In [1]:
```
import os
os.chdir("/Users/hromerol/Desktop/TFM")
print(os.getcwd())
```
```
/Users/hromerol/Desktop/TFM
```

In [2]:
```
# Importando bibliotecas de trabajo
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Lectura del dataset

In [3]:
```
# Importando el dataset
df = pd.read_json("act_grupal_News_Category_Dataset_v2.json", lines=True)
print (df.shape)
df.head()
```
```
(200853, 6)
```

Out[3]:

	category	headline	authors	link	short_description	date
0	CRIME	There Were 2 Mass Shootings In Texas Last Week,...	Melissa Jeltsen	https://www.huffingtonpost.com/entry/texas-ama...	She left her husband. He killed their children...	2018-05-26
1	ENTERTAINMENT	Will Smith Joins Diplo And Nicky Jam For The 2...	Andy McDonald	https://www.huffingtonpost.com/entry/will-smit...	Of course it has a song.	2018-05-26
2	ENTERTAINMENT	Hugh Grant Marries For The First Time At Age 57	Ron Dicker	https://www.huffingtonpost.com/entry/hugh-gran...	The actor and his longtime girlfriend Anna Ebe...	2018-05-26
3	ENTERTAINMENT	Jim Carrey Blasts 'Castrato' Adam Schiff And D...	Ron Dicker	https://www.huffingtonpost.com/entry/jim-carre...	The actor gives Dems an ass-kicking for not fi...	2018-05-26
4	ENTERTAINMENT	Julianna Margulies Uses Donald Trump Poop Bags...	Ron Dicker	https://www.huffingtonpost.com/entry/julianna-...	The "Dietland" actress said using the bags is ...	2018-05-26

Ahora que ya hemos importado correctamente el archivo y lo hemos convertido de formato JSON a dataframe, es necesario solo quedarnos con las siguientes categorías:

1. WELLNESS
2. PARENTING
3. TRAVEL
4. BUSINESS
5. SPORTS

In [4]:
```
# Creando copia del dataset original
data = df.copy()
```

In [5]:
```
# Creando un subset unicamente con las variables indicadas
data = data[(data.category == 'WELLNESS') |
            (data.category == 'PARENTING') |
            (data.category == 'TRAVEL') |
            (data.category == 'BUSINESS') |
            (data.category == 'SPORTS')]
```

In [6]:
```
# Visualizando si el nuevo subset se hizo de forma correcta
data["category"].value_counts()
```

Out[6]:
```
WELLNESS     17827
TRAVEL        9887
PARENTING     8677
BUSINESS      5937
SPORTS        4884
Name: category, dtype: int64
```

Efectivamente podemos observar como nuestro dataset ahora solo cuenta con los factores indicados en la variable "category".

Ahora bien, en esta práctica lo que se quiere es crear un modelo de Naive-Bayes que consiga clasificar artículos atendiendo a sus titulares, es por ello que vamos a proceder a eliminar las otras columnas ya que no van a ser utilizadas en el modelo.

In [7]:
```
# Creando subset solo con la variable "category" y "headline"
data = data[['category', 'headline']]
```

In [8]:
```
# Visualizando resultado
data.head()
```

Out[8]:

	category	headline
80	SPORTS	Jets Chairman Christopher Johnson Won't Fine P...
87	BUSINESS	U.S. Launches Auto Import Probe, China Vows To...
101	SPORTS	Trump Posthumously Pardons Boxer Jack Johnson
126	TRAVEL	14 Ways To Make Family Road Trips Easier, From...
135	SPORTS	Anna Kournikova Dancing With Her Bouncing Baby...

Podemos observar como efectivamente pudimos hacer el preprocesamiento requerido y ahora solo tenemos la variable a predecir junto con la variable explicativa.

In [9]:
```
# Evaluando balanceo de clases
round((data["category"].value_counts()/len(data))*100, 2)
```

Out[9]:
```
WELLNESS     37.76
TRAVEL       20.94
PARENTING    18.38
BUSINESS     12.58
SPORTS       10.34
Name: category, dtype: float64
```

Podemos observar como las clases se encuentran desbalanceadas, ya que, el factor que tiene la mayor cantidad de registros es el de "Wellness" con un 37.76%, seguido de "travel" con un 20.94% y los otros dos tienen menos de 20%.

Lo ideal sería poder tener un 20% de cada una de las clases para que el modelo pueda tener la misma cantidad de registros, sin embargo vamos a ver cómo se comporta el modelo.

In [10]:
```
# Visualizando valores
data["headline"].values
```

Out[10]:
```
array(["Jets Chairman Christopher Johnson Won't Fine Players For Anthem Protests",
       'U.S. Launches Auto Import Probe, China Vows To Defend Its Interests',
       'Trump Posthumously Pardons Boxer Jack Johnson', ...,
       'Giants Over Patriots, Jets Over Colts Among  Most Improbable Super Bowl Upsets Of All Time (VIDEOS)',
       'Aldon Smith Arrested: 49ers Linebacker Busted For DUI',
       'Dwight Howard Rips Teammates After Magic Loss To Hornets'],
      dtype=object)
```

Aplicando TF-IDF-Vectorizer

In [11]:
```
# Ahora vamos a descargar el vectorizador para los textos
import nltk
nltk.download('stopwords')
```
```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/hromerol/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[11]: True

