Universidad
Carlos III de Madrid

Universidad Carlos III de Madrid

Métodos Bayesianos, Grado en Estadística y Empresa

Análisis y clasificación de textos

Fabio Scielzo Ortiz

Índice

1 Introducción

En este trabajo se va a realizar un análisis y clasificación de textos. Para ellos se utilizaran dos lenguajes de programación, `Python` y `R`. El trabajo puede dividirse en dos partes bien diferenciadas, una primera parte en la que se trabaja con `Python` y una segunda en la que se usa `R`.

En la primera parte, en la que trabajamos con `Python`, se llevará acabo una descripción y preprocesado del data-set con el que trabajaremos, posteriormente se llevara acabo un análisis de texto, y para finalizar se realizaran tareas de clasificación aplicando algoritmos de clasificación supervisada, especialmente el algoritmo de clasificación ingenua bayesiana.

En la parte en la que trabajamos con `R` se seguirán los pasos del ejemplo ilustrado en clase.

2 Carga de los datos (`Python`)

El data-set con el que vamos a trabajar contiene como observaciones noticias fechadas entre el 31 de marzo de 2015 y el 18 de febrero de 2018, y como variables la fecha, el título y el texto de la noticia, y si es una noticia falsa (fake new) o es verdadera (no fake new). La variable respuesta será `Fake` . Las variables predictoras que se usaran en el apartado de aplicación de algoritmos de clasificación no aparecen en el data-set original, pero serán creadas usando la información de la variable `texto`.

El data set ha sido obtenido de la pagina web Kaggle

Importamos la libreria `pandas`, que es la liberia de `Python` mas usada para la manipulación y manejo de datos en formato de tabla, es decir, data-frames.

```
import pandas as pd
```

Ahora importamos los datos, que originalmente estan distribuidos en dos data-sets, uno que contiene las fake news (`df_Fake`) y otro que contiene las no fake news (`df_True`):

```
df_Fake = pd.read_csv('Fake.csv')
df_True = pd.read_csv('True.csv')
```

Creamos una variable que indicará en nuestro data-set final si la noticia es fake o no fake:

```
df_Fake['Fake'] = 1
df_True['Fake'] = 0
```

Si para una noticia la nueva variable creada `Fake` toma el valor 1 , indica que es fake new, y si toma el 0 indica que no es fake new.

Ahora concatenamos (por filas) los dos data-sets anteriores, para generar el data-set con el que trabajaremos:

```
Fake_News_Data = pd.concat([df_Fake, df_True])
```

Seleccionamos las columnas (variables) de nuestro interés:

```
Fake_News_Data = Fake_News_Data.loc[: , ['Fake', 'title', 'text', 'date']
↪  ]
```

Añadimos un índice al data-set:

```
Fake_News_Data.index = range(0 , len(Fake_News_Data))
```

Ahora vamos a ver de qué tipo son nuestras variables en `Python` :

```
Fake_News_Data.dtypes
```

```
Fake      int64
title    object
text     object
date     object
dtype: object
```

El tipo `object` es propio de variables no cuantitativos, como categoricas o texto, y el tipo `int64` es propio de variables enteras.

En este caso dejaremos los types como están, salvo el de la variable `Fake` que es categorica y por tanto es más adecuado que su type sea `object`

```
Fake_News_Data['Fake'] = Fake_News_Data['Fake'].astype('object')
```

Calculamos el numero de valores faltantes (NA) en cada una de las variables:

```
Fake_News_Data.isnull().sum()
```

```
Fake     0
title    0
text     0
date     0
```

Vamos a imprimir el data set para hacernos una mejor idea de su contenido:

```
Fake_News_Data
```

```
      Fake                                              title

0        1   Donald Trump Sends Out Embarrassing New Year'...
1        1   Drunk Bragging Trump Staffer Started Russian ...
2        1   Sheriff David Clarke Becomes An Internet Joke...
3        1   Trump Is So Obsessed He Even Has Obama's Name...
4        1   Pope Francis Just Called Out Donald Trump Dur...
...    ...                                               ...
44893    0  'Fully committed' NATO backs new U.S. approach...
44894    0  LexisNexis withdrew two products from Chinese ...
44895    0  Minsk cultural hub becomes haven from authorities
```

```
44896    0  Vatican upbeat on possibility of Pope Francis ...
44897    0  Indonesia to buy $1.14 billion worth of Russia...
```

	text	date
0	Donald Trump just couldn t wish all Americans ...	December 31, 2017
1	House Intelligence Committee Chairman Devin Nu...	December 31, 2017
2	On Friday, it was revealed that former Milwauk...	December 30, 2017
3	On Christmas day, Donald Trump announced that ...	December 29, 2017
4	Pope Francis used his annual Christmas Day mes...	December 25, 2017
...	...	...
44893	BRUSSELS (Reuters) - NATO allies on Tuesday we...	August 22, 2017
44894	LONDON (Reuters) - LexisNexis, a provider of l...	August 22, 2017
44895	MINSK (Reuters) - In the shadow of disused Sov...	August 22, 2017
44896	MOSCOW (Reuters) - Vatican Secretary of State ...	August 22, 2017
44897	JAKARTA (Reuters) - Indonesia will buy 11 Sukh...	August 22, 2017

3 Descripción estadistica de los datos (Python)

Hacemos una breve descripción estadistica de las variables del data-set:

```
Fake_News_Data.describe(include='all')
```

	Fake	title
count	44898	44898
unique	2	38729
top	1	Factbox: Trump fills top jobs for his administ...
freq	23481	14

	date	text
count	44898	44898
unique	2397	38646
top	December 20, 2017	(no se muestra por tamaño excesivo)
freq	182	627

Esta tabla nos da alguna informacion relevante, como que en el data-set hay mas fake news que no fake news. Concretamente hay 44898 noticias, de las cuales 23481 son fakes y 44898-23481 = 21417 son no fakes.

Vamos ahora a realizar un análisis descriptivo del data-set algo más profundo.

3.1 Gráfico de barras de la variable respuesta (Fake)

Importamos algunas librerias necesarias para realizar este análisis en Python

Concretamente la libreria numpy da soporte para crear vectores y matrices grandes multi-dimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas. En general es una de las librerias de Python más empleadas junto con pandas

Tambien importamos las librerias `seaborn` y `matplotlib`que son muy empleadas para visualización de datos (creación de gráficos).

```
import numpy as np

import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt

sns.set(rc={'figure.figsize':(8,8)})
```

Vamos a calcular un gráfico de barras para la variable `Fake`:

```
prop_Fake_yes = len( Fake_News_Data.loc[ Fake_News_Data['Fake']== 1 , :] )
↪  / len(Fake_News_Data)

prop_Fake_no = len( Fake_News_Data.loc[ Fake_News_Data['Fake']== 0 , :] )
↪  / len(Fake_News_Data)
```

```
Fake_News_Data['proportion_Fakes'] = 0


for i in range(0, len(Fake_News_Data)):

    if Fake_News_Data['Fake'][i] == 1 :

        Fake_News_Data['proportion_Fakes'][i] = prop_Fake_yes

    else :

        Fake_News_Data['proportion_Fakes'][i] = prop_Fake_no
```

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()


p = sns.barplot(x='Fake', y='proportion_Fakes', data=Fake_News_Data,
↪  palette="Spectral")
p.set_yticks( np.arange(0, 0.85, 0.1)  )
p.set_xticklabels(['No', 'Yes'])
p.axes.set(xlabel='Fakes', ylabel='proportion')

fig.savefig('p.png', format='png', dpi=1200)
```

Las proporciones exactas de fake y no fake news son:

```
[prop_Fake_no , prop_Fake_yes]
```

```
[0.47701456635039424, 0.5229854336496058]
```

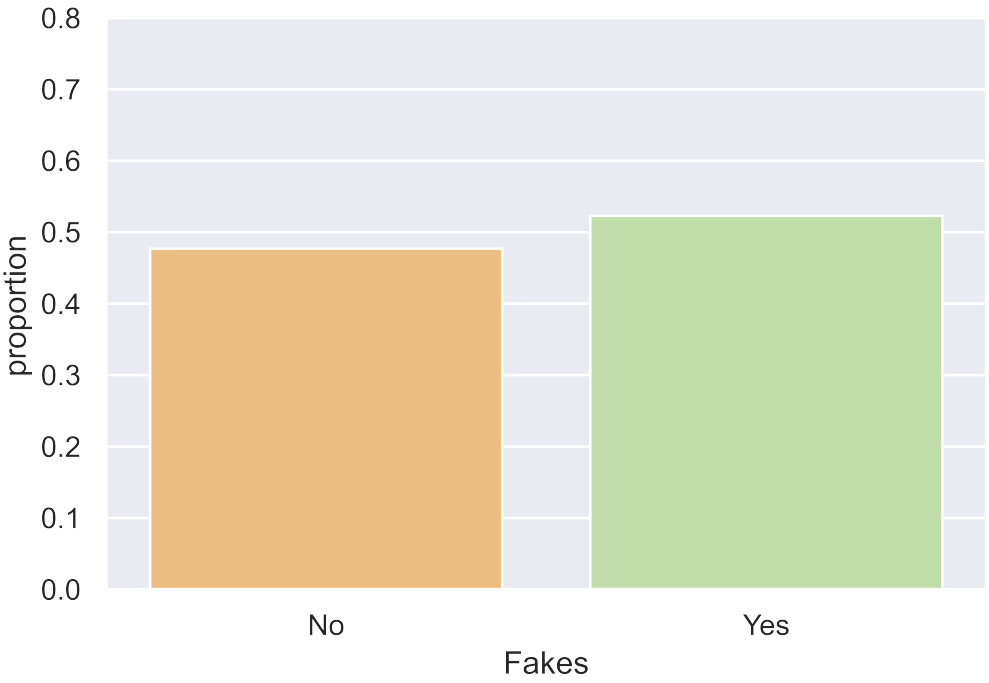

Figure 1: Gráfico de barras de la variable `Fake`

El número exacto de fake y no fake news es:

```
[prop_Fake_no*len(Fake_News_Data) , prop_Fake_yes*len(Fake_News_Data)]
```

```
[21417.0, 23481.0]
```

Eliminamos la columna `proportion_Fakes` del data-set, que ha sido creada solamente de manera auxiliar para poder generar el gráfico de barras anterior:

```
Fake_News_Data = Fake_News_Data.loc[ : , Fake_News_Data.columns !=
↪  'proportion_Fakes']
```

3.2 Número de palabras por noticia

Una forma de calcular en `Python` el número de palabras de cada notica es la siguiente:

```
Fake_News_Data['word_count'] =
↪  Fake_News_Data['text'].str.split().str.len()
```

Vamos a ver el data-set con la nueva columna `word_count` que contiene el nº de palabras por noticia

```
Fake_News_Data
```

```
      Fake   title
```

```
0        1   Donald Trump Sends Out Embarrassing New Year'...
1        1   Drunk Bragging Trump Staffer Started Russian ...
2        1   Sheriff David Clarke Becomes An Internet Joke...
3        1   Trump Is So Obsessed He Even Has Obama's Name...
4        1   Pope Francis Just Called Out Donald Trump Dur...

...    ...                                                ...

44893    0  'Fully committed' NATO backs new U.S. approach...
44894    0  LexisNexis withdrew two products from Chinese ...
44895    0  Minsk cultural hub becomes haven from authorities
44896    0  Vatican upbeat on possibility of Pope Francis ...
44897    0  Indonesia to buy $1.14 billion worth of Russia...
```

```
       text                                                date

0      Donald Trump just couldn t wish all Americans ...  December 31, 2017
1      House Intelligence Committee Chairman Devin Nu...  December 31, 2017
2      On Friday, it was revealed that former Milwauk...  December 30, 2017
3      On Christmas day, Donald Trump announced that ...  December 29, 2017
4      Pope Francis used his annual Christmas Day mes...  December 25, 2017

...                                                  ...                ...

44893  BRUSSELS (Reuters) - NATO allies on Tuesday we...   August 22, 2017
44894  LONDON (Reuters) - LexisNexis, a provider of l...   August 22, 2017
44895  MINSK (Reuters) - In the shadow of disused Sov...   August 22, 2017
44896  MOSCOW (Reuters) - Vatican Secretary of State ...   August 22, 2017
44897  JAKARTA (Reuters) - Indonesia will buy 11 Sukh...   August 22, 2017
```

```
       word_count

0             495
1             305
2             580
3             444
4             420

...           ...

