PRÁCTICA I TEXT MINING

CLASIFICACIÓN DE TWEETS

Carlos Grande Núñez

UNIVERSIDAD REY JUAN CARLOS

[ÍNDICE]

01 Introducción a la práctica

01

Descripción

La práctica propuesta consiste en apoyarnos en las librerías de procesamiento del lenguaje natural vistas en clase, para realizar un análisis de sentimientos sobre un conjunto de datos obtenidos de twitter. Para ello se nos proporcionan 1008 tweets en español y el script de un modelo de clasificación Naive Bayes.

02

Durante la práctica aplicaremos diferentes procesados en el texto probando y comprendiendo cómo estos cambios afectan en la predicción del modelo propuesto.

03

Planeamiento inicial

El análisis de sentimientos que se plantea en esta práctica es mediante aprendizaje automático supervisado, es decir, utilizando algún algoritmo de clasificación que aprenderá utilizando datos anotados en el entrenamiento y será capaz de predecir después una etiqueta de las cuatro posibles para cada "tweet" de un conjunto de pruebas. Por tanto, los pasos mínimos que se pueden dar para resolver el problema son los siguientes:

04

- Procesamiento del corpues en formato XML para convertir los "tweets" a texto.
- Procesar el texto para generar una representación común de cada "tweet", por ejemplo, una representación vectorial donde cada vector representa un "tweet". Cada componente del vector representa un rasgo del vocabulario del problema (todos aquellos rasgos únicos de todos los "tweet" que intervienen en el problema) y tiene un peso concreto, que mide la importancia de ese rasgo en el "tweet".

4.1

- A partir de una matriz de vectores de "tweet" se utilizaría un algoritmo de clasificación que clasificaría los "tweet" en las cuatro posibles categorías según su polaridad.
- Medir el rendimiento del algoritmo con el conjunto de datos de prueba.

4.2

Se proporciona un código base en Python que ya realiza algunas de las tareas del esquema anterior.

4.3

02 Objetivo general y objetivos específicos

Objetivo general

4.4

Procesar los tweets proporcionados mediante las herramientas de procesamiento de lenguaje natural.

Objetivos específicos

4.5

- Realizar una exploración previa de los datos.
- Optimizar el modelo de clasificación de Naive Bayes mediante el procesamiento de los tweets.
- Construir un nuevo modelo de clasificación basado en SVM.

4.6

4.7

4.8

A

03 Metodología

01 **Los datos**

Los datos proporcionados consisten en 1008 tweets en español clasificados en 4 categorías sobre un fichero XML. Usando el script de la práctica se genera un directorio con el corpus de los datos, almacenando los tweets por separado en ficheros de texto.

02

Los tweets serán agrupados por las categorías de polaridad de sentimientos: 418 negativos (N), 318 positivios (P), 133 neutrales (Neu) y 139 no definidos (None)

03 **Procedimiento**

Como paso previo a la resolución de la práctica se han traducido los tweets al inglés conservando sus categorías en un fichero CSV anexo. Este fichero se ha generado debido a que las librerías de NLP que vamos a usar en la práctica están optimizadas para la lengua inglesa. De esta manera la traducción podría ayudar a un mejor procesamiento de los tweets.

04

Una vez cargados los datos sin ningún procesado, hemos ejecutado el modelo de clasificación Bayesiana proporcionado en la práctica para ver la capacidad de predicción inicial del modelo.

4.1 A continuación, llevamos a cabo una estructuración de los datos mediante un data frame de exploración, donde cada columna supone algún tipo de cambio o procesamiento sobre los tweets originales. Esta tabla de datos nos servirá a lo largo de la práctica para ir almacenando los resultados y poder aplicarlos al final de la memoria en los modelos. El data frame inicial contendrá los tweets orignales y procesamientos sencillos como palabras sin signos de puntuación y sin stopwords y lemas.

4.2

Para poder justificar los procesamientos realizados en la práctica, en el siguiente paso realizaremos analisisi de exploración de los tweets. en este análisis vamos a analizar los siguientes puntos:

- Distribución de los tweets por categoría

4.3

- Procesado y filtrado de stopwords
- Normalización de onomatopeyas comunes (ja, ja, ja)

4.4

- Análisis y procesamiento de menciones
- Análisis y procesamiento de de hashtags
- Análisis y conversión de emoticonos a texto

4.5

- Palabras no reconocidas

Finalmente se entrenaran los modelos de clasificación mediante Naive Bayes y SVM probando los diferentes procesamientos llevados a cabo sobre los datos para ver cual de éstos da el mejor resultado.

4.6

4.7

4.8

04 Elaboración de la práctica

4.1 Situación de los archivos originales y generados

Los archivos usados para la elaboración de la práctica son:

- *baselinePractica1_19_20_v3.py* (script de python facilitado par la creación del corpus)
- *Translator.py* (script para la traducción de los tweets al inglés)
- *Practica.ipynb* (cuaderno de jupyter notebook con el código y resultados de la práctica)
- */CorpusTrainTASS* (directorio con el contenido de los tweers en ficheros .txt)
- */helpers* (directorio con los ficheros de texto auxiliares a la práctica)
- /helpers/ tweets_en.csv (fichero csv con la lista de tweets en inglés)
- /helpers/ false_stops_es.json (fichero json con la lista de stopwords seleccionados en español)
- /helpers/ false_stops_en.json (fichero json con la lista de stopwords seleccionados en inglés)

Mejoras previas

Antes de comenzar la práctica, se ha optado por traducir al inglés los tweets originales proporcionados en español con el fin ajustar mejor los modelos de análisis de sentimientos. Esto se debe a que la librería spacy usada para la práctica, aunque permite el análisis de téxtos en español está mejor entrenada en inglés lo que podría arrojar mejores resultados a la hora de procesar los textos.

Para poder llevar a cabo la traducción de los tweets se ha usado la librería de googletrans, una librería gratuita que permite usar los mismo sevidores del traductor de google.

googletrans: https://pypi.org/project/googletrans/

El script Translator.py generado para esta práctica permite traducir los tweets de español a inglés

```
from googletrans import Translator

# Declaring variables
tweets_en = []
errors = []
translator = Translator()
times = []

# Translating tweets
for i in range(len(X)):
    rand = rd.randrange(20,25)/10
    try:
        tweet_en = translator.translate(X[i], src='es').text
        tweets_en.append(tweet_en)
    except:
        print('Translation failed in tweet number {}'.format(i))
        errors.append(i)
        tweets_en.append(None)
    t.sleep(rand)

# Saving results
df_tweets_en = pd.DataFrame({'tweets_en': tweets_en})
df_errors = pd.DataFrame({'errors index': errors})
df_tweets_en.to_csv('tweets_en.csv')
```

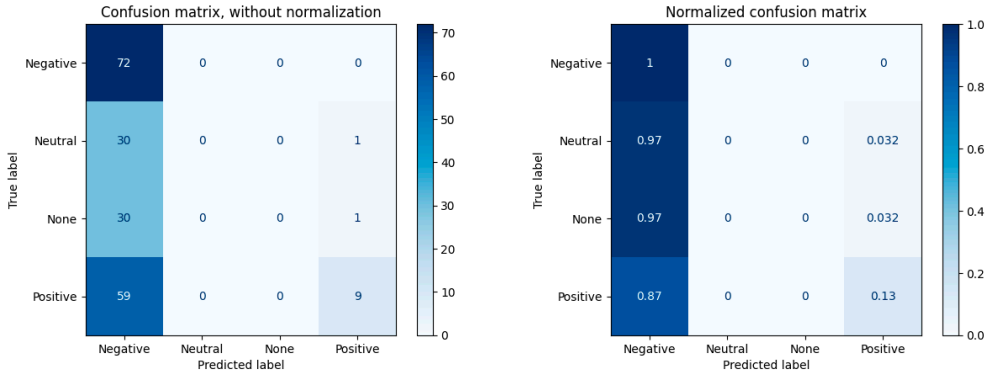
4.2 Entenamiento del modelo Naive Bayes con los tweets sin procesar

Para confirmar la mejora del procesamiento de texto a lo largo de la práctica se aplica primeramente el modelo de Naive Bayes sobre los tweets en crudo obteniendo los siguientes resultados. Además se genera un data frame de resultados para ir almacenando los parámetros obtenidos de los modelos usados en la práctica.

Ejecuta el entrenamiento del clasificador bayesiano.

```
tweets_es = X
target = y
acc = nbayes(tweets_es, target)
```

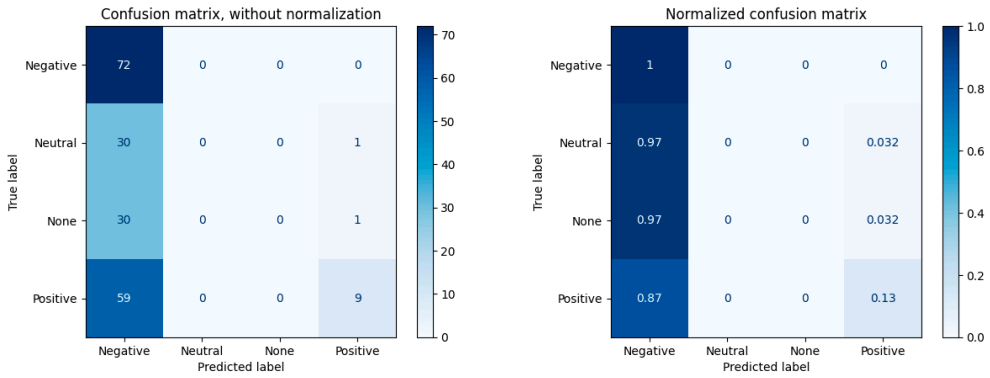
accuracy 0.400990099009901 *nbayes*



Ejecuta el entrenamiento del clasificador bayesiano.

```
acc = nbayes(tweets_en, target)
```

accuracy 0.400990099009901 *nbayes*



Al extraer los resultados, podemos observar como el modelo con los tweets en español tiene un accuracy del 0.40. Este parámetro indica que el modelo ha sido capáz de clasificar un 40% de los datos de TEST (202 tweets).

