

# Clasificación reviews Amazon

Julen Rodríguez, Carlos Ruiz y Axel Kamp

# Índice

Introducción.....	3
Nuestras reviews.....	3
Construcción de los Datasets .....	4
Preprocesamiento de textos .....	4
Tokenización.....	4
Palabras .....	5
Byte-Pair Encoding (BPE) .....	5
Bigram .....	5
Wordpiece .....	5
Codificación numérica .....	6
Métodos a utilizar .....	6
Discusión resultados obtenidos.....	6
Distribución tokens total .....	6
Distribución tokens por clases .....	8
Conclusiones .....	9
División dataset Train/Validación/Test .....	9
Modelos de aprendizaje .....	9
NaiveBayes Multinomial.....	10
Regresión Logística .....	10
Red Neuronal .....	11
SVM.....	12
Ensembles – GradientBoosting .....	13
Regresión – GradientBoosting.....	13
Conclusión .....	14

# Introducción

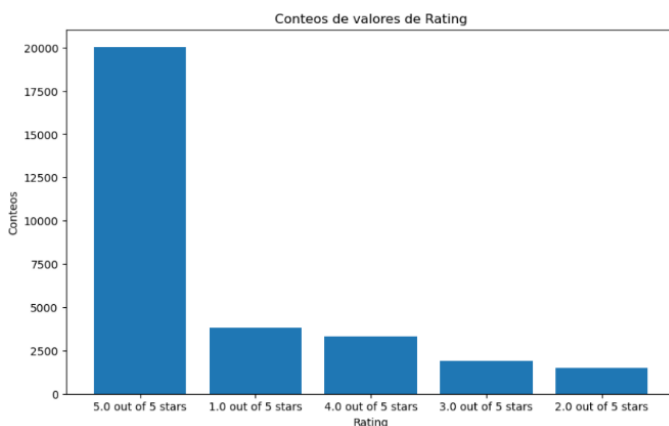
El problema trata sobre la realización de un programa el cual puede averiguar, únicamente sabiendo el comentario de la review, cuál es su rating. Para ello hemos tenido que hacer una búsqueda en la plataforma de Amazon sobre varios productos de una misma clase, para poder obtener sus reviews. Lo perfecto hubiera sido encontrar productos que tuvieran un balanceo de comentarios en base sus ratings pero como esto es la vida real y eso es muy complicado ya que si un producto tiene malas reviews, “normalmente” no se compra. Después de esta selección adjuntamos todas nuestras reviews y le aplicamos diferentes métodos de limpiezas de textos para que no nos cause problemas en un futuro. Luego realizamos un tokenizado diferente al de Bag of Words para ver si vemos resultados más prometedores como puede ser Byte-Pair Encoding por ejemplo. Luego aplicaremos Bag of Words para pasar de tener Strings a tener un Dataset formado por números. Por último, veremos qué resultados nos proporcionan los diferentes clasificadores que obtengamos y si nos clasifica bien una review que no tenga rating.

Esto podría ser una breve descripción de que hay que realizar en este trabajo, pero a continuación vamos a mostraros la manera en la que nosotros hemos afrontado el problema y los diferentes resultados que hemos obtenidos en todo. También mostraremos resultados de los procesos en diferentes apartados y porque es importante realizar esos procesos.

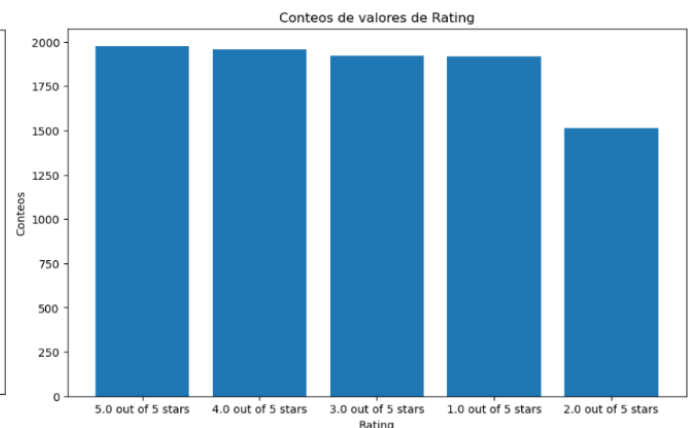
## Nuestras reviews

Nosotros decidimos hacer el trabajo en base cables de tipo USB y C ya que pensamos que al ser elementos tan cotidianos del día a día veríamos más tipos de reviews, pero más o menos los objetos que tenían muchas reviews normalmente solían estar dentro de una media de rating de 4,2. Sabiendo esto, decidimos coger 12 objetos de este tipo y nos encargáramos nosotros de realizar el balance de reviews. Lo primero que hicimos fue leer los URLs de los productos y poner cada review en un csv, pero nuestra siguiente tarea era realizar el balance.

Para realizar este balance medimos nosotros los porcentajes y en vez de borrar reviews, cogimos el csv de datos donde aplicamos los porcentajes calculados previamente y elegimos aleatoriamente por producto el número determinado de reviews que teníamos que poner a cada clase. Después de realizar este proceso manual para balancear los datos por clases obtenemos lo siguiente:



*Disposición de la cantidad de datos por clases sin realizar el balanceo.*



*Disposición de los datos por clases balanceada*

# Construcción de los Datasets

## Preprocesamiento de textos

En esta parte, vamos a comentar varios apartados que están dentro de la construcción de los datasets. La primera de estas se denomina **“Limpieza de textos”** de limpiar caracteres non-ascii, limpiar contracciones y más cosas que comentaremos a continuación. Primero os vamos a mostrar como una review evoluciona mostrando también los pasos realizados. Este es el comentario original:

*“At first we liked these. but after only 4 months, they were fraying and 1 month later, they no longer are fast charging. 🙄”*

Ahora os mostramos los 7 procesos realizadas dentro de esta parte:

1. Poner todo en **minúsculas**: lo realizamos con gracias al ‘lower ()’.
2. Tratar las **contracciones** como “there’s” para que ese “ ’ ” no nos cree problemas, esto lo realizamos con la librería contractions.
3. Eliminar **signos de puntuación y números**.
4. Eliminar **caracteres non-ascii** ya que queremos trabajar con el lenguaje Ingles que está dentro del rango Ascii.
5. Eliminamos palabras sobrantes con un proceso llamado **“stop-words removal”**.