44893         466
44894         125
44895         320
44896         205
44897         210
```

3.3 Numero medio de palabras por noticia en función de si son fake o no

Calculamos ahora la media de palabras de las fakes news y de la sno fake news. Es decir, el nº medio de palabras en el cojuntos de las noticias fake, y por otro lado en el conjutno de las no fake:

```
Fake_News_Data.groupby('Fake')['word_count'].mean()
```

```
Fake   Mean word_count

0        385.640099

1        423.197905
```

4 Preprocesado de texto

En este apartado se vana a hacer una serie de operaciones orientadas al preprocesado de texto, para poder posteriormente realizar analasis mas profundos, y para poder implementar algoritmos de clasificación sobre texto.

Este tipo de preprocesado es básico y fundamental en areas de la ciencia de datos que trabajan con texto, como son la mineria de texto (text minning), el procesamiento del lenguaje natural (PLN) y la recuperación de información (information retrival).

Una de las operaciones centrales del preproceso de textos es la `tokenización`.

4.1 Tokenizacion

Existen algunas librerias de `Python` que tienen funciones para realizar operaciones de tokenizacion, como por ejemplo las librerias `sklearn`, `nltk` o `spaCy`

En este caso no usaremos ninguna función de alguna de esas librerias, sino que crearemos nuestra propia función para realizar la tokenización.

Esta función esta totalmente inspirada en la función creada por el cientifico de datos Joaquín Amat Rodrigo, el cual es el creador del excelente blog sobre ciencia de datos Cienciadedatos.net. En este blog Joaquin tiene un articulo sobre analisis de texto en `Python` en el cual se encuentra la función que ahora vamos a presentar. Ademas muchas otras partes de este trabajo estan basadas en dicho articulo, es por ello que s ele hace una especial mención tanto aqui como en el apartado de bibliografia.

La función `limpiar_tokenizar` toma como input texto y devuelve como output un vector de tokens asociado a ese texto, es decir, un vector con las cadenas caracteres del texto, pero no con cualquier tipo, sino que la función no considera signos de puntuación , palabras que empiezan por “http”, números, espacios en blancos múltiples, tokens con longitud menor que 2.

Un token aqui es considerado como una cadena de caracteres, es decir, una concatenacion de símbolos (sin considerar el espacio en blanco como un símbolo).

Veamos un ejemplo de lo que consideramos tokens:

Dado el siguiente texto:

” Esto es 1 ejemplo de l’limpieza de6 TEXTO https://t.co/rnHPgyhx4Z @cienciadedatos #textmining ”

Los tokens (en sentido estricto, no en el sentido restrictivo que considera la función `limpiar_tokenizar`) asociados a dicho texto son:

[Esto , es , 1 , ejemplo , de , l'limpieza , de6 , TEXTO , https://t.co/rnHPgyhx4Z , @cienciadedatos , #textmining]

```
def limpiar_tokenizar(texto):

    import re

    '''
    Esta función limpia y tokeniza el texto en palabras individuales.
    El orden en el que se va limpiando el texto no es arbitrario.
    El listado de signos de puntuación se ha obtenido de:
    print(string.punctuation)
    y re.escape(string.punctuation)
    '''

    # Se convierte todo el texto a minúsculas:

    nuevo_texto = texto.lower()


    # Eliminacion de paginas web (palabras que empiezan por "http"):

    ## Las cadenas de caracteres que sean enlaces a webs no serán
    ↪  consideradas como tokens

    nuevo_texto = re.sub('http\S+', ' ', nuevo_texto)


    # Eliminacion de signos de puntuación:

    ## Si una cadena de caractrer contiene un signo de puntuacion estos
    ↪  serán eliminados y sustituidos por un  espacio en blanco. Si  por
    ↪  ejemplo tenemos las cadenas  '@FabioScielzo' y 'Fabio@Scielzo' ,
    ## la funcion las transforma en  'FabioScielzo' en el primer caso  y
    ↪  en el par de cadenas 'Fabio' , 'Scielzo' en el segundo. Y si
    ↪  tenemos
    ## una cadena de signos d puntuacion  como '@#!' la elimina
    ↪  directamente.

    regex =
↪   '[\\!\\"\\#\\$\\%\\&\\'\\(\\)\\*\\+\\,\\-\\.\\/\\:\\;\\<\\=\\>\\?\\@\\[\\\\\\]\\^_\\

    nuevo_texto = re.sub(regex , ' ', nuevo_texto)


    # Eliminacion de numeros:

    ## Si una cadena de caracter tiene numeros estos serán eliminados y
    ↪  sustituidos por un espacio en blanco. Si por ejemplo tenemos las
    ↪  cadenas '4FabioScielzo' y 'Fabio44Scielzo' la funcion las
    ↪  transforma en 'FabioScielzo' y 'Fabio' , 'Scielzo' ,
    ↪  respectivamente. Ademas si una cadena solo contienen numeros, por
    ↪  ejemplo '123' la elimina directamente.
```

```
    nuevo_texto = re.sub("\d+", ' ', nuevo_texto)



    # Eliminacion de espacios en blanco multiples:

    ## Si tenemos en un texto dos o mas espacios en blanco consecutivos la
    ↪  funcion los transforma en un solo espacio en blanco. Por ejemplo
    ↪  si tenemos el texto "Fabio     es abogado" la funcion lo
    ↪  transforma en "Fabio es abogado".

    nuevo_texto = re.sub("\\s+", ' ', nuevo_texto)


    # Una vez que a un texto se le han aplicado las operaciones anteriores
    ↪  ya solo quede considerar las cadenas de caracteres de ese texto
    ↪  como tokens, ya que son cadenas con buenas propiedades, a saber,
    ↪  sin signos de puntuacion, sin numeros, sin links de web. Ademas la
    ↪  eliminacion de espacios en blanco multiples es fundamental para
    ↪  que la siguiente operacion funcione bien, ya que en el texto final
    ↪  resultante todas las cadenas estan separadas entre si por un solo
    ↪  espacio, y la siguiente operacion utiliza esa propiedad para
    ↪  identificar a las cadenas, que ya serán considerados tokens en
    ↪  sentido estricto.

    # Obtención de tokens:

    nuevo_texto = nuevo_texto.split(sep = ' ')


    # Eliminacion de tokens con una longitud menor que  2:

    ## Una ultima operacion es solo considerar los tokens obteenidos tras
    ↪  las operaciones anteriores que tengan un tamaño (nº de caracteres)
    ↪  igual o superior a 2 , es decir, dejar fuera tokens con solo un
    ↪  caracter.

    nuevo_texto = [token for token in nuevo_texto if len(token) >=  2]

    return(nuevo_texto)
```

Probamos el funcionamiento de la función `limpiar_tokenizar` con el mismo texto que fue usado antes como ejemplo ilustrativo.

```
test = "Esto es 1 ejemplo de l'limpieza de6 TEXTO  https://t.co/rnHPgyhx4Z
↪  @cienciadedatos #textmining"

print(limpiar_tokenizar(texto=test))
```

```
['esto', 'es', 'ejemplo', 'de', 'limpieza', 'de', 'texto', 'cienciadedatos', 'textmining']
```

Ahora probamos la función `limpiar_tokenizar` con la primera noticia del data-set `Fake_News_Data`:

```
Fake_News_Data['text'][0]
```

```
'Donald Trump just couldn t wish all Americans a Happy New Year and leave it at that. Ins
```

```
print(limpiar_tokenizar(texto=Fake_News_Data['text'][0]))
```

```
['donald', 'trump', 'just', 'couldn', 'wish', 'all', 'americans', 'happy', 'new', 'year',
```

Ahora aplicamos la función `limpiar_tokenizar` a cada una de las noticias del data-set `Fake_News_Data`

```
Fake_News_Data['text_tokenizado'] = Fake_News_Data['text'].apply(
↪  limpiar_tokenizar )
```

Creamos una columna que identifique las noticias:

```
Fake_News_Data['id_text'] = range(0, len(Fake_News_Data))
```

Vemos como queda tras estos cambios el data-set `Fake_News_Data`:

```
Fake_News_Data
```

	Fake	title	text	date
0	1	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn t wish all Americans ...	December 31, 2017
1	1	Drunk Bragging Trump Staffer Started Russian ...	House Intelligence Committee Chairman Devin Nu...	December 31, 2017
2	1	Sheriff David Clarke Becomes An Internet Joke...	On Friday, it was revealed that former Milwauk...	December 30, 2017
3	1	Trump Is So Obsessed He Even Has Obama's Name...	On Christmas day, Donald Trump announced that ...	December 29, 2017
4	1	Pope Francis Just Called Out Donald Trump Dur...	Pope Francis used his annual Christmas Day mes...	December 25, 2017
...	...	...		
44893	0	'Fully committed' NATO backs new U.S. approach...		
44894	0	LexisNexis withdrew two products from Chinese ...		
44895	0	Minsk cultural hub becomes haven from authorities		
44896	0	Vatican upbeat on possibility of Pope Francis ...		
44897	0	Indonesia to buy $1.14 billion worth of Russia...		

```
...                    ...                                          ...

44893  BRUSSELS (Reuters) - NATO allies on Tuesday we...   August 22, 2017
44894  LONDON (Reuters) - LexisNexis, a provider of l...   August 22, 2017
44895  MINSK (Reuters) - In the shadow of disused Sov...   August 22, 2017
44896  MOSCOW (Reuters) - Vatican Secretary of State ...   August 22, 2017
44897  JAKARTA (Reuters) - Indonesia will buy 11 Sukh...   August 22, 2017
```

```
       word_count              text_tokenizado                        id_text

0         495     [donald, trump, just, couldn, wish, all, ameri...      0
1         305     [house, intelligence, committee, chairman, dev...      1
2         580     [on, friday, it, was, revealed, that, former, ...      2
3         444     [on, christmas, day, donald, trump, announced,...      3
4         420     [pope, francis, used, his, annual, christmas, ...      4

...       ...                          ...                            ...

44893     466     [brussels, reuters, nato, allies, on, tuesday,...    44893
44894     125     [london, reuters, lexisnexis, provider, of, le...    44894
44895     320     [minsk, reuters, in, the, shadow, of, disused,...    44895
44896     205     [moscow, reuters, vatican, secretary, of, stat...    44896
44897     210     [jakarta, reuters, indonesia, will, buy, sukho...    44897
```

Creamos un nuevo data-frame solo con las columnas (variables) `id_text` , `text_tokenizado` y `Fake`, en ell que la columna `text_tokenizado` esta expandida, es decir, al ser una columna cuyos elementos son vectores, lo que se hace con la operacion `explode` es expandir cada uno de esos vectores en un nuevo data-frame, es decir, para cada uno de esos vectores se crean tantas filas en el nuevo data-frame como elementos hay en el vector, y en cada una de esas filas la columna `text_tokenizado` contendra un elemento del vector expandido. Visualmente es mas facil de entenderlo como se verá a continuación:

```
Fake_News_Tokens = Fake_News_Data.loc[:, ['id_text', 'text_tokenizado',
↪  'Fake'] ].explode(column='text_tokenizado')


# Renombramos la columna `text_tokenizado` como `token` :

Fake_News_Tokens =
↪  Fake_News_Tokens.rename(columns={'text_tokenizado':'token'})
```

Imprimimos el nuevo data-frame creado `Fake_News_Tokens` al expandir la columna `text_tokenizado` del data-frame `Fake_News_Data`

```
Fake_News_Tokens
```

```
       id_text       token    Fake
0            0      donald       1
0            0       trump       1
0            0        just       1
```

```
0            0      couldn      1
0            0        wish      1

...        ...         ...     ...

44897    44897  technology      0
44897    44897         and      0
44897    44897    aviation      0
44897    44897       among      0
44897    44897      others      0
```

5 Descripción estadística de los datos tras la tokenización

5.1 Numero de tokens del conjunto de noticias en funcion de si son fake o no

```
# nº de palabras (tokens) en el conjunto de textos clasificados como fake
↪  y en los no fake

Fake_News_Tokens.groupby(by='Fake')['token'].count()
```

```
Fake
0    7891501
1    9611544
Name: token, dtype: int64
```

5.2 Numero de tokens *únicos* del conjunto de noticias en funcion de si son fake o no

```
# nº de palabras (tokens) *unicos* en el conjunto de textos clasificados
↪  como fake y en los no fake

Fake_News_Tokens.groupby(by='Fake')['token'].nunique()
```

```
Fake
0    78020
1    85642
Name: token, dtype: int64
```

5.3 Numero de tokens en cada una de las noticias individualmente

```
# nº de palabras (tokens) en cada texto individual clasificados como fake
↪  y en los no fake

df1 = pd.DataFrame( Fake_News_Tokens.groupby(by = ["id_text" , "Fake"]
↪  )["token"].count().rename('nº_tokens') )
```

```
df1
```

		nº_tokens
id_text	Fake	
0	1	447
1	1	294
2	1	563
3	1	426
4	1	415

```
 ...       ...          ...

