+1] = list(word_index.keys())
doc_matrix = pd.DataFrame(one_hot_results,columns=cols)
doc_matrix
```

```
Found 13 unique tokens.
```

[68]:

	0	el	se	mi	vecina	perro	comió	tarea	pelea	con	otra	gato	\
0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	

	toca	piano	0

```
0   0.0    0.0  0.0
1   0.0    0.0  0.0
2   1.0    1.0  0.0
```

6.2 Vector space model (Salton et al., 1975), también conocido como Bag of Words (BOW, Joachims 1998).

Detalle curioso:

Nunca usa el término *Bag of words*...

- Un documento es entonces, un punto $d \in \mathbb{R}^{|V|}$, donde V es un vocabulario. Documentos similares tenderán a tener palabras similares, y por lo tanto, vectores parecidos. En nuestro ejemplo, vectores rengl'on.
- De forma equivalente, una palabra es un punto $w \in \mathbb{R}^{|N|}$, donde N es el número de documentos del corpus. Palabras similares tenderán a ocurrir en documentos similares, entonces tendrán vectores parecidos. En nuestro ejemplo, vectores columna.

[59]:
```
import keras
from keras.preprocessing.text import Tokenizer
import pandas as pd

samples = ['El perro se comió mi tarea.', 'Mi vecina se pelea con otra vecina.
 ↪', 'El gato toca el piano.']

max_words = 15
tokenizer = Tokenizer(num_words=max_words)
# word index
tokenizer.fit_on_texts(samples)
# vectorizacion (one-hot u otro)
one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary') #"binary",␣
 ↪count", "tfidf", "freq"

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
pos = list(word_index.values())
cols = ['0' for i in range(15)]
cols[pos[0]:pos[len(pos)-1]+1] = list(word_index.keys())
doc_matrix = pd.DataFrame(one_hot_results,columns=cols)
doc_matrix
```

```
Found 13 unique tokens.
```

[59]:

	0	el	se	mi	vecina	perro	comió	tarea	pelea	con	otra	gato	\
0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	

```
   toca  piano    0
0   0.0    0.0  0.0
1   0.0    0.0  0.0
2   1.0    1.0  0.0
```

6.2.1 Ejemplo

Textos en español que corresponden a opiniones de usuarios en los siguientes productos: automóviles, hoteles, lavadoras, libros, teléfonos celulares, música, computadoras y películas (Julian Brooke and Maite Taboada. https://www.sfu.ca/~mtaboada/SFU_Review_Corpus.html).

[2]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import my_functions
from my_functions import *

t_data = pd.read_csv('../data/spanish_reviews/reviews_text_caract.csv',␣
 ↪header=0)
t_data
```

[2]:
```
                       file  categoria sentimiento
0       coches_no_1_11.txt     coches          no
1       coches_no_1_13.txt     coches          no
2       coches_no_1_15.txt     coches          no
3       coches_no_1_18.txt     coches          no
4       coches_no_1_19.txt     coches          no
..                     …        …         …
395  peliculas_yes_5_23.txt  peliculas         yes
396   peliculas_yes_5_4.txt  peliculas         yes
397   peliculas_yes_5_5.txt  peliculas         yes
398   peliculas_yes_5_7.txt  peliculas         yes
399   peliculas_yes_5_9.txt  peliculas         yes

[400 rows x 3 columns]
```

[3]:
```
dir_data = '../data/spanish_reviews/all_files/'
# leer y preprocesar textos
preprocesador = preprocesaTexto(idioma='es', _tokeniza=False,␣
 ↪_muestraCambios=False, _quitarAcentos=True,
                                 _remueveStop=True, _stemming=False)

files_txt = dir_data+t_data['file']
files_txt = files_txt.tolist()
```

```
corpus = []
for f in files_txt:
    file = open(f, 'r', encoding='latin-1')
    txt = file.read()
    txt_prep = preprocesador.preprocesa(txt)
    corpus.append(txt_prep)

y1 = t_data['categoria'].astype('category').cat.codes
y2 = t_data['sentimiento'].astype('category').cat.codes
```

[14]:
```
corpus[10]
```

[14]:
```
'hola scenic rx dos años medio km mayoria defectos citan desilusionados
entregaron coche fallo turbo creo hicieran revision preentrega si hicieron dio
igual darme coche averia sic rotura turbo km pese garantia kilometraje renault
hizo cargo solo pague euros infinitos fallos sensor presion neumaticos reconozco
chorrada averia obligo llevar coche concesionario menos veces sistema anclaje
tapa gasolina motor cierre hechos rompan tiempo solo verlo ultimo famoso sistema
escape cambiado veces catalizador dos veces tramo intermedio silencioso final
acoples cambios diseño escape servido todas semanas reviso bajos coche si miedo
pusieran bomba vaya solo conozco caso propietario vehiculo roto verdad aun pocos
km conozco personalmente casos peores fundidas embrague roturas suspensiones etc
espero afecten unidad fin coche bonito malo si renault dejado fabricarlo honor
verdad decir salvo tapa gasolina reparacion turbo resto reparaciones hecho cargo
renault incluso producido periodo garantia sintoma inequivoco reconocen chapuza
coche comercializan cierto ampliacion garantia años km costaba euros regalo
renault tercera vez lleve coche taller definitiva pedi quedara casa coche diera
valor actual mercado solo hacian si compraba renault logicamente pase disgusto
producido birria casi millones pesetas pienso volver hacer saludos perdon rollo
apetecia desahogarme '
```

Para éste ejemplo, usaremos las funciones de procesamiento de textos de CountVectorizer (scikit learn)

[4]:
```
vectorizer = CountVectorizer(lowercase=False, ngram_range= (1,1),␣
 ↪max_features=10000)
#vectorizer = CountVectorizer(lowercase=True, ngram_range= (2,2),␣
 ↪max_features=10000)
X = vectorizer.fit_transform(corpus)
bow = X.toarray()
bow_df = pd.DataFrame(bow,columns=vectorizer.get_feature_names())
bow_df[55:65]
```

[4]:

	aaron	abajo	abandona	abandonado	abandonar	abandono	abarcan	abate	\
55	0	0	0	0	0	0	0	0	
56	0	0	0	0	0	0	0	0	

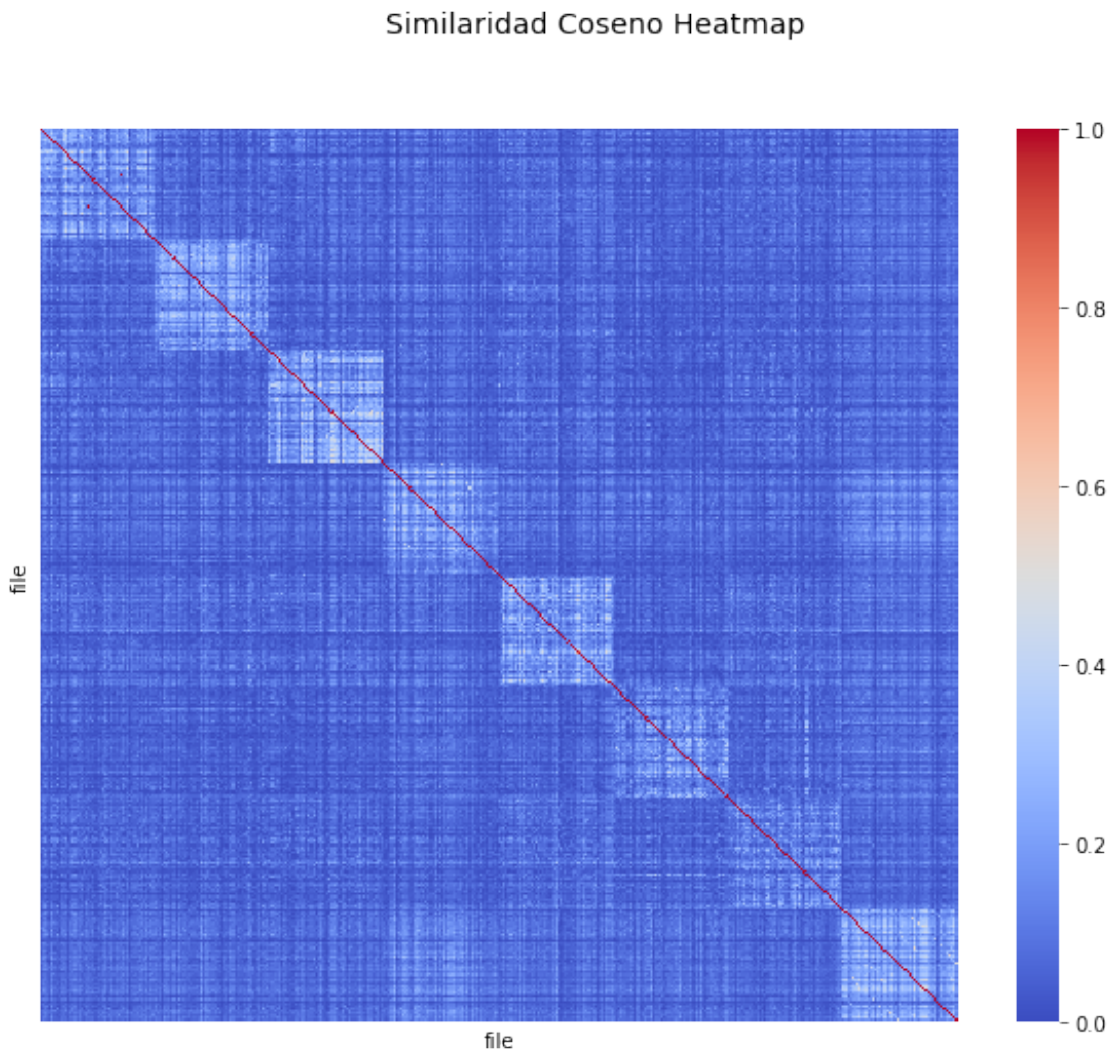
```
57      0      0         0           0          0         0       0
58      0      0         0           0          0         0       0
59      0      0         0           0          0         0       0
60      0      0         0           0          0         0       0
61      0      0         0           0          0         0       0
62      0      0         0           0          0         0       0
63      0      0         0           0          0         0       0
64      0      0         0           0          0         1       0      0

    abatibles  abatir  …  zi  zona  zonas  zoom  zumbido  zumo  zumos  zwan  \
55          0       0  …   0     0      0     0        0     0      0     0
56          0       0  …   0     0      0     0        0     1      0     0
57          0       0  …   0     1      0     0        0     0      0     0
58          0       0  …   0     0      3     0        0     0      0     0
59          0       0  …   0     0      0     0        0     0      0     0
60          0       0  …   0     0      0     0        0     0      0     0
61          0       0  …   0     0      0     0        0     0      0     0
62          0       0  …   0     0      0     0        0     0      0     0
63          0       0  …   0     0      0     0        0     0      0     0
64          0       0  …   0     0      0     0        0     0      0     0

    ñoñas  ñoño
55      0     0
56      0     0
57      0     0
58      0     0
59      0     0
60      0     0
61      0     0
62      0     0
63      0     0
64      0     0

[10 rows x 10000 columns]
```