Hasta aquí vamos a mostrar cómo ha evolucionado nuestra review:

*“first liked months fraying month later longer fast charging”*

Los últimos pasos para realizar son los siguiente:

6. Realizamos **Lemmitazitation o Porter Stemmer**, si usamos una u otra obtenemos diferentes salidas:

Lemmitazitation → *“first liked month fraying month later longer fast charging”*

Porter Stemmer → *“first like month fray month later longer fast charg”*

7. Por último, **eliminamos posibles ejemplos que nos queden vacíos** después de realizar todos estos cambios. A nosotros nos han salido 16 ejemplos vacíos así que nos disponemos a eliminarlos para que no nos dañe nuestro proceso.

Todos estos procesos son los que hemos realizado para dejar nuestras reviews listas para aplicarles lo siguiente y conseguir finalmente nuestro datasets numérico.

## Tokenización

El siguiente proceso que vamos a realizar se denomina **“Tokenizer”**. Este proceso se basa en obtener un número determinado de tokens para poder estudiar los textos con ellos y existen muchos tipos de estos para obtener nuestros tokens. Nosotros lo hemos planteado de la manera la cual introducimos que tipo de tokenizer vamos a usar, el data de los textos y el número de tokenizers

que vamos a generar. Lo que obtenemos es un diccionario ordenado descendientemente donde la KEY es el token y el valor es la cantidad de veces que aparece ese token dentro de la clase. A continuación, os vamos a mostrar que tokenizers hemos usado para solventar nuestro problema:

## Palabras

El primero de ellos realiza una tokenización el cual nos genera en este caso tokens igual a **palabras**, un ejemplo sería este:

*"I like very much play football"* → ["I"," like","very","much","play","football"]

Este modelo es muy simple y es el más básico ya que lo único que devuelve es un string de caracteres cuyo contenido son las palabras. Pero no por ello es menos importante y lo hemos elegido para en un futuro ver si los resultados podrían variar.

## Byte-Pair Encoding (BPE)

El siguiente se llama **Byte-Pair Encoding** el cual trabaja diferente que el primero. Las diferencias son que en este separamos por letras la frase, y de ahí por cada iteración va juntando caracteres hasta ver los que más se repiten, por ejemplo:

*"I play football"*

*"I have been playing tennis"*

Con estas dos frases primero obtendríamos todas las letras separadas y luego vemos que la "I\_" se repite, luego que también vemos que "play" se repite 2 veces... Es decir, nos capta la secuencia de caracteres que más veces se repiten. Elegimos contar con ella ya que, al ser diferente, puede que en los resultados veamos cosas que nos gusten.

## Bigram

La tercera se llama **Bigram** que es muy semejante al primero que hemos comentado también nos realiza una división de tokens por palabras, pero no del todo. En una de ellas se realiza una división íntegra de la palabra, pero en la de Bigram, la tokenización también puede ser en subcadenas o en caracteres como, por ejemplo:

*"catlike"*

Palabras → ["catlike"]

Unigram → ["cat"," like"]

## Wordpiece

El último se llama **WordPiece**, donde aquí si hace subsecuencias de palabras, nos muestra con un # que es consecuencia de otra sub-cadena.

En resumidas cuentas, hemos usado estos 4 tokenizadores para ver que obtenemos si realizamos de diferente manera la disposición de formar el diccionario con el cual nos vamos a apoyar para construir nuestro dataset. Lo único, tendremos que definir cuantos tokens queremos por clase.

# Codificación numérica

Se trata de convertir el corpus de reviews (textos) en una matriz de números, donde cada fila de la matriz representa uno de los textos del corpus y cada columna representa cada una de las palabras (tokens).

## Métodos a utilizar

Los métodos que nos ofrece scikit-learn para la codificación suman 4, nosotros solo realizaremos las pruebas con `CountVectorizer()` y `TfidfVectorizer()`. Hemos descartado `TfidfTransformer()`, porque al ser combinado con `CountVectorizer()` obtenemos `TfidfVectorizer()`. También hemos descartado `HashingVectorizer()` porque no hemos tenido el tiempo de investigar el funcionamiento del mismo.

`CountVectorizer()` solo tiene en cuenta el número de veces que aparece cada palabra a lo largo del texto, denominado "Term Frequency" o  $TF(t)$ , donde  $t$  es la palabra en concreto. `TfidfVectorizer()` realiza un par de modificaciones a `CountVectorizer()`, estas modificaciones son:

- El cálculo de  $IDF(t)$ , se basa en aumentar la ponderación de aquellas palabras que se repiten en pocas reviews. Hace uso de  $df(t)$  para su cálculo,  $df(t)$  es el número de reviews en el que aparece la palabra ( $t$ ) en concreto.
- Normalizar el producto de  $TF(t)$  con  $IDF(t)$ , para dejar todos los valores de la matriz en la misma escala.

Hemos decidido hacer uso de histogramas para teorizar la diferencia de resultados entre `CountVectorizer()` y `TfidfVectorizer()`.