Si nos fijamos en la matríz de confusión veremos que más del 90% de los tweets han sido clasificados como negativos habiendo acertado 81 de los 202 tweets, 72 como negativos y 9 como positivos.

Por otro lado, podemos observar que la traducción sin procesamiento no arroja ningún cambio en el entrenamiento del modelo, esto es una buena señal ya que indica que la traducción no ha generado una pérdida notable. Recordemos que la mejora en la utilización de tweets traducidos, si se produce será en las fases posteriores de procesamiento de texto.

4.3 Carga de los modelos de spacy y filtrado de stopwords

En este apartado además de cargar los módulos de spacy, realizamos un filtro personalizado de stopwords a partir de dos ficheros json generados en la práctica que explicaremos más adelante.

Carga de los módulos con el vocabulario de spacy y posterior modificación

```
# loading spacy module
nlp_es_spacy = spacy.load('es_core_news_md')
nlp_en_spacy = spacy.load('en_core_web_lg')
nlp_es = spacy.load('es_core_news_md')
nlp_en = spacy.load('en_core_web_lg')

# Loading selected stopwords in Spanish
with open('helpers/false_stops_es.json', 'r') as myfile:
    data=myfile.read()
false_stops_es = json.loads(data)

# Excluding selected stopwords in Spanish
for fstop in false_stops_es:
    nlp_es.vocab[fstop].is_stop = False
```

4.4 Estructuración de los datos en un data frame de exploración

Para poder realizar distintos procesamientos a los tweets y tener cierta flexibilidad en la aplicación de los modelos, vamos a generar un data frame de exploración que permita almacenar todos los cambios realizados a lo largo de la práctica.

El data frame restultante consiste en una tabla de 1008 filas, una por cada tweet y 11 columnas con diferentes resultados de los procesos realizados a cada uno de los tweets. Esta tabla no es estática e irá variando a lo largo de la práctica.

Los parámetros iniciales de la tabla son los siguientes:

- tweets_es Contiene los tweets originales en español.
- tweets_en Contiene los tweets traducidos en inglés.
- tokens_es Contiene la palabras de los tweets en español sin stopwords.
- tokens_es* Contiene lo mismo que la anterior pero filtrando los stopwords.
- tokens_en Contiene la palabras de los tweets en inglés sin stopwords.
- toknes_en* Contiene lo mismo que la anterior pero filtrando los stopwords.
- lemas_es Contiene los lemas de los tweets en español filtrando los stopwords.
- lemas_en Contiene los lemas de los tweets en español filtrando los stopwords.
- stops_es Contiene la lista de stopwords en español de ese tweet.
- stops_en Contiene la lista de stopwords en inglés de ese tweet.
- target Contiene la clasificación de los tweets en un rango entre 0 y 3 (Neg, N, None, P)

El data frame de exploración inicial contendrá los procesamientos básicos de los tweets por duplicado en español e inglés y posteriormente se añadiran elementos comunes como emoticonos, hashtags o menciones.

La ventaja de estructurar los datos de esta manera es que permite diferentes combinaciones de procesamientos por tweet, lo que posteriormente nos dará flexibilidad en los modelos.

Generación del data frame de exploración

```
tweets_es = X
tweets_en = list_tweets_en
target = y

# Processing tokens with default stops
tokens_es = [get_tokens(tweet, nlp_es_spacy) for tweet in tweets_es]
tokens_en = [get_tokens(tweet, nlp_en_spacy) for tweet in tweets_en]

# Processing tokens with selected stops
tokens_es_selected = [get_tokens(tweet, nlp_es) for tweet in tweets_es]
tokens_en_selected = [get_tokens(tweet, nlp_en) for tweet in tweets_en]

# Processing lemas
lemas_es = [get_lemas(tweet, nlp_es) for tweet in tweets_es]
lemas_en = [get_lemas(tweet, nlp_en) for tweet in tweets_en]
print('lemas processed')

# Procesing stops
stops_es = [get_stops(tweet, nlp_es_spacy) for tweet in tweets_es]
stops_en = [get_stops(tweet, nlp_en_spacy) for tweet in tweets_en]

# Data frame creation
df = pd.DataFrame({'tweets_es': tweets_es,
                   'tweets_en': tweets_en,
                   'tokens_es': tokens_es,
                   'tokens_es*': tokens_es_selected,
                   'tokens_en': tokens_en,
                   'tokens_en*': tokens_en_selected,
                   'lemas_es': lemas_es,
                   'lemas_en': lemas_en,
                   'stops_es': stops_es,
                   'stops_en': stops_en,
                   'target': target})
```

get_tokens, get_lemas, get_stops

	tweets_es	tweets_en	tokens_es	tokens_es*	tokens_en	tokens_en*	lemas_es	lemas_en	stops_es	stops_en	target
0	@sosagraphs Pues tan sencillo porque en su día...	For @sosagraphs so simple because at the time ...	[y, estilos, encargos, aparte, @sosagraphs, se...	[y, estilos, encargos, aparte, porque, hacerlo...	[makes, simple, styles, time, @sosagraphs, com...	[makes, simple, styles, time, @sosagraphs, com...	[y, poner, porque, apartar, hacerlo, @sosagrap...	[simple, time, @sosagraph, style, start, crack...	[pues, tan, porque, en, su, día, me, hacerlo, ...	[for, so, because, at, the, i, to, do, and, wh...	3
1	1477. No pero porque apenas hay confi	1477. No but because there just confi	[confi, 1477]	[confi, porque, no, 1477]	[confi, 1477]	[confi, 1477, because]	[confi, porque, no, 1477]	[confi, 1477, because]	[no, pero, porque, apenas, hay]	[no, but, because, there, just]	0
2	Vale he visto la tia bebiendose su regla y me ...	It've seen the aunt drinking his rule and hs g...	[y, regla, bebiendose, muchs, vale, visto, tia...	[y, regla, bebiendose, dado, muchs, vale, vist...	[rule, muchs, it've, aunt, seen, given, grima,...	[rule, muchs, it've, aunt, seen, hs, given, gr...	[y, bebiendose, vestir, muchs, grima, tia, dar...	[give, rule, -PRON-, muchs, it've, aunt, hs, d...	[he, la, su, me, dado]	[the, his, and, me]	0
...	...	...	...	...	...	...	...	...	...	...	...
1005	@anayainfantil @ayuntamientoibi @Bibliotecalbi...	@anayainfantil @ayuntamientoibi @Bibliotecalbi...	[duda, @anayainfantil, límites, edad, @ayuntam...	[duda, @anayainfantil, límites, edad, @ayuntam...	[@anayainfantil, perfect, @ayuntamientoibi, su...	[@anayainfantil, perfect, @ayuntamientoibi, su...	[@anayainfantil, edad, @ayuntamientoibi, dudar...	[@anayainfantil, perfect, limit, @ayuntamiento...	[sí, pero, mi, es, si, hay, de, para, a]	[but, my, is, whether, there, are, to, be, to,...	2
1006	@CharlieReckless llévame a tu casa, la otra ve...	@CharlieReckless take me to your home, and tod...	[y, dragonite, casa, blastoise, @charliereckle...	[y, dragonite, casa, blastoise, @charliereckle...	[dragonite, home, blastoise, @charliereckless,...	[dragonite, home, blastoise, @charliereckless,...	[y, dragonite, blastoise, @charliereckless, ca...	[dragonite, home, blastoise, @charliereckless,...	[tu, la, otra, vez, hoy]	[take, me, to, your, and, again]	2
1007	@IgnacioJurado a mi a las 8.15 que son las 7.1...	@IgnacioJurado me at 8.15 to 7.15 are not too bad	[7.15, 8.15, bad, too, @ignaciojurado, a]	[7.15, 8.15, bad, too, @ignaciojurado, no, a]	[bad, @ignaciojurado, 7.15, 8.15]	[7.15, 8.15, bad, too, @ignaciojurado, not]	[7.15, 8.15, bad, toar, @ignaciojurado, no, a]	[7.15, 8.15, bad, too, @ignaciojurado, not]	[mi, las, que, son, las, no]	[me, at, to, are, not, too]	2

1008 rows × 11 columns

4.5 Exploración y tratamiento de los datos

En el siguiente apartado vamos a realizar un pequeño análisis exploratorio que nos permita conocer mejor como se organizan los tweets y su distribución. De esta manera podremos procesar mejor los datos.