44893       0          433
44894       0          120
44895       0          307
44896       0          196
44897       0          197
```

Hay noticias que no tienen tokens :

```
df1.loc[df1['nº_tokens'] == 0, :]
```

```
                nº_tokens
id_text  Fake

9358      1           0
10923     1           0
11041     1           0
11190     1           0
11225     1           0

 ...     ...         ...

21857     1           0
21869     1           0
21870     1           0
21873     1           0
32451     0           0
```

Algunos ejemplos de estas noticias son los siguientes:

```
Fake_News_Data.loc[Fake_News_Data.id_text == 9358]
```

```
        Fake   title
9358     1     https://100percentfedup.com/served-roy-moore-v...


        text
9358    https://100percentfedup.com/served-roy-moore-v...


        date  word_count
9358              1


      text_tokenizado   id_text
9358        []           9358
```

```
Fake_News_Data.loc[Fake_News_Data.id_text == 10923]
```

```
       Fake      title
10923    1       TAKE OUR POLL: Who Do You Think President Trum...

       text     date            word_count   text_tokenizado   id_text
10923           May 10, 2017        0              []           10923
```

Nos quedamos por tanto solo con las noticias que tienen algun token :

```
df2 = df1.loc[df1['nº_tokens'] != 0, :]

df2
```

```
                nº_tokens
id_text   Fake

0          1          447
1          1          294
2          1          563
3          1          426
4          1          415

...       ...         ...

44893      0          433
44894      0          120
44895      0          307
44896      0          196
44897      0          197
```

Calculamos el numero medio de tokens para las noticas que tienen uno o mas tokens en funcion se si son fake o no:

```
df2.groupby("Fake")["nº_tokens"].agg(['mean'])
```

```
            mean
Fake
0     368.486225
1     422.169983
```

Se puede interpretar como la longitud media de las noticas fake y de las no fake

Hay diferencias entre lo obtenido mediante esta operación y lo obtenido al usar el siguiente código, que fue visto anteriormente:

```
Fake_News_Data['word_count'] =
↪  Fake_News_Data['text'].str.split().str.len()

Fake_News_Data.groupby('Fake')['word_count'].mean()
```

```
Fake   Mean word_count

0        385.640099

1        423.197905
```

Y esto es debido a que el código `Fake_News_Data['text'].str.split()` hace una operacion similar a la realizada por nuestra funcion `limpiar_tokenizar` pero **no exactamente igual**, y esto lleva a que con la primera opcion se obtiene un conjunto de tokens diferente al obtenido con la funcion `limpiar_tokenizar`, en los distintos documentos, y esto lleva a que la longitud de los documentos sea diferente si se consideran los tokens obtenidos con `Fake_News_Data['text'].str.split()` a si se usan los obtenidos con `limpiar_tokenizar`, llo que lleva a diferencias en las longitudes medias obtenidas.

```
Fake_News_Data['text'].str.split()
```

```
0        [Donald, Trump, just, couldn, t, wish, all, Am...
1        [House, Intelligence, Committee, Chairman, Dev...
2        [On, Friday,, it, was, revealed, that, former,...
3        [On, Christmas, day,, Donald, Trump, announced...
4        [Pope, Francis, used, his, annual, Christmas, ...
                              ...
44893    [BRUSSELS, (Reuters), -, NATO, allies, on, Tue...
44894    [LONDON, (Reuters), -, LexisNexis,, a, provide...
44895    [MINSK, (Reuters), -, In, the, shadow, of, dis...
44896    [MOSCOW, (Reuters), -, Vatican, Secretary, of,...
44897    [JAKARTA, (Reuters), -, Indonesia, will, buy, ...
```

Como se pueden ver con el código anterior se obtiene por ejemplo que '-' y ', Donald' son tokens , cuando con la función `limpiar_tokenizar` no serían considerados un tokens.

Otra forma de calcular lo anterior:

```
m0 = (
↪  Fake_News_Tokens.loc[Fake_News_Tokens['Fake']==0].groupby('id_text')['token'].count()
↪  ).mean()
```

```
m1 = (
↪  Fake_News_Tokens.loc[Fake_News_Tokens['Fake']==1].groupby('id_text')['token'].count()
↪  ).mean()
```

```
pd.DataFrame({'fake_new': [0,1] , 'tokens_mean':[m0 , m1]})
```

```
   fake_new  tokens_mean
0         0   368.469020
1         1   409.332822
```

5.4 Número de veces que aparece cada token en el conjunto de las noticias en funcion de si es fake o no

```
df = pd.DataFrame(  (Fake_News_Tokens.groupby(by = ["Fake", "token"])["token"].count().unstack(fill_value=0).stack().reset_index(name='frecuencia_token')

# .unstack(fill_value=0).stack() para que tambien aparezcan los tokens con count = 0 , si no solo aprecerian los que tienen count > 0.

df
```

	Fake	token	frecuencia_token
0	0	aa	22
1	0	aaa	7
2	0	aaaaaaaand	0
3	0	aaaaackkk	0
4	0	aaaaapkfhk	0
...	...	...	...
251605	1	""it	0
251606	1	""when	0
251607	1	•if	0
251608	1	$emoji1$	0
251609	1	$emoji2$	0

La salida anterior nos da para cada token el numero de veces que aparece en el conjunto de las fake news por un lado (Fake = 1), y por otro lado en el conjunto de las no fake (Fake=0)

Veamos algunos ejemplos para tokens concretos:

En la siguiente salida vemos el nº de veces que aparece el token 'yes' en eñ conjunto de las fake news (1775), asi como en el conjunto de las no fake news (336).

```
df.loc[df['token']=='yes' , ] # El token 'yes' aprece 1775 veces en el conjunto de las fake news y 336 en el de las no fake news
```

	Fake	token	frecuencia_token
116577	0	yes	336
242382	1	yes	1775

En la siguiente salida vemos el nº de veces que aparece el token 'true' en eñ conjunto de las fake news (2595), asi como en el conjunto de las no fake news (412).

```
df.loc[df['token']=='true' , ] # El token 'true' aparece 2595 veces en el conjunto de las fake news y 412 en el de las no fake news
```

	Fake	token	frecuencia_token
106608	0	true	412
232413	1	true	2595

En la siguiente salida podemos ver el nº de veces que aparece cada token en el conjunto de las no fake news.

```
df.loc[df['Fake']==0 , ]
```

```
         Fake       token  frecuencia_token

0           0          aa                22
1           0         aaa                 7
2           0  aaaaaaaand                 0
3           0   aaaaackkk                 0
4           0  aaaaapkfhk                 0

...       ...         ...               ...

125800      0        ""it                 1
125801      0      ""when                 1
125802      0         •if                 3
125803      0      $emoji1$               3
125804      0      $emoji2$               1
```

Y en la siguiente salida podemos ver el nº de veces que aparece cada token en el conjunto de las fake news.

```
df.loc[df['Fake']==1 , ]
```

```
        Fake       token  frecuencia_token
125805     1          aa                24
125806     1         aaa                 9
125807     1  aaaaaaaand                 1
125808     1   aaaaackkk                 1
125809     1  aaaaapkfhk                 1
...      ...         ...               ...
251605     1        ""it                 0
251606     1      ""when                 0
251607     1         •if                 0
251608     1      $emoji1$               0
251609     1      $emoji2$               0
```

Ahora vamos a ordenar los dos data-frames anteriores en función de la columna `frecuencia_token` , de mayor a menor, para así poder ver cuales son los tokens con mayor frecuencia tanto en el conjunto de las fake news, como en el de las no fake news.

```
df_fake_sort = df.loc[df['Fake']==1 ,
↪  ].sort_values(by=["frecuencia_token"],
↪  ascending=False).reset_index(drop=False)
```

```
df_no_fake_sort = df.loc[df['Fake']==0 ,
↪  ].sort_values(by=["frecuencia_token"],
↪  ascending=False).reset_index(drop=False)
```

Imprimimos las primeras 15 filas de cada uno de los nuevos data-frames ordenados:

```
df_fake_sort.head(15)
```

```
      index  Fake  token  frecuencia_token

0    229301     1    the            544521
1    230713     1     to            290882
2    199217     1     of            236735
3    129697     1    and            227349
4    174372     1     in            171433
5    229261     1   that            151789
6    176603     1     is            111278
7    162672     1    for             93538
8    176868     1     it             83693
9    199777     1     on             83661
10   232444     1  trump             79922
11   169936     1     he             79124
12   238650     1    was             67865
13   240547     1   with             63441
14   229776     1   this             58581
```

```
df_no_fake_sort.head(15)
```

```
      index  Fake token  frecuencia_token

0    103496     0   the            478548
1    104908     0    to            245378
2     73412     0    of            205193
3      3892     0   and            181715
4     48567     0    in            181082
5     73972     0    on            108459
6     90350     0  said             99054
7    103456     0  that             86723
8     36867     0   for             79705
9     50798     0    is             55298
10   114742     0  with             54327
11    44131     0    he             52605
12   112845     0   was             47892
13    14219     0    by             47871
14     5659     0    as             46935
```

Se puede observar que en ambas tablas la mayoria de los 15 tokens mas frecuentees se corresponden con artículos, preposiciones, pronombres, etc. En general, palabras que no aportan información relevante sobre el texto. A estas palabras se les conoce como **stopwords**. Para cada idioma existen distintos listados de stopwords, además, dependiendo del contexto, puede ser necesario adaptar el listado. Con frecuencia, a medida que se realiza un análisis se encuentran palabras que deben incluirse en el listado de stopwords.

5.5 Stop words

Vamos a obtener un listado de **stopwords** en ingles, ya que nuestros textos (noticias) están en ingles. Si estuvieran en varios idiosmas habra que formar un listado de stopwords para todos esos idomas.

Para ontener el listado de stopwords usaremos la libreria `nltk` (Natural Language Toolki) , una de las librerias mas importantes en `Python` en el área de procesamiento de lenguaje natural.

```
# pip install nltk
```

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
```

Obtenemos el listado de stopwords que provee `nltk` para el idioma inglés, y ademas le añadimos una lista extra de palabras que también vamos a considerar stopwords:

```
# Obtencion de listado de stopwords del ingles

stop_words = stopwords.words('english') + ["pic" , "getty", "quot", "acr",
↪  "filessupport", "flickr", "fjs", "js", "somodevilla", "var",
↪  "henningsen",
"ck", "cdata", "subscribing", "mcnamee", "amp", "wfb", "screenshot",
↪  "hesher","nyp", "cking", "helton", "raedle", "donnell",
"getelementbyid", "src", "behar", "createelement", "getelementsbytagname",
↪  "parentnode", "wnd","insertbefore",
"jssdk", "nowicki", "xfbml", "camerota", "sdk",  ""i" , ""the", ""we",
↪  "it's", "don't", ""this", ""it", ""a",
""if",  ""it's", "we're", "that's",  ""he", ""there", "i'm",  "he's",
↪  ""we're", "doesn't", "can't", ""i'm", ""in",
"suu", ""they", "you're", ""but", "didn't", ""you", "they're", ""no",
↪  ""as", ""very" , "there's", ""what",  ""and", "won't",
  ""to", ""that", ""one", "we've", ""when" , ""our", ""not", "'""
   ↪  ,""that's", ""these", ""there's", ""he's", "we'll", 'one',
    'would', 'like', 'us', 'even', 'could', 'two', 'many', 'angerer',
     ↪  'reilly']
```

Imprimimos la lista de stopwords que se van a considerar en este trabajo:

```
print(stop_words)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've
```

De los data-frames `df_fake_sort` y `df_no_fake_sort` eliminamos aquellos tokens que estan en la lista de stopwords:

```
df_fake_sort_not_StopWords = df_fake_sort[ ~
↪  df_fake_sort['token'].isin(stop_words) ] # ranking de tokens para las
↪  fake news sin stop words
```

```
df_no_fake_sort_not_StopWords = df_no_fake_sort[ ~
↪  df_no_fake_sort['token'].isin(stop_words) ] # ranking de tokens para
↪  las no fake news sin stop words
```