In [12]:
```
# Importando todas aquellas palabras que son monosilabos o que no anaden informacion al modelo
from nltk.corpus import stopwords
len(stopwords.words('english'))
```

Out[12]: 179

In [13]:
```
# Importando la biblioteca especifica para vectorizar
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words=stopwords.words('english'))
vect = vectorizer.fit_transform(data["headline"].values)
```

In [14]:
```
# Visualizando resultados
vect.todense()
```

Out[14]:
```
matrix([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]])
```

In [15]:
```
# Aplicando vectorizador a todos los textos
vocab = np.sort(list(vectorizer.vocabulary_.keys()))
df_tf = pd.DataFrame(vect.todense(), columns = vocab)
print(df_tf.shape)
df_tf.head()
```
```
(47212, 26770)
```

Out[15]:

	00	000	000th	007	01	012	013	014	02	080	...	zyberk	zylowska	zynga	zyola	zytiga	zzs	zzz	zzzs	élysées	étretat
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 26770 columns

In [16]:
```
# Visualizando toda la lista de filas
df_tf.max(axis=1)
```

Out[16]:
```
0        0.421677
1        0.403366
2        0.493686
3        0.439682
4        0.476555
           ...   
47207    0.492960
47208    0.411799
47209    0.396632
47210    0.423291
47211    0.434877
Length: 47212, dtype: float64
```

In [17]:
```
# Extrayendo representacion por fila
df_tf.idxmax(axis=1)
```

Out[17]:
```
0         christopher
1              import
2        posthumously
3              easier
4          kournikova
             ...     
47207      investment
47208        azarenka
47209      improbable
47210           aldon
47211         hornets
Length: 47212, dtype: object
```

Creando split en train/test de los datos

In [18]:
```
from sklearn.model_selection import train_test_split

X = df_tf.values
y = data["category"].values

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=42, test_size=0.2)
print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)
```
```
(37769, 26770) (37769,)
(9443, 26770) (9443,)
```

Creación del Modelo de Naibe Bayes Multinomial NB

Seguidamente vamos a proceder a realizar el modelo, para ello vamos a apoyarnos en la biblioteca GridSearch, la que nos va a ayudar a determinar cuáles son los mejores parámetros a utilizar en el hipertuneo, así podremos garantizar que nuestro algoritmo utilizará los mejores parámetros posibles.

Al ser un proceso computacional costoso, vamos a hacerlo con solo aquellos parámetros que son más populares y no utilizaremos todos de forma exhaustiva, ya que esto puede llevar a que se dure bastante determinando los mejores parámetros, por ello vamos a utilizar solamente:

1. Alpha
2. Fit_prior

In [19]:
```
# Importando biblioteca
from sklearn.naive_bayes import MultinomialNB
```

In [20]:
```
# Creando el modelo Naibe Bayes MultinomialNB para utilizar con GridSearch
multi= MultinomialNB()
```

In [21]:
```
# Creando estimadores de parametros
param_grid = {
    'alpha':[1.0, 3.0, 5.0, 7.0, 10.0],
    'fit_prior':[True, False]
}
```

In [22]:
```
# Realizando Cross Validation con 3 y probando parametros
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
CV_1 = GridSearchCV(estimator = multi, param_grid=param_grid, cv= 3)
CV_1.fit(X_train, y_train)
```

Out[22]:
```
GridSearchCV(cv=3, estimator=MultinomialNB(),
             param_grid={'alpha': [1.0, 3.0, 5.0, 7.0, 10.0],
                         'fit_prior': [True, False]})
```

In [23]:
```
# Comprobando cuales son los mejores parametros a utilizar
CV_1.best_params_
```

Out[23]:
```
{'alpha': 1.0, 'fit_prior': False}
```

Aquí podemos observar que el GridSearch recomienda usar un alpha = 1 y un fit_prior = False, por ende vamos a utilizar dichos parámetros.