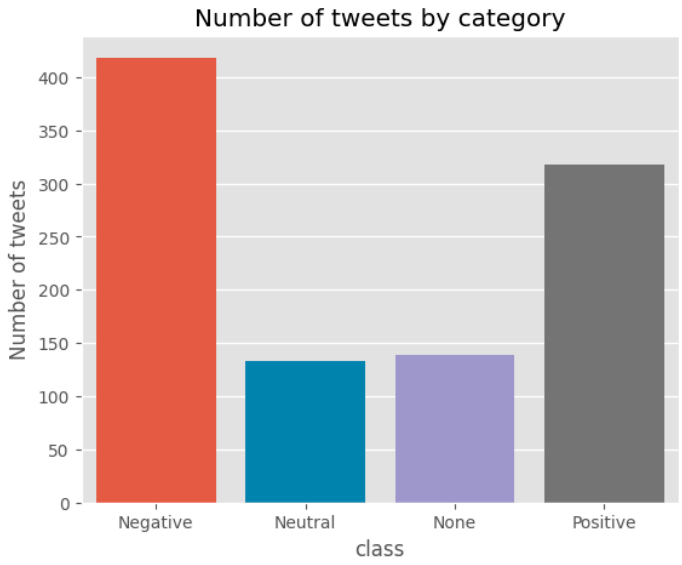
_1 Número de tweets por categoría

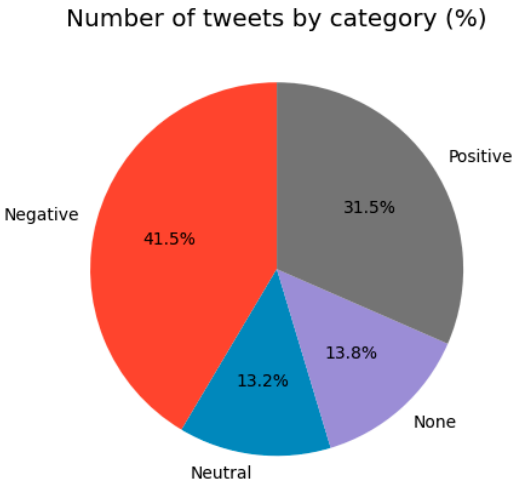
En este punto realizaremos un análisis previo del número de tweets en función a su repetición y categoría, mostrando los resultados mediante los siguientes diagramas.

Muestra de la distribución de los tweets por categoría

```
countplot(df, title = 'Number of tweets by category')
```

countplot





_2 Análisis de stopwords

En este apartado realizaremos y clasificaremos las stopwords por frecuencia y categoría.

Muestra de la distribución de los tweets por categoría

```
df_stops_freq_es = freq_table(df[['stops_es', 'target']], 'stopwords')
df_stops_freq_en = freq_table(df[['stops_en', 'target']], 'stopwords')
df_stops_freq_es.head()
```

freq_table

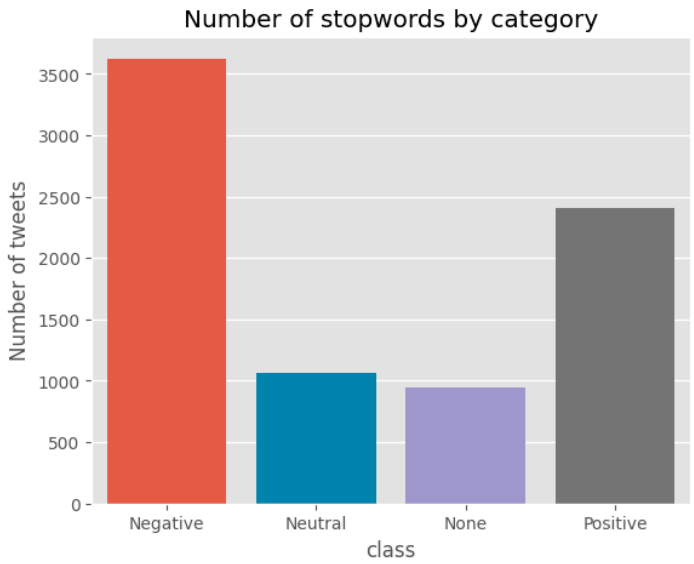
	stopwords	target
0	pues	3
1	tan	3
2	porque	3
3	en	3
4	su	3

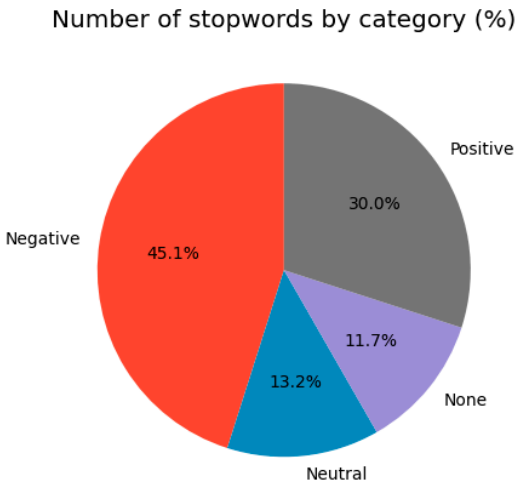
Este paso nos permite generar una lista con todas las stopwors de los tweets manteniendo la categoría a la que pertenecen. Esta nueva tabla nos permitirá medir su frecuencia y su distribución por categorías.

Muestra de la distribución de las stopwords por categoría

```
countplot(df_stops_freq_es, title = 'Number of stopwords by category')
```

countplot





Aunque podemos observar que un 45% de los stopwords son negativos, mientras que un 30% son positivos, estos datos no aportan mucha información ya que coinciden con los porcentajes de la cantidad de tweets por categoría. Llegados a este punto una idea interesante sería estudiar las frecuencias de los stopwords más repetidos por categoría y compararlos mutuamente.

Muestra la frecuencia de repetición de las stopwords negativos

```
negative_stops_es = df_stops_freq_es[df_stops_freq_es.target == 0].groupby('stopwords').target.count().reset_index()\
        .sort_values(by=['target'], ascending = False).reset_index(drop=True).rename(columns={'target': 'freq'})
negative_stops_es.head(5)
```

	stopwords	freq
0	que	250
1	de	226
2	no	207
3	me	154
4	la	134

stopwords negativos

	stopwords	freq
0	de	152
1	que	138
2	la	94
3	el	84
4	en	78

stopwords positivos

La idea de generar dos tablas de stopwords positivas y negativas por separado con sus frecuencias nos permite cruzarlas posteriormente, con el fin de descubrir que stopwords tienen una alta repetición en una tabla y una baja o nula repetición en la tabla opuesta. De esta manera descubriremos que stopwords nos conviene eliminar ya que no aportan información y que stopwords debemos mantener. A estas nuevas stopwords, las llamremos "false stopwords". El siguiente script permite un outer join obteniendo stopwors con una relación en su frecuencia de un 30% o menor, obteniendo las stopwords más desiguales.

```
threshold = 0.3
df_merged_es = pd.merge(negative_stops_es, positive_stops_es, on='stopwords', how = 'outer')
df_merged_es = df_merged_es[(df_merged_es.freq_x.isna()) | (df_merged_es.freq_y.isna()) | (df_merged_es.freq_x/
df_merged_es.freq_y <= threshold) | (df_merged_es.freq_y/df_merged_es.freq_x <= threshold)]
df_merged_es.head()
```

	stopwords	freq_x	freq_y
2	no	207.0	60.0
16	porque	43.0	10.0
39	ni	16.0	1.0
40	le	16.0	3.0
45	están	15.0	3.0

Tabla de frecuencias menores al 30%

	stopwords	freq_x	freq_y
2	no	207.0	60.0
16	porque	43.0	10.0
39	ni	16.0	1.0
40	le	16.0	3.0
45	están	15.0	3.0

Tabla ordenada por mayor freq negativos

	stopwords	freq_x	freq_y
2	no	207.0	60.0
100	mejor	6.0	21.0
266	buen	1.0	15.0
16	porque	43.0	10.0
324	buena	NaN	10.0

Tabla ordenada por mayor freq positivos

```
false_stops_es = list(df_merged_es[(df_merged_es.freq_x > 1) | (df_merged_es.freq_y >1)].stopwords)
false_stops_en = list(df_merged_en[(df_merged_en.freq_x > 1) | (df_merged_en.freq_y >1)].stopwords)
```

['no', 'porque', 'ni', 'le', 'estan']

Tras cruzar las tablas obtenemos las mayores frecuencias de stopwords negativos y positivos en los que hay una disparidad de más de 1/7 obtenida por el threshold de 0'3. Esto nos permite generar dos listas de false stopwords que guardaremos en formato Json. Estos ficheros Json se han cargado al inicio de la práctica excluyendo estas stopwords del vocabulario de Spacy.

_3 Normalización de onomatopeyas comunes de la risa ja, ja, ja

Otro procesamiento que podría ayudarnos en el entrenamiento de modelos, sería el de normalizar las variaciones de la onomatopeya de la risa. De esta manera apoyándonos en expresiones regulares reduciremos las variaciones de esta onomatopeya para aumentar así su frecuencia sustituyéndola por **'risa'** y **'laugh'**.