Podemos realizar tareas de aprendizaje supervisado y no supervisado usando alguna métrica apropiada. Generalmente, se usa la distancia de coseno.

[5]:
```
cos_sim = cosine_similarity(bow, bow)
sim_df = pd.DataFrame(cos_sim)
sim_df.columns = t_data['file']
sim_df.index = t_data['file']
```

[7]:
```
f, ax = plt.subplots(figsize=(10, 8))
hm = sns.heatmap(sim_df, annot=False, ax=ax, cmap="coolwarm",fmt='.2f',
            xticklabels = False, yticklabels = False)
#f.subplots_adjust(top=0.93)
```

```
t= f.suptitle('Similaridad Coseno Heatmap', fontsize=14)
```



Information retrieval

[8]:
```
## QUERY
#q = 'Los saltos con breves reflexiones, de lectura ágil y sin problemas de␣
 ↪comprensión de una historia que se cuenta hacia atrás y en peequeñas dosis.␣
 ↪Al libro le sobran páginas, divagaciones que no aportan demasiado, sin␣
 ↪embargo me gustó bastante'
q = 'una lavadora muy buena para mi ropa'
q_prep = [preprocesador.preprocesa(q)]
q_bow = vectorizer.transform(q_prep)
sim_q = cosine_similarity(bow,q_bow.toarray())
ind_sort = np.argsort(sim_q,0)[::-1] ## sort con reverse mediante slicing
# top 5
```

```
cerca_ind = np.ndarray.flatten(ind_sort[:5])
ff = [t_data['file'][i] for i in cerca_ind]
ff
```

[8]:
```
['lavadoras_yes_4_21.txt',
 'lavadoras_no_1_5.txt',
 'lavadoras_yes_5_16.txt',
 'lavadoras_yes_5_7.txt',
 'lavadoras_no_1_9.txt']
```

Clustering

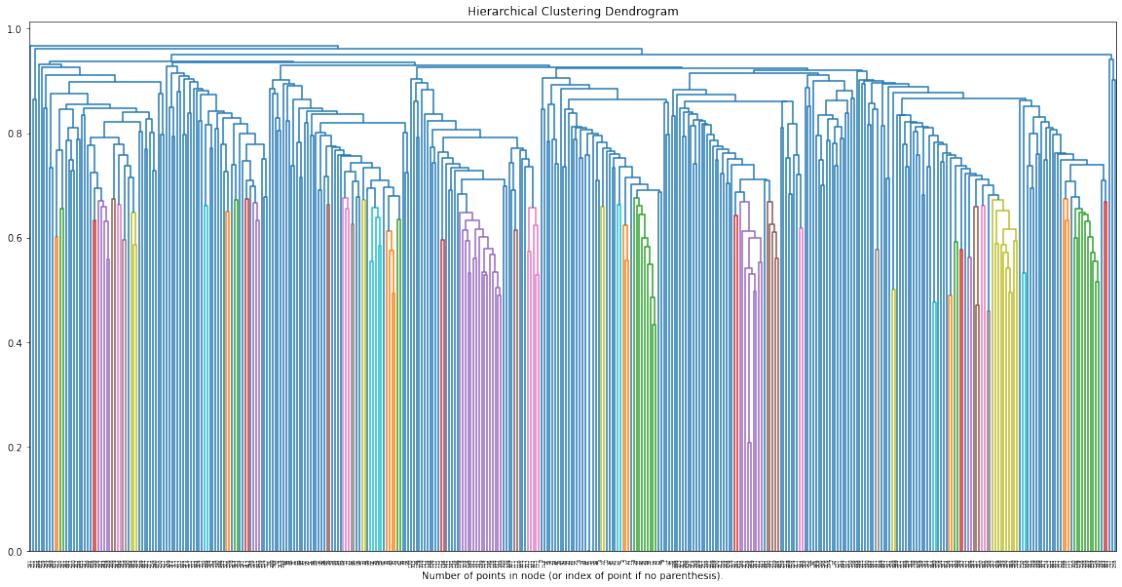
[73]:
```
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=None, distance_threshold=0,␣
 ↪affinity='cosine', linkage='average')
cluster.fit_predict(bow)

print('Clusters: ',cluster.n_clusters_)
```

```
Clusters:  400
```

[74]:
```
from scipy.cluster.hierarchy import dendrogram

def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)
```

[75]:
```
plt.figure(figsize=(20, 10))
plt.title('Hierarchical Clustering Dendrogram')
```

```
plot_dendrogram(cluster) #, truncate_mode='level', p=5)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()
```



K-Means y visualización con PCA

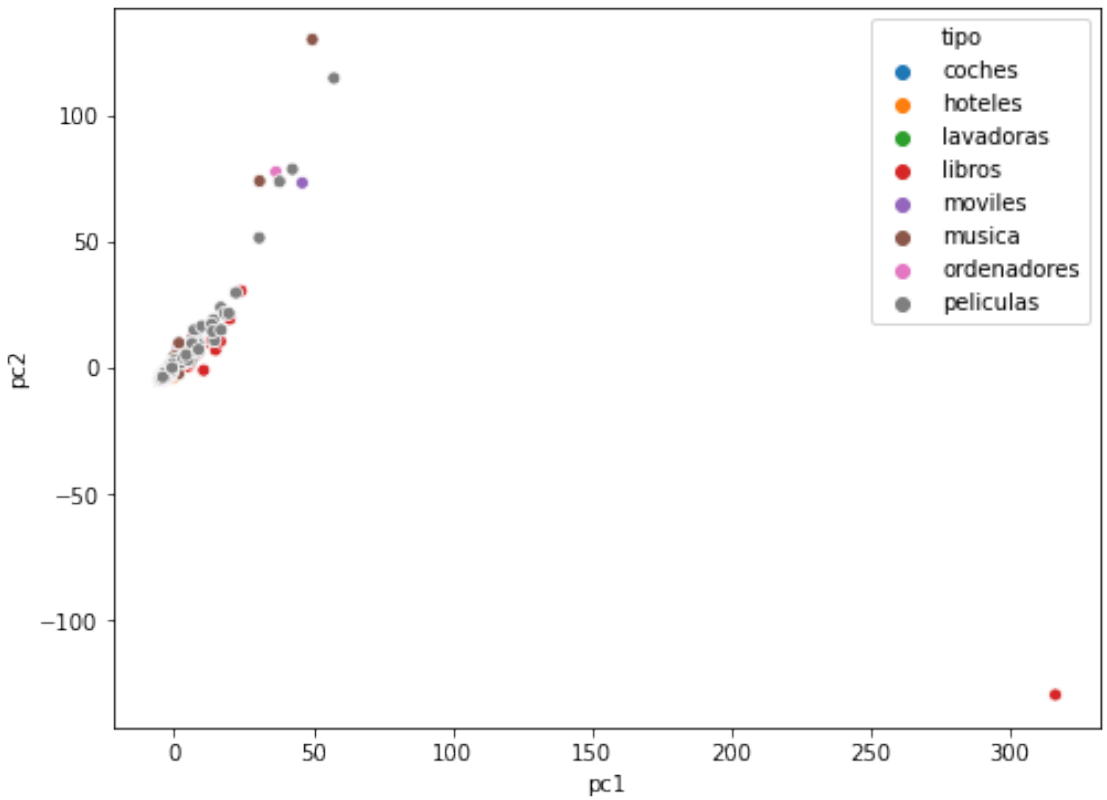
[40]:
```
from sklearn import cluster
from sklearn.decomposition import PCA, KernelPCA
from sklearn.preprocessing import StandardScaler
import plotly.express as px

k = 8
k_means = cluster.MiniBatchKMeans(n_clusters=k, random_state=10)
k_means.fit(bow)
cl_pred = k_means.predict(bow)
cl_pred
```