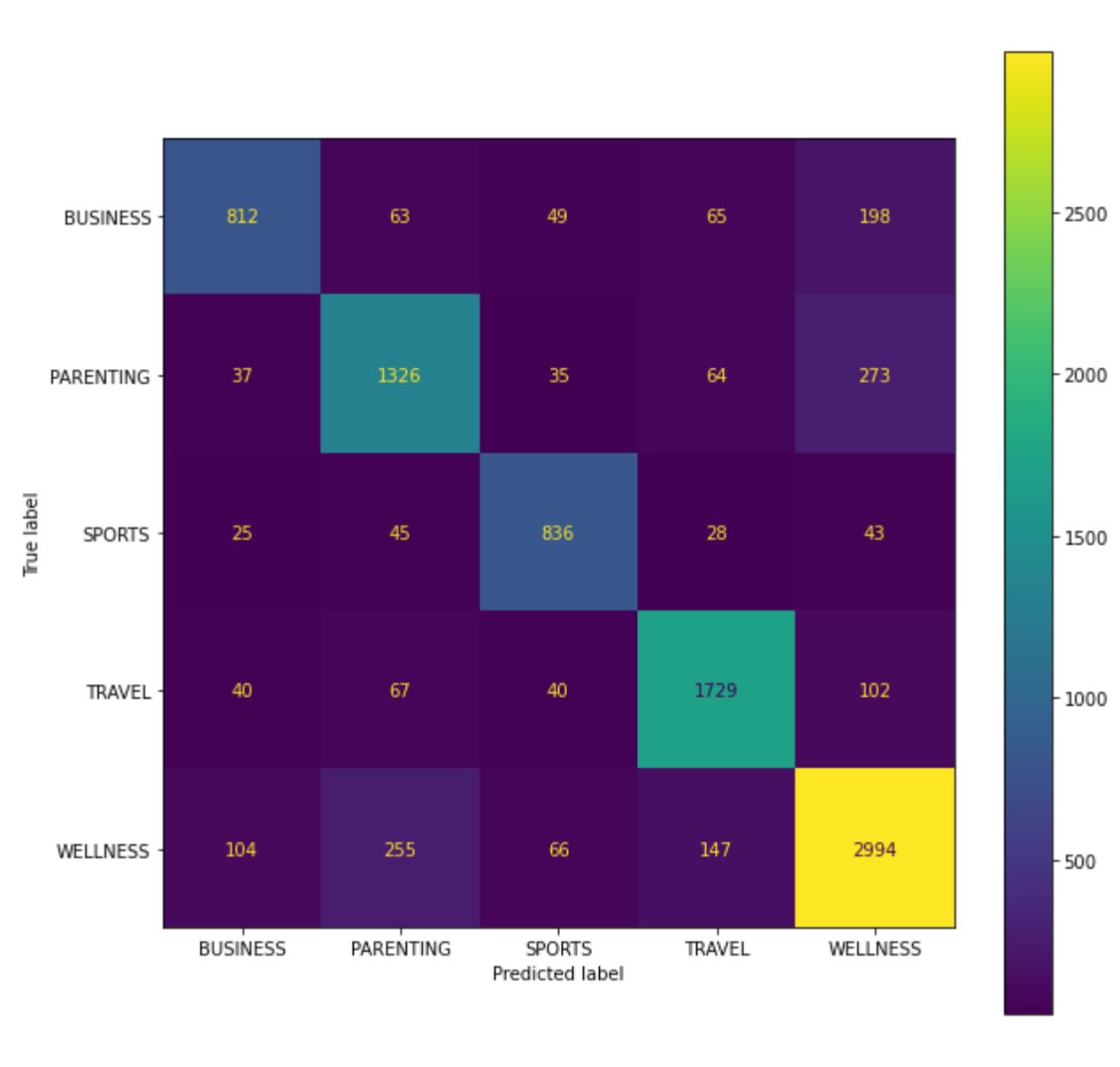
In [24]:
```
# Construyendo modelo con parametros seleccionados
mnb_model = MultinomialNB(alpha=1.0, fit_prior = False)
mnb_model.fit(X_train, y_train)
pred_train = mnb_model.predict(X_train)
pred_test = mnb_model.predict(X_test)
print("Precisión sobre los datos de entrenamiento: {:.2f}".format(100.0*mnb_model.score(X_train, y_trai
n)))
print("Precisión sobre los datos de test: {:.2f}".format(100.0*mnb_model.score(X_test, y_test)))
```
```
Precisión sobre los datos de entrenamiento: 88.69
Precisión sobre los datos de test: 81.51
```

Aquí podemos observar que el modelo sobre los datos de entrenamiento da un resultado de un 88.69% mientras que para los de test de un 81.51% sin embargo es necesario calcular el recall, precision y el F1 score para ver si el modelo puede producir resultados decentes.

In [25]:
```
# Creando visualizacion de la matriz de confusion
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
from sklearn import metrics
print(f"Confusion matrix:\n")
fig, ax = plt.subplots(figsize=(10, 10))
metrics.plot_confusion_matrix(mnb_model, X_test, y_test, cmap=plt.cm.viridis, ax=ax)
plt.show()

print("\n")

print(f"Confusion matrix:\n{confusion_matrix(y_test, pred_test)}")
```
```
Confusion matrix:
```




```
Confusion matrix:
[[ 812   63   49   65  198]
 [  37 1326   35   64  273]
 [  25   45  836   28   43]
 [  40   67   40 1729  102]
 [ 104  255   66  147 2994]]
```

Análisis: podemos observar de forma rápida que en la categoría travel es en donde suceden la mayor cantidad de aciertos, seguido de sports y wellness, mientras que las predicciones más pobres son realizadas por business.