Imprimimos las primeras 15 filas de los nuevos data-frames creados:

```
df_fake_sort_not_StopWords.head(15)
```

	index	Fake	token	frecuencia_token
10	232444	1	trump	79922
31	216155	1	said	33763
34	206880	1	president	27801
35	203392	1	people	26591
56	144568	1	clinton	19209
59	198761	1	obama	18833
62	154174	1	donald	17789
67	128977	1	also	15420
69	196554	1	news	14688
73	196507	1	new	14414
75	171064	1	hillary	14184
77	230293	1	time	13854
79	224427	1	state	13471
82	239806	1	white	13194
84	237031	1	via	12830

```
df_no_fake_sort_not_StopWords.head(15)
```

	index	Fake	token	frecuencia_token
6	90350	0	said	99054
17	106639	0	trump	42755
26	87534	0	reuters	28880
28	81075	0	president	27128
36	98622	0	state	19912
41	41076	0	government	18484
44	70702	0	new	16849
47	46493	0	house	16480
48	98655	0	states	16380
49	86922	0	republican	16175
50	3172	0	also	15948
51	109089	0	united	15584
53	77587	0	people	14945
54	116463	0	year	14276
55	105051	0	told	14245

5.6 Ranking de tokens mas frecuentes en el conjunto de las noticas en funcion de si son fake y no fake tras eliminar stopwords

Una vez eliminadas las stopwords vamos a crear unos graficos de barras para representar el ranking de los 15 tokens mas frecuentes en el conjunto de las fake news por un lado, y por otro las no fake news:

```
fig, ax = plt.subplots()

sns.barplot(data=df_fake_sort_not_StopWords.head(15),
↪  x='frecuencia_token', y='token', color='tomato').set(title='Ranking 15
↪  Tokens in Fake News')

fig.savefig('p1.png', format='png', dpi=1200)
```

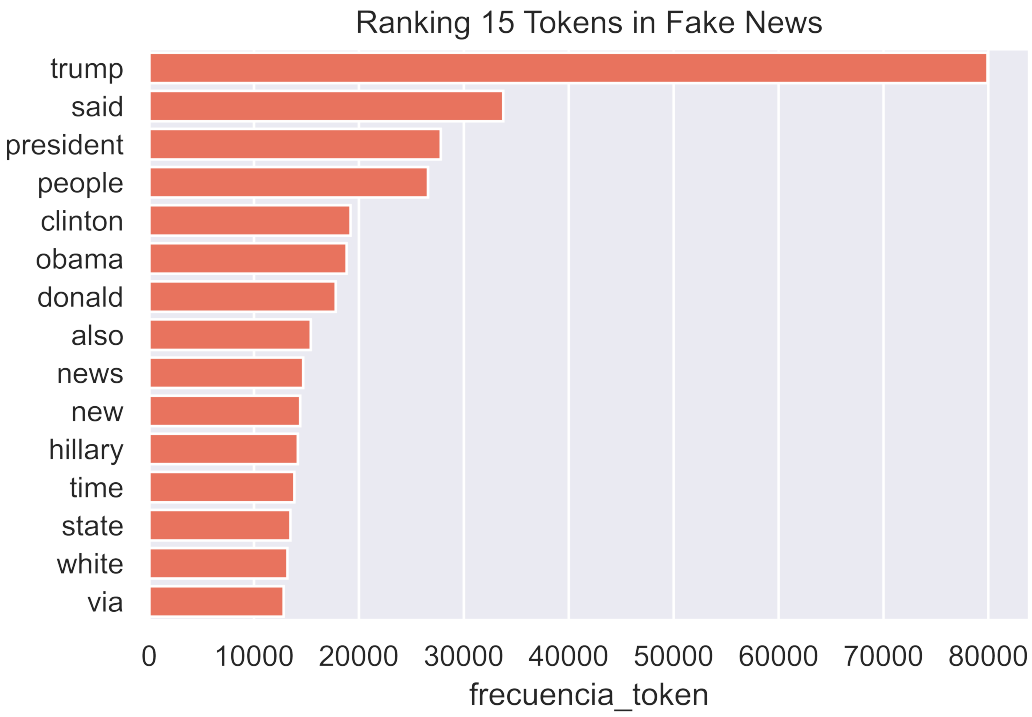

Figure 2: Ranking 15 Tokens in Fake News

```
fig, ax = plt.subplots()

sns.barplot(data=df_no_fake_sort_not_StopWords.head(15),
↪  x='frecuencia_token', y='token', color='cyan').set(title='Ranking 15
↪  Tokens in Not Fake News')

fig.savefig('p2.png', format='png', dpi=1200)
```

5.7 Odds Ratio

A continuación, se estudia qué palabras se utilizan de forma más diferenciada en cada tipo de noticia (fake / no fake), es decir, palabras que utiliza mucho en las fake news y que no se utilizan tanto en las no fakes, y viceversa.

Una forma de hacer este análisis es mediante el odds ratio de las frecuencias.

Sea $p_k1 = \frac{n_{k1}+1}{N_1+1}$ y $p_k0 = \frac{n_{k0}+1}{N_0+1}$

$$OR(Fake|NoFake, k) = \frac{p_{k1}}{p_{k0}}$$

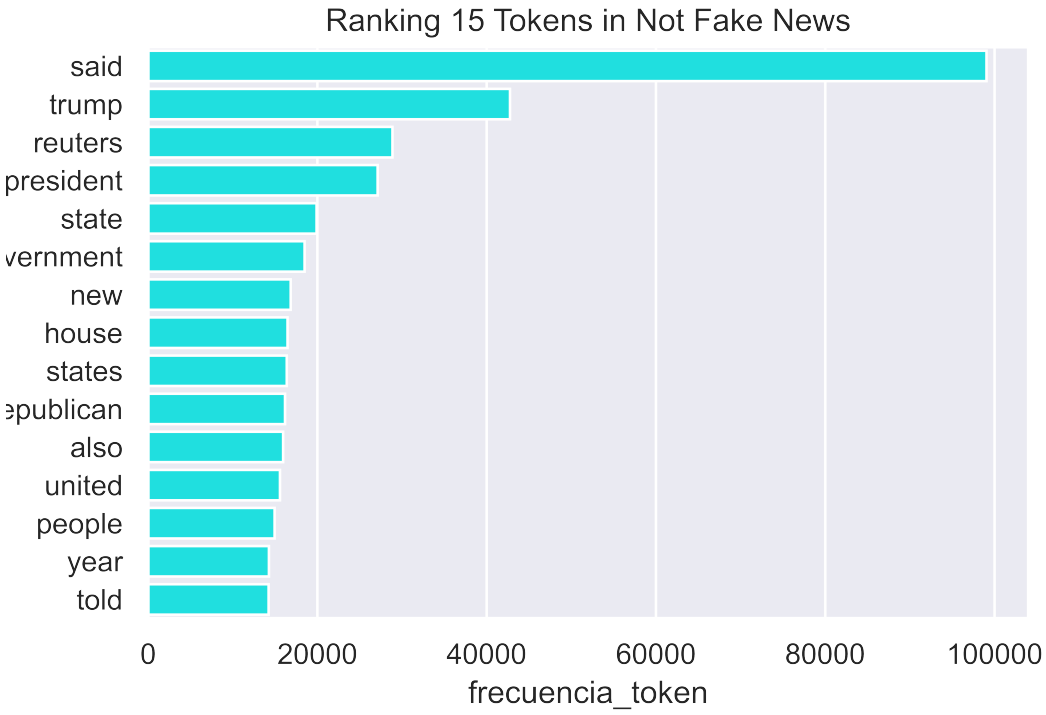

Figure 3: Ranking 15 Tokens in Not Fake News

Donde:

n_{k1} el número de veces que aparece el token k en las **fake news**.

n_{k0} el numero de veces que aparece el termino k en las **no fake news**.

N_1 es el número de tokens, contando repeticiones, que aparecen en las **fake news**.

N_0 es el número de tokens, contando repeticiones, que aparecen en las **no fake news**

Por tanto:

$p_{k1} \approx$ proporcion de apariciones del token k en las **fake news**

$p_{k0} \approx$ proporcion de apariciones del token k en las **no fake news**

Si $OddsRatio(k) = \dfrac{p_k 1}{p_k 0} = h$, entonces:

Si $h > 1 \Rightarrow$ el token k es h veces mas frecuente en las **fake news** que en las **no fake news**, ya que $p_{k1} = h \cdot p_{k0}$

Si $h \in (0, 1) \Rightarrow$ el token k es $1/h$ veces mas frecuente en las **no fake news** que en las **fake news**, ya que $p_{k0} = (1/h) \cdot p_{k1}$, donde $(1/h) > 1$

Si $h = 1 \Rightarrow$ el token k es igual de frecuente en las **fake news** que en las **no fake news**, ya que $p_{k1} = p_{k0}$

A continuacion definimos funciones para calcular n_{k1} y n_{k0} en `Python`

```
def n_k1(token) :

    n_k1 = df_fake_sort_not_StopWords.loc[
↪   df_fake_sort_not_StopWords['token']==token , 'frecuencia_token']

    return(n_k1)
```

```
def n_k0(token) :

    n_k0 = df_no_fake_sort_not_StopWords.loc[
↪   df_no_fake_sort_not_StopWords['token']==token , 'frecuencia_token']

    return(n_k0)
```

Probamos las funciones para algunos tokens concretos:

```
n_k0('trump')
```

```
17    42755
Name: frecuencia_token
```

```
n_k1('trump')
```

```
10    79922
Name: frecuencia_token
```

Estas salidas nos indican que el nº de veces que aparece el token 'trump' en el conjunto de las fake news es 79922 , mientras que en el conjunto de las no fake news es 42755.

N_0 y N_1 coinciden con el nº de tokens, contando repeticiones y sin considerar las stopwords, que aparecen el las no fake y fake news, respectivamente:

```
Fake_News_Tokens_not_StopWords = Fake_News_Tokens[ ~
↪   Fake_News_Tokens['token'].isin(stop_words) ]

Fake_News_Tokens_not_StopWords
```

```
       id_text       token Fake

0            0      donald    1
0            0       trump    1
0            0        wish    1
0            0   americans    1
0            0       happy    1

...        ...         ...  ...

44897    44897      energy    0
44897    44897  technology    0
44897    44897    aviation    0
44897    44897       among    0
44897    44897      others    0
```

```
Fake_News_Tokens_not_StopWords.groupby(by='Fake')['token'].count()
```

```
Fake
0    4782198
1    5396339

Name: token
```

```
N0 = Fake_News_Tokens_not_StopWords.groupby(by='Fake')['token'].count()[0]

N1 = Fake_News_Tokens_not_StopWords.groupby(by='Fake')['token'].count()[1]
```

```
N0
```

```
4782198
```

```
N1
```

```
5396339
```

Como ejemplo vamos a calcular el Odds Ratio fake - no fake para el toke 'trump' :

```
n_k0('trump') / N0
```

```
17    0.00894
Name: frecuencia_token, dtype: float64
```

```
n_k1('trump') / N1
```

```
10    0.01481
Name: frecuencia_token, dtype: float64
```

```
# Odds Ratio fake - no fake para el token 'trump'

float( n_k0('trump') / N0 ) / float( n_k1('trump') / N1 )
```

```
1.6565622548396417
```