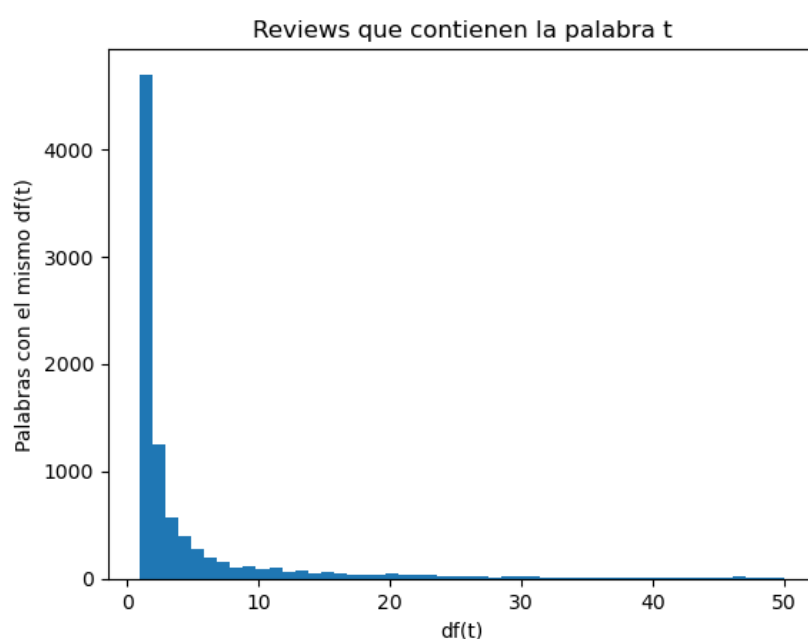
## Discusión resultados obtenidos

### Distribución tokens total

Primero, queremos saber si la diferencia de  $df(t)$  es considerable entre el conjunto de palabras que disponemos, en caso afirmativo, hará que  $IDF(t)$  juegue un rol importante en la codificación. Recordemos que las palabras con un alto  $df(t)$ , tendrán un bajo  $IDF(t)$  ya que es considerado como una palabra que no es determinante al aparecer en muchas reviews.

count	9329.000000
mean	15.102155
std	79.873471
min	1.000000
25%	1.000000
50%	1.000000
75%	5.000000
max	2511.000000

Observamos que la gran mayoría de palabras aparecen en muy pocas reviews, el 75% de las palabras aparecen en 5 reviews o menos, haciendo que la diferencia de  $df(t)$  no sea alta. Aun así, habrá casos en los que  $df(t)$  sea excepcionalmente alto, como el ejemplo con  $df(t)=2511$ , lo que hace que su ponderación sea muy baja.

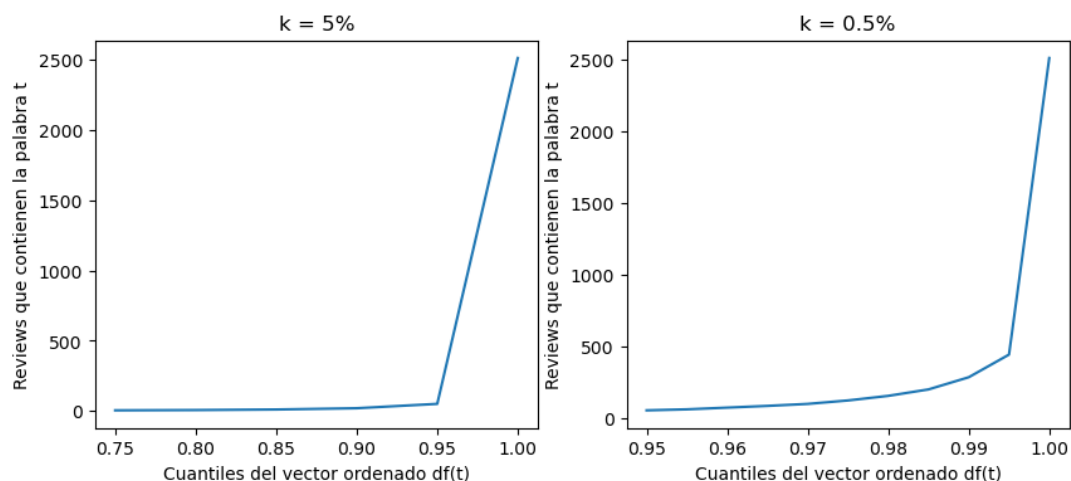


Ya que tanto `CountVectorizer()` como `TfidfVectorizer()` poseen los parámetros `min_df` y `max_df`, vamos a aplicar una técnica improvisada para tratar de averiguar valores coherentes para ambos parámetros. `Min_df` es el número mínimo de reviews en el que debe de aparecer cada palabra y `max_df` el número máximo de reviews.

Al tener al menos el 50% de las palabras en una única review ( $q_2=1$ ), no es conveniente alterar el valor por defecto de `min_df=1` (default) ya que sino habrá una gran cantidad de ceros en la matriz de codificación, dando resultados poco coherentes.

Para el valor de `max_df` entra la técnica improvisada que hemos ideado, se trata de la regla del codo (técnica vista en aprendizaje automático). Es una técnica visual y sigue tres pasos:

- Ordenamos todos los valores de  $df(t)$ , tal que:  $df(t_1) \leq df(t_2) \leq \dots \leq df(t_n)$ .
- Damos saltos de  $k$  en  $k$  sobre el vector ordenado y ordenamos los valores de dichos cuantiles en una lista.
- Por último, graficamos los resultados para elegir el umbral conveniente.



En la primera gráfica empezamos del cuantil correspondiente al 75% del vector y observamos que pega un cambio en el rango [95%-100%], por lo que decidimos analizarlo más a fondo. En la segunda gráfica (la de la derecha) realizamos los saltos de 0.5% en 0.5% para mirar con detenimiento el codo, observamos que esta al final, así que buscamos un valor de `max_df` para descartar ese conjunto de palabras. Ya que son abundantes respecto al resto de palabras, no resultan ser relevantes en la clasificación.