In [26]:
```
# Imprimiendo resultados para ver el accuracy, precision y recall
print(
    f"Classification report for classifier {mnb_model}:\n"
    f"{metrics.classification_report(y_test, pred_test)}\n"
)
```
```
Classification report for classifier MultinomialNB(fit_prior=False):
              precision    recall  f1-score   support

    BUSINESS       0.80      0.68      0.74      1187
   PARENTING       0.76      0.76      0.76      1735
      SPORTS       0.81      0.86      0.83       977
      TRAVEL       0.85      0.87      0.86      1978
    WELLNESS       0.83      0.84      0.83      3566

    accuracy                           0.82      9443
   macro avg       0.81      0.80      0.81      9443
weighted avg       0.81      0.82      0.81      9443
```

Análisis: gracias al resumen que se nos muestra podemos ver que la predicción a nivel de precisión, que recordemos que nos responde la siguiente pregunta: ¿qué porcentaje de las categorías que identifiquemos estarán realmente correcto? (es decir, mide la calidad del modelo). En este caso podemos ver que la categoría que tiene el score más alto es travel, con un 85% de acierto, es decir, el 85% de las veces el algoritmo será capaz de idenficar la categroía travel, por ende se equivocará en un 15% de las veces, mientras que la más baja es la de parenting con un 76%, es decir, para el algoritmo es tarea fácil identificar la categoría travel, pero le cuesta un poco más predecir parenting (tiende a cometer más error tipo I).

A nivel del recall (esta nos informa sobre la cantidad que el modelo es capaz de identificar) que recordemos que este responde a la pregunta: ¿qué porcentaje de las categorías correctas somos capaces de identificar? Podemos ver que la categoría Travel con un 87%, es decir, se equicará con travel un 13% de las veces, mientras que la más baja es business con un 68%, es decir la categoría Business conlleva una mayor cantidad de FN que el algoritmo predice incorrectamente (error tipo II).

Finalmente podemos ver como el F1 que, recordemos que hace más fácil el poder comparar el rendimiento combinado de la precisión y la exhaustividad entre varias soluciones, obtenemos que el que tiene mayores problemas idenficando es la categoría business 74%, mientras que el que idenfica de forma bastante aceptable es travel.

Con esta evaluación podemos observar que en realidad nuestro modelo está dando resultados bastante decentes, excepto por la categoría business que es a la que más le cuesta identificar.

En caso de que tuviesemos un dataset con solo dos clases podríamos calcular la Curva ROC y la curva de Precision y Recall, pero al ser más de dos categorías se debe analizar con las métricas antes presentadas.

Creando segundo modelo con datos scrappeados de Huffpost y más actuales

In [27]:
```
# Importando bibliotecas necesarias
import requests
import json
import re
from time import sleep
from newspaper import Article
from urllib.request import Request, urlopen
from bs4 import BeautifulSoup as soup

# Guardando codigo HTML por cada pagina
store_pages=[]

for page_number in range(1,5):
    base_url_huff='https://www.huffpost.com/life/healthy-living'
    r_huff = Request(base_url_huff, headers={'User-Agent': 'Mozilla/5.0'})
    webpage = urlopen(r_huff).read()
    page_soup=soup(webpage,"html.parser")
    store_pages.append(page_soup)
    sleep(3)
```

In [28]:
```
# Creando lista vacia para guardar links
links_huff = []
i=0
for i in range(0, len(store_pages)):
    for link in store_pages[i].findAll("a", {"class": "card__headline card__headline--long"}):
        links_huff.append(link.get("href"))
```

In [29]:
```
from newspaper.article import ArticleException, ArticleDownloadState

# Creando lista para alojar datos
article_info_huff=[]
df_huff=[]

for i in range(0,len(links_huff)):
    article_huff = Article(links_huff[i])

    slept = 0
    article_huff.download()
    while article_huff.download_state == ArticleDownloadState.NOT_STARTED:
        if slept > 9:
            raise ArticleException('Download never started')
    sleep(1)
    slept += 1

    article_huff.parse()

    article_info_huff={'Titulo' : article_huff.title,
                       'Categoria' : 'Wellness'}

    df_huff.append(article_info_huff)
    Wellness = pd.DataFrame(df_huff)
```

In [30]:
```
# Visualizando datos de wellness
Wellness.head()
```

Out[30]:

	Titulo	Categoria
0	10 Common Behaviors That Are Making You More F...	Wellness
1	Why Am I Having Nightmares? Here Are A Few Sne...	Wellness
2	If You're Going To Clean Out Your Ears Yoursel...	Wellness
3	When Will It Be Time For Another COVID Booster...	Wellness
4	Without Roe V. Wade, Pregnant Women May Face A...	Wellness

In [31]:
```
store_pages=[]

for page_number in range(1,5):
    base_url_huff='https://www.huffpost.com/life/parents'
    r_huff = Request(base_url_huff, headers={'User-Agent': 'Mozilla/5.0'})
    webpage = urlopen(r_huff).read()
    page_soup=soup(webpage,"html.parser")
    store_pages.append(page_soup)
    sleep(3)
```

In [32]:
```
links_huff = []
i=0
for i in range(0, len(store_pages)):
    for link in store_pages[i].findAll("a", {"class": "card__headline card__headline--long"}):
        links_huff.append(link.get("href"))
```

In [33]:
```
from newspaper.article import ArticleException, ArticleDownloadState

article_info_huff=[]
df_huff=[]

for i in range(0,len(links_huff)):
    article_huff = Article(links_huff[i])

    slept = 0
    article_huff.download()
    while article_huff.download_state == ArticleDownloadState.NOT_STARTED:
        if slept > 9:
            raise ArticleException('Download never started')
    sleep(1)
    slept += 1

    article_huff.parse()

    article_info_huff={'Titulo' : article_huff.title,
                       'Categoria' : 'Parenting'}

    df_huff.append(article_info_huff)
    Parenting = pd.DataFrame(df_huff)
```