Muestra de la distribución de los tweets por categoría

```
tweets_jaja_es = []
tweets_jaja_en = []
pattern_es = re.compile('j\w*j\w*j\w*|a*ja+j[ja]*|lo+l')
pattern_en = re.compile('a*ha+h[ha]*|lo+l')

for index, row in df.iterrows():
    tweet_es = row.tweets_es
    tweet_en = row.tweets_en
    finded = re.findall('j\w*j\w*j\w*|a*ja+j[ja]*|lo+l', tweet_es)
    if finded != []:
        new_tweet_es = pattern_es.sub(' risa ', tweet_es)
        new_tweet_en = pattern_en.sub(' laugh ', tweet_en)
        tweets_jaja_es.append(new_tweet_es)
        tweets_jaja_en.append(new_tweet_en)
        jaja_checker.append(True)
    else:
        tweets_jaja_es.append(tweet_es)
        tweets_jaja_en.append(tweet_en)
        jaja_checker.append('')
```

['@ElSpeaker La vida nos lleva por caminos inextricables **risa** Me alegro de que te haya molado el programa de hoy ', '@ mamiauryner1 espero sobretodo tenerla con mi nino **risa** . gracias guapa']

["@ElSpeaker Life takes us through inextricable ways **laugh** I'm glad you have molado today's program", '@ Mamiauryner1 especially hope to have it with my **laugh** child. thanks beautiful ']

_4 Analisis de menciones de usuarios (@usuario)

01 En este apartado hacemos una extracción de los usuarios mencionados en los tweets para estudiar su distribución, una posible actuación sería la sustitución de las menciones por una palabra clave.

Genera una nueva columna en el dataframe que contenga las mensiones en una lista

```
mentions = [only_mentions(lema) for lema in df.lemas_es]
df['mentions'] = mentions
df.iloc[0:4, [0, 1, -1]]
```

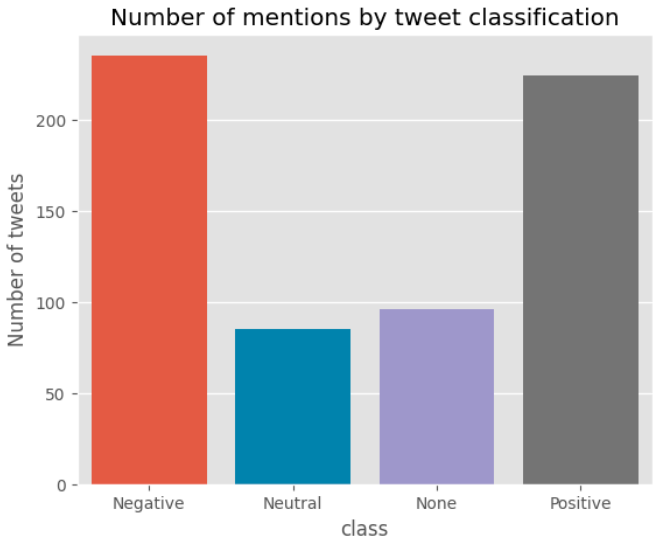
only_mentions

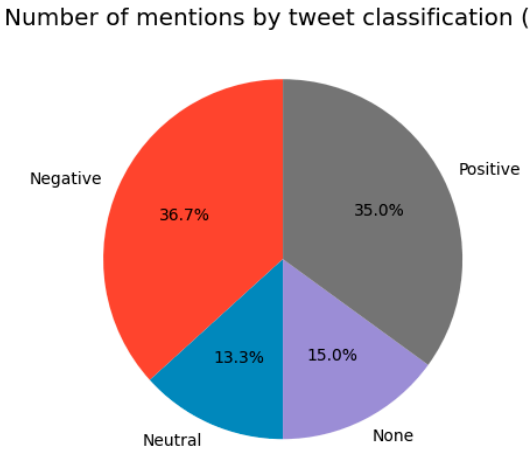
	tweets_es	tweets_en	mentions
0	@sosagraphs Pues tan sencillo porque en su día...	For @sosagraphs so simple because at the time ...	[@sosagraphs]
1	1477. No pero porque apenas hay confi	1477. No but because there just confi	
2	Vale he visto la tia bebiendose su regla y me ...	It've seen the aunt drinking his rule and hs g...	
3	@Baronachor El concepto es de lo más bonico, p...	@Baronachor The concept is most bonico, but th...	[@baronachor]

Muestra de la distribución de menciones por categoría

```
df_mentions = df[df.mentions != ''].reset_index(drop=True)
countplot(df_mentions, title = 'Number of mentions by tweet classification')
```

countplot

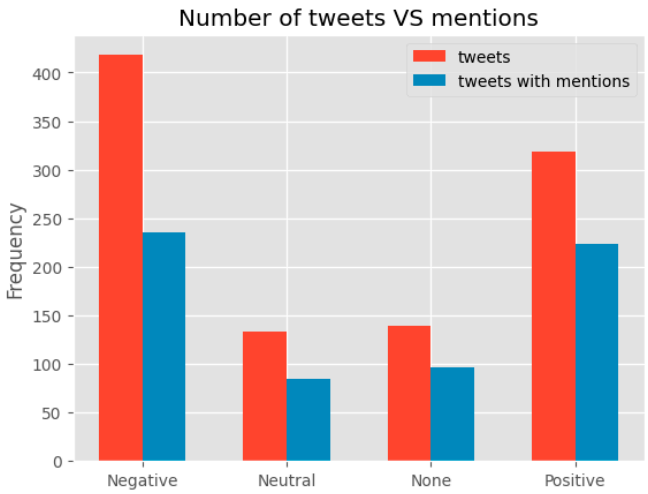




Muestra la proporcion de tweets frente al número de menciones por categoría

```
comparison_plot(df, 'mentions')
```

comparison_plot



_5 Analisis de hashtags (#hashtag)

El objetivo de este apartado es realizar un análisis en la distribución de los hashtags y comprobar si es posible su conversion a palabras ya que muchos de los hashtags son conjuntos de palabras sin espacios.

Muestra de la distribución de los tweets por categoría

```
hashtags = [only_hash(tweet) if len(only_hash(tweet)) != 0 else '' for tweet in df.tweets_es]
df['hashtags'] = hashtags
df.iloc[[7, 15, 26], [0, 2, -1]]
```

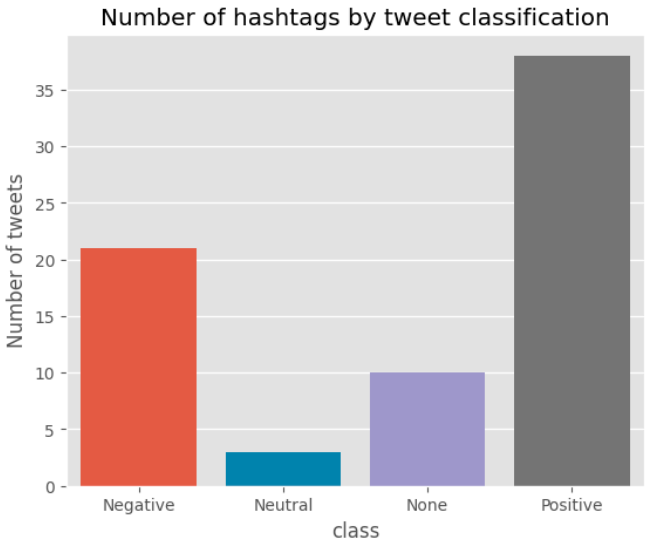
only_hash

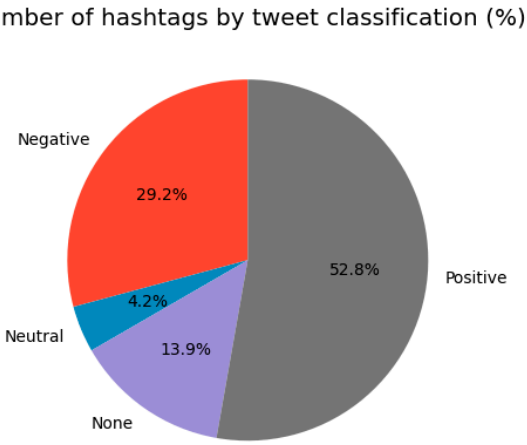
	tweets_es	tokens_es	hashtags
7	@Diego_FDM @el_pais La noticia perfecta para #...	[@el_pais, notasdelmisterio, noticia, perfecta...	[NotasDelMisterio]
15	Hoy microaventura en #kayak con dos expertos k...	[https://t.co/y7j88brmyk, expertos, puntal, vi...	[kayak]
26	6 momentos en los que usar lentes de contacto ...	[lunes, simplemente, https://t.co/x06g1nkah1, ...	[optometria, lentesdecontacto]

Muestra de la distribución de los hashtags por categoría

```
df_mentions = df[df.mentions != ''].reset_index(drop=True)
countplot(df_mentions, title = 'Number of mentions by tweet classification')
```

countplot

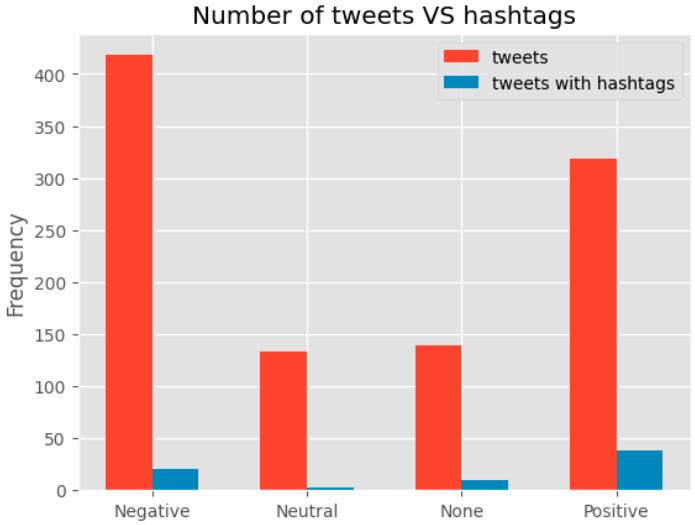




Muestra la proporcion de tweets frente al número de hashtags por categoría

```
comparison_plot(df, 'mentions')
```

comparison_plot



A continuación haremos un análisis de los hashtags por repetición y clasificación, por si encontásemos un patrón común de hashtags en alguna de las categorías principales.