[40]:
```
array([3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5, 3, 3,
       3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 6, 3, 3, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 6, 6, 6, 3, 3,
       6, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 6, 3, 3, 3, 3, 6, 3, 3, 3, 3,
       0, 3, 3, 3, 0, 0, 3, 3, 3, 0, 0, 3, 3, 3, 3, 3, 2, 3, 0, 3, 3, 3,
       0, 3, 0, 0, 0, 3, 3, 3, 0, 3, 0, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
```

```
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 3, 3, 0, 3, 3, 3, 3, 0, 3, 3,
3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 3, 3,
3, 3, 3, 3, 0, 0, 0, 3, 0, 3, 0, 3, 3, 0, 0, 3, 0, 3, 0, 0, 3, 3,
3, 0, 0, 4, 3, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 3, 0, 3, 0, 3, 7, 3,
0, 0, 0, 0], dtype=int32)
```

[41]:
```
X = StandardScaler().fit_transform(bow)
pca = PCA(n_components=5)
bow_pca = pca.fit_transform(X)
proj = pd.DataFrame(bow_pca,columns = ['pc1','pc2','pc3','pc4','pc5'])

pca_bow = pd.DataFrame({'pc1': proj['pc1'], 'pc2': proj['pc2'], 'tipo':␣
↪t_data['categoria'],
                        'clus':cl_pred})
```

[42]:
```
import seaborn as sns
sns.scatterplot(data = pca_bow, x='pc1', y='pc2', hue='tipo')
sns.set(style='whitegrid',)
plt.show()
```



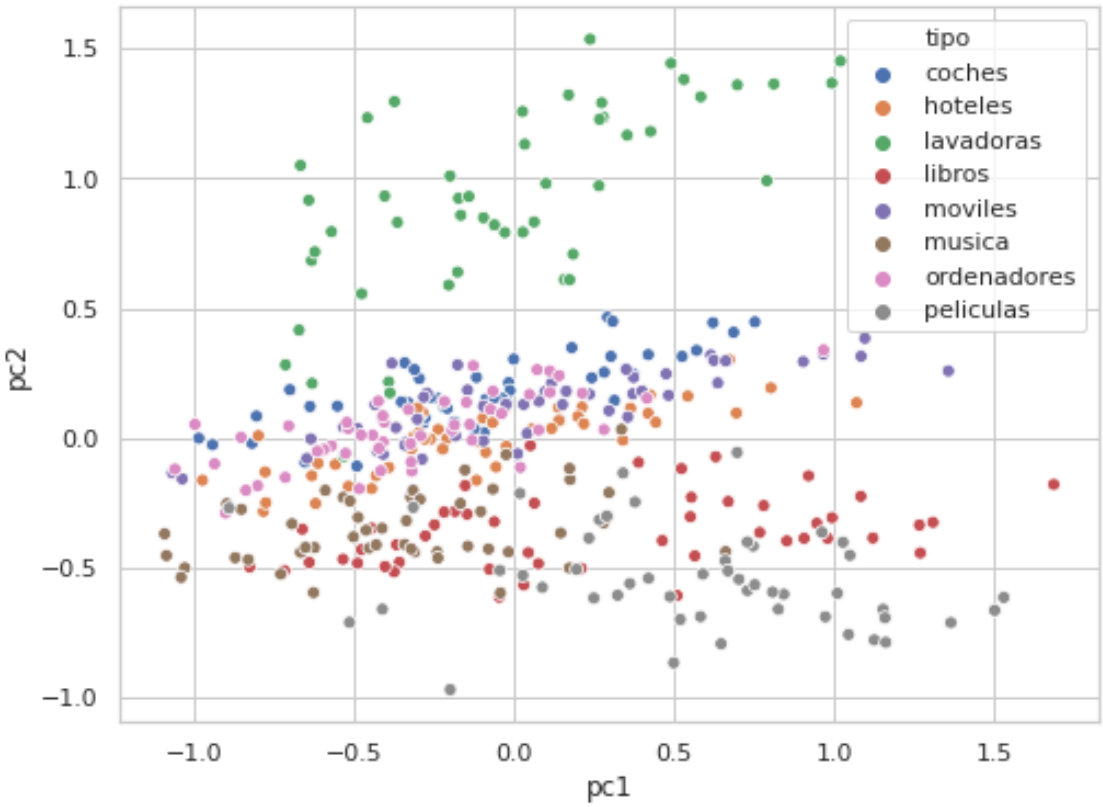
Gráfica interactiva

[79]:
```
fig = px.scatter(pca_bow, x='pc1', y='pc2', hover_data=['tipo'], color = 'tipo')
fig.update_layout(
    autosize=False,
    width=600,
    height=600,
)
fig.show()
```

Lo mismo pero ahora en la matriz de similaridades obtenida

[43]:
```
pca = PCA(n_components=5)
pca.fit(cos_sim)
# proyectar datos
proj = pd.DataFrame(pca.transform(cos_sim),columns =␣
↪['pc1','pc2','pc3','pc4','pc5'])

pca_bow = pd.DataFrame({'pc1': proj['pc1'], 'pc2': proj['pc2'], 'tipo':␣
↪t_data['categoria'],
                        'clus':cl_pred})
```

[44]:
```
import seaborn as sns
sns.scatterplot(data = pca_bow, x='pc1', y='pc2', hue='tipo')
sns.set(style='whitegrid',)
plt.show()
```



[81]:
```
# Grafica interactiva
fig = px.scatter(pca_bow, x='pc1', y='pc2', hover_data=['tipo'], color = 'tipo')
fig.update_layout(
    autosize=False,
    width=600,
    height=600,
)
fig.show()
```

[86]:
```
# K-MEANS en los componentes principales de la matriz de similaridad
k = 5
k_means = cluster.MiniBatchKMeans(n_clusters=k, random_state=10)
k_means.fit(proj)
#cl_pred = pd.DataFrame(k_means.predict(proj)).astype('category')
cl_pred = pd.Series(k_means.predict(proj), dtype="category")
#cl_pred = k_means.predict(proj)
cl_pred
```

[86]:
```
0      2
1      2
2      2
```

```
3      2
4      2
      ..
395    0
396    0
397    0
398    0
399    0
Length: 400, dtype: category
Categories (5, int64): [0, 1, 2, 3, 4]
```

[87]:
```
# grafica interactiva
pca_bow = pd.DataFrame({'pc1': proj['pc1'], 'pc2': proj['pc2'], 'tipo':␣
 ↪t_data['file'],
                        'clus':cl_pred})

fig = px.scatter(pca_bow, x='pc1', y='pc2', hover_data=['tipo'], color = 'clus')
fig.update_layout(
    autosize=False,
    width=600,
    height=600,
)
fig.show()
```

Clasificación Veamos primero la clasificación del sentimiento (positivo 1, negativo 0)

[9]:
```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC # Support vector classifier

X_train, X_test, y_train, y_test = train_test_split(bow, y2, test_size=0.2,␣
 ↪random_state=42)

svm = SVC(kernel='linear', C=1E10)
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
y_pred
```

[9]:
```
array([1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1], dtype=int8)
```

Métricas de evaluación.

La Exactitud (Accuracy) es simplemente la proporción de datos clasificados correctamente.

$$Accuracy = \frac{VP + VN}{TOTAL}$$

donde VP es el número de predicciones positivas correctamente clasificadas, VN es el número de predicciones negativas correctamente clasificadas y $TOTAL$ es el número total de casos u observaciones.

Es la primera métrica que se comprueba al evaluar un clasificador; sin embargo, si los datos están desbalanceados o si se está más interesado detectar una de las clases, la Exactitud no captura realmente la eficacia de un clasificador.

La precisión (Precision) nos ayuda a cuantificar la fracción de positivos verdaderos entre el total de los clasificados como positivos. Se utiliza principalmente para medir qué tan efectivo es el modelo en detectar la categoría de interés, es decir, la categoría positiva.

$$Precision = \frac{VP}{VP + FP}$$

donde FP es el número de predicciones positivas calificadas incorrectamente (falsos positivos). \end{frame}

La recuperación (Recall) mide cuántos positivos verdaderos se predijeron como positivos.

$$Recall = \frac{VP}{VP + FN}$$

donde FN es el número de predicciones negativas calificadas incorrectamente.

Observa que mientras más cercano a 1 se encuentre el recall, más datos de la categoría real fueron bien clasificados.

La medida F1 (F1 score) es la media armónica de la precisión y la recuperación (Recall). Esta medida es una buena forma de resumir la evaluación en un número único.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Existe una relación entre la precisión y el recall: un modelo que predice todo como positivo tendrá un recall de 1, pero una precisión muy baja pues tendría muchos falsos positivos, mientras que un modelo que solo predijera un positivo y el resto negativos tendría un recall muy bajo, pero una precisión muy alta.