In [34]:
```
# Visualizando datos de Parenting
Parenting.head()
```

Out[34]:

	Titulo	Categoria
0	Kim Kardashian Reveals When She Decided To Int...	Parenting
1	Harry Styles Helps Concert Attendee Come Out A...	Parenting
2	More Than 200 Abortion Clinics Will Close If T...	Parenting
3	10 Common Behaviors That Are Making You More F...	Parenting
4	I Carried And Delivered Our Baby. So Why Did G...	Parenting

In [35]:
```
store_pages=[]

for page_number in range(1,5):
    base_url_huff='https://www.huffpost.com/life/travel'
    r_huff = Request(base_url_huff, headers={'User-Agent': 'Mozilla/5.0'})
    webpage = urlopen(r_huff).read()
    page_soup=soup(webpage,"html.parser")
    store_pages.append(page_soup)
    sleep(3)
```

In [36]:
```
links_huff = []
i=0
for i in range(0, len(store_pages)):
    for link in store_pages[i].findAll("a", {"class": "card__headline card__headline--long"}):
        links_huff.append(link.get("href"))
```

In [37]:
```
from newspaper.article import ArticleException, ArticleDownloadState

article_info_huff=[]
df_huff=[]

for i in range(0,len(links_huff)):
    article_huff = Article(links_huff[i])

    slept = 0
    article_huff.download()
    while article_huff.download_state == ArticleDownloadState.NOT_STARTED:
        if slept > 9:
            raise ArticleException('Download never started')
    sleep(1)
    slept += 1

    article_huff.parse()

    article_info_huff={'Titulo' : article_huff.title,
                       'Categoria' : 'Travel'}

    df_huff.append(article_info_huff)
    Travel = pd.DataFrame(df_huff)
```

In [38]:
```
# Visualizando datos de travel
Travel.head()
```

Out[38]:

	Titulo	Categoria
0	How To Take The Ultimate Vacation In Glasgow, ...	Travel
1	Are Airport Lounges Actually Worth It?	Travel
2	How To Take The Ultimate Vacation In Savannah,...	Travel
3	CLEAR, Global Entry And TSA PreCheck: What You...	Travel
4	24 Useful Travel Products That Won't Take Up A...	Travel

In [39]:
```
store_pages=[]

for page_number in range(1,5):
    base_url_huff='https://www.huffpost.com/impact/business'
    r_huff = Request(base_url_huff, headers={'User-Agent': 'Mozilla/5.0'})
    webpage = urlopen(r_huff).read()
    page_soup=soup(webpage,"html.parser")
    store_pages.append(page_soup)
    sleep(3)
```

In [40]:
```
links_huff = []
i=0
for i in range(0, len(store_pages)):
    for link in store_pages[i].findAll("a", {"class": "card__headline card__headline--long"}):
        links_huff.append(link.get("href"))
```

In [41]:
```
from newspaper.article import ArticleException, ArticleDownloadState

article_info_huff=[]
df_huff=[]

for i in range(0,len(links_huff)):
    article_huff = Article(links_huff[i])

    slept = 0
    article_huff.download()
    while article_huff.download_state == ArticleDownloadState.NOT_STARTED:
        if slept > 9:
            raise ArticleException('Download never started')
    sleep(1)
    slept += 1

    article_huff.parse()

    article_info_huff={'Titulo' : article_huff.title,
                       'Categoria' : 'Business'}

    df_huff.append(article_info_huff)
    Business = pd.DataFrame(df_huff)
```

In [42]:
```
# Visualizando datos de business
Business.head()
```

Out[42]:

	Titulo	Categoria
0	U.S. Pools Close, Go Without Lifeguards Amid L...	Business
1	Maryland Apple Workers Face Hurdles After Vote...	Business
2	Apple Store Workers In Maryland Become First T...	Business
3	Elon Musk Offers Thoughts On Finances, Space A...	Business
4	Senator Ron Johnson Caught Faking Phone Call T...	Business

In [43]:
```
store_pages=[]

for page_number in range(1,5):
    base_url_huff='https://www.huffpost.com/section/sports'
    r_huff = Request(base_url_huff, headers={'User-Agent': 'Mozilla/5.0'})
    webpage = urlopen(r_huff).read()
    page_soup=soup(webpage,"html.parser")
    store_pages.append(page_soup)
    sleep(3)
```

In [44]:
```
links_huff = []
i=0
for i in range(0, len(store_pages)):
    for link in store_pages[i].findAll("a", {"class": "card__headline card__headline--long"}):
        links_huff.append(link.get("href"))
```

In [45]:
```
from newspaper.article import ArticleException, ArticleDownloadState

article_info_huff=[]
df_huff=[]

for i in range(0,len(links_huff)):
    article_huff = Article(links_huff[i])

    slept = 0
    article_huff.download()
    while article_huff.download_state == ArticleDownloadState.NOT_STARTED:
        if slept > 9:
            raise ArticleException('Download never started')
    sleep(1)
    slept += 1

    article_huff.parse()

    article_info_huff={'Titulo' : article_huff.title,
                       'Categoria' : 'Sports'}

    df_huff.append(article_info_huff)
    Sports = pd.DataFrame(df_huff)
```