Genera una tabla con cada hashtag y la categoría a la qu pertenece

01
```
df_hashtags_freq = freq_table(df[['hashtags', 'target']], 'hashtags')
df_hashtags_freq.head()
```

freq_table

Permite extraer y medir las repeticiones de los hashtags negativos ordenándolos de mayor a menor

02
```
negative_hashtags = df_hashtags_freq[df_hashtags_freq.target == 0].reset_index(drop=True)\
    .groupby('hashtags').target.count().reset_index().sort_values(by=['target'], ascending = False)\
    .reset_index(drop=True).rename(columns={'target': 'freq'})
negative_hashtags.head(10)
```

03

	hashtags	target
0	playa	3
1	NotasDelMisterio	3
2	kayak	3
3	optometria	2
4	lentesdecontacto	2

04

Tabla de frecuencias

	hashtags	freq
0	askalvarogango	2
1	Algeciras	1
2	Sangüesa	1
3	rcde	1
4	justice4hombres	1

Tabla de hastags negativos

	hashtags	freq
0	HableConEllas	3
1	AcapulcoShore3	1
2	enlacabezaNo	1
3	TonyYSorayaEnMalaga	1
4	TrendingEstreno	1

Tabla de hastags positivos

4.1

Posteriormente procuraremos separar aquellos hashtags que debido a su patrón permitan procesarlos en palabras para probar esta modificación posteriormente en el modelo y ver si produce alguna mejoría.

4.2

Genera un diccionario con los hashtags como claves y su posible separación en palabras

4.3
```
all_hashtags = [hash for list in df.hashtags for hash in list]
h2word = [hash2word(hash) for hash in all_hashtags]
hash_dic = {h:w for h, w in zip(all_hashtags, h2word)}
hash_dic['HableConEllas']
```

hash2word

4.4
```
'hable con Ellas'
```

Con este diccionario podremos recorrer la lista de hashtags por tweet y crear una nueva columna con los hashtags convertidos en palabras siempre que cumplan el patrón de mayúscula por cada palabra.

Posteriormente eliminaremos los stopwords de cada hashtags dejando una lista de tokens procesados que pueden ser unidos al tweet original.

4.5

Genera un diccionario con los hashtags como claves y su posible separación en palabras

4.6
```
# Tokenizing hashtags
hash2tokens = [get_tokens(word, nlp_es) for word in words]
df['hash2tokens'] = hash2tokens
df.iloc[[5, 7, 26], [0, -2, -1]]
```

get_tokens, hash2tokens

4.7

	tweets_es	hash2words	hash2tokens
5	Como siempre mi tortilla triunfa mas que otros...	playa	[playa]
7	@Diego_FDM @el_pais La noticia perfecta para #...	notas del Misterio	[notas, misterio]
26	6 momentos en los que usar lentes de contacto ...	optometria lentesdecontacto	[lentesdecontacto, optometria]

4.8

_6 Analisis de emoticonos

01 En este apartado nos ayudaremos de la librería *spacymoji* de python para poder procesar a texto cada uno de los emoticonos y ver si se produce una mejora en el modelo.

spacyemoji: https://spacy.io/universe/project/spacymoji

Carga de la librería spacymoji y su diccionario

02
```
from spacymoji import Emoji
emoji = Emoji(nlp_en)
nlp_en.add_pipe(emoji, first=True)
```

03 *Detección y conversión de emoticonos a texto*

```
emojis_en = []
for tweet in df.tweets_es:
    doc = nlp_en(tweet)
    if doc._.has_emoji:
        emojis = doc._.emoji
        emojis = [emoji[2] for emoji in emojis]
        emojis_en.append(emojis)
    else:
        emojis_en.append('')
df['emojis_en'] = emojis_en
```

4.1

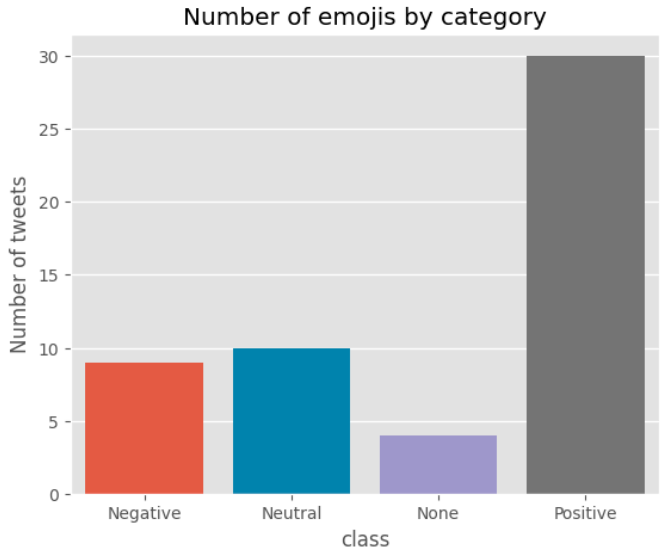
4.2 *Traducción de las descripciones de los emoticonos al español*

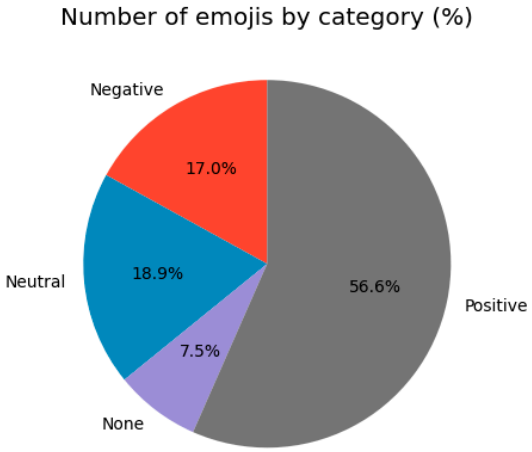
```
emojis_es = [[translate(emj) for emj in lst] if len(lst) > 0 else '' for lst in emojis_en]
print(emojis_en[67])
print(emojis_es[67])
```

4.3 *get_tokens, hash2tokens*

```
['face with tears of joy', 'face with tears of joy']
['cara con lagrimas de alegria', 'cara con lagrimas de alegria']
```

4.4 Apoyándonos en la función countplot realizaremos un estudio de la distribución de los emoticonos en función de su categoría.





4.5

4.6

4.7

4.8 Aunque la mayoría de los emoticonos se encuentran en los tweets positivos tras un posterior análisis de repetición del emoticono por categoría descubrimos que no hay un emoticono común para cada categoría. Esto quiere decir que a pesar de la conversión a texto de los emoticonos, probablemente no supongan una gran mejoría, salvo por su alta frecuencia en la categoría de positivos.

A

_7 Analisis de palabras no reconocidas en español

01 Otra idea que podría ser de ayuda es la detección de erratas o abreviaturas comunes que al ser corregidas contabilicen de forma positiva en las frecuencias del modelo bayesiano.

Traducción de las descripciones de los emoticonos al español

```
# Words unknown
all_tokens = [' '.join(token) for token in df.tokens_jaja_es]
tokens_es = nlp_es(' '.join(all_tokens))
tokens_unknown = [ukn_cleaner(token.text) for token in tokens_es if token.is_oov and len(token.text) > 2]

# Words unknown by frequency
tokens_freq = counter(tokens_unknown)
tokens_unknown = list(set(tokens_unknown))
freq_tokens_unknown = [tokens_freq[token] for token in tokens_unknown]
df_unknown = pd.DataFrame({'tokens': tokens_unknown, 'freq': freq_tokens_unknown})
df_unknown = df_unknown[df_unknown.tokens != ''].sort_values(by = 'freq', ascending = False)
```

counter, tokens_freq, ukn cleaner

There are 212 tokens unknown

	tokens	freq
24	hableconellas	3
121	askalvarogango	3
62	niñuca	2
101	felizmartes	2
175	vmas	2
200	gemeliers	2
1	grassias	1

Como podemos observar la palabra no reconocida usada con mayor frecuencia es 'hableconellas' repetida tan solo 3 veces. Este apartado no nos sirve de mucha utilidad en este caso debido al bajo número de erratas.

Probablemente si se hubiesen aportado un corpus mayor, este apartado cobraría más sentido.