Por tanto el token 'trump' es 1.66 veces mas frecuente en las fake news que en las no fake.

```
df1 =
↪  df_fake_sort_not_StopWords.sort_values(by=["token"]).reset_index(drop=True)
df1
```

```
         index  Fake       token  frecuencia_token
0       125805     1          aa                24
1       125806     1         aaa                 9
2       125807     1  aaaaaaaand                 1
3       125808     1   aaaaackkk                 1
```

```
4       125809     1  aaaaapkfhk                 1
...        ...   ...         ...               ...
125561  251605     1        ""it                 0
125562  251606     1      ""when                 0
125563  251607     1          if                 0
125564  251608     1      emoji1                 0
125565  251609     1      emoji2                 0
```

```
df0 =
↪  df_no_fake_sort_not_StopWords.sort_values(by=["token"]).reset_index(drop=True)
df0
```

```
         index  Fake       token  frecuencia_token
0            0     0          aa                22
1            1     0         aaa                 7
2            2     0  aaaaaaaand                 0
3            3     0   aaaaackkk                 0
4            4     0  aaaaapkfhk                 0
...        ...   ...         ...               ...
125561  125800     0        ""it                 1
125562  125801     0      ""when                 1
125563  125802     0          if                 3
125564  125803     0       emoji1                3
125565  125804     0       emoji2                1
```

```
n_k0_vector = df0['frecuencia_token']

n_k1_vector = df1['frecuencia_token']


Odds_ratio = ( ( n_k1_vector + 1 ) / ( N1 + 1) ) / ( ( n_k0_vector + 1 ) /
↪  ( N0 + 1) )
```

```
df0['Odds_ratio_Fake_NotFake'] = Odds_ratio
df1['Odds_ratio_Fake_NotFake'] = Odds_ratio

df0['Odds_ratio_NotFake_Fake'] = 1 / df0["Odds_ratio_Fake_NotFake"]
df1['Odds_ratio_NotFake_Fake'] = 1 / df1["Odds_ratio_Fake_NotFake"]
```

```
df0
```

```
         index  Fake       token  frecuencia_token  Odds_ratio_Fake_NotFake
0            0     0          aa                22                 0.963253
1            1     0         aaa                 7                 1.107741
2            2     0  aaaaaaaand                 0                 1.772386
3            3     0   aaaaackkk                 0                 1.772386
4            4     0  aaaaapkfhk                 0                 1.772386
...        ...   ...         ...               ...                      ...
125561  125800     0        ""it                 1                 0.443097
```

```
125562  125801     0      ""when                 1                  0.443097
125563  125802     0          if                 3                  0.221548
125564  125803     0    $emoji1$                 3                  0.221548
125565  125804     0    $emoji2$                 1                  0.443097

        Odds_ratio_NotFake_Fake
0                      1.038149
1                      0.902738
2                      0.564211
3                      0.564211
4                      0.564211
...                         ...
125561                 2.256845
125562                 2.256845
125563                 4.513689
125564                 4.513689
125565                 2.256845
```

```
df1
```

```
         index  Fake       token  frecuencia_token  Odds_ratio_Fake_NotFake  \
0       125805     1          aa                24                 0.963253
1       125806     1         aaa                 9                 1.107741
2       125807     1  aaaaaaaand                 1                 1.772386
3       125808     1   aaaaackkk                 1                 1.772386
4       125809     1  aaaaapkfhk                 1                 1.772386
...        ...   ...         ...               ...                      ...
125561  251605     1        ""it                 0                 0.443097
125562  251606     1      ""when                 0                 0.443097
125563  251607     1          if                 0                 0.221548
125564  251608     1    $emoji1$                 0                 0.221548
125565  251609     1    $emoji2$                 0                 0.443097

        Odds_ratio_NotFake_Fake
0                      1.038149
1                      0.902738
2                      0.564211
3                      0.564211
4                      0.564211
...                         ...
125561                 2.256845
125562                 2.256845
125563                 4.513689
125564                 4.513689
125565                 2.256845
```

```
df0.sort_values(by=["Odds_ratio_Fake_NotFake"],
↪  ascending=False).reset_index(drop=True).head(5)
```

```
    index  Fake           token  frecuencia_token  Odds_ratio_Fake_NotFake
0   35830     0         finicum                 0               320.801884
1  114264     0       wikimedia                 0               200.279629
```

```
2  109040     0  uninterruptible                 0               189.645313
3   78372     0     philosophers                 0               186.100540
4   60711     0           lovable                0               183.441961

   Odds_ratio_NotFake_Fake
0                 0.003117
1                 0.004993
2                 0.005273
3                 0.005373
4                 0.005451
```

```
df0.sort_values(by=["Odds_ratio_NotFake_Fake"],
↪  ascending=False).reset_index(drop=True).head(5)
```

```
    index  Fake      token  frecuencia_token  Odds_ratio_Fake_NotFake
0  106864     0    trump’s             11629                 0.000076
1   72989     0    obama’s              2132                 0.000415
2   18791     0  clinton’s              1604                 0.000552
3   76630     0    party’s              1101                 0.000804
4   98675     0    state’s              1010                 0.000877

   Odds_ratio_NotFake_Fake
0            13123.551362
1             2406.924768
2             1811.117793
3             1243.521376
4             1140.834946
```

```
df1.sort_values(by=["Odds_ratio_Fake_NotFake"],
↪  ascending=False).reset_index(drop=True).head(5)
```

```
    index  Fake            token  frecuencia_token  Odds_ratio_Fake_NotFake
0  161635     1          finicum               361               320.801884
1  240069     1        wikimedia               225               200.279629
2  234845     1  uninterruptible               213               189.645313
3  204177     1     philosophers               209               186.100540
4  186516     1           lovable               206               183.441961

   Odds_ratio_NotFake_Fake
0                 0.003117
1                 0.004993
2                 0.005273
3                 0.005373
4                 0.005451
```

```
df1.sort_values(by=["Odds_ratio_NotFake_Fake"],
↪  ascending=False).reset_index(drop=True).head(5)
```

```
    index  Fake      token  frecuencia_token  Odds_ratio_Fake_NotFake
0  232669     1    trump’s                 0                 0.000076
```

```
1  198794     1    obama's                0                 0.000415
2  144596     1  clinton's                0                 0.000552
3  202435     1    party's                0                 0.000804
4  224480     1    state's                0                 0.000877

   Odds_ratio_NotFake_Fake
0             13123.551362
1              2406.924768
2              1811.117793
3              1243.521376
4              1140.834946
```

Notese que en ambos data sets las columnas Odds_ratio_Fake_NotFake y Odds_ratio_NotFake_Fake son las mismas, por tanto podemos construir un nuevo data set solo con esas columnas y otra para los tokens, a partir de cualquiera de esos dos data-sets.

```
Odds_ratio_df = df1.loc[: , ['token', 'Odds_ratio_Fake_NotFake' ,
↪  'Odds_ratio_NotFake_Fake']]

Odds_ratio_df
```

```
             token  Odds_ratio_Fake_NotFake  Odds_ratio_NotFake_Fake
0               aa                 0.963253                 1.038149
1              aaa                 1.107741                 0.902738
2       aaaaaaaand                 1.772386                 0.564211
3        aaaaackkk                 1.772386                 0.564211
4       aaaaapkfhk                 1.772386                 0.564211
...            ...                      ...                      ...
125561        ""it                 0.443097                 2.256845
125562      ""when                 0.443097                 2.256845
125563          if                 0.221548                 4.513689
125564    $emoji2$                 0.221548                 4.513689
125565    $emoji1$                 0.443097                 2.256845
```

```
Odds_ratio_df.sort_values(by=["Odds_ratio_Fake_NotFake"],
↪  ascending=False).head(15)
```

```
                 token  Odds_ratio_Fake_NotFake  Odds_ratio_NotFake_Fake
35775          finicum               320.801884                 0.003117
114071       wikimedia               200.279629                 0.004993
108870  uninterruptible               189.645313                 0.005273
78242      philosophers               186.100540                 0.005373
60612           lovable               183.441961                 0.005451
91113           savants               182.555768                 0.005478
67583         moralists               182.555768                 0.005478
97785            spore               182.555768                 0.005478
84324          rascals               181.669575                 0.005504
32976       evangelists               181.669575                 0.005504
63302        masochists               181.669575                 0.005504
11482            boiler               172.586096                 0.005794
13727             bundy               170.813710                 0.005854
92025        screengrab               167.490486                 0.005970
113747           whined               166.604293                 0.006002
```

```
Odds_ratio_df.sort_values(by=["Odds_ratio_NotFake_Fake"],
↪  ascending=False).head(15)
```

	token	Odds_ratio_Fake_NotFake	Odds_ratio_NotFake_Fake
106696	trump’s	0.000076	13123.551362
72874	obama’s	0.000415	2406.924768
18756	clinton’s	0.000552	1811.117793
76500	party’s	0.000804	1243.521376
98529	state’s	0.000877	1140.834946
80975	president’s	0.000979	1021.222183
83999	rakhine	0.000987	1013.323226
1242	administration’s	0.001157	864.371483
88673	rohingya	0.001294	772.969276
117944	zuma	0.001298	770.712432
82344	puigdemont	0.001372	728.960807
17524	china’s	0.001400	714.291317
89715	russia’s	0.001439	695.108137
21888	country’s	0.001541	648.842823
69047	myanmar	0.001579	633.496280

```
fig, ax = plt.subplots()

sns.barplot(data=Odds_ratio_df.sort_values(by=["Odds_ratio_Fake_NotFake"],
↪  ascending=False).head(15) ,
                x='Odds_ratio_Fake_NotFake', y='token',
↪  color='tomato').set(title='Ranking 15 most representative tokens in
↪  Fake News')

fig.savefig('p3.png', format='png', dpi=1200)
```

Vamos a hacer un pequeño analisis de los tokens que son los mas representativos para las fake news, es decir, aquellos tokens con mayor odds ratio fake - no fake, esto es, aquellos que son mucho mas frecuentes en las fake news que en las no fake news.

- El token mas representativa de las fake news analizadas es ‘finicum’
  - El token ‘finicum’ podria hace referencia a Robert LaVoy Finicum, que según Wikipedia fue uno de los militantes estadounidenses que organizaron una ocupación armada del Refugio Nacional de Vida Silvestre Malheur en enero de 2016. Después de que comenzó, la fuerza de ocupación se organizó como Ciudadanos por la Libertad Constitucional. , de la que Finicum fue portavoz. Fue la única víctima mortal de la ocupación. El 26 de enero de 2016, agentes del orden público intentaron arrestar a Finicum y a otros líderes de la ocupación mientras viajaban por una carretera remota para reunirse con simpatizantes en el condado vecino. Cuando el camión de Finicum finalmente fue detenido por una barricada, salió del vehículo hacia la nieve profunda y le dispararon, pero los oficiales fallaron. Finicum hizo dos movimientos a su chaqueta mientras le gritaba a la policía que tendrían que dispararle. Luego, Finicum fue asesinado a tiros. Más tarde, los oficiales encontraron un arma cargada en su bolsillo.
- El segundo token mas representativo de las fakes news es ‘wikipedia’

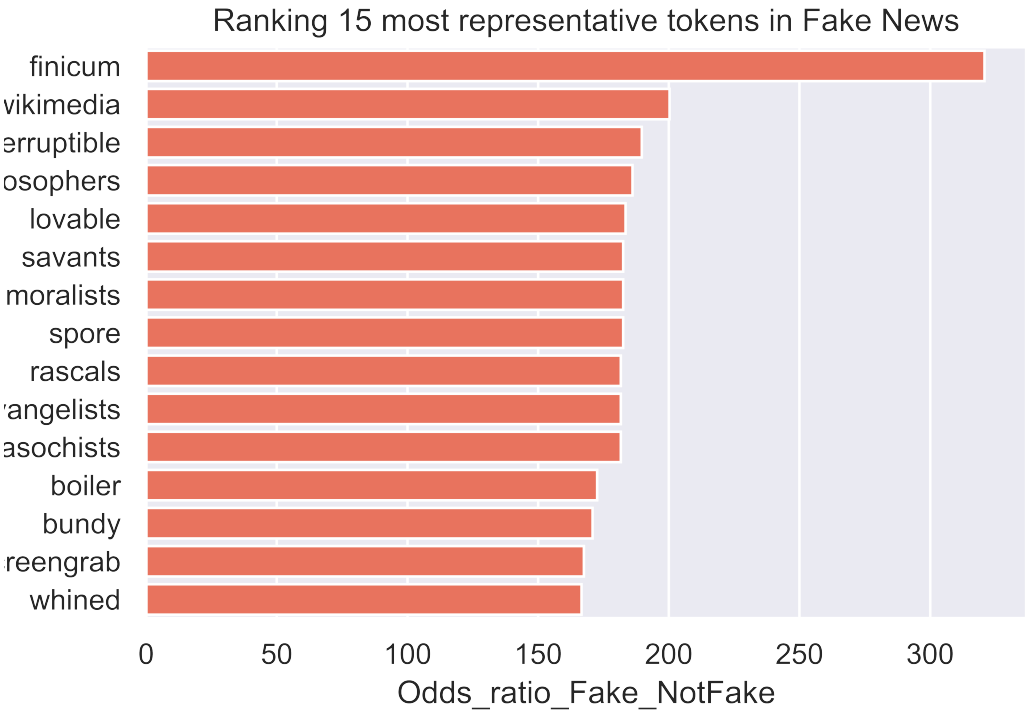

Figure 4: Ranking de los 15 tokens mas representativos de las Fake News

  - La Fundación Wikimedia (en inglés: Wikimedia Foundation, Inc.) es una organización sin ánimo de lucro. Es la organización matriz de Wikipedia

- El tercer token mas representativo de las fakes news es 'uninterruptible' que significa 'ininterrumpible' en español.
- El cuarto token mas representativo de las fakes news es 'philosophers' que significa 'filosofos' en español.
- El quinto token mas representativo de las fakes news es 'lovable' que significa 'amable' en español.
- El sexto token mas representativo de las fakes news es 'savants' que significa 'sabios' en español.
- El septimo token mas representativo de las fakes news es 'moralist' que significa 'moralistas' en español.
- El octavo token mas representativo de las fakes news es 'spore' que significa 'espora' en español.
- El noveno token mas representatico de las fake news es 'rascals' que podria hacer alusion a la pelicula 'The Little Rascals' en la cual participo como parte del elenco de actores el ex-presidente de Estados Unidos Donald Trump.
- El decimo token mas representativo es 'evangelist' que significa 'evangelistas' en español.
- El undecimo token mas representativo es 'masochist' que significa 'masoquista' en español.
- El duodecimo token mas representativo es 'boiler' que significa 'caldera' en español.