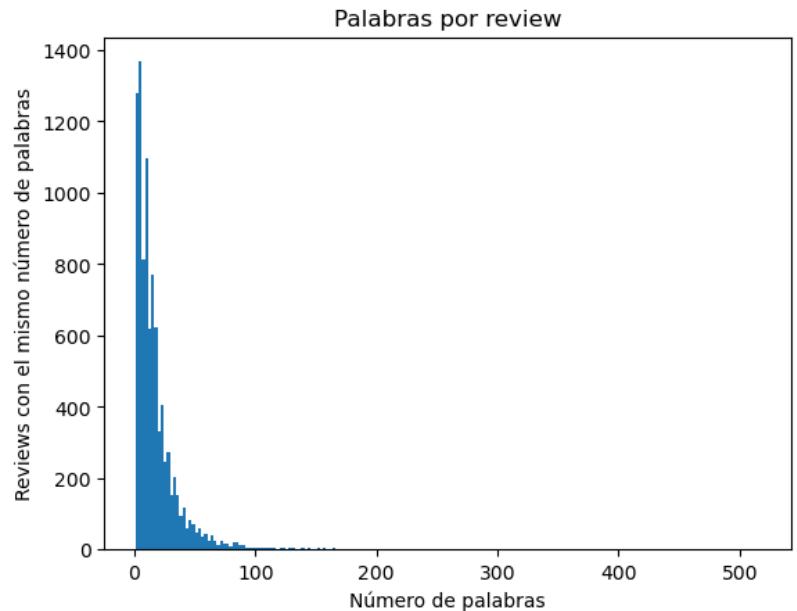
El valor que hemos decidido probar es 0.05, ya que al multiplicar el número de palabras por 0.05 obtenemos un valor que separa los últimos cuantiles, es decir, donde se encuentra el codo.

## Distribución tokens por clases

Segundo, queremos saber si la cantidad de palabras entre las diferentes reviews es notoria o no. En caso afirmativo, esto hará que la normalización juegue un rol importante en la codificación, ya que el valor del denominador al normalizar viene condicionado por el número de palabras (sumandos) y la frecuencia de aparición de cada palabra.

```
count    9269.000000
mean      17.683245
std       21.792920
min        1.000000
25%        6.000000
50%       12.000000
75%       22.000000
max      517.000000
```

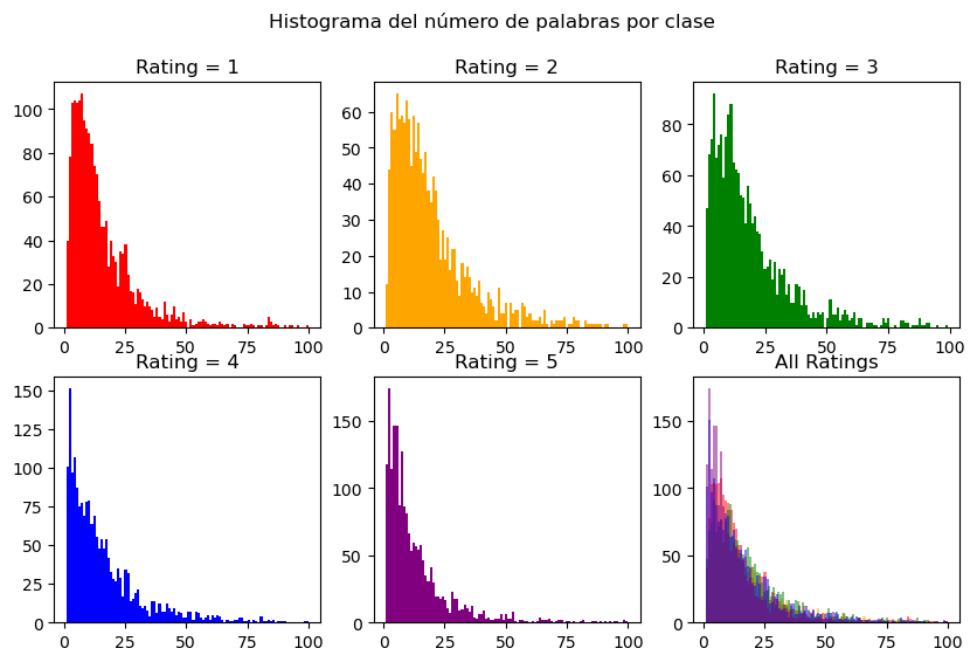
Observamos que la distribución también es agrupada en un rango pequeño, la gran mayoría de reviews poseen 100 palabras o menos. Pero sigue habiendo excepciones, donde hay reviews que llegan hasta 517 palabras, haciendo que los valores tengan una ponderación muy baja.



Aparte de ello, quiero comprobar si existe alguna dependencia entre la valoración impuesta y la cantidad de palabras en una review, para esto graficaré los histogramas de las cinco clases (tipos de valoraciones) junto a una gráfica que solape las cinco.

Como podemos ver, no existe dependencia notoria entre el número de palabras por review y la valoración impuesta.

También observamos que las reviews con rating 2 y 3 tienen valores más bajos en el eje y ¿Esto que significa? Teniendo en cuenta que el número de reviews por clase están balanceados, significa que hay menos coincidencias en el número de palabras en dichas reviews, es decir, más variedad, siguiendo así una distribución más uniforme.







## Conclusiones

Finalmente, vamos a realizar las pruebas de rendimiento con los dos métodos de preprocesamiento de textos: Lemmatizer y Porter Stemmer, junto a los tokenizers: Words, Byte-Pair Encoding, Unigram y WordPiece. El modelo usado es el de Naive Bayes Multinomial, ya que no requiere de hiperparametros y se basa en estadística.