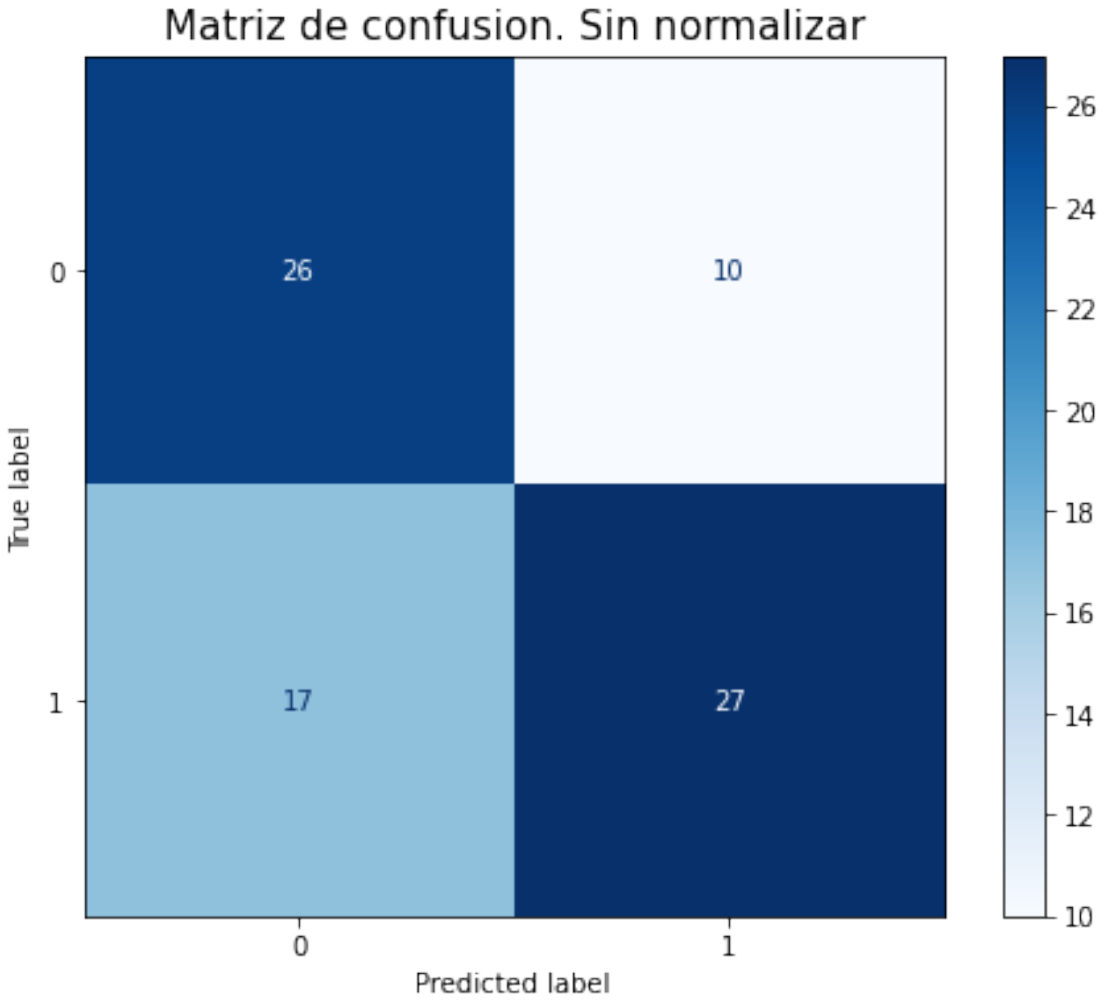
Es por ello que se recurre a la medida F1, ya que mitiga el impacto de las tasas altas y acentúa el de las tasas bajas.

[10]:
```
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.60      0.72      0.66        36
```

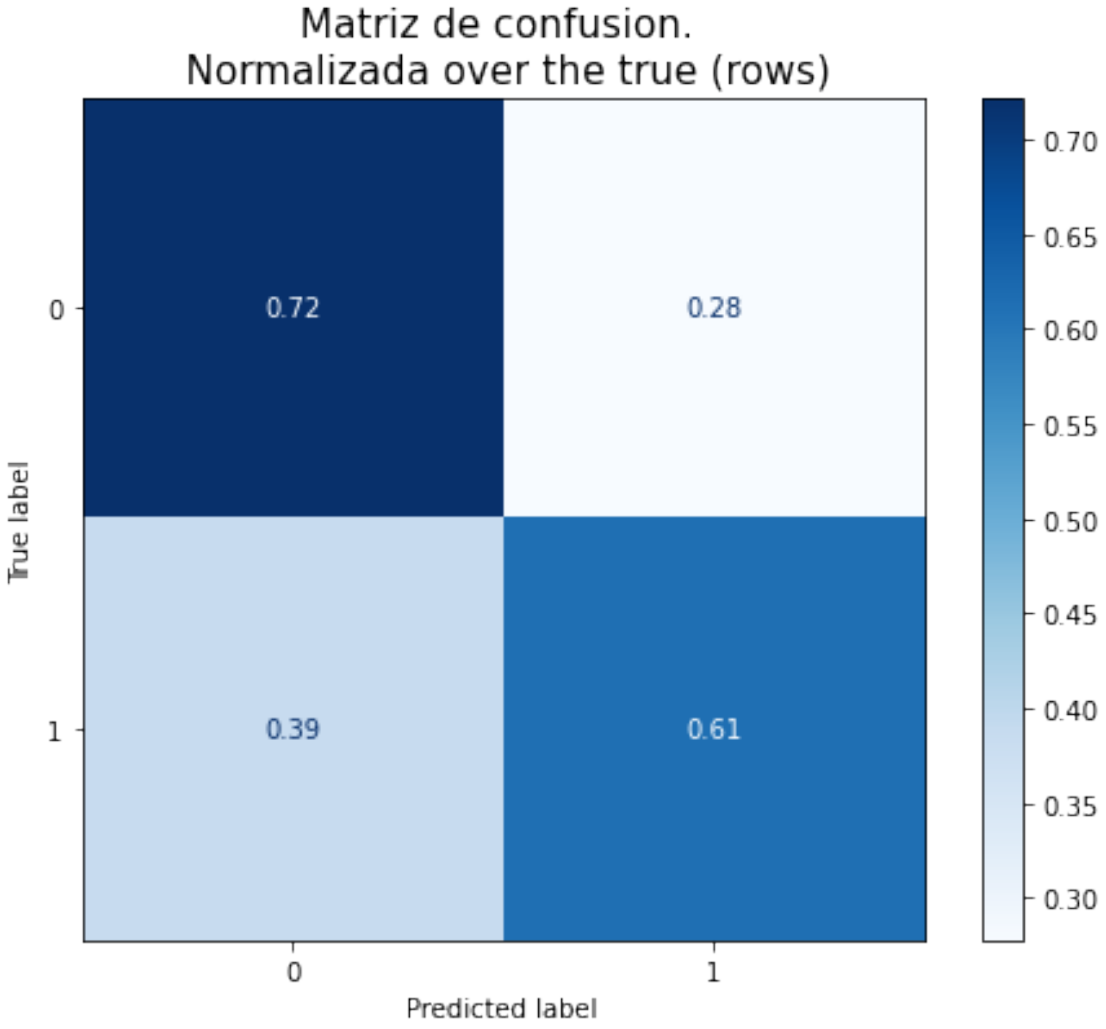
```
              1       0.73      0.61      0.67        44

    accuracy                           0.66        80
   macro avg       0.67      0.67      0.66        80
weighted avg       0.67      0.66      0.66        80
```

[11]:
```
plt.rcParams['figure.figsize'] = (8, 6)
disp1 = metrics.plot_confusion_matrix(svm, X_test, y_test, cmap=plt.cm.Blues,␣
 ↪normalize = None)
disp1.ax_.set_title('Matriz de confusion. Sin normalizar',{'fontsize':15})
plt.show()
```



[29]:
```
plt.rcParams['figure.figsize'] = (8, 6)
disp1 = metrics.plot_confusion_matrix(svm, X_test, y_test, cmap=plt.cm.Blues,␣
 ↪normalize = 'true')
```

```
disp1.ax_.set_title('Matriz de confusion. \n Normalizada over the true␣
↪(rows)',{'fontsize':15})
plt.show()
```



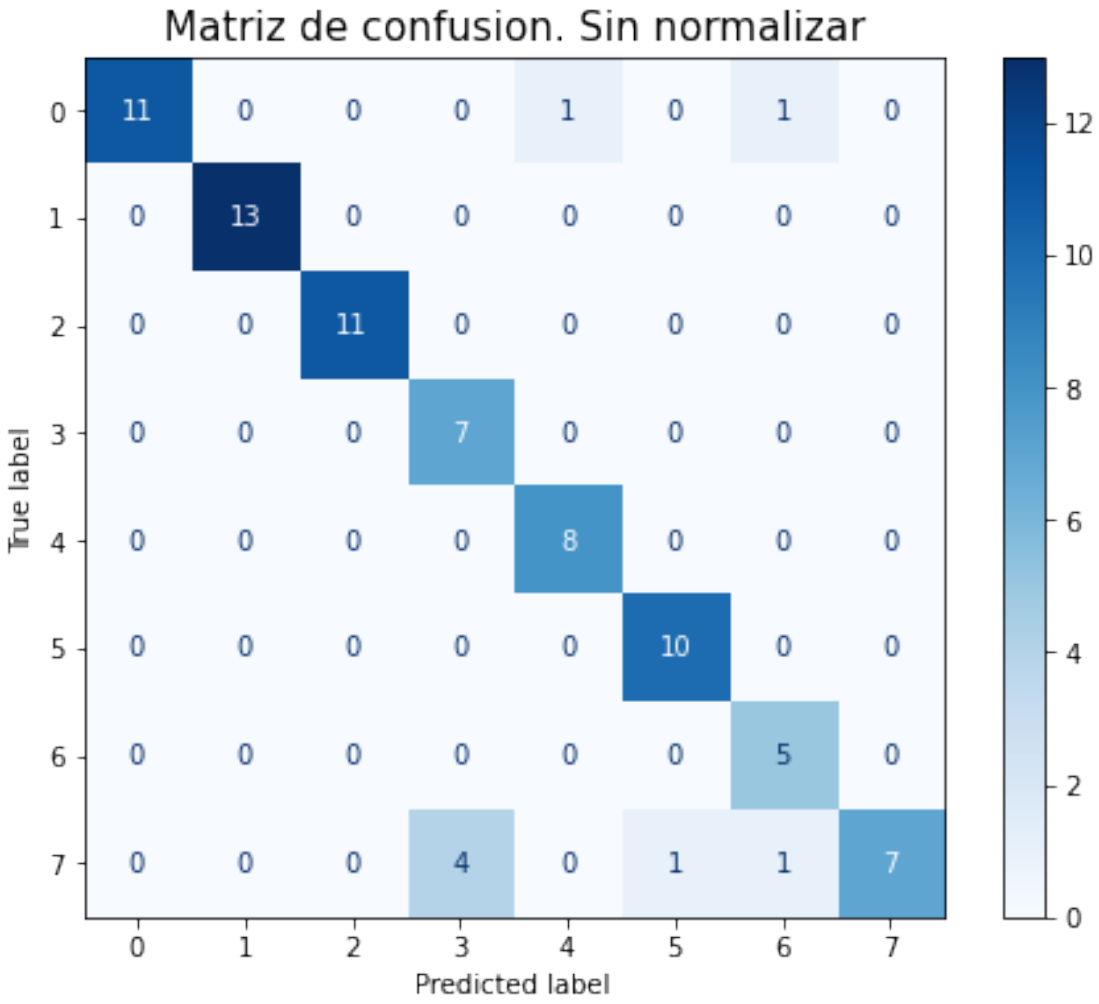
Ahora, veamos la clasificación respecto al tópico o categoría (8 categorías)