- El decimotercero token mas representativo es 'bundy' que significa 'paquete' en español.
- El decimocuarto token mas representativo es 'screengrab' que significa 'captura de pantalla' en español.
- El decimoquinto token mas representativo es 'whined' que significa 'quejarse' en español.

```
fig, ax = plt.subplots()

sns.barplot(data=Odds_ratio_df.sort_values(by=["Odds_ratio_NotFake_Fake"],
↪  ascending=False).head(15) ,
                x='Odds_ratio_NotFake_Fake', y='token',
↪  color='cyan').set(title='Ranking 15 most representative tokens in Not
↪  Fake News')

fig.savefig('p4.png', format='png', dpi=1200)
```

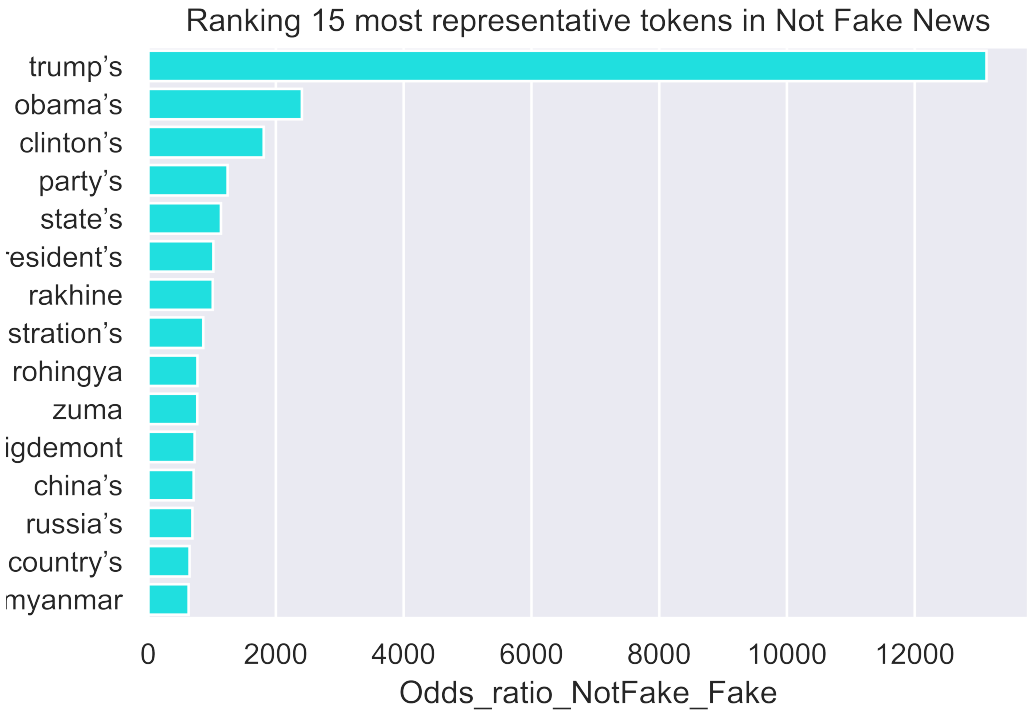

Figure 5: Ranking de los 15 tokens mas representativos de las No Fake News

Vamos a hacer un pequeño analisis de los tokens que son los mas representativos para las fake news, es decir, aquellos tokens con mayor odds ratio fake - no fake, esto es, aquellos que son mucho mas frecuentes en las fake news que en las no fake news.

- El token mas representativa de las fake news analizadas es 'trump's'
  - Este token hace referencia a Donald John Trump es un empresario, director ejecutivo, inversor en bienes inmuebles, personalidad televisiva y político estadounidense que ejerció como el 45.º presidente de los Estados Unidos de América desde el 20 de enero de 2017 hasta el 20 de enero de 2021

- El segundo token mas representativo de las fakes news es 'obama's'
  - Este token hace referencia a Barack Hussein Obama es un político estadounidense que ejerció como el 44.º presidente de los Estados Unidos de América desde el 20 de enero de 2009 hasta el 20 de enero de 2017
- El tercer token mas representativo de las fakes news es 'clinton's'
  - William Jefferson Clinton es un político y abogado estadounidense que ejerció como el 42.º presidente de los Estados Unidos de América de 1993 a 2001
- El cuarto token mas representativo de las fakes news es 'party's' que hace referencia a 'partidos politicos' en español.
- El quinto token mas representativo de las fakes news es 'state's' que significa 'estados' en español.
- El sexto token mas representativo de las fakes news es 'president's' que significa 'presidentes' en español.
- El septimo token mas representativo de las fakes news es 'rakhine'
  - Rakhine es un estado de Birmania.
- El octavo token mas representativo de las fakes news es 'administration's' que significa 'gobierno' en español.
- El noveno token mas representatico de las fake news es 'rohingya'
  - Los rohingya son un grupo étnico musulmán de Birmania (Myanmar) que desde 2017 fue objeto, según la ONU, de una limpieza étnica por parte de las autoridades birmanas que obligó a la mayoría de sus integrantes a refugiarse en la vecina Bangladés.
- El decimo token mas representativo es 'zuma' que podria hacer referencia a Jacob Zuma

```
- Jacob Gedleyihlekisa Zuma  es un político sudafricano que ejerció como el cuarto Pr
```

- El undecimo token mas representativo es 'puigdemont' que hace referencia a Carles Puigdemont

```
- Carles Puigdemont i Casamajón es un político y periodista español, diputado al
```

- El duodecimo token mas representativo es 'china's' que hace referencia a China.
- El decimotercero token mas representativo es 'russia's' que hace referencia a Rusia.
- El decimocuarto token mas representativo es 'county's' que significa 'país' en español.
- El decimoquinto token mas representativo es 'myanmar'

```
- Birmania o Myanmar  denominada oficialmente República de la Unión de Myanmar, es u
```

La interpretacion de estos términos no es una tarea en la que necesariamente el cientifico de datos que los ha obtenido pueda aportar un gran valor. Desde mi punto de vista, es mas adecuado que la interpretacion de estas palabras se haga por parte de expertos en el contexto en el que se encuentran las noticias analizadas, a saber, en el contexto politico, economico y social que rodeaba a Estados Unidos entre los años 2015 y 2018.

6 Term frequency – Inverse document frequency (Tf - Idf)

Siguiendo a Joaquin Amat Rodrigo , creador de Cienciadedatos.net y la entrada de Wikipedia

Uno de los principales intereses en **text mining**, **natural language processing** e **information retrieval** es cuantificar la temática de un texto, así como la importancia de cada término que lo forma. Una manera sencilla de medir la importancia de un término dentro de un documento es utilizando la frecuencia con la que aparece (tf, term-frequency). Esta aproximación, aunque simple, tiene la limitación de atribuir mucha importancia a aquellas palabras que aparecen muchas veces aunque no aporten información selectiva. Por ejemplo, si la palabra matemáticas aparece 5 veces en un documento y la palabra página aparece 50, la segunda tendrá 10 veces más peso a pesar de que no aporte tanta información sobre la temática del documento.

Para solucionar este problema se pueden ponderar los valores tf multiplicándolos por la inversa de la frecuencia con la que el término en cuestión aparece en el resto de documentos (idf). Asi se obtienee el estadístico **tf-idf**, que se consigue reducir el valor de aquellos términos que aparecen en muchos documentos y que, por lo tanto, no aportan información selectiva.

El estadístico **tf-idf** es una medida numérica que expresa cuán relevante es un término para un documento dentro de una colección de documentos. Esta medida se utiliza a menudo como un factor de ponderación en la recuperación de información y la minería de texto. El valor **tf-idf** aumenta proporcionalmente al número de veces que una palabra aparece en el documento, pero es compensada por la frecuencia de la palabra en la colección de documentos, lo que permite manejar el hecho de que algunas palabras son generalmente más comunes que otras.

Variaciones del esquema de peso **tf-idf** son empleadas frecuentemente por los motores de búsqueda como herramienta fundamental para medir la relevancia de un documento dada una consulta del usuario, estableciendo así una ordenación o ranking de los mismos. **Tf-idf** también puede utilizarse exitosamente para el filtrado de las denominadas stop-words (palabras que suelen usarse en casi todos los documentos).

6.1 Definición formal del estadistico tf-idf

Term Frequency (tf)

- Versión simple:

$$tf(k,d) = \frac{n(k,d)}{size(d)}$$

  Donde:

  $n(k,d)$ es el número de veces que aparece el término k en el documento d

  $size(d)$ es el nº de terminos del documento d

- Versión normalizada (para evitar una predisposición hacia los documentos largos):

$$tf_{norm}(k,d) = \frac{tf(k,d)}{Max\{tf(k,d) \ / \ k \in d\}}$$

Inverse Document Frequency (idf)

- Versión simple:

$$idf(k) = log\left(\frac{n(D)}{n(k,D)}\right)$$

  Donde :

  $n(D) = \#D$ es el número total de documentos , donde D es el conjunto de los documentos

  $n(D,k) = \#\{d \in D / k \in d\}$ el número de documentos que contienen el término k

  $log()$ es la función logaritmo en base e

- Versión `sklearn` si smooth_idf = True

$$idf(k) = log\left(\frac{n(D)}{n(k,D)}\right) + 1$$

- Versión `sklearn` si smooth_idf = False

$$idf(k) = log\left(\frac{n(D)+1}{n(k,D)+1}\right) + 1$$

Estadístico tf-idf

- Versión simple:

$$tfidf(k,d) = tf(k,d) \cdot idf(k)$$

- Versión normalizada :

$$tfidf_{norm}(k,d) = tf_{norm}(k,d) \cdot idf(k)$$

- Versión `sklearn`

$$tfidf(k,d) = \frac{tfidf(k,d)}{\sum_{k \in T(D)} tfidf(k,d)^2}$$

Donde:

$T(D)$ es el conjunto de términos del conjunto de documentos (D).

Sea $tfidf(k \in T, d) = (tfidf(k, d))_{k \in T}$ el vector que contiene como componentes los valores de tf-idf para los terminos $k \in T$ en el documento d , entonces:

$||tfidf(k \in T, d)||_2 = \sqrt{\sum_{k \in T(D)} tfidf(k, d)^2}$, es decir, es la norma euclidea del vector $tfidf(k \in T, d)$

Notese que si $k \notin d$, entonces $tfidf(k, d) = 0$, por tanto , $\sum_{k \in T} tfidf(k, d)^2 = \sum_{k \in T(d)} tfidf(k, d)^2$, donde $T(d)$ es el conjunto de los terminos del documento $d \in D$

6.2 Cálculo de tf-idf en Python

6.2.1 Cálculo de tf

```
# nº de veces que aparece cada termino (token) en cada noticia (n_k)

df_tf = pd.DataFrame(
↪  Fake_News_Tokens_not_StopWords.groupby(['id_text','token'])['token'].count().reset_in
↪  )

# nº de terminos (tokens) en cada noticia (size(d))

df_tf['size(d)'] = df_tf.groupby('id_text')['n_k'].transform(sum)

# Calculo de term-frequency (tf)

df_tf['tf'] = df_tf['n_k'] / df_tf['size(d)']


# Calculo de term-frequency normalizado (tf_norm)

df_tf['max_tf'] = df_tf.groupby('id_text')['tf'].transform(max)

df_tf['tf_norm'] = df_tf['tf'] / df_tf['max_tf']
```

Veamos como queda el data-frame creado:

```
df_tf
```

	id_text	token	n_k	size(d)	tf	max_tf	tf_norm
0	0	accept	1	251	0.003984	0.059761	0.066667
1	0	alan	1	251	0.003984	0.059761	0.066667
2	0	alansandoval	1	251	0.003984	0.059761	0.066667
3	0	allow	1	251	0.003984	0.059761	0.066667

```
4              0          also    1          251  0.003984  0.059761  0.066667
...          ...           ...  ...          ...       ...       ...       ...
7155407    44897         union    1          132  0.007576  0.030303  0.250000
7155408    44897        volume    1          132  0.007576  0.030303  0.250000
7155409    44897         wants    1          132  0.007576  0.030303  0.250000
7155410    44897         worth    2          132  0.015152  0.030303  0.500000
7155411    44897         years    1          132  0.007576  0.030303  0.250000
```