Estos han sido los resultados:

	Palabras	BPE	Wordpiece	Bigram
LEMMATIZATION	42,23%	42,88%	42,13%	41,96%
PORTER STEMMER	40,83%	41,15%	40,99%	41,64%
LEMMATIZATION	43,37%	<b>43,64%</b>	43,04%	43,42%
PORTER STEMMER	42,34%	42,23%	41,96%	42,18%

 -> CountVectorizer()  
 -> TfidfVectorizer()

Tras los histogramas obtenidos, hemos observado que la mayoría de datos se centran en un rango reducido, haciendo que TfidfVectorizer() no altere mucho los valores de la matriz resultante de CountVectorizer(). Lo que se ve, es una mejora en los resultados, esto indica que tener en cuenta la frecuencia de aparición de las palabras en las distintas reviews impacta positivamente en el rendimiento de la clasificación.

## División dataset Train/Validación/Test

Después de crear el dataset numérico tenemos que realizar las particiones para poder trabajar con ello. Para eso vamos a realizar tres divisiones que son el Train, el Test y la Validación. Lo primero que hacemos es dividir el dataset para obtener el test y después de todo esto realizamos un Stratified Shuffle Split con la parte del “train” para de ahí sacar la parte del train y del val.

Gracias a este método de división podemos realizar todas las pruebas que queramos con el train y el val para obtener los mejores hiperparametros de nuestro proceso y después de realizar todo este proceso podremos decir que finalmente hemos obtenido nuestros datasets para poder trabajar con ellos y así obtener resultados. De estos resultados sacaremos nuestras conclusiones.

## Modelos de aprendizaje

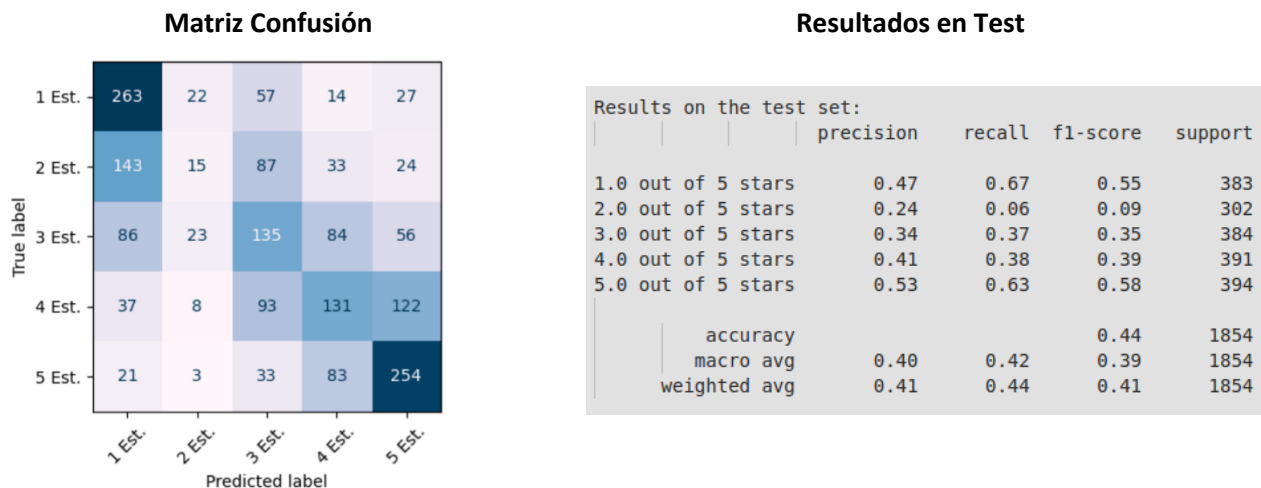
Una vez seleccionado nuestro método de *vectorización* de textos a número, lo que nos queda en último lugar es utilizar varios modelos de aprendizaje automático para poder realizar una clasificación correcta.

Es importante mencionar que usaremos el *accuracy* como medida de decisión. Por otro lado, el *recall* será un elemento importante a tener en cuenta en el total de nuestros modelos.

Hemos decidido utilizar 5 modelos distintos, siendo uno de ellos el **NaiveBayes Multinomial** utilizado anteriormente. Hemos optado por realizar una búsqueda con la función **GridSearchCV** la mejor combinación de hiper-parametros. Ha continuación se van a exponer todos ellos con una pequeña explicación de porque se ha decidido su elección, como se han utilizado y los datos obtenidos. Todos los resultados obtenidos (matriz de confusión y datos varios) han sido obtenidos con el conjunto de Test realizado anteriormente.

## NaiveBayes Multinomial

Como hemos explicado anteriormente y sin entrar más en detalles, este modelo al basarse puramente en porcentaje de apariciones de *tokens*, es muy útil para nuestro problema de clasificación. Hemos visto que el **accuracy** obtenido es un **43.64%**. Mas ampliamente podemos observar los siguientes datos:



No se han tomado en cuenta ningún tipo de hiper-parametro. Mantenemos el resultado obtenido en la prueba anterior.

Como ya se ha mencionado anteriormente, el porcentaje de acierto se acerca ligeramente al 50%. Observamos que no es el mejor modelo de todos, pero sirve en cierta medida. En cuanto a todos los datos obtenidos, es bastante interesante fijarse en el *recall* que ha obtenido cada una de las clases. Observamos una clara tendencia a obtener un **mayor acierto** en las **clases extremas**, siendo estas la de *1 Estrella* y la de *5 Estrellas*, un acierto medio-bajo en las clases intermedias y un inesperadamente bajo acierto en la clase 2. Nos damos cuenta de que en cuanto a esta última clase la gran mayoría de ejemplos han sido clasificados erróneamente. Este dato es sospechosamente inferior a los demás, por lo que es un dato a tener en cuenta respecto a los resultados obtenidos con los siguientes modelos.