In [46]:
```
# Visualizando datos de Sports
Sports.head()
```

Out[46]:

	Titulo	Categoria
0	NFL Seeks Arbitration For Flores' Racial Discr...	Sports
1	Jaylon Ferguson, Baltimore Ravens Linebacker, ...	Sports
2	Longtime MLB Pitcher Kyle Farnsworth Is Now A ...	Sports
3	Senator Ron Johnson Caught Faking Phone Call T...	Sports
4	Gloria Estefan Shades Jennifer Lopez Over Supe...	Sports

In [47]:
```
# Concatenando dataset con todas las posibles categorias
ds = pd.concat([Wellness, Travel, Parenting, Business, Sports], ignore_index = True)
```

In [48]:
```
# Aleatorizando base de datos
df_1 = ds.sample(frac = 1)
```

In [49]:
```
# Mostrando la nueva base
df_1.head(10)
```

Out[49]:

	Titulo	Categoria
334	Andrew Gillum, Former Candidate For Florida Go...	Sports
357	Longtime MLB Pitcher Kyle Farnsworth Is Now A ...	Sports
78	5 Common Morning Habits That Actually Ruin You...	Wellness
143	How To Take The Ultimate Vacation In Philadelp...	Travel
304	Apple Store Workers In Maryland Become First T...	Business
127	Shoe Bags: The Packing Essential Not Enough Pe...	Travel
200	Kim Kardashian Reveals When She Decided To Int...	Parenting
43	Sleep Tips For When The News Cycle Is Bad And ...	Wellness
176	If Airplane Seat Belts Don't Fit You, Here's H...	Travel
291	Elon Musk Offers Thoughts On Finances, Space A...	Business

In [50]:
```
# Visualizando si el nuevo subset se hizo de forma correcta
df_1["Categoria"].value_counts()
```

Out[50]:
```
Wellness     112
Travel        88
Parenting     60
Business      56
Sports        52
Name: Categoria, dtype: int64
```

Para este ejercicio se solicitaba recuperar no menos de 20 datos de cada categoría y como podemos ver el más bajo corresponde a Sports.

También se solicitaba solamente extraer el titular de las noticias, por lo que, el presente dataset cumple con lo que se solicita.

In [51]:
```
# Evaluando balanceo de clases
round((df_1["Categoria"].value_counts()/len(df_1))*100, 2)
```

Out[51]:
```
Wellness     30.43
Travel       23.91
Parenting    16.30
Business     15.22
Sports       14.13
Name: Categoria, dtype: float64
```

Se puede observar que la clase mayoritaria es la de Wellness y las que tienen la menor cantidad de observaciones es Sports.

Aplicando TF-IDF-Vectorizer

In [79]:
```
# Ahora vamos a aplicar el vectorizador a los textos
import nltk
nltk.download('stopwords')
```
```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/hromerol/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[79]: True

In [80]:
```
# Importando todas aquellas palabras que son monosilabos o que no anaden informacion al modelo
from nltk.corpus import stopwords
len(stopwords.words('english'))
```

Out[80]: 179

In [81]:
```
# Importando la biblioteca especifica para vectorizar
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words=stopwords.words('english'))
vect = vectorizer.fit_transform(df_1["Titulo"].values)
```

In [82]:
```
# Visualizando resultados
vect.todense()
```

Out[82]:
```
matrix([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]])
```

In [83]:
```
# Aplicando vectorizador a todos los textos
vocab = np.sort(list(vectorizer.vocabulary_.keys()))
df_tf_1 = pd.DataFrame(vect.todense(), columns = vocab)
print(df_tf_1.shape)
df_tf_1.head()
```
```
(368, 455)
```

Out[83]:

	10	11	19	20	200	24	25	26	32	40	...	without	women	wondering	work	workers	working	world	worth	year	york
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.347172	0.0	0.0	0.0	0.0	0.0

5 rows × 455 columns

In [84]:
```
# Visualizando toda la lista de filas
df_tf_1.max(axis=1)
```

Out[84]:
```
0      0.339029
1      0.361032
2      0.465824
3      0.561282
4      0.423504
         ...   
363    0.419886
364    0.335259
365    0.502359
366    0.321179
367    0.577350
Length: 368, dtype: float64
```

```
In [85]: # Extrayendo representacion por fila
df_tf_1.idxmax(axis=1)
```

```
Out[85]: 0             andrew
1        bodybuilder
2             habits
3       pennsylvania
4             become
            ...
363            clear
364           arrest
365          capitol
366           aliens
367           causes
Length: 368, dtype: object
```

Creando split en train/test de los datos

```
In [86]: from sklearn.model_selection import train_test_split

X = df_tf_1.values
y = df_1["Categoria"].values

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=42, test_size=0.3)
print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)
```