6.2.2 Cálculo de idf

```
# Calculo del nº de documentos en los que aparece cada termino (token)
↪  (n(D,k))

df_Idf = pd.DataFrame(
↪  Fake_News_Tokens_not_StopWords.groupby(['token'])['id_text'].nunique().reset_index(na
↪  )

# Ojo, si se usa count en lugar de nunique no se estaria contando el nº de
↪  documentos
# en los que aparece cada termino, si no el nº de veces en total (contando
↪  repeticiones) que
# aparece un termino en el conjunto de documentos. Por ejemplo, dado un
↪  termino k que aparece 10 veces en el documento d1 y 3 veces en el d3 ,
↪  usando count() la cuenta sale 13 , que es el  nº de veces que aparece
↪  k en el conjunto de los documentos, en cambio usando nunique()  la
↪  cuenta sale 2 que es el nº de documentos en los que aparece el termino
↪  k, que es lo que buscamos

# Calculo del nº total de documentos (n_D)

df_Idf['n_D'] = len(Fake_News_Data)

# Calculo de Idf

df_Idf['Idf'] = np.log( (df_Idf['n_D'] ) / (df_Idf['n_d_k']) ) + 1
```

Vemos como queda el data-frame creado:

```
df_Idf
```

```
             token  n_d_k    n_d        Idf
0               aa     28  44898   8.379944
1              aaa     11  44898   9.314253
2       aaaaaaaand      1  44898  11.712149
3        aaaaackkk      1  44898  11.712149
4       aaaaapkfhk      1  44898  11.712149
...            ...    ...    ...        ...
125561        ""it      1  44898  11.712149
125562      ""when      1  44898  11.712149
125563          if      1  44898  11.712149
125564    $emoji1$      1  44898  11.712149
125565    $emoji2$      1  44898  11.712149
```

6.2.3 Cálculo de tf-idf

```
df_tf_Idf = pd.merge(df_tf, df_Idf, on='token')

df_tf_Idf['tf_Idf'] = df_tf_Idf['tf'] * df_tf_Idf['Idf']

df_tf_Idf['tf_Idf_norm'] = df_tf_Idf['tf_norm'] * df_tf_Idf['Idf']

df_tf_Idf = df_tf_Idf.sort_values(by="id_text")

#####################################

def euclidean_norm( v ):

  return np.sqrt( (v**2).sum() )

#####################################

df_tf_Idf['euclidean_norm'] =
↪  df_tf_Idf.groupby('id_text')['tf_Idf'].transform( euclidean_norm)

df_tf_Idf['tf_Idf_sklearn'] = df_tf_Idf['tf_Idf'] /
↪  df_tf_Idf['euclidean_norm']
```

Vemos con es el nuevo data-frame creado:

```
df_tf_Idf
```

	id_text	token	n_k	size(d)	tf	max_tf
0	0	accept	1	251	0.003984	0.059761
304262	0	pollitt	1	251	0.003984	0.059761
304263	0	power	1	251	0.003984	0.059761
309320	0	president	3	251	0.011952	0.059761
332891	0	presidential	1	251	0.003984	0.059761
...	...	...	...	...	...	...
5516302	44897	amid	1	132	0.007576	0.030303
5350014	44897	string	1	132	0.007576	0.030303
2043348	44897	state	2	132	0.015152	0.030303
1423490	44897	delivered	1	132	0.007576	0.030303
7155411	44897	suhkoi	1	132	0.007576	0.030303

	tf_norm	n_d_k	n_d	Idf	tf_Idf	tf_Idf_sklearn
0	0.066667	1395	44898	4.471499	0.017815	0.298100
304262	0.066667	1	44898	11.712149	0.046662	0.780810
304263	0.066667	5057	44898	3.183620	0.012684	0.212241
309320	0.200000	23571	44898	1.644376	0.019654	0.328875

```
332891   0.066667   8823  44898   2.627031  0.010466     0.175135

...           ...    ...    ...        ...       ...          ...

5516302  0.250000   1316  44898   4.529796  0.034317     1.132449
5350014  0.250000    404  44898   5.710734  0.043263     1.427683
2043348  0.500000  14226  44898   2.149322  0.032565     1.074661
1423490  0.250000    882  44898   4.929956  0.037348     1.232489
7155411  0.250000      1  44898  11.712149  0.088728     2.928037
```

6.3 Matriz Tf-Idf

Para poder aplicar algoritmos de clasificación a un texto, es necesario crear una representación numérica del mismo. Para ello se utiliza una matriz que tiene como filas los documentos y como columnas los tokens. Existen diferentes criterios para definir los elementos internos de esta matriz. Sea (i, j) el elemento de la fila $i y columna j distinguimos varias aproximaciones. Una es que (i,j) sea la frecuencia del token j en el documento i , es decir, tf(j,i) , otra aproximacion es que (i, j) sea 0 si el token j no aparece en el documento i y 1 en el caso de que si aparece. Otra aprozimacion es que (i,j) sea tfidf(j,i).

El criterio seguido en esta seccion del trabajo es que $(i, j) = tfidf(j, i)$, ya que es el criterio seguido por la libreria `sklearn`, la cual será empleada para calcular la matriz tf-idf. Además es uno de los criterios mas habituales para definir dicha matriz.

Se ha intentado construir esta matriz a traves de bucles, pero dado que es una matriz con 44898 filas (documentos) y 125565 columnas (tokens) , la sola operacion de crear la primera fila no ha podido ser ejecutada por el computador por sobrepasar la memoria necesaria para ello. Por tanto deben usarse métodos de programación mas eficientes, o usar opciones eficientes ya implementadas por desarrolladores profesionales, como el equipo de `sklearn`. Esta segunda opción es la que seguiresmos, es decir, usaremos dicha funciones de dicha libreria para crear la matriz tf-idf.

Para crear la matriz tf-idf con `sklearn` necesitamos construir por un un vector con los documentos (en este caso noticias). Además también vamos a crear otro con la variable respuesta (en este caso la variable binaria `Fake` que indica si las noticias son o no fakes), que será necesario para la parte de clasificación de texto.

```
X_data = Fake_News_Data.loc[ : , 'text']

X_data
```

```
0        Donald Trump just couldn t wish all Americans ...
1        House Intelligence Committee Chairman Devin Nu...
2        On Friday, it was revealed that former Milwauk...
3        On Christmas day, Donald Trump announced that ...
4        Pope Francis used his annual Christmas Day mes...

                              ...

44893    BRUSSELS (Reuters) - NATO allies on Tuesday we...
44894    LONDON (Reuters) - LexisNexis, a provider of l...
44895    MINSK (Reuters) - In the shadow of disused Sov...
44896    MOSCOW (Reuters) - Vatican Secretary of State ...
44897    JAKARTA (Reuters) - Indonesia will buy 11 Sukh...
```

```
Y_data = Fake_News_Data.loc[ : , 'Fake']

Y_data
```

```
0        1
1        1
2        1
3        1
4        1

        ...

44893    0
44894    0
44895    0
44896    0
44897    0
```

Importamos la funcion `TfidfVectorizer` de `sklearn` la cual nos permitirá generar la matriz tf-idf.

Es recomendable ver la documentación de `sklearn` para esta función documentación

En este caso usaremos como funcion de tokenizacion la anteriormente creada `limpiar_tokenizar` , podria usarse la que usa por defecto `sklearn`, además la lista de stopwords que también se ha usado anteriormente. Ademas usamos los argumentos min_df = 0 , lo cual significa que se van a considerar todos los tokens generados por la funcion tokenizer, si min_df=h la función solo consideraria los tokens que aparecen en mas de h documentos. Por ultimo usaremos el argumento smooth_idf=False , el cual ya fue mencionado en la seccion *definición formal del estadistico tf-idf*.

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizador = TfidfVectorizer(tokenizer  = limpiar_tokenizar,
↪  min_df = 0, stop_words = stop_words, smooth_idf=False)
```

Ahora necesitamos usar el método `fit` con el vector de documentos `X_data`

```
tfidf_vectorizador.fit(X_data)
```

Creamos la matriz tf-idf con el metodo `transform`

```
tfidf_matrix = tfidf_vectorizador.transform(X_data)
```

Comprobamos el tamaño de la matriz

```
tfidf_matrix.shape
```

```
(44898, 125566)
```

Podemos obtener los nombres de las columnas de la matriz, a saber, los tokens, con el metodo `get_feature_names_out` , en este caso imprimimos los 50 primeros:

```
print(tfidf_vectorizador.get_feature_names_out()[0:50])
```

```
['aa' 'aaa' 'aaaaaaaand' 'aaaaackkk' 'aaaaapkfhk' 'aaaahhhh' 'aaaand'
 'aaaarrgh' 'aaab' 'aaarf' 'aab' 'aaba' 'aabfsv' 'aabge' 'aabo'
 'aaccording' 'aachen' 'aacnr' 'aadhaar' 'aadhar' 'aadl' 'aaf' 'aafn'
 'aag' 'aahd' 'aahwuhvvnh' 'aai' 'aaj' 'aaja' 'aal' 'aalberg' 'aalberts'
 'aaldef' 'aaliyah' 'aamer' 'aamin' 'aammir' 'aamom' 'aamrrd' 'aan'
 'aaofj' 'aapa' 'aapi' 'aapl' 'aapxim' 'aar' 'aardal' 'aardvark'
 'aardvarks' 'aargh']
```

`sklearn` no permite imprimir la matriz tf-idf obtenida debido a sus dimension excesiva, pero si podemos acceder a sus elementos del siguiente modo:

Por ejemplo, podemos acceder al tf-idf del token asociado a la columna 645 y el documento asociado a la columna 0

```
tfidf_matrix[0, 645]
```

```
0.034889784479772486
```

La salida anterior nos indicaa que el valor del estadistico tf-idf para el token asociado a la columna 645 en el documento (noticia) asociada a la fila 0 (la fila 0 en Python es la fila 1) es 0.0349

Vamos a crear un data-frame para identificar cada columna con su token asociado:

```
df_index_token = pd.DataFrame( {'index' :
↪  range(0,len(tfidf_vectorizador.get_feature_names_out())) , 'token' :
↪  tfidf_vectorizador.get_feature_names_out() })

df_index_token
```

	index	token
0	0	aa
1	1	aaa
2	2	aaaaaaaand
3	3	aaaaackkk
4	4	aaaaapkfhk
...	...	...
125561	125561	""it
125562	125562	""when
125563	125563	if
125564	125564	$emoji1$
125565	125565	$emoji2$

Utilizando este da-taframe vamos a comparar algunos valores de la matriz tf-idf obtenida con `sklearn`con los valores que calculamos nosotros en las secciones anteriores y que se encuentran registrados en el data-frame `df_tf_Idf`

```
df_tf_Idf
```

```
         id_text         token  n_k  longitud(d)        tf    max_tf

0              0        accept    1          251  0.003984  0.059761
304262         0       pollitt    1          251  0.003984  0.059761
304263         0         power    1          251  0.003984  0.059761
309320         0     president    3          251  0.011952  0.059761
332891         0  presidential    1          251  0.003984  0.059761
...          ...           ...  ...          ...       ...       ...
5516302    44897          amid    1          132  0.007576  0.030303
5350014    44897        string    1          132  0.007576  0.030303
2043348    44897         state    2          132  0.015152  0.030303
1423490    44897     delivered    1          132  0.007576  0.030303
7155411    44897        suhkoi    1          132  0.007576  0.030303

          tf_norm  n_d_k    n_d        Idf    tf_Idf  tf_Idf_norm
0        0.066667   1395  44898   4.471499  0.017815     0.298100
304262   0.066667      1  44898  11.712149  0.046662     0.780810
304263   0.066667   5057  44898   3.183620  0.012684     0.212241
309320   0.200000  23571  44898   1.644376  0.019654     0.328875
332891   0.066667   8823  44898   2.627031  0.010466     0.175135
...           ...    ...    ...        ...       ...          ...
5516302  0.250000   1316  44898   4.529796  0.034317     1.132449
5350014  0.250000    404  44898   5.710734  0.043263     1.427683
2043348  0.500000  14226  44898   2.149322  0.032565     1.074661
1423490  0.250000    882  44898   4.929956  0.037348     1.232489
7155411  0.250000      1  44898  11.712149  0.088728     2.928037

         euclidean_norm  tf_Idf_sklearn
0              0.510600        0.034890
304262         0.510600        0.091386
304263         0.510600        0.024841
309320         0.510600        0.038492
332891         0.510600        0.020498
...                 ...             ...
5516302        0.585168        0.058644
5350014        0.585168        0.073933
2043348        0.585168        0.055651
1423490        0.585168        0.063825
7155411        0.585168        0.151629
```

Vamos a comparar concretamente los tf-idf obtenidos “manualmente” y con `sklearn` para los tokens ‘accept’ , ‘pollit’ , ‘string’, ‘never’, e ‘investigation’ :

Primero tenemos que identificar la columna asociada al token ‘accept’

```
df_index_token.loc[df_index_token.token == 'accept', ]
```

```
 index   token
```

645 645 accept

Ahora vemos cual es el valor del estadistico tf-idf calculado por `sklearn` para el token 'accept' en el documento asociado a la fila 0.