[12]:
```
X_train, X_test, y_train, y_test = train_test_split(bow, y1, test_size=0.2,␣
↪random_state=42)

svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support
```

```
              0       1.00      0.85      0.92        13
              1       1.00      1.00      1.00        13
              2       1.00      1.00      1.00        11
              3       0.64      1.00      0.78         7
              4       0.89      1.00      0.94         8
              5       0.91      1.00      0.95        10
              6       0.71      1.00      0.83         5
              7       1.00      0.54      0.70        13

       accuracy                           0.90        80
      macro avg       0.89      0.92      0.89        80
   weighted avg       0.93      0.90      0.90        80
```

[13]:
```
plt.rcParams['figure.figsize'] = (8, 6)
disp1 = metrics.plot_confusion_matrix(svm, X_test, y_test, cmap=plt.cm.Blues,␣
 ↪normalize = None)
disp1.ax_.set_title('Matriz de confusion. Sin normalizar',{'fontsize':15})
plt.show()
```



6.3 BOW extensiones. TF-IDF (Salton and Buckley, 1988).

Una extensión a la frecuencia de términos es una versión **pesada** de los mismos.

El más usado es Term Frequency - Inverse Document Frequency (TF-IDF).

- Frecuencia de términos: $tf_{t,d} = \text{count}(t, d)$, o también: $tf = log(\text{count}(t, d) + 1)$
- Frecuencia de documentos: $df_t = \text{count}(d : t \in d)$
- Frecuencia de documentos inversa: $idf_t = \log \frac{N}{1+df_t}$

Entonces

$$\text{TF-IDF} : tf_{t,d} \times idf_t$$

Existen varias implementaciones (Keras, SKlearn, NLTK) que consideran diferentes esquemas de pesado de la matriz BOW, incluyendo TF-IDF.

[14]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(lowercase=False, ngram_range= (1,1),␣
 ↪max_features=10000, min_df=.02)
#vectorizer = TfidfVectorizer(lowercase=True, ngram_range= (2,2))
X = vectorizer.fit_transform(corpus)
tfidf = X.toarray()
tfidf_df = pd.DataFrame(tfidf,columns=vectorizer.get_feature_names())
#tfidf_df.describe()
tfidf_df[55:65]
```

[14]:

	abajo	abierto	abre	abrir	absolutamente	absoluto	absurda	\
55	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	
56	0.0	0.0	0.105531	0.000000	0.000000	0.0	0.0	
57	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	
58	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	
59	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	
60	0.0	0.0	0.000000	0.000000	0.082280	0.0	0.0	
61	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	
62	0.0	0.0	0.000000	0.101472	0.000000	0.0	0.0	
63	0.0	0.0	0.000000	0.000000	0.148767	0.0	0.0	
64	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	

	aburrida	acaba	acabado	…	vuelven	vuelvo	web	will	windows	\
55	0.0	0.0	0.0	…	0.0	0.000000	0.000000	0.0	0.0	
56	0.0	0.0	0.0	…	0.0	0.000000	0.000000	0.0	0.0	
57	0.0	0.0	0.0	…	0.0	0.000000	0.000000	0.0	0.0	
58	0.0	0.0	0.0	…	0.0	0.000000	0.000000	0.0	0.0	
59	0.0	0.0	0.0	…	0.0	0.000000	0.000000	0.0	0.0	
60	0.0	0.0	0.0	…	0.0	0.000000	0.000000	0.0	0.0	
61	0.0	0.0	0.0	…	0.0	0.121589	0.104483	0.0	0.0	
62	0.0	0.0	0.0	…	0.0	0.000000	0.000000	0.0	0.0	
63	0.0	0.0	0.0	…	0.0	0.000000	0.000000	0.0	0.0	
64	0.0	0.0	0.0	…	0.0	0.000000	0.000000	0.0	0.0	

	xp	york	you	your	zona
55	0.0	0.0	0.0	0.0	0.000000
56	0.0	0.0	0.0	0.0	0.000000
57	0.0	0.0	0.0	0.0	0.060404
58	0.0	0.0	0.0	0.0	0.000000
59	0.0	0.0	0.0	0.0	0.000000
60	0.0	0.0	0.0	0.0	0.000000
61	0.0	0.0	0.0	0.0	0.000000
62	0.0	0.0	0.0	0.0	0.000000
63	0.0	0.0	0.0	0.0	0.000000
64	0.0	0.0	0.0	0.0	0.000000

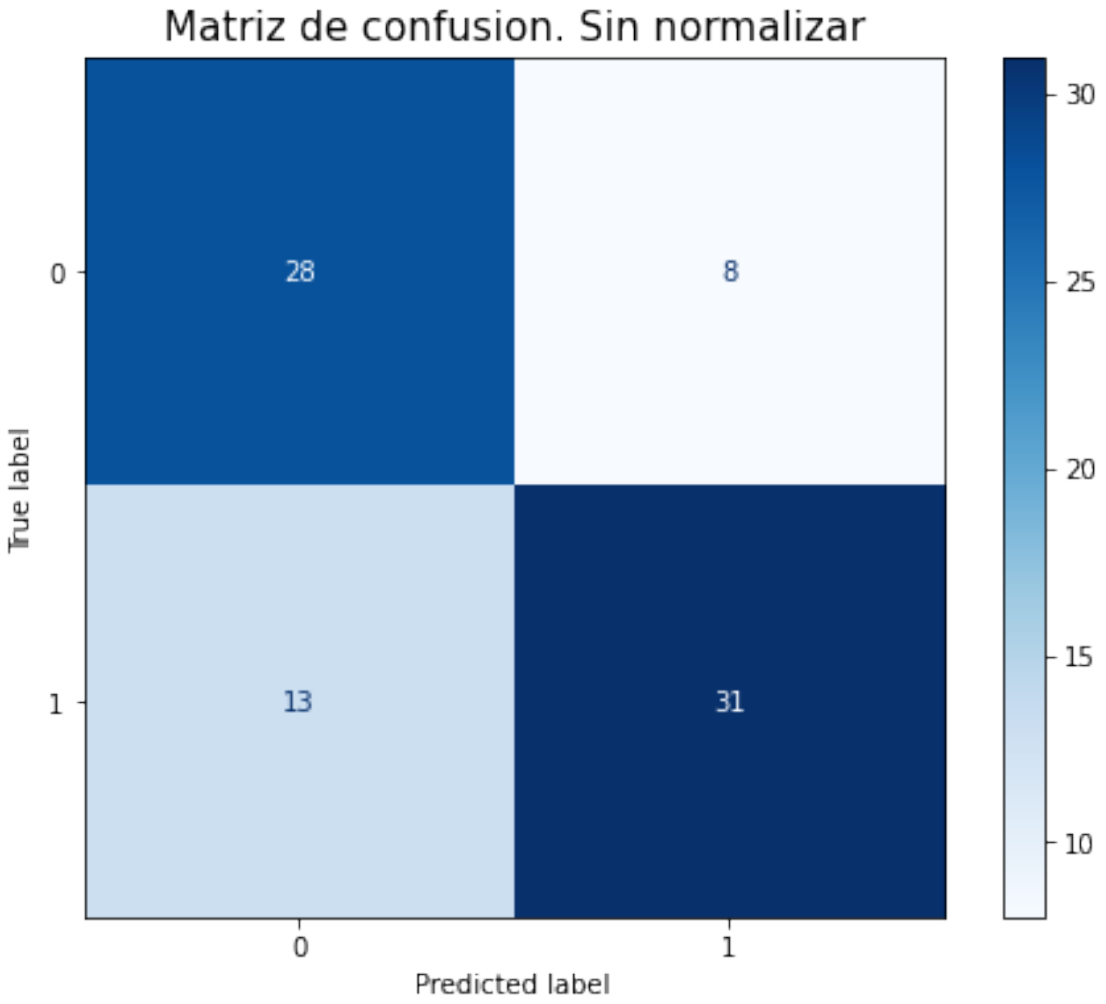
[10 rows x 1721 columns]

Clasificación con TF-IDF Usando el sentimiento (positivo, negativo) como respuesta.

[17]:
```
X_train, X_test, y_train, y_test = train_test_split(tfidf, y2, test_size=0.2,␣
↪random_state=42)

svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.68      0.78      0.73        36
           1       0.79      0.70      0.75        44