```
(257, 455) (257,)
(111, 455) (111,)
```

Creación del Modelo de Naibe Bayes Multinomial NB

Nuevamente vamos a proceder a realizar el modelo, para ello vamos a apoyarnos en la biblioteca GridSearch, la que nos va a ayudar a determinar cuáles son los mejores parámetros a utilizar en el hipertuneo, así podremos garantizar que nuestro algoritmo utilizará los mejores parámetros posibles.

Al ser un proceso computacional costoso, vamos a hacerlo con solo aquellos parámetros que son más populares y no utilizaremos todos de forma exhaustiva, ya que esto puede llevar a que se dure bastante determinando los mejores parámetros, por ello vamos a utilizar solamente:

1. Alpha
2. Fit_prior

```
In [95]: # Importando biblioteca
from sklearn.naive_bayes import MultinomialNB
```

```
In [96]: # Creando el modelo Naibe Bayes MultinomialNB para utilizar con GridSearch
Nai = MultinomialNB()
```

```
In [97]: # Creando estimadores de parametros
param_grid = {
    'alpha':[1.0, 3.0, 5.0, 7.0, 10.0],
    'fit_prior':[True, False]
}
```

```
In [98]: # Realizando Cross Validation con 5 y probando parametros
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
CV_1 = GridSearchCV(estimator = Nai, param_grid=param_grid, cv= 5)
CV_1.fit(X_train, y_train)
```

```
Out[98]: GridSearchCV(cv=5, estimator=MultinomialNB(),
             param_grid={'alpha': [1.0, 3.0, 5.0, 7.0, 10.0],
                         'fit_prior': [True, False]})
```

```
In [99]: # Comprobando cuales son los mejores parametros a utilizar
CV_1.best_params_
```

```
Out[99]: {'alpha': 1.0, 'fit_prior': False}
```

Aquí podemos observar que el GridSearch recomienda usar un alpha = 1 y un fit_prior = False, por ende vamos a utilizar dichos parámetros.

```
In [103]: # Construyendo modelo con parametros seleccionados
mnb_model_1 = MultinomialNB(alpha=1.0, fit_prior = False)
mnb_model_1.fit(X_train, y_train)
pred_train = mnb_model_1.predict(X_train)
pred_test = mnb_model_1.predict(X_test)
print("Precisión sobre los datos de entrenamiento: {:.2f}".format(100.0*mnb_model_1.score(X_train, y_tr
ain)))
print("Precisión sobre los datos de test: {:.2f}".format(100.0*mnb_model_1.score(X_test, y_test)))
```

```
Precisión sobre los datos de entrenamiento: 93.00
Precisión sobre los datos de test: 87.39
```

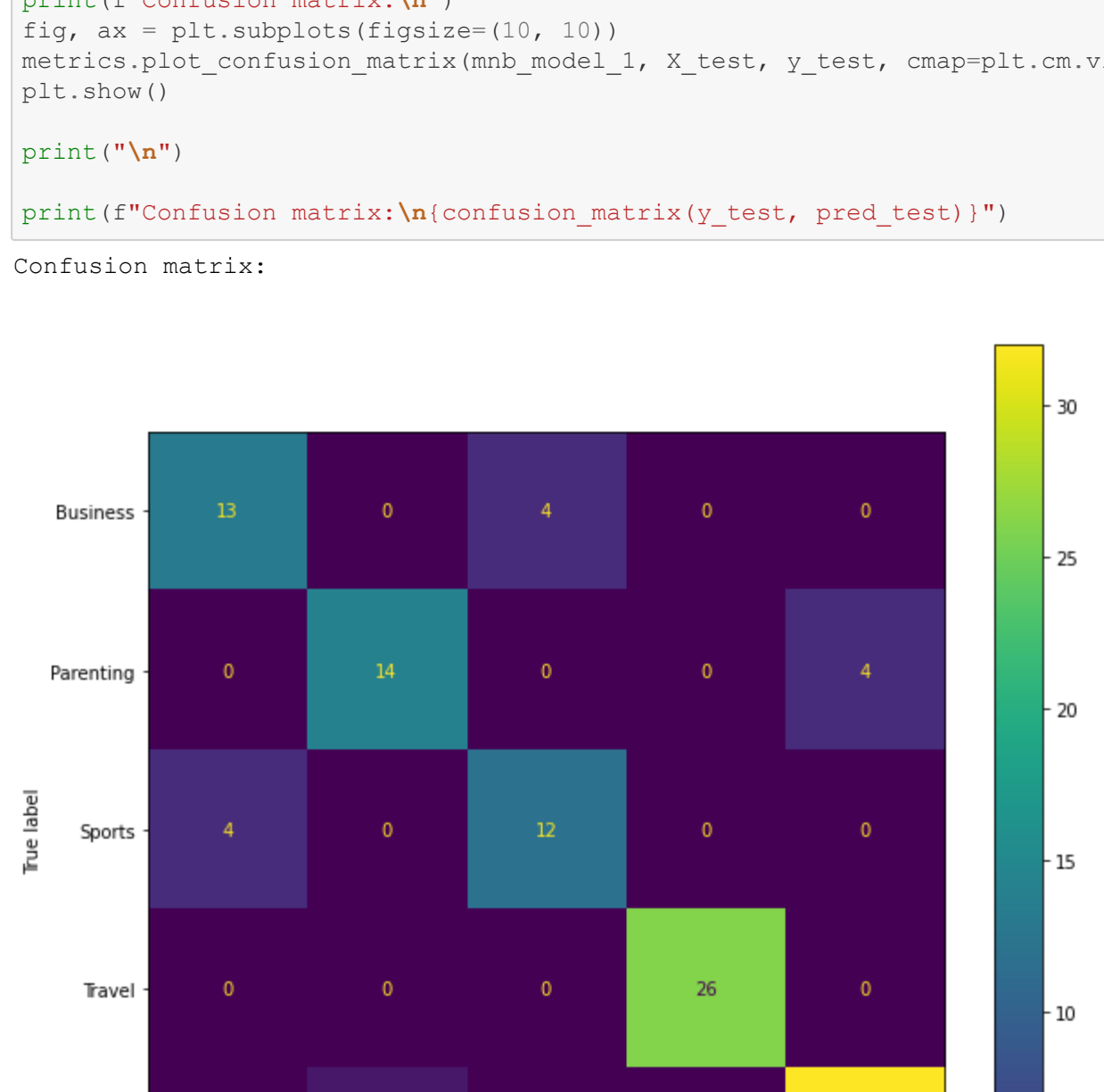
Aquí podemos observar que el modelo sobre los datos de entrenamiento da un resultado de un 93% mientras que para los de test de un 87.39%, sin embargo es necesario calcular el recall, precision y el F1 score para ver si el modelo puede producir resultados decentes.