4.6 Entrenamiento del modelo Naive Bayes

_1 Entrenamiento del modelo con tokens en español y en inglés (Spacy stopwords)

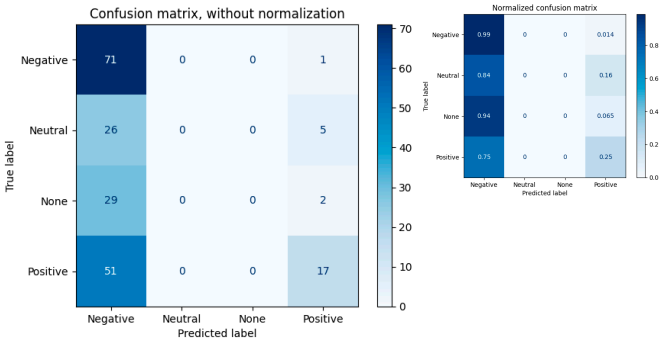
Para un primer entrenamiento tras el procesado vamos indroducir como muestra los tokens generados eliminando los simbolos de puntuación y stopwords que spacy tiene por defecto.

Selección de muestra de entrada para el modelo

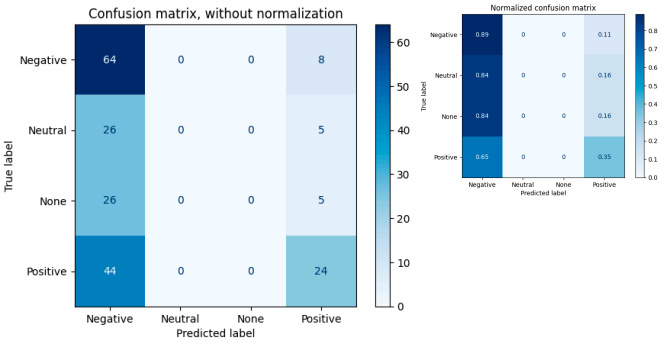
```
X_tokenized_es = [' '.join(tokens) for tokens in list(df.tokens_es)]; acc = nbayes(X_tokenized_es, target)
X_tokenized_en = [' '.join(token) for token in list(df.tokens_en)]; acc = nbayes(X_tokenized_en, target)
```

nbayes

accuracy 0.43564356435643564



accuracy 0.43564356435643564



Tras entrenar el modelo bayesiano eliminando los símbolos de puntuación y las stopwords de la base de datos de Spacy conseguimos una mejoría de un 3% con respecto al 0.40 que teníamos inicialmente. Probablemente la eliminación de estos elementos hayan mejorado las frecuencias, además de simplificar los cálculos al modelo al reducir las dimensiones de la matríz.

_2 Entrenamiento del modelo con tokens en español e inglés (usando false stopwords)

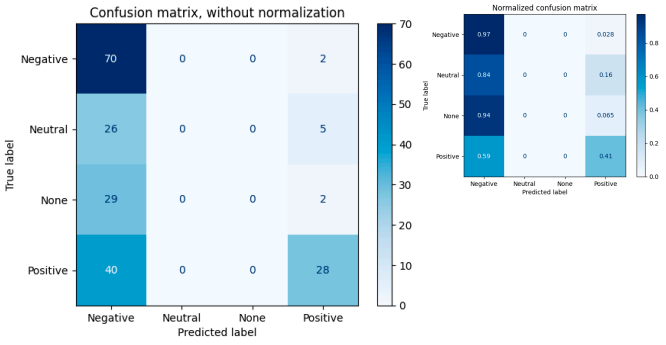
Uno de los procesamientos más importantes que hemos llevado a cabo anteriormente, consistía en el tratamiento de las stopwords. De esta manera se eliminan solo las stopwords seleccionadas previamente en el apartado (3.5_2)

Selección de muestra de entrada para el modelo

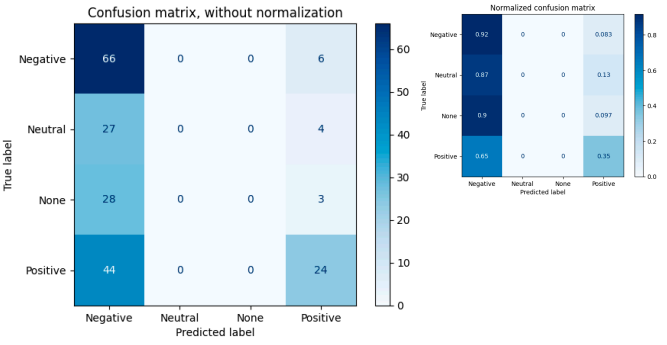
```
X_tokenized_es_2 = [' '.join(lema) for lema in list(df['tokens_es*'])]; acc = nbayes(X_tokenized_es_2, target)
X_tokenized_en_2 = [' '.join(lema) for lema in list(df['tokens_en*'])]; acc = nbayes(X_tokenized_en_2, target)
```

nbayes

accuracy 0.48514851485148514



accuracy 0.44554455445544555



Como veníamos intuyendo en apartados anteriores, la exclusión de algunos stopwords seleccionados por su frecuencia en relación a sus categorías ha producido una mejora de un 8.5% con respecto al 0.40% que teníamos inicialmente. Por otro lado, la mejora que esperábamos en los tweets traducidos al inglés no ha sido la esperada, probablemente el hecho de que las traducciones no son literales y Spacy no se ha entrenado con textos traducidos si no con nativos provoque una pérdida de información en el modelo.

_3 Entrenamiento del modelo con lemas en español e inglés (usando false stopwords)

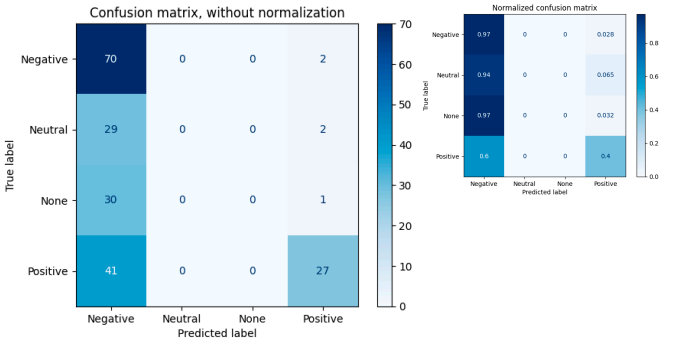
01 Aplicando el mismo filtro que en el apartado anterior, hemos lematizado los tokens, para ver si la reducción de todas las palabras a lemas ayuda en el entrenamiento del modelo.

```
X_lematized_es = [' '.join(lema) for lema in list(df.lemas_es)]; acc = nbayes(X_lematized_es, target)
X_lematized_en = [' '.join(lema) for lema in list(df.lemas_en)]; acc = nbayes(X_lematized_en, target)
```

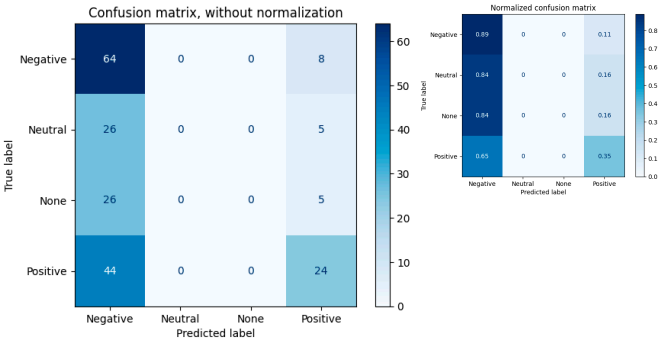
02

nbayes

accuracy 0.4801980198019802



accuracy 0.43564356435643564



03 04

4.1 Curiosamente tras la lematización el resultado empeora muy levemente con respecto al apartado anterior. Esto puede producirse porque al lematizar los tiempos verbales pasan a convertirse en infinitivos y se pierden pasados y futuros los cuales pueden estar relacionados con las categorías de los tweets.

4.2

_4 Entrenamiento del modelo con la transcripción de emoticonos en español

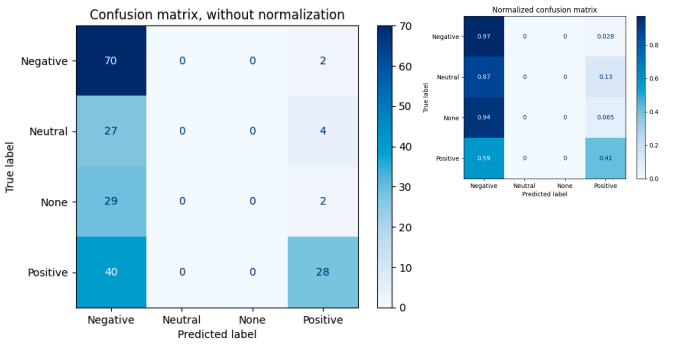
4.3 Continuando con el procesamiento que ha dado mejores resultados, los tokens con false stopwords, vamos a añadir las transcripciones de los emoticonos a los tweets para comprobar si produce una mejora.

```
X_tokens_emojis_es = [' '.join(token) + ' ' + ' '.join(emoji) for token, emoji in zip(df['tokens_es*'], df.emojis_es)]
acc = nbayes(X_tokens_emojis_es, target)
```

4.4

nbayes

accuracy 0.48514851485148514



4.5 4.6

4.7 Comprobamos que al añadir las descripciones de los emoticonos a los tweets no se produce ningún cambio con respecto a al procesamiento anterior. Aunque, como habíamos visto anteriormente un 60% de los emoticonos se encontraban en tweets positivos, también descubrimos que no había un emoticono común a una categoría, sino que todos se cruzaban indistintamente en las diferentes categorías.