    accuracy                           0.74        80
   macro avg       0.74      0.74      0.74        80
weighted avg       0.74      0.74      0.74        80
```

[18]:
```
plt.rcParams['figure.figsize'] = (8, 6)
disp1 = metrics.plot_confusion_matrix(svm, X_test, y_test, cmap=plt.cm.Blues,␣
↪normalize = None)
disp1.ax_.set_title('Matriz de confusion. Sin normalizar',{'fontsize':15})
plt.show()
```



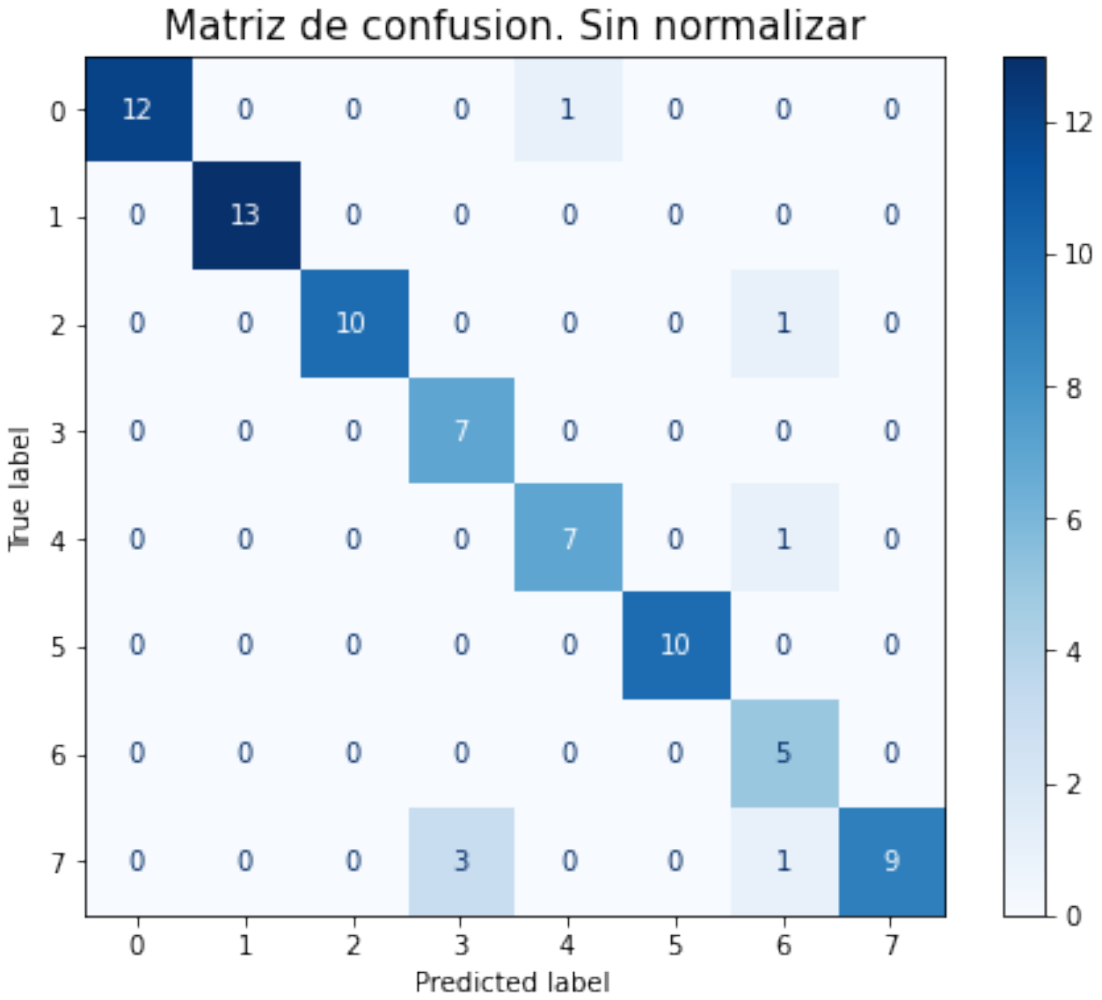
Ahora para las 8 categorías que corresponden a los tópicos. ¿Cómo es respecto a BOW?

[19]:
```
X_train, X_test, y_train, y_test = train_test_split(tfidf, y1, test_size=0.2,␣
↪random_state=42)

svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.92      0.96        13
           1       1.00      1.00      1.00        13
           2       1.00      0.91      0.95        11
           3       0.70      1.00      0.82         7
```

4	0.88	0.88	0.88	8
5	1.00	1.00	1.00	10
6	0.62	1.00	0.77	5
7	1.00	0.69	0.82	13
accuracy			0.91	80
macro avg	0.90	0.92	0.90	80
weighted avg	0.94	0.91	0.92	80

[20]:
```
plt.rcParams['figure.figsize'] = (8, 6)
disp1 = metrics.plot_confusion_matrix(svm, X_test, y_test, cmap=plt.cm.Blues,␣
 ↪normalize = None)
disp1.ax_.set_title('Matriz de confusion. Sin normalizar',{'fontsize':15})
plt.show()
```



Si bien los modelos basados en representaciones tipo BOW son muy útiles para ciertas tareas y ciertos corpus de textos, para otros no es la mejor opción.

Básicamente, los problemas de BOW y sus extensiones son - Altamente dimensional - Sparse

6.4 Word embeddings y semántica distribucional

El esquema que hemos usado para análisis de textos:

Las representaciones vectoriales que veremos en ésta parte del curso, se construyen para que nos den más información relacionada con las palabras **y su uso** en el lenguaje.

- Semántica: el estudio del significado de los signos lingüisticos. Examina las relaciones entre las palabras y lo que representan.
- El significado de una palabra está dado por las palabras que aparecen cercanas frecuentemente: "*You shall know a word by the company it keeps*" (J. R. Firth 1957: 11)
- La semántica vectorial nos ofrece una manera cuantitativa para representar diversas cualidades útiles de las palabras (significado, similaridad, asociaciones, etcétera):

[18]:
```
from nltk.book import text1, text4
print('Corpus: ',text4.name,'\n')