```
In [104]: # Creando visualizacion de la matriz de confusion
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
from sklearn import metrics
print(f"Confusion matrix:\n")
fig, ax = plt.subplots(figsize=(10, 10))
metrics.plot_confusion_matrix(mnb_model_1, X_test, y_test, cmap=plt.cm.viridis, ax=ax)
plt.show()

print("\n")

print(f"Confusion matrix:\n{confusion_matrix(y_test, pred_test)}")
```

```
Confusion matrix:
```




```
Confusion matrix:
[[13  0  4  0  0]
 [ 0 14  0  0  4]
 [ 4  0 12  0  0]
 [ 0  0  0 26  0]
 [ 0  2  0  0 32]]
```

Análisis: podemos observar de forma rápida que en la categoría travel es en donde suceden la mayor cantidad de aciertos, seguido de wellness y parenting, mientras que las predicciones más pobres son realizadas por sports.

```
In [105]: # Imprimiendo resultados para ver el accuracy, precision y recall
print(
    f"Classification report for classifier {mnb_model_1}:\n"
    f"{metrics.classification_report(y_test, pred_test)}\n"
)
```

```
Classification report for classifier MultinomialNB(fit_prior=False):
              precision    recall  f1-score   support

    Business       0.76      0.76      0.76        17
   Parenting       0.88      0.78      0.82        18
      Sports       0.75      0.75      0.75        16
      Travel       1.00      1.00      1.00        26
    Wellness       0.89      0.94      0.91        34

    accuracy                           0.87       111
   macro avg       0.86      0.85      0.85       111
weighted avg       0.87      0.87      0.87       111
```

Análisis: gracias al resumen que se nos muestra podemos ver que la predicción a nivel de precisión, que recordemos que nos responde la siguiente pregunta: ¿qué porcentaje de las categorías que identifiquemos estarán realmente correcto? (es decir, mide la calidad del modelo). En este caso podemos ver que la categoría que tiene el score más alto es travel, con un 100% de acierto, es decir: nunca se equivoca, mientras que la más baja es la de sports con un 75%, es decir, para el algoritmo es tarea fácil identificar la categoría travel, pero le cuesta un poco más predecir Sports (tiende a cometer más errores tipo I).

A nivel del recall (esta nos informa sobre la cantidad que el modelo es capaz de identificar) que recordemos que este responde a la pregunta: ¿qué porcentaje de las categorías correctas somos capaces de identificar? Podemos ver que la categoría Travel con un 100%, es decir, nunca se equivocó, mientras que la más baja es Sports con un 75%, es decir la categoría Sports conlleva una mayor cantidad de FN que el algoritmo predice incorrectamente (error tipo II).

Finalmente podemos ver como el F1 que, recordemos que hace más fácil el poder comparar el rendimiento combinado de la precisión y la exhaustividad entre varias soluciones, obtenemos que el que tiene mayores problemas identificando es la categoría Sports con un 75%, mientras que el que identifica de forma bastante aceptable es travel.

Con esta evaluación podemos observar que en realidad nuestro modelo está dando resultados bastante decentes, excepto por la categoría business que es a la que más le cuesta identificar.

En caso de que tuviesemos un dataset con solo dos clases podríamos calcular la Curva ROC y la curva de Precision y Recall, pero al ser más de dos categorías se debe analizar con las métricas antes presentadas.

Conclusiones:

Se puede observar como ambos modelos dan resultados bastante aceptables, incluso cuando hay desbalanceos de clases.

Algo importante de mencionar es que este algoritmo trabaja bien en datasets pequeños como en el caso del segundo.

Para el presente trabajo no se calculó la gráfica de accuracy y precision así como la curva ROC ya que contaba con más categorías que las dos estándar, sin embae