Todo lo mencionado se puede observar en la propia matriz de confusión, siendo que la mayoría de los ejemplos de la clase 2 se han clasificado como clase 1, mayoritariamente, o como las demás.

## Regresión Logística

En segundo lugar, vamos a utilizar el modelo de la *Regresión Logística*. Esta utiliza una regresión para predecir mediante porcentajes a que clase pertenece cada uno de los ejemplos introducidos a predecir. Los hiper-parametros usados, asi como los mejores, y los resultados son los siguientes:

Hiper-parametros				Ganador
Penalty	l2      None			l2
C	1e+101	1	1e-101	1
Solver	newton-cg	sag	saga	newton-cg
Multi_class	auto	ovr	multinomial	ovr

**Matriz Confusión**

1 Est.	232	43	52	22	34
2 Est.	112	48	74	33	35
3 Est.	71	44	124	86	59
4 Est.	29	21	86	132	123
5 Est.	19	5	30	80	260
	1 Est.	2 Est.	3 Est.	4 Est.	5 Est.

**Resultados en Test**

Results on the test set:				
	precision	recall	f1-score	support
1.0 out of 5 stars	0.50	0.61	0.55	383
2.0 out of 5 stars	0.30	0.16	0.21	302
3.0 out of 5 stars	0.34	0.32	0.33	384
4.0 out of 5 stars	0.37	0.34	0.35	391
5.0 out of 5 stars	0.51	0.66	0.57	394
accuracy			0.43	1854
macro avg	0.40	0.42	0.40	1854
weighted avg	0.41	0.43	0.41	1854

El **accuracy** obtenido es de un **42.93%**. Este es sorprendentemente parecido al obtenido con el modelo *NaiveBayes*. Igualmente llegamos a, prácticamente, los mismos resultados obtenidos en cuanto al **recall**. La diferencia más notable es el aumento del resultado obtenido en cuanto al obtenido en la **clase 2** en esta última variable. Existe un **aumento del 266.67%** respecto al obtenido anteriormente. Es un aumento relativamente grande, pero el valor obtenido, 16%, sigue siendo despreciable respecto a las demás clases. Siguen existiendo sospechas respecto a esta clase.

Observando la matriz, existe la misma tendencia que la anterior, pero mejorando lo obtenido en la clase 2, teniendo así sentido los resultados del *recall* del modelo.

## Red Neuronal

Mediante este modelo, intentaremos montar una red neuronal que aprenda de las matrices de apariciones mediante neuronas interconectadas como clasificar los diversos ejemplos. Los hiper-parametros usados, así como los mejores, y los resultados son los siguientes:

**Hiper-parametros**

**Ganador**

<b>Hidden_layer_sizes</b>	(15, 30)	(50, 50)	(25, 75)	(50, 50)
<b>Solver</b>	sgd	adam		adam
<b>Alpha</b>	0.1	0.2	0.35	0.2
<b>Learning_rate</b>	constant	adaptative		constant

**Matriz Confusión**

1 Est.	186	85	59	30	23
2 Est.	89	76	66	47	24
3 Est.	62	71	112	97	42
4 Est.	28	33	76	172	82
5 Est.	19	17	31	135	192
	1 Est.	2 Est.	3 Est.	4 Est.	5 Est.

**Resultados en Test**

Results on the test set:				
	precision	recall	f1-score	support
1.0 out of 5 stars	0.48	0.49	0.49	383
2.0 out of 5 stars	0.27	0.25	0.26	302
3.0 out of 5 stars	0.33	0.29	0.31	384
4.0 out of 5 stars	0.36	0.44	0.39	391
5.0 out of 5 stars	0.53	0.49	0.51	394
accuracy			0.40	1854
macro avg	0.39	0.39	0.39	1854
weighted avg	0.40	0.40	0.40	1854

En este caso, nos podemos dar cuenta de algo interesante. Efectivamente, nuestro porcentaje de **accuracy** baja, siendo en esta ocasión un **39.81%**. Una vez más obtenemos un dato que no supera la mitad de aciertos. Lo curioso aparece en cuanto observamos la matriz de confusión. Hemos logrado predecir de manera más precisa las clases *intermedias* sacrificando las clases *extremas*. Sigue existiendo un problema claro con la clase 2 ya que se sigue clasificando principalmente como clase 1. Todo esto también se puede observar en lo obtenido en cuanto al *recall*. Se consigue, una vez más, un aumento considerable en aciertos en cuanto a la clase 2. En este caso, un **156.25%** más respecto a la *Regresión Logística*. A su vez, se consiguen mejores resultados en cuanto a la clase 4, pero un poco peores en la clase 3. Sin ningún tipo de duda, se puede observar el claro descenso de aciertos en las clases 1 y 5.

Podemos de cierta manera dictaminar que existiría cierta **balanza**. Podemos obtener más predicciones de las clases que no sean las de 1 o 5 estrellas, logrando de esta manera no ser tan *extremos*. Sin embargo, esto no ayudaría a obtener un mejor *accuracy*.

## SVM

*SVM* se trata de un modelo el cual, a rasgos generales, es una mejora de la *Regresión Logística*, por lo que debería de mejorar los resultados obtenidos. Los hiper-parametros usados, así como los mejores, y los resultados son los siguientes:

Hiper-parametros	Ganador		
<b>C</b>	0.5	1	2
<b>Kernel</b>	linear	rbf	sigmoid
<b>Gamma</b>	3	5	10
<b>Decision_function_shape</b>	ovo	ovr	ovo

Matriz Confusión

1 Est.	226	36	63	37	21
2 Est.	112	49	77	40	24
3 Est.	65	35	131	116	37
4 Est.	21	17	93	166	94
5 Est.	16	5	32	125	216
	1 Est.	2 Est.	3 Est.	4 Est.	5 Est.