text4.concordance('president',lines=10) #president, city
```

```
Corpus:  Inaugural Address Corpus

Displaying 10 of 89 matches:
artment it is made the duty of the President " to recommend to your considerati
ecution of any official act of the President the Constitution requires an oath
o taking upon myself the duties of President of the United States for another t
he unequaled services of the first President , it was a common sentiment that t
orrection is in the power of every President , and consequently in mine , it wo
s would be found to constitute the President a part of the legislative power .
ent have been entertained that the President , placed at the capital , in the c
r conferring the veto power on the President . This argument acquires additiona
 the nation ," as affording to the President sufficient authority for his consi
to increase itself . By making the President the sole distributer of all the pa
```

Word embeddings:

Representaciones vectoriales densas de una longitud predefinida, donde podemos inferir propiedades semánticas de las palabras, tales como su significado, mediante la semántica distribucional.

Para obtenerlos, usamos un **modelo de lenguaje**.

6.4.1 Modelos de lenguaje

Los modelos que asignan **probabilidades** a **secuencias de palabras** se les llaman modelos de lenguaje.

Introducir probabilidades nos permite inferir características semánticas de las palabras, tales como el **significado**, al observar la distribución de las palabras alrededor de alguna de interés.

Modelos de lenguaje "everywhere"…

Sea $w_1, w_2, \ldots, w_T$ una secuencia de palabras, donde $w \in V$ (Vocabulario).

Un modelo de lenguaje obtiene probabilidades de palabras

$$P(w_t) = P(w_t|w_1^{t-1})$$

en la secuencia.

Por ejemplo, si $V = \{..., perro, gato, suegro, refrigerador, ...\}$

P(perro|cuando llego a casa, tengo que dar de comer al)

P(gato | cuando llego a casa, tengo que dar de comer al)

P(suegro|cuando llego a casa, tengo que dar de comer al)

P(estufa|cuando llego a casa, tengo que dar de comer al)

También, podemos calcular la probabilidad de obtener una secuencia de n palabras:

$$P(w_1^n) = \prod_{t=1}^{n} P(w_t|w_1^{t-1}).$$

Esto se vuelve muy complicado a medida que la longitud de la secuencia aumenta.

Una simplificación es el modelo $n-$gram:

$$P(w_t|w_1^{t-1}) \approx P\left(w_t|w_{t-N+1}^{t-1}\right)$$

donde definimos un contexto basado solo en las N palabras previas.

Diferentes modelos de lenguaje se basan en enfoques particulares para calcular ésas probabilidades.

Por ejemplo, usando Máxima Verosimilitud, los estimadores para las probabilidades son conteos. Por ejemplo, para el modelo bigrama (Markov):

$$P\left(w_t|w_{t-1}\right) = \frac{\#(w_{t-1}, w_t)}{\sum_w \#(w_{t-1}, w)}$$

Las estimaciones, por supuesto, depende del tamaño del Corpus. Anteriormente (hasta inicios de los 2000s), los métodos más usados a partir de éste esquema eran modelos de Markov, tales como Hidden Markov Models o Máximum Entropy Markov Models.

Aunque fueron usados extensívamente (y aún se usan), tiene varias problemáticas, sobre todo computacionales.

6.4.2 Modelo neuronal de lenguaje (NNLM)

En este paper seminal, Bengio et al. introducen el uso de Redes Neuronales Feed Forward para secuencias de textos.

¿Cómo funciona? Versión simplificada…

(Jurafsky & Martin, 2019)

Si queremos construir los embeddings, llegamos al modelo de Bengio:

(Jurafsky & Martin, 2019)

La propuesta de Bengio et al.

- Asociar un vector en $\mathbb{R}^d$ a cada palabra del vocabulario
- Expresar la probabilidad conjunta de secuencias de palabras en términos de los vectores de las palabras en la secuencia.
- Aprender al mismo tiempo, la representación vectorial de las palabras y los parámetros de las probabilidades (máxima verosimilitud)

Dado un vocabulario V, la probabilidad conjunta se modela (y se aprende) mediante $P(w_t|w_1^{t-1}) = f(w_t, \ldots, w_{t-n+1})$. Consta de dos partes: 1. Un mapeo $E(i)$ para cualquier elemento i de V a un vector $E(i) = \mathbf{e} \in \mathbb{R}^d$. Este mapeo es representado por la matriz $\mathbf{E}_{d\times|V|}$ (en el paper de Bengio se representa por $C_{d\times|V|}$). 2. Una función de probabilidad obtenida mediante una función $g(\cdot)$:

$$f(i, w_{t-1}, \ldots, w_{t-n+1}) = g(i, E(w_{t-1}), \ldots, E(w_{t-n+1})).$$

Observa que $\mathbf{E}$ se **comparte** en todas las palabras del contexto.

Paso Forward:

1. Capa de proyección: dados los índices de las n palabras del contexto, construye los vectores one-hot $\mathbf{x}_i$ correspondientes y su proyección (mapeo o embedding) $\mathbf{e}_i = \mathbf{E}\mathbf{x}_i$. La capa de proyección es la concatenación de los embeddings de las palabras del contexto:

$$\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n)'$$

2. Capa oculta:

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{e} + \mathbf{b})$$

3. Capa de salida:

$$\begin{aligned} \mathbf{z} &= \mathbf{U}\mathbf{h} \\ \mathbf{y} &= \text{softmax}(\mathbf{z}) \end{aligned}$$

Paso Backward (backpropagation):

Se realiza iterativamente usando gradiente estocástico:

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \gamma \frac{\partial \log P(w_t|w_{t-1}, \ldots, w_{t-n+1})}{\partial \boldsymbol{\theta}}$$

después de observar la palabra t del corpus de entrenamiento.

Los detalles los omitimos. Si tienes curiosidad, me preguntas…

En el ajuste, $\boldsymbol{\theta}$ representa todos los parámetros:

Sean d la dimensión del embedding de las palabras, h el número de unidades ocultas y n la ventana de contexto. Los parámetros de la NNLM son entonces

$$\boldsymbol{\theta} = (\mathbf{c}, \mathbf{b}, \mathbf{V}, \mathbf{U}, \mathbf{W}, \mathbf{E}),$$

donde

$\mathbf{E} \in \mathbb{R}^{d \times |V|}$, $\mathbf{W} \in \mathbb{R}^{h \times n \cdot d}$, $\mathbf{U} \in \mathbb{R}^{|V| \times h}$, $\mathbf{V} \in \mathbb{R}^{|V| \times n \cdot d}$, $\mathbf{b} \in \mathbb{R}^{h}$, $\mathbf{c} \in \mathbb{R}^{|V|}$

Nota: $\mathbf{V}$ y $\mathbf{c}$ son opcionales, ya que son parámetros de conexiones de los embeddings a la capa de salida (detalles en el paper de Bengio).

Word embeddings derivados del NNLM de Bengio:

- word2vec
- GloVe
- fastText

6.4.3 word2vec (T. Mikolov et al., 2013)

- Una de las representaciones vectoriales mas usadas
- Nos proporciona representaciones densas (entradas diferentes de cero) y con vectores cortos ($\mathbb{R}^{100}$ a $\mathbb{R}^{300}$, generalmente).
- Usa los pesos de una red neuronal como los embeddings. Este clasificador representa una medida cuantitativa de la Probabilidad de que una palabra w_c aparezca **cerca** de una palabra objetivo w_t.
- Aporta principalmente, ventajas computacionales
- Presenta dos algoritmos: CBOW (Continuous Bag-of-Words Model) y Skip-Gram

Mikolov et. al., 2013

CBOW

Xin Rong, 2016.

CBOW:

- One-hot encoding en la capa de entrada
- Usa activación lineal en la capa oculta
- Dos embeddings (pesos): el de la palabra objetivo y el del contexto (input-hidden y hidden-output).

Skip-Gram

Xin Rong, 2016.

- En la capa de salida, se calculan n distribuciones multinomiales con softmax, donde n es el número de palabras del contexto

T. Mikolov et al., 2013 (b). Extensiones del modelo

- Mejora sustancial en el proceso de ajuste, computacionalmente hablando.
- Se enfoca en el modelo Skip-Gram
- Simplifica el modelo multinomial por un problema de clasificación binario. Equivalente a regresión logística.
- Introduce un proceso de Negative-Sampling para generar palabras objetivo y del contexto.
- El método resultante es llamado Skip-Gram with Negative Sampling (SGNS)

Los pasos del algoritmo SGNS son:

- Considera la palabra objetivo y las palabras vecinas w_c como objetos con clase positiva
- Escoge aleatoriamente otras palabras en el vocabulario para obtener objetos con clase negativa
- Entrenar un clasificador binario con una red neuronal (eq. regresión logística) en las clases definidas antes
- Usa los pesos de la regresión como los embeddings.

Los parámetros del modelo, es decir, los embeddings.

- El embedding de las palabras objetivo w_t: **W**
- El embedding de las palabras del contexto w_c: **C**
- Generalmente, se toma **W** como los embeddings, también pueden sumarse…

Ejemplos

Demo interactivo: Xin Rong demo, y su artículo correspondiente: word2vec Parameter Learning Explained.

Cristian Cardellino: Spanish Billion Words Corpus and Embeddings (March 2016), ejemplo en el notebook.

6.4.4 Otros embeddings relacionados

- fastText: Enriching Word Vectors with Subword Information. Bojanowski et al. 2017.

  Skip-Gram (word2vec) a nivel de subpalabras: `<where>, --> <wh, whe, her, ere, re>`

- GloVe: Global Vectors for Word Representation. Pennington et al., 2014.

  `Global matrix factorization + Local context window.`

7 Algunas arquitecturas de redes profundas para análisis de textos

7.1 Redes neuronales profundas.

¿Cuál es el objetivo de hacer "*profunda*" una red? Veamos…

El esquema de Machine Learning

El esquema de Machine Learning

El esquema de Machine Learning

El esquema Deep Learning

El esquema Deep Learning

El esquema Deep Learning: deep encoder-decoder

Aprender representaciones de los datos de manera útil:

7.1.1 Representation learning

Una de las características principales de los métodos de aprendizaje profundo, es no solo aprender una función que mapee un conjunto de datos de entrada a uno de salida, sino **la representación de los datos que sea *mejor* para el problema a resolver**.

Estas representaciones son aprendidas en forma jerárquica, expresadas en términos de otras representaciones más simples. De esta forma, Deep Learning permite a la computadora construir conceptos complejos mediante complejos simples.

Algunas aplicaciones.

Image captioning.

Resúmen automático abstractivo

Pronóstico de series de tiempo multivariadas

7.1.2 Aprendizaje basado en gradientes

El aprendizaje para éste tipo de modelos sigue el mismo esquema que ya conocemos:

Recuerda el paso Forward y Backwards.

En el caso de redes profundas, el ajuste incluye conceptos como:

- Representación y capas ocultas
- Capas de salida
- Funciones de costo
- Ajuste
- Optimización

7.1.3 Capas y unidades ocultas

7.1.4 Unidades de salida

Tipo de salida	Distribución	Capa de salida	Función de costo
Binaria	Bernoulli	Sigmoid	Binary cross-entropy
Discreta	Multinomial	Softmax	Discrete cross-entropy
Contínua	Gausiana	Lineal	MSE (Gaussian cross-entropy)
Contínua	Mezcla de Gaussianas	Mezcla de densidades	Cross-entropy

7.1.5 Funciones de costo

- Error cuadrático medio

$$L(\boldsymbol{\theta}) = \|\mathbf{y} - f(\mathbf{x}; \boldsymbol{\theta})\|$$

- Cross entropy

$$L(\boldsymbol{\theta}) = -E_{\mathbf{x}} \log P_{\text{model}}(\mathbf{y}|\mathbf{x})$$

7.1.6 Regularización