4.8

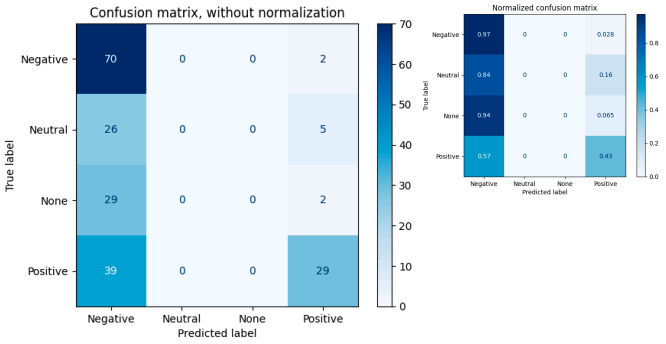
_5 Entrenamiento del modelo con tokens, jaja normalizados

Otro de los cambios que podría producir una mejora es la normalización de la onomatopeya común jaja y sus variantes, llevada a cabo en el apartado 4.

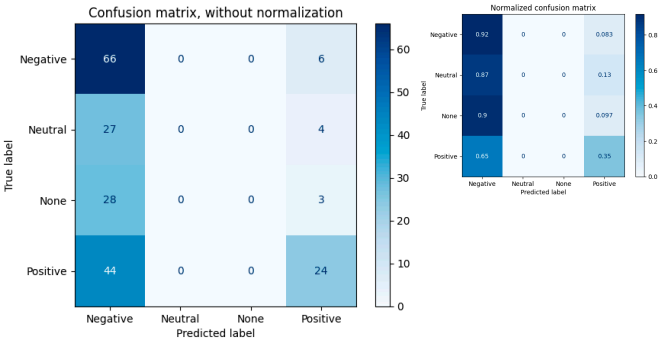
```
X_tokenized_jaja_es = [' '.join(token) for token in list(df.tokens_jaja_es)]; acc = nbayes(X_tokenized_jaja_es, target)
X_tokenized_jaja_en = [' '.join(token) for token in list(df.tokens_jaja_en)]; acc = nbayes(X_tokenized_jaja_en, target)
```

nbayes

accuracy 0.4900990099009901



accuracy 0.44554455445544555



Tras la eliminación de símbolos y stopwords, exluyendo las false stopwords, y la normalización de la risa en los tweets. conseguimos aumentar el porcentaje de acierto al 0.49%. Esto es una leve mejoría ya que la clasificación solo ha mejorado acertando un tweet positivo.

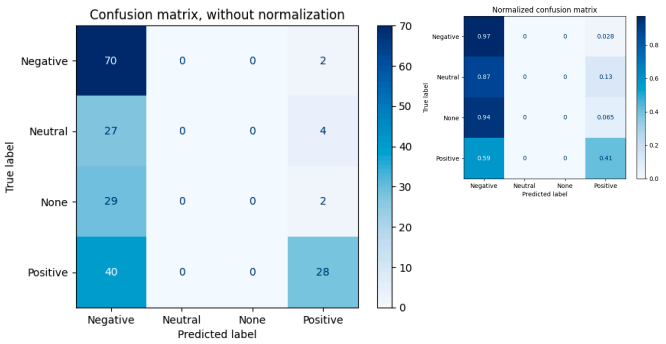
_6 Entrenamiento del modelo con tokens, jaja normalizados y hashtags

Finalmente vamos a unificar todas las mejoras de procesamiento que hemos ido haciendo, unido a los hashtags separados a palabras cuando ha sido posible.

```
X_tokens_emojis_es = [' '.join(token) + ' ' + ' '.join(emoji) for token, emoji in zip(df['tokens_es*'], df.emojis_es)]
acc = nbayes(X_tokens_emojis_es, target)
```

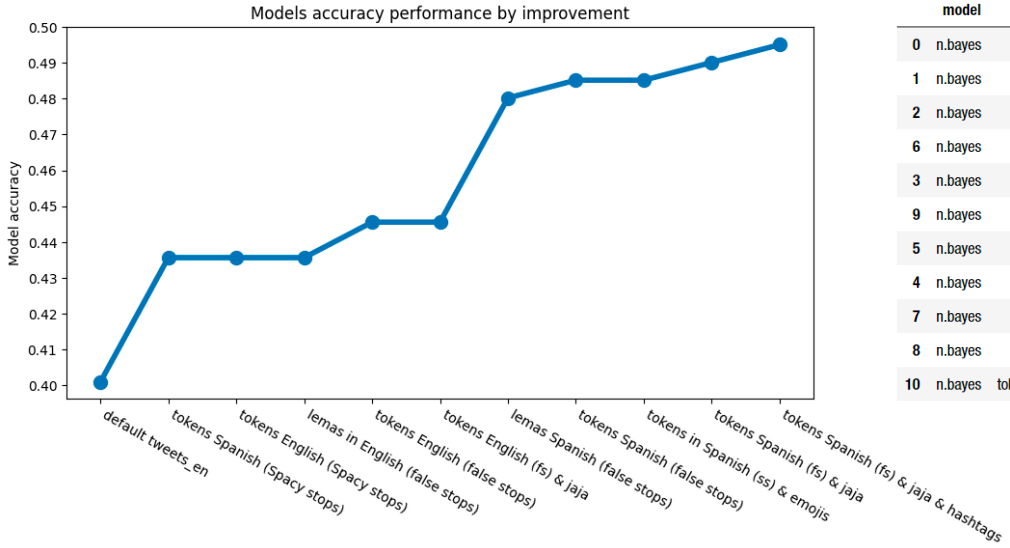
nbayes

accuracy 0.48514851485148514



Comprobamos que al añadir las descripciones de los emoticonos a los tweets no se produce ningún cambio con respecto a al procesamiento anterior. Aunque, como habíamos visto anteriormente un 60% de los emoticonos se encontraban en tweets positivos, también descubrimos que no había un emoticono común a una categoría, sino que todos se cruzaban indistintamente en las diferentes categorías.

_9 Cuador resumen de las mejoras en la predicción del modelo



	model	improve	accuracy
0	n.bayes	default tweets_en	0.400990
1	n.bayes	tokens Spanish (Spacy stops)	0.435644
2	n.bayes	tokens English (Spacy stops)	0.435644
6	n.bayes	lemas in English (false stops)	0.435644
3	n.bayes	tokens English (false stops)	0.445545
9	n.bayes	tokens English (fs) & jaja	0.445545
5	n.bayes	lemas Spanish (false stops)	0.480198
4	n.bayes	tokens Spanish (false stops)	0.485149
7	n.bayes	tokens in Spanish (ss) & emojis	0.485149
8	n.bayes	tokens Spanish (fs) & jaja	0.490099
10	n.bayes	tokens Spanish (fs) & jaja & hashtags	0.495050

En la figura resumen con los resultados de todos los entrenamientos llevados a cabo podemos ver que mejora ha supuesto cada uno. Si nos centramos en los grandes cambios, lo que más ha afectado al modelo de clasificación Naive Bayes ha sido en primer lugar la eliminación de simbolos y tokens y en segundo lugar el filtrado de stopwords dejando algunas de las que venían por defecto en Spacy.

4.7 Entrenamiento del modelo SVM

Dado que el objetivo principal de la práctica era principalmente el procesamiento de los tweets no vamos a extendernos como lo hemos hecho en Naive Bayes con SVM. De esta manera presentaremos el cuadro resumen de resultados similar al presentado con el modelo bayesiando manteniendo los mismo experimentos.

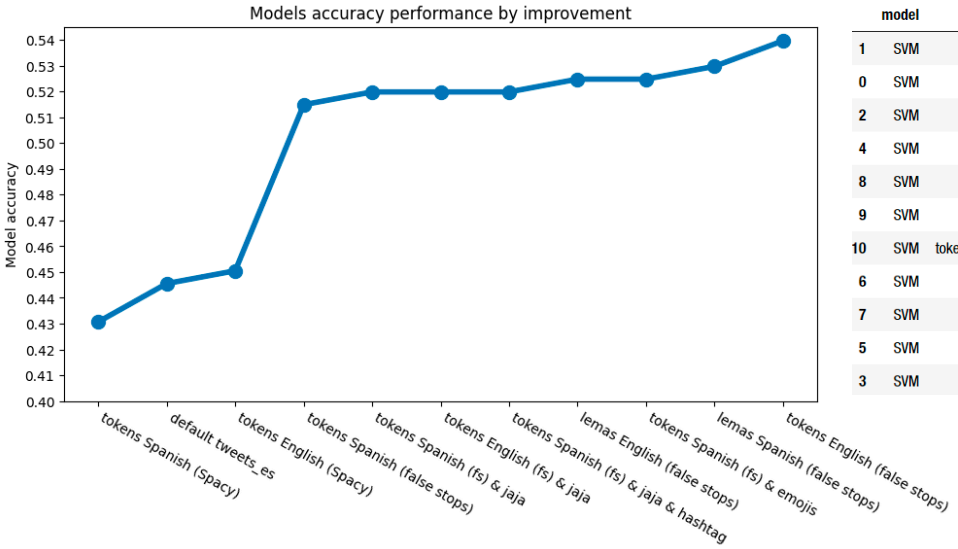
Cuador resumen de las mejoras en la predicción del modelo

Para la realización del modelo SVM se ha aplicado en cada uno de los casos una búsqueda del kernel sobre un grid de parámetros que permitan la búsqueda del mejor resultado de entrenamiento en cada caso. Estos parametros se pueden consultar en la tabla de resultados.