Resultados en Test

Results on the test set:				
	precision	recall	f1-score	support
1.0 out of 5 stars	0.51	0.59	0.55	383
2.0 out of 5 stars	0.35	0.16	0.22	302
3.0 out of 5 stars	0.33	0.34	0.34	384
4.0 out of 5 stars	0.34	0.42	0.38	391
5.0 out of 5 stars	0.55	0.55	0.55	394
accuracy			0.43	1854
macro avg	0.42	0.41	0.41	1854
weighted avg	0.42	0.43	0.42	1854

El valor resultante es de **42.5%** de *accuracy*. Se acaban obteniendo prácticamente los mismos resultados que en la *Regresión Logística*. Al ser una “mejora” del mismo, se han intentado estabilizar más las clases 2, 3 y 4, perdiendo valor en el *recall* en las clases 1 y 5, dando más pistas de que efectivamente existe una especie de “balanza”. Se ha acabado obteniendo como resultado que el mejor **kernel** posible sea el **lineal**. Esto se puede deber a diversas razones, pero según lo que hemos podido ir viendo en los resultados hasta ahora, existe cierto solapamiento entre las clases *intermedias* por lo que lo más probable es que si se usara el *kernel rbf* o *sigmoid*, se acabaría obteniendo cierto sobre entrenamiento, dando por sí resultados peores en cuanto a la validación del modelo.

## Ensembles – GradientBoosting

Por último, vamos a utilizar como *Ensemble* el *GradientBoosting*. Al momento de construir los árboles secuenciales, este trata de corregir los errores cometidos por el anterior. De esta manera se intenta conseguir ser mucho mas preciso que otros tipos de *Ensembles*.

Hiper-parametros	Ganador			
Learning_rate	0.05	0.1	1	1
N_estimators	50	100	500	500
Min_samples_split	2	5	10	2
Min_samples_leaf	1	2	5	10
Max_features	auto	sqrt	log2	None

**Matriz Confusión**

1 Est.	204	63	49	22	45
2 Est.	94	72	67	37	32
3 Est.	64	61	109	88	62
4 Est.	21	29	74	126	141
5 Est.	16	11	37	79	251
	1 Est.	2 Est.	3 Est.	4 Est.	5 Est.

**Resultados en Test**

Results on the test set:	precision	recall	f1-score	support
1.0 out of 5 stars	0.54	0.45	0.49	1913
2.0 out of 5 stars	0.34	0.26	0.30	1512
3.0 out of 5 stars	0.38	0.35	0.36	1918
4.0 out of 5 stars	0.35	0.46	0.40	1955
5.0 out of 5 stars	0.46	0.49	0.48	1971
accuracy			0.41	9269
macro avg	0.41	0.40	0.40	9269
weighted avg	0.42	0.41	0.41	9269

Como último modelo, vemos que obtenemos un total de **41.1% de accuracy**. Si nos fijamos únicamente en este valor, no estamos hablando del mejor modelo de clasificación. En cambio, como hemos ido haciendo a lo largo de los diversos clasificadores, si miramos la variable **recall** nos damos cuenta que *GradientBoosting* ha sido el que mas balanceo de resultados ha habido. Una vez más, y fuera de todo tipo de sorpresas, el problema reside en la clase 2 sobre todo. Se obtienen resultados relativamente parejos las clases *extremas* y en varias *intermedias*, dando a entender que la teoría de la “balanza” empieza decantarse cada vez más a que sería el verdadero problema.

## Regresión – GradientBoosting

Como último modelo, hemos decidido realizar pruebas con un modelo de regresión. Estos tipos de modelos no se deben de usar para un problema como el que estamos planteando, ya que devuelven resultados continuos y nosotros buscamos resultados discretos, ósea clasificación entre las 5 clases. Igualmente, hemos decidido hacer una prueba donde las variables continuas las redondeamos a su valor mas cercano, siendo que las clases sean valores del 1 al 5 (como sus ratings mismos), e intentar averiguar si de esta manera se puede realizar una cierta “clasificación”

Hiper-parametros	Ganador			
Learning_rate	0.05	0.1	1	0.1
N_estimators	50	100	500	500
Min_samples_split	2	5	10	2
Min_samples_leaf	1	2	5	5
Max_features	auto	sqrt	log2	Auto

Para decidir la mejor combinación se ha decidido utilizar **r2score** como método de elección. Una vez teniendo este mismo, siendo un **37.09%**, realizamos la predicción y redondeamos, dando como resultado lo siguiente:

**Matriz Confusión**

1 Est.	41	159	169	14	0
2 Est.	9	107	165	21	0
3 Est.	5	60	254	65	0
4 Est.	0	18	226	135	12
5 Est.	0	3	179	176	36
	1 Est.	2 Est.	3 Est.	4 Est.	5 Est.

Predicted label

**Resultados en Test**

Results on the test set:					
	precision	recall	f1-score	support	
0	0.75	0.11	0.19	383	
1	0.31	0.35	0.33	302	
2	0.26	0.66	0.37	384	
3	0.33	0.35	0.34	391	
4	0.75	0.09	0.16	394	
accuracy			0.31	1854	
macro avg	0.48	0.31	0.28	1854	
weighted avg	0.49	0.31	0.27	1854	

No hay mucho que se pueda comentar de los mismos. Mediante esta simple prueba se ha podido comprobar que utilizar la regresión como clasificación que los resultados obtenidos no son para nada decentes.