Un procedimiento de suma importancia. El objetivo es restringir el modelo para **reducir el error de generalización** , no precisamente el de entrenamiento.

$$L(\boldsymbol{\theta}; \mathbf{X}, \mathbf{y}) + \alpha\Omega(\boldsymbol{\theta})$$

La forma más común es regularizar mediante alguna norma $\|\cdot\|_L$, pero en redes profundas, es necesario otro tipo de métodos. Solo por mencionar algunos: - Early stopping - Dropout - Data augmentation

7.1.7 Ajuste

Backpropagation: aplicación recursiva de la regla de la cadena.

Sea $x = f(w), y = f(x), z = f(y)$

7.1.8 Ajuste

Optimización. Se usa descenso por gradiente, pero en éste caso, dada la complejidad del procedimiento (muchos parámetros, muchos datos), se utilizan métodos estocásticos.

7.1.9 Ajuste

Optimización. La optimización en éstos modelos es compleja y generalmente, mal condicionada. Varios métodos se han propuesto para tratar de solucionar los problemas que surjen tomando en cuenta el costo computacional.

- RMSprop
- AdaGrad
- ADAM
- entre otros...

7.1.10 Recursos de cómputo. Software y hardware

Hardware. CPU (central process unit), GPU (graphic process unit), TPU (tensor process unit).

Software. Actualmente, hay software especializado en manejo y operaciones con estructuras de datos apropiadas para redes neuronales profundas. Algunas tienen separado backend y frontend. Keras, por ejemplo, es una API que puede ejecutarse con ciertos backends, como tensorflow, theano, entre otros.

Ver también Comparación de software para DL.

Nosotros usaremos

7.1.11 Conceptos relacionados

Representación de datos mediante tensores. Tensores para imágenes

Tensores para secuencias

- **Batch y mini batch**. Son una muestra, generalmente pequeña (entre 8 y 128) de los datos. Representan una **partición** de los datos de entrenamiento o prueba, y se almacenan en un tensor de dimensión arbitraria, pero el primer eje (axis) identifica el batch (tensorflow).
- **Época**. Es un término para indicar que todas las muestras del conjunto de entrenamiento han sido usadas para actualizar los parámetros del modelo, es decir, han completado el paso forward y backward. En consecuencia, todos los batches han sido usados para el procedimiento de optimización.
- **Iteración**. El número de batches necesario para completar una época.

Ejemplo. Clasificación de sentimiento en reseñas de películas (En Notebook)

7.2 Redes convolucionales

- Tipo especializado de red neuronal para procesar datos con topología tipo grid (1-D, 2-D, 3-D, etc…)
- Características a resaltar:
  - conexiones sparse
  - compartir parámetros
- Generalizan automáticamente para traslaciones espaciales de las entradas
- Inicialmente diseñadas para su aplicación en imágenes, pero su uso se ha extendido en textos

7.2.1 Convolución

$$h(t) = (f * g)(t) = \int f(s)g(t-s)ds$$

Y para el caso discreto:

$$h(t) = (f * g)(t) = \sum_{s=-\infty}^{\infty} f(s)g(t-s)$$

En CNNs, - La función f es el *input* I (Tensor) - La función g es el *kernel* K (Tensor de parámetros) - La salida h es el *mapeo de características (feature map)*

En CNNs, generalmente realizamos convolución en dos o más ejes. Por ejemplo, una imagen I en 2-D con un kernel K 2-D.

$$H(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

Y también puede expresarse como una forma de cross-correlation:

$$H(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

Pero, en términos prácticos ¿qué es la convolución? ¿para qué nos sirve?

En el caso discreto, la convolución es como una multiplicación de matrices, pero con otra dinámica:

(animación)

La dinámica de las operaciones de convolución depende de varios factores. Los dos más importantes son `stride` y `zero padding` (ver animación).

Un esquema común de una capa de convolución se ve así:

7.2.2 Pooling

Una operación para reducir el tamaño de la transformación o mapeo usando alguna función que resume la información contenida en ciertas regiones.

Observa que, el pooling tiene como input una capa que puede ser el resultado de una convolución o *una transformación de ella*. Es común realizar pooling a una capa de convolución a la que se le aplica alguna función de activación.

$$\text{pooling}(relu(conv))$$

Demos ilustrativos: - MNIST Stanford - Adam Harley, Carnegie Mellon University - Convolución en tensores. Stanford

Otros temas interesantes relacionados con CNN (que no veremos por ahora):

- Visualización de las capas de convolución. ¿Qué está aprendiendo la red?
- Transfer learning. Usar pesos aprendidos en otros corpus de imágenes para un problema específico, aunque tengamos pocos datos.

7.2.3 Redes convolucionales para textos

Una convolución para datos tipo secuencia (de cualquier tipo), puede definirse como una convolución 1-D:

Zhang et. al., Dive into Deep Learning, 2020

Y como antes, podemos incluir más canales:

Zhang et. al., Dive into Deep Learning, 2020

En datos tipo secuencia, es más común considerar los canales como variables indexados también por el tiempo t, lo que resulta en una secuencia multivariada. Este caso es similar a la convolución 2-D con un solo canal:

Zhang et. al., Dive into Deep Learning, 2020

Convolución 1-D para textos con padding.

Curso Stanford (2020)

Convolución 1-D con múltiples filtros y padding.

Curso Stanford (2020)

Convolución 1-D con múltiples filtros, padding y max pooling over time.

Curso Stanford (2020)

Convolución 1-D con múltiples filtros, padding y avg pooling over time.

Curso Stanford (2020)

Una arquitectura general, puede ilustrarse mediante:

Yoon Kim, 2014.

En este caso, el canal estático representa embeddings pre-entrenados, y el canal dinámico embeddings que se entrenan en el proceso vía fine-tunning.

O también:

Ye Zhang et al., 2016.

Ejemplo en Notebook

7.3 Redes recurrentes

(Rumelhart et al., 1986)

Redes neuronales especializadas para procesar datos secuenciales

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(T)},$$

donde $t \in 1, \ldots, T$ representa la posición en la secuencia (por ejemplo, el tiempo, o la palabra en un texto).

Las secuencias, pueden ser *muy grandes* o ser minibatches de *longitud variable*.

Considera un sistema dinámico

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \boldsymbol{\theta}),$$

donde incorporamos información a través de una señal $\mathbf{x}$ en el tiempo t. El sistema es recurrente, ya que el estado $\mathbf{h}^{(t)}$ depende del estado anterior $\mathbf{h}^{(t-1)}$.

Por ejemplo,

$$\mathbf{h}^{(3)} = f(\mathbf{h}^{(2)}, \mathbf{x}^{(3)}; \boldsymbol{\theta}) = f(f(\mathbf{h}^{(1)}, \mathbf{x}^{(3)}; \boldsymbol{\theta}); \boldsymbol{\theta})$$

Observa que $\boldsymbol{\theta}$ es el mismo para todo t, es decir, se comparten los parámetros.

Gráficamente (RNN simple, también llamada "*vanilla*":

Goodfellow et al., 2016.

Podemos pensar en las RNN simples como una *extensión* de las redes neuronales donde se les agrega una dinámica temporal:

Curso Stanford.

Podemos pensar en las RNN simples como una *extensión* de las redes neuronales donde se les agrega una dinámica temporal:

Curso Stanford.

Podemos pensar en las RNN simples como una *extensión* de las redes neuronales donde se les agrega una dinámica temporal:

Curso Stanford.

Podemos pensar en las RNN simples como una *extensión* de las redes neuronales donde se les agrega una dinámica temporal:

Curso Stanford.

Podemos pensar en las RNN simples como una *extensión* de las redes neuronales donde se les agrega una dinámica temporal:

Curso Stanford.

Podemos pensar en las RNN simples como una *extensión* de las redes neuronales donde se les agrega una dinámica temporal:

Curso Stanford.

Podemos pensar en las RNN simples como una *extensión* de las redes neuronales donde se les agrega una dinámica temporal:

Curso Stanford.

Podemos pensar en las RNN simples como una *extensión* de las redes neuronales donde se les agrega una dinámica temporal:

Curso Stanford.

¿Qué tipo de secuencias podemos modelar?

Básicamente, todas… por ejemplo, recuerda nuestro modelo de lenguaje:

Ajuste. Como todas las redes neuronales, el ajuste tiene dos pasos: Forward propagation y Back propagation:

- Forward propagation.

donde $J(\theta)$ es una función de costo.

- Forward propagation.

$$
\begin{aligned}
\mathbf{a}^{(t)} &= \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b} \\
\mathbf{h}^{(t)} &= \tanh(\mathbf{a}^{(t)}) \\
\mathbf{o}^{(t)} &= \mathbf{V}\mathbf{h}^{(t)} + \mathbf{c} \\
\hat{\mathbf{y}}^{(t)} &= \text{softmax}(\mathbf{o}^{(t)})
\end{aligned}
$$

- Backpropagation through time (BPTT).
- Backpropagation through time (BPTT).
  - NO entrenamos con la historia completa! (puede ser muy costoso)
  - Usamos secuencias $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)}$ como una "ventana" móvil
  - Usamos gradiente estocástico
  - Usamos batches de secuencias
  - Usamos máxima verosimilitud, por lo tanto, la función de costo es cross-entropy:

$$CE(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) = -\sum_i y_i^{(t)} \log \hat{y}_i^{(t)}.$$

    Por ejemplo, en el caso de modelos de lenguaje,

$$CE(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) = -\sum_{i \in V} y_i^{(t)} \log \hat{y}_i^{(t)}.$$

- Backpropagation through time (BPTT).
- Backpropagation through time (BPTT).
- Backpropagation through time (BPTT).
- Backpropagation through time (BPTT).
- Backpropagation through time (BPTT).

7.3.1 El problema de las dependencias a largo plazo

Considera un modelo de lenguaje para predecir la siguiente palabra:

- "*Las carreras de ciclismo son* _____"
- "*Aprendí a programar en Java hace 20 años. Trabajé en varios lugares haciendo diferentes actividades, siendo la programación solo una de ellas, luego decidí estudiar una carrera en computación donde he usado diferentes lenguajes de programación, sin embargo, mi lenguaje de programación favorito sigue siendo* _____"

En muchos casos, es necesario incorporar información del pasado "*antiguo*"

Intuitivamente

Conclusión:

Ocurren dos fenómenos:

- El gradiente explota: usamos *clipping gradients*

Ocurren dos fenómenos:

- El gradiente se desvanece: "*olvidar…*": Gated RNNs:
  - LSTM (Long Short-Term Memory, Hochreiter et al., 1997)
  - GRU (Gated Recurrent Units, Cho et al., 2014)

Antes: arquitectura RNN.

Arquitectura LSTM.

Nos permite controlar dinámicamente la escala de tiempo y olvidar el comportamiento de diferentes unidades.

Arquitectura LSTM (Resumen):

Arquitectura GRU. Simplifican las celdas de las LSTM al *combinar* las compuertas forget y update.

7.3.2 Redes recurrentes multicapa y bidireccionales

RNN multicapa (stacked):

RNN bidireccionales. Aplicables cuando tenemos acceso a la secuencia completa, por ejemplo, en clasificación de textos.

RNN bidireccionales. Aplicables cuando tenemos acceso a la secuencia completa, por ejemplo, en clasificación de textos.