Selección de muestra de entrada para el modelo

```
nlps = [tweets_es, X_tokens, X_tokenized_en, X_tokenized_en_2, X_tokenized_es_2, X_lematized_es, X_lematized_
en, X_tokens_emojis_es, X_tokenized_jaja_es, X_tokenized_jaja_es, X_tokenized_jaja_hash_es]
for nlp in nlps:
    acc = svm_model(nlp, target)
```

svm_model



	model	improve	accuracy	parameters
1	SVM	tokens Spanish (Spacy)	0.430693	(rbf, 1000, 0.0001)
0	SVM	default tweets_es	0.445545	(rbf, 1000, 0.0001)
2	SVM	tokens English (Spacy)	0.450495	(rbf, 1000, 0.0001)
4	SVM	tokens Spanish (false stops)	0.514851	(rbf, 100, 0.001)
8	SVM	tokens Spanish (fs) & jaja	0.519802	(rbf, 100, 0.001)
9	SVM	tokens English (fs) & jaja	0.519802	(rbf, 100, 0.001)
10	SVM	tokens Spanish (fs) & jaja & hashtag	0.519802	(rbf, 100, 0.001)
6	SVM	lemas English (false stops)	0.524752	(rbf, 1000, 0.0001)
7	SVM	tokens Spanish (fs) & emojis	0.524752	(rbf, 100, 0.001)
5	SVM	lemas Spanish (false stops)	0.529703	(rbf, 100, 0.001)
3	SVM	tokens English (false stops)	0.539604	(rbf, 100, 0.001)

4.8 Conclusión

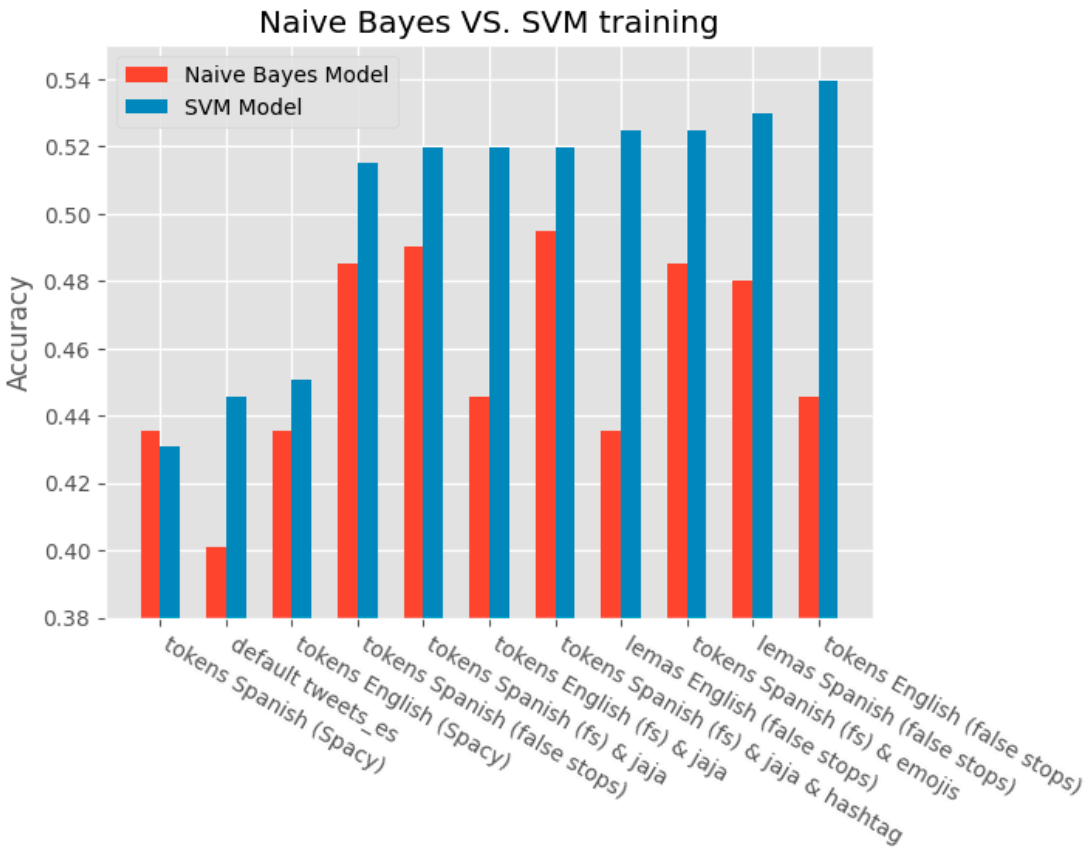
Curiosamente si nos centramos en el mejor entrenamiento de cada modelo, los mismos procesamientos realizados les afectan de diferente manera. Esto se puede ver claramente en los resultados, mientras el mejor entrenamiento del **clasificador bayesiano** ha sido con los **tokens en español, jaja normalizados y hashtags**; en el clasificador **SVM** curiosamente el mejor entrenamiento ha sido el de los **tokens en inglés**.

Para poder explicar esta diferencia conviene entender bien el funcionamiento de los dos modelos, a priori el modelo Naive Bayes funciona por una matriz de frecuencias y asume cada variable como independiente, esto permite simplificar mucho los cálculos ya que cada distribución se puede estimar de forma independiente sin verse afectado por el efecto Huges.

Por otro lado un modelo Support Vector Machine funciona añadiendo nuevas dimensiones a los vectores originales, clasificando las muestras mediante un hiperplano en un espacio de nivel superior. Esto permite aplicar ecuaciones lineales, polinomiales o gausianas en un espacio de dimensión superior que favorecen segun el teorema de Mercer la probabilidad de encontrar una clasificación más precisa.

Atendiendo a estos aspectos posiblemente el clasificador bayesiano obtuvo su mejor resultado donde había más varianza ya que su ventaja es la independencia de variables, en cambio, el clasificador SVM se ve favorecido por una normalización mayor de lemas obteniendo mayores repeticiones y contando con que para este clasificador las variables si pueden ser dependientes unas de otras. Este último párrafo es tan solo una suposición habría que seguir realizando pruebas a los resultados para comprobar esta posibilidad.

Finalmente se muestra una tabla donde se puede ver esa comparativa de resultados en los dos modelos propuestos.



ANEXO: LIBRERÍAS Y FUNCIONES USADAS

Librerías usadas para la práctica

```
import spacy
from googletrans import Translator
import pandas as pd
import random as rd
import time as t
from sklearn.datasets import load_files
import numpy as np
import re
from googletrans import Translator
import time
import json
import matplotlib.pyplot as plt
```

Funciones NLP usadas para la práctica

Funciones NLP

```
def get_stops(text, nlp):
    nlp.max_length = 5000000
    text = text.lower()
    # stops generator
    doc = nlp(text)
    stops = [t.text for t in doc if t.is_stop]
    return stops

def get_tokens(text, nlp):
    nlp.max_length = 5000000
    text = text.lower()
    # tokens generator
    doc = nlp(text, disable=['parser', 'ner'])
    tokens = [t.text for t in doc if
            (t.is_stop is False and t.is_punct is False and not t.text.isspace())]
    tokens = list(set(tokens))
    return tokens

def get_lemas(text, nlp):
    nlp.max_length = 5000000
    text = text.lower()
    # lemas generator
    doc = nlp(text, disable=['parser', 'ner'])
    tokens = [t.lemma_ for t in doc if
            (t.is_stop is False and t.is_punct is False and not t.text.isspace())]
    tokens = list(set(tokens))
    return tokens
```

Modelos Naive Bayes y SVM

Función para el modelo Naive Bayes

```
def nbayes(X, Y):
    plt.style.use('default')
    labels = ['Negative', 'Neutral', 'None', 'Positive']
    def basic_processing(X):
        documents = []
        for sen in range(0, len(X)):
            document = str(X[sen])
            documents.append(document)
        return documents
    documents = basic_processing(X)
    print('Documents processed', '\n')
    vectorizer = CountVectorizer()
    X = vectorizer.fit_transform(documents).toarray()
    tfidfconverter = TfidfTransformer()
    X = tfidfconverter.fit_transform(X).toarray()
    # The data is divided into 20% test set and 80% training set.
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
    # Training the model
    clf = MultinomialNB().fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    return accuracy_score(y_test, y_pred)
```

Función para el modelo SVM

```
from sklearn.svm import SVC
def svm_model(X, Y):
    labels = ['Negative', 'Neutral', 'None', 'Positive']
    def basic_processing(X):
        documents = []
        for sen in range(0, len(X)):
            document = str(X[sen])
            documents.append(document)
        return documents
    documents = basic_processing(X)
    print('Documents processed', '\n')
    vectorizer = CountVectorizer()
    X = vectorizer.fit_transform(documents).toarray()
    # The data is divided into 20% test set and 80% training set.
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
    # Create the parameter grid based on the results of random search
    params_grid = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                    'C': [1, 10, 100, 1000]},
                   {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
    clf = GridSearchCV(SVC(decision_function_shape='ovr'), params_grid, cv=5)
    final_model = clf.fit(X_train, y_train)
    y_pred = final_model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    return acc
```