```
tfidf_matrix[0, 645]
```

0.034889784479772486

Ahora comprobamos el valor del estadistico tf-idf calculado "manualmente" por nosotros para el token 'accept' y la noticia con identificador 0 (notese que la noticia con identificador i es la que esta asociada a la fila i de la matriz creada por `sklearn` , para $i = 0, 1, 2, ...$)

```
df_tf_Idf.loc[ ( df_tf_Idf.id_text == 0 ) &  ( df_tf_Idf.token == 'accept'
↪  ) , ]
```

```
   id_text   token  n_k  longitud(d)        tf    max_tf   tf_norm  n_d_k
0        0  accept    1          251  0.003984  0.059761  0.066667   1395

     n_d       Idf    tf_Idf  tf_Idf_norm  euclidean_norm  tf_Idf_sklearn
0  44898  4.471499  0.017815       0.2981          0.5106         0.03489
```

Podemos ver que de todos los estadisticos tf-idf calculados (el simple, el normalizado y la versión de sklearn) el único que coincide con el obtenido al usar sklearn es justanmente `tf_Idf_sklearn` , como cabria esperar.

En este punto hay que hacer una mención especial a la entrada de analyticsvidhya , la cual me permitio resolver una disparidad de resultados que obtuve inicialmente, al no ser consciente de que `sklearn` normalizaba el tf-idf (la version simple) dividiendolo entre la norma euclidea, tal y como se ha explicado anteriormente con más detalle.

Ahora repetimos el mismo proceso para el token 'pollitt'

```
df_index_token.loc[df_index_token.token == 'pollitt', ]
```

```
       index    token
79741  79741  pollitt
```

```
tfidf_matrix[0, 79741]
```

```
   0.09138643507615184
```

```
df_tf_Idf.loc[ ( df_tf_Idf.id_text == 0 ) &  ( df_tf_Idf.token ==
↪  'pollitt' ) , ]
```

```
        id_text    token  n_k  longitud(d)        tf    max_tf   tf_norm  \
304262        0  pollitt    1          251  0.003984  0.059761  0.066667

        n_d_k    n_d        Idf    tf_Idf  tf_Idf_norm  euclidean_norm  \
304262      1  44898  11.712149  0.046662      0.78081          0.5106

        tf_Idf_sklearn
304262        0.091386
```

```
df_index_token.loc[df_index_token.token == 'string', ]
```

```
   index   token
```

99546 99546 string

```
tfidf_matrix[44897, 99546]
```

0.07393279140214064

```
df_tf_Idf.loc[ ( df_tf_Idf.id_text == 44897 ) &  ( df_tf_Idf.token ==
↪  'string' ) , ]
```

```
     id_text   token  n_k  longitud(d)        tf    max_tf  tf_norm  \
```

5350014 44897 string 1 132 0.007576 0.030303 0.25

```
     n_d_k    n_d       Idf    tf_Idf  tf_Idf_norm  euclidean_norm  \
```

5350014 404 44898 5.710734 0.043263 1.427683 0.585168

```
     tf_Idf_sklearn
```

5350014 0.073933

```
df_index_token.loc[df_index_token.token == 'never', ]
```

```
   index  token
```

70560 70560 never

```
tfidf_matrix[3, 70560]
```

0.022176846230040667

```
df_tf_Idf.loc[ ( df_tf_Idf.id_text == 3 ) &  ( df_tf_Idf.token == 'never'
↪  ) , ]
```

```
    id_text  token  n_k  longitud(d)        tf    max_tf   tf_norm  n_d_k  \
```

990208 3 never 1 249 0.004016 0.044177 0.090909 6077

```
      n_d       Idf    tf_Idf  tf_Idf_norm  euclidean_norm  tf_Idf_sklearn
```

990208 44898 2.999882 0.012048 0.272717 0.543257 0.022177

```
df_index_token.loc[df_index_token.token == 'investigation', ]
```

```
   index          token
```

50314 50314 investigation

```
tfidf_matrix[1522, 50314]
```

0.2598673157066844

```
df_tf_Idf.loc[ ( df_tf_Idf.id_text == 1522 ) &  ( df_tf_Idf.token ==
↪  'investigation' ) , ]
```

```
    id_text          token  n_k  longitud(d)        tf    max_tf  \
```

635701 1522 investigation 7 210 0.033333 0.052381

```
     tf_norm  n_d_k    n_d       Idf    tf_Idf  tf_Idf_norm  \
```

635701 0.636364 3753 44898 3.481838 0.116061 2.215715

```
    euclidean_norm  tf_Idf_sklearn
```

635701 0.446617 0.259867

7 Métodos Naive Bayes

Los métodos de naive Bayes son un conjunto de algoritmos de aprendizaje supervisado basados en aplicar el teorema de Bayes con el supuesto "naive" de independencia condicional entre cada par de predictores dada una clase de la variable respuesta (que debe ser categorica).

Sean $Y, X_1, ..., X_p$ la respuesta categorica y los predictores, y sean $x_i = (x_i1, x_i2, ..., x_ip)^t$ y y_i la i-esima observación de los predictores y de la respuesta, respectivamente.

Si consideramos $Y, X_1, ..., X_p$ como variables aleatorias, por el teorema de Bayes tenemos que :

$$P(Y = y_i|X_1 = x_{i1}, ..., X_p = x_{ip}) = \frac{P(Y = y_i) \cdot P(X_1 = x_{i1}, ..., X_p = x_{ip}|Y = y_i)}{P(X_1 = x_{i1}, ..., X_p = x_{ip})}$$

Usando el supuesto naive de independencia entre cada par de predictores

$$X_r \perp X_j, \forall r \neq j = 1, ..., p$$

tenemos que:

$$P(X_1 = x_{i1}, ..., X_p = x_{ip}|Y = y_i) = \prod_{j=1}^{p} P(X_j = x_{ij}|Y = y_i)$$

Por tanto, podemos reformular el teorema de Bayes como:

$$P(Y = y_i|X_1 = x_1, ..., X_p = x_p) = \frac{P(Y = y_i) \cdot \prod_{j=1}^{p} P(X_j = x_{ij}|Y = y_i)}{P(X_1 = x_{i1}, ..., X_p = x_{ip})}$$

$$P(Y = y_i|X_1 = x_1, ..., X_p = x_p) \propto ¶(Y = y_i) \cdot \prod_{j=1}^{p} P(X_j = x_{ij}|Y = y_i)$$

El algoritmo de naive Bayes predice la respuesta Y para un vector de observaciones de los predictores $x_i = (x_i1, x_i2, ..., x_ip)^t$ como la solucion del siguiente problema de optimización:

$$\underset{Max}{y} P(Y = y|X_1 = x_1, ..., X_p = x_p) = \underset{Max}{y} \frac{P(Y = y_i) \cdot \prod_{j=1}^{p} P(X_j = x_{ij}|Y = y_i)}{P(X_1 = x_{i1}, ..., X_p = x_{ip})} = \underset{Max}{y} P(Y = y) \cdot \prod_{j=1}^{p} P(X_j =$$

Notese que $P(X_1 = x_{i1}, ..., X_p = x_{ip})$ no depende del valor de y por lo que puede sacarse del problema de maximización.

Es decir, la prediccion de Y para el vector de observaciones de los predictores $x_i = (x_i1, x_i2, ..., x_ip)^t$ es

$$\hat{y_i} = arg \underset{Max}{y} P(Y = y|X_1 = x_1, ..., X_p = x_p) = arg \underset{Max}{y} P(Y = y) \cdot \prod_{j=1}^{p} P(X_j = x_{ij}|Y = y)$$

Es decir, la observación i-esima $x_i = (x_i1, x_i2, ..., x_ip)^t$ se clasifica en la clase/categoria/grupo de maxima probabilidad para esa observación.

Problemas

Dados y y x_{ij}

- ¿Cómo estimar P(Y=y) ?

$$\widehat{P}(Y = y) = \frac{\#\{r = 1, ..., n / y_r = y\}}{n}$$

Es decir, P(Y=y) se estima como la proporcion de observaciones (del conjunto de entrenamiento) que pertenecen a la clase/categoria/grupo y , es decir, se estima como la proporción de observaciones para las que la respuesta toma la categoria y

- ¿ Cómo estimar P(X_j=x_{ij} | Y=y) ?

Podria seguirse la solución del problema anterior, a saber:

$$\widehat{P}(X_j = x_{ij}|Y = y) = \frac{\#\{r = 1, ..., n \ y_r = y y x_{rj} = x_{ij}\}}{\#\{r = 1, ..., n \ y_r = y\}}$$

Problema: en la practica en cuanto haya algun predictor X_j tal que el valor observado x_{ij} no este en el set de observaciones de entrenamiento con Y=y se tendrá $\widehat{P}(X_j = x_{ij}|Y = y) = 0$, lo que conducirá a $P(Y = y) \cdot \prod_{j=1}^{p} P(X_j = x_{ij}|Y = y) = 0$, y esto llevará a no clasificar la observacion $x_i = (x_i1, x_i2, ..., x_ip)^t$ en la clase y , independientemente de los valores observados para el resto de predictores. Por lo que es razonable pensar que esta aproximacion no conduciria a buenas predicciones de la respuesta.

Ademas si nos enfocamos en un problema de clasificación de texto, en el que se usa la matriz tf-idf como matriz de predictores, esta aproximacion queda en clara evidencia, ya que para cualquier categoria y habria algunas palabras (tokens) que tienen para cada documento un valor del estadistico tf-idf diferente al valor correspondiente de la observacion $x_i = (x_i1, x_i2, ..., x_ip)^t$, es decir, para cada y habria algun predictor X_j tal que $\widehat{P}(X_j = x_{ij}|Y = y) = 0$, por lo que la observacion $x_i = (x_i1, x_i2, ..., x_ip)^t$ seria clasificada indistintamente en cualquier categoria de la respuesta, lo cual no es razonable en absoluto.

Si en lugar de una matriz tf-idf se usase un matriz con el conteo de ocurrencia de cada palabra en los textos la situacion seria similar, puesto que para cada categoria y de la respuesta habria algunas palabras que aparecen 0 veces en los textos de dicha categoria (es decir, habria algunos predictores X_j tales que $\widehat{P}(X_j = x_{ij}|Y = y) = 0$), y esto llevaria a la misma conclusion que antes.

La solución estandar a este problema pasa por estimar $\widehat{P}(X_j = x_{ij}|Y = y)$ usando la funcion de probabilidad/densidad de una distribución conocida. En este trabajo distinguiremos dos casos (los estandar), uno en el que se usa la distribucion normal Gaussiana (Gaussian Naive Bayes) y otro en el que se usa la distribución multinomial (multinomial naive Bayes).

7.1 Gaussian Naive Bayes

Es un naive Bayes classifier en el que se considera el siguiente supuesto:

$$\widehat{P}(X_j = x_{ij}|Y = y) = \frac{1}{\sqrt{2\pi\sigma^2(x_{ij}|y)}} \cdot exp\{-\frac{(x_{ij} - \mu(x_{ij}|y))^2}{2\sigma^2(x_{ij}|y)}\}$$

Donde:

$\mu(x_{ij}|y) = Mean(x_{rj}/r = 1, .., n; y_r = y)$, es decir, es la media de los valores de X_j asociados a la clase y de la variable respuesta Y

$\sigma^2(x_{ij}|y) = Var(x_{rj}/r = 1, .., n; y_r = y)$, es decir, es la varianza de los valores de X_j asociados a la clase y de la variable respuesta Y

7.2 Multinomial Naive Bayes

7.3 Gaussian Naive Bayes aplicado con `Python`

8 Bibliografía

https://scikit-learn.org/stable/modules/naive_bayes.html

https://en.wikipedia.org/wiki/Naive_Bayes_classifier

https://www.cienciadedatos.net/documentos/py25-text-mining-python.html

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

https://www.analyticsvidhya.com/blog/2021/11/how-sklearns-tfidfvectorizer-calculates-tf-idf-values/

https://github.com/scikit-learn/scikit-learn/blob/f3f51f9b6/sklearn/feature_extraction/text.py#L1717

https://es.wikipedia.org/wiki/Tf-idf

web sklearn, numpy , pandas , wikipedia