## Conclusión

En conclusión, podemos observar que el **accuracy** de los modelos varía siempre entre el **40%-45%**. Este no es el resultado que esperábamos desde el principio de ninguna manera. Si rescatamos varios resultados de lo visto hasta ahora podemos darnos cuenta de diversos factores importantes. Por última vez, tenemos que recaer en el *recall* de los mismos, del cual destacan los siguientes:

Modelo	1 Estrella	2 Estrellas	3 Estrellas	4 Estrellas	5 Estrellas	Accuracy
<b>NB Multinomial</b>	67%	6%	37%	38%	63%	43.64%
<b>GradientBoosting</b>	45%	26%	35%	46%	49%	41.1%

Recordamos que el *recall* trata en saber la cantidad de ejemplos se han clasificado correctamente, y el *accuracy* simplemente si los que se han dicho que son de cierta clase, ciertamente lo sean. Una vez recordado esto, a simple vista se pueden observar los cambios de los modelos. En cuanto al *NB Multinomial*, este intenta obtener el máximo parecido con la realidad, ya que el propio modelo simplemente se basa en porcentaje de apariciones de *tokens* sin tener en cuenta como son las clases en general, consiguiendo de esta manera el valor más alto de *accuracy*. En cambio, el *GradientBoosting* trata de que cada una de las clases tenga unos buenos resultados, al mismo tiempo de obtener un buen valor de *accuracy*.

Sabiendo esto, los resultados obtenidos en el *recall* son bastante claros. Si queremos basarnos únicamente en la aparición de tokens sin un “trabajo previo”, está claro que existe una **clara diferencia** entre la **clase 1 y 5**. Tanto la clase 3 y 4 tienen sus diferencias, pero el **verdadero problema** aparece al tratar con la **clase 2**, como hemos venido prediciendo a lo largo de los diversos modelos. Usando un modelo el cual trata de balancear más los resultados para cada clase, vemos que se ha logrado aumentar un **433.33%** el *recall*, dando muchísimos más ejemplos como clasificados correctamente. Para lograr esto, sin embargo, se ha tenido que sacrificar un alto porcentaje en cuanto a las clases *extremas*, dando una reducción de un **70%** de aciertos originales. El cambio producido en la clase 4 también es notable y la clase 3 ha obtenido resultados

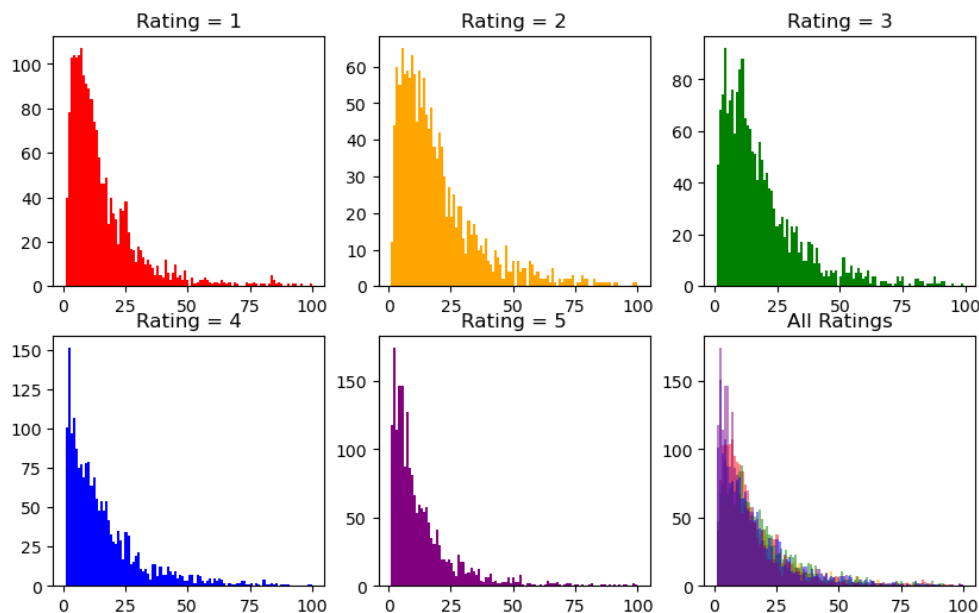
parejos, dando a entender que al ser la clase intermedia esta no va a tener tantos problemas como se esperaría.

Una vez visto todo esto, la teoría de la “balanza” que se ha comentado anteriormente cobra más sentido que antes. Se podría considerar que los resultados obtenidos con el primer modelo se tratarían de un sobre entrenamiento y con el segundo se reduciría el mismo dando más igualdad entre clases.

Finalmente, podemos decir sin atisbo de duda que los modelos obtienen estos resultados gracias a los datos introducidos. Si retrocedemos en los pasos que hemos seguido hasta conseguir los mismos podemos decir las siguientes conclusiones:

- Puede que **no** se haya realizado una **limpieza correcta**. Se hubieran tomado en cuenta, por ejemplo, la posible existencia de reviews en otros idiomas, lo cual afectaría en la creación de *tokens*, o, por el contrario, haber eliminado ciertas características importantes de las propias reviews.
- Si volvemos a observar esta gráfica obtenida en el momento del análisis de los textos ya procesados, podemos volver a dar énfasis a la poca longitud de las reviews en la clase de 2 estrellas, siendo esta vez un problema verdadero.

Histograma del número de palabras por clase



- Por último, esto podría considerarse como un problema del mundo real. Una opinión *extrema* de una persona está claramente mucho más diferenciada ya que se hablan directamente de lo mejor o lo peor del producto usando palabras descriptivas y antónimas como “good” y “bad”. En cambio, las opiniones intermedias, como bien dice la palabra, no se decantan por ningún extremo. Estas usan expresiones neutras, incluso las mismas que se podrían considerar *extremas* siendo así que cuesten mucho más diferenciales. No existe forma objetiva de decidir si una review es de 1 o 2 estrellas, como si lo podría haber para una de 1 o 5 estrellas.