**PROYECTO  
REVIEW**

****

**Alvaro Larraya & Eugen Hamuraru**

**INTRODUCCIÓN:**

El proyecto “Reviews amazon” tiene como objetivo predecir la valoración de una serie de comentarios en base a sus reseñas, las reseñas en este caso eran de libros que podemos encontrar en amazon.

Los objetivos que se buscaban conseguir con este trabajo eran :

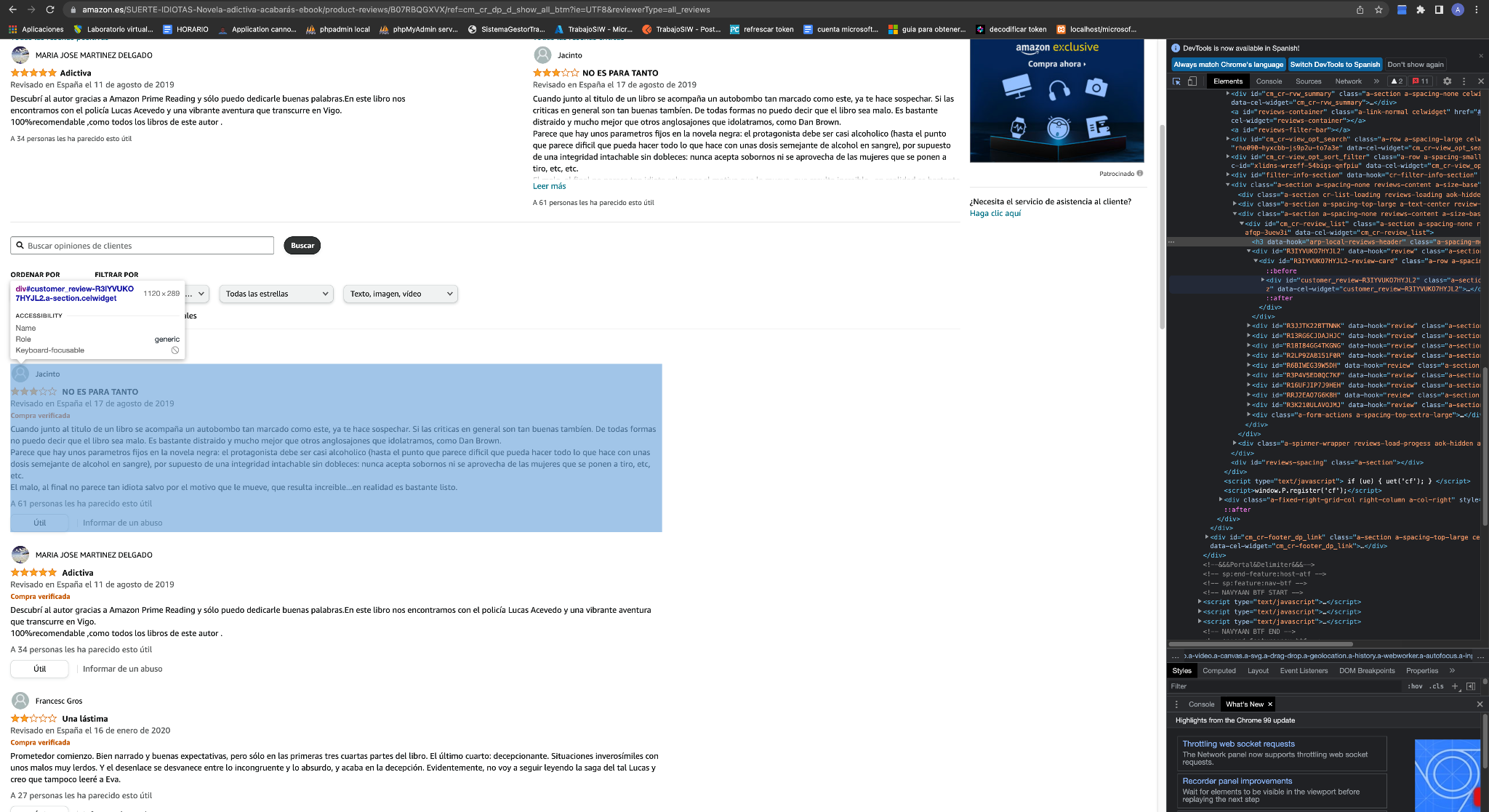
* Poder predecir con la mayor exactitud las valoraciones según las palabras utilizadas en las reseñas
* Emplear los métodos estudiados de preprocesado de datos.
* Observar los resultados obtenidos y sacar conclusiones en base a ellos.

**OBTENCIÓN DE DATOS:**

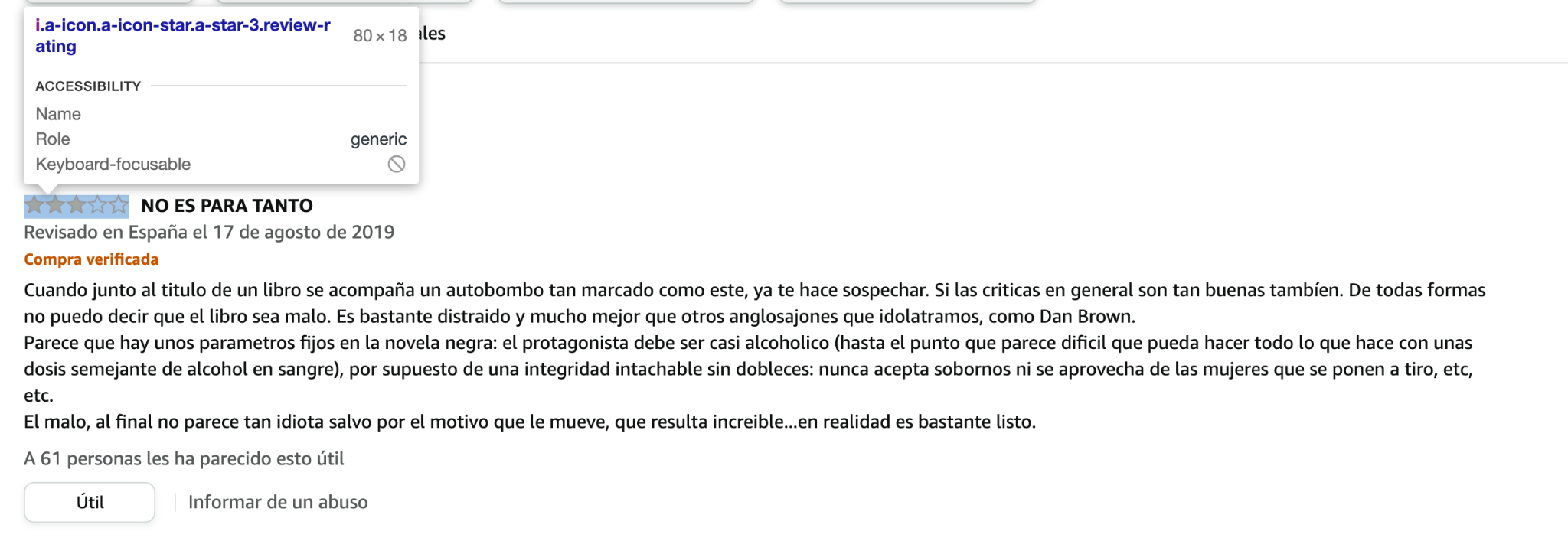
Nuestro dataset se compone de una matriz columna de comentarios escritos por los usuarios y otra con la valoración que le dan al producto. ¿Cómo hemos obtenido esto?

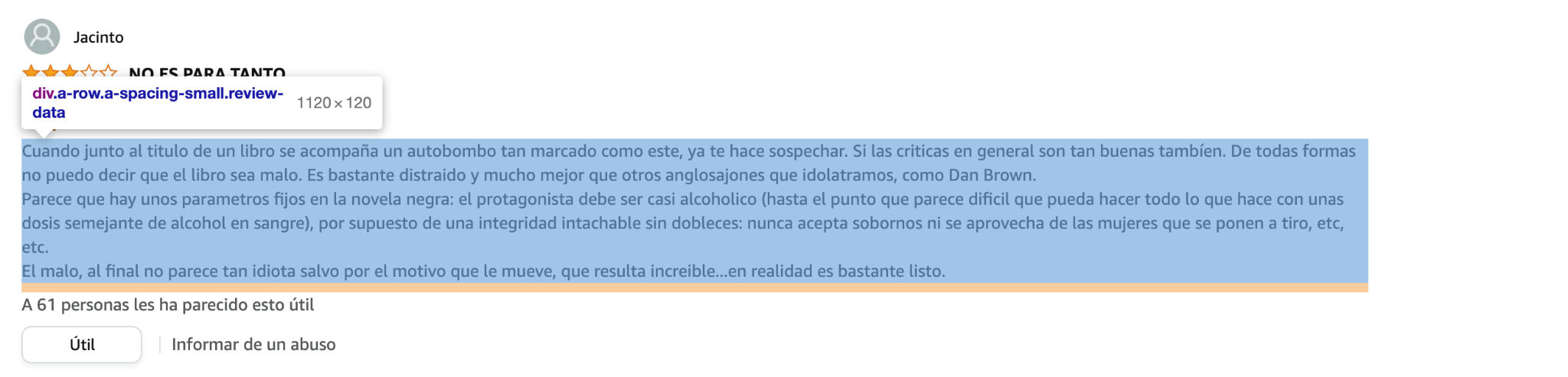
Para empezar hemos buscado libros con tres requisitos: que tengan una cantidad importante de reseñas, que sean en español y que, dentro de lo posible, estén balanceadas la cantidad de ejemplos que pertenecen a cada clase, en este caso la valoración que se le ha dado al libro.

Una vez seleccionados los libros con los que trabajar aislamos la información de la página web que nos interesaba, en nuestro caso la que estaba en este contenedor



Pero todavía podemos ¨limpiar¨ más el html desde el que estamos buscando nuestros datos. Para ello especificamos dentro del contenedor qué sub-etiqueta nos interesa obtener, osea estas dos:

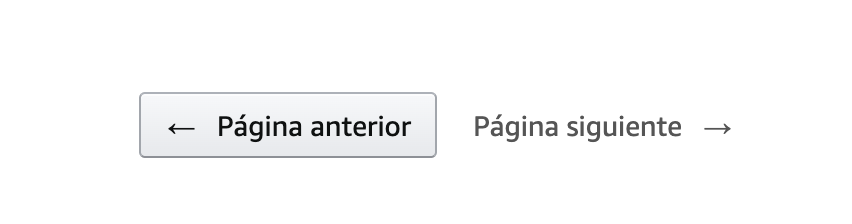




De estos dos contenedores cogemos el texto y ya tenemos las características de un ejemplo, iteramos sobre la página para tener los atributos de todos los comentarios de la página. Para hacernos con todas las páginas lo que hacemos es: darle al botón de siguiente página:



Hasta que llegamos a la última página y ya no está disponible el botón:

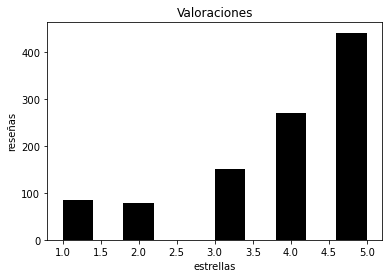


Con esto ya tenemos nuestro dataset disponible.

**ANÁLISIS DEL PROBLEMA:**

Uno de los principales inconvenientes que nos encontramos en este tipo de problemas es que los datos que tenemos son reseñas escritas por personas, eso hace que sean datos poco objetivos y que no sigan un patrón fijo que nos permita utilizar una solución que siempre funcione bien, por ejemplo puede darse el caso de que dos personas tengan reseñas muy parecidas entre sí pero que cada persona haya decidido darle una puntuación totalmente diferente al libro, esto pasa porque la opinión de una persona es algo sumamente subjetivo y difícil de prever.

Nos encontramos ante un problema muy desbalanceado. Aunque hayamos intentado seleccionar libros con valoraciones balanceadas es difícil, ya que hay muy pocos libros así y Amazon pone más facilidades para encontrar libros con valoraciones más altas. Así que partimos de este inicio:

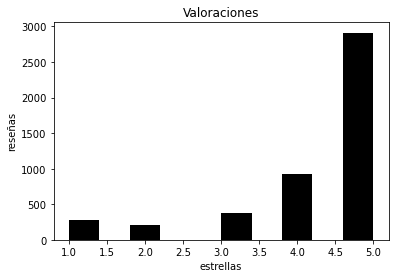


Lo que provoca esto es que al entrenar un modelo una posibilidad muy a tener en cuenta es predecir todo como 5 estrellas. Teniendo en cuenta lo que ha comentado Eugen anteriormente sobre la variabilidad de criterios de una persona a otra y que gran parte de los datos son de cinco estrellas, consigue porcentajes de acierto relativamente buenos. Pero no nos interesa que esto pase, así que hemos tenido que penalizar este tipo de razonamientos.

**SOLUCIÓN IMPLEMENTADA :**

* **AUMENTO DEL DATASET:**

Ya que tenemos tan fácil aumentar el tamaño del dataset decidimos hacerlo. De esta forma tenemos más ejemplos tanto para entrenar, como para testearlo. Lo cual va a hacer que mejore la forma de pensar de nuestro modelo. Además es una forma de combatir unos de los problemas más grandes que nos hemos encontrado en este proyecto: el sobreaprendizaje. El dataset después del aumento de ejemplos nos queda así:



* **PREPROCESADO DE LOS TEXTOS:**

El primer paso ha sido tokenizar, para ello hemos utilizado el “word tokenize” puesto que era el tokenizador que mejor encajaba con las características y las necesidades de nuestro problema.

Además hemos eliminado de las reseñas todo lo que no fuesen letras, puesto que elementos como los emoticonos, símbolos y otros elementos no resultaban importantes para la clasificación de las reseñas en un contexto como este.

* **STEMMING :**

A los textos le aplicamos stemming, para quedarnos con las raíces de las palabras en vez de las palabras como tal. Esto hace que palabras parecidas se guarden como la misma raíz, lo cual reduce la dimensionalidad del problema y nos ayuda a mejorar el modelo ya que palabras con raíces parecidas suelen darnos información parecida.

* **COUNT VECTORIZER :**

Previamente a obtener la tabla de frecuencias de palabras con count vectorizer hemos pensado que sería buena idea eliminar palabras que se utilicen con mucha frecuencia en castellano y que realmente no aportan información importante para posteriormente clasificar, es por ello que hemos utilizado “stop words” consiguiendo asi reducir el dataset considerablemente sin perder información relevante.

Hecho esto hemos pasado a realizar el count vectorizer con el que hemos obtenido nuestro corpus y nuestra tabla de frecuencias. Una vez hemos obtenido el corpus utilizando el dataset de entrenamiento lo hemos aplicado también sobre el dataset de test.

* **TF-IDF :**

El bag of words lo hemos complementado aplicando TF-IDF, el cual es un proceso que asigna pesos a cada palabra en el texto. A diferencia del bag of words de serie que solo ve cuantas veces aparece la palabra en el texto. TF-IDF da más importancia a aquellas palabras que aparecen mucho dentro del mismo texto y que están pocas veces en otros textos.

* **HOLD OUT :**

Partimos nuestros datos en entrenamiento y test, con hold out, pero para no entrenar modelos con unas clases y luego encontrarnos con clases nuevas en el test decidimos repartir los ejemplos de manera que a cada partición le caiga el mismo número de comentarios de cada clase.

* **REGRESIÓN LOGÍSTICA:**

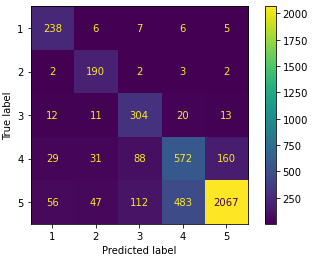
Hemos probado varios modelos el que mejor nos ha funcionado es la regresión logística con pesos para cada clase balanceados. Esto lo que hace es darle mayor peso a clases minoritarias. Así que evitamos que en caso de duda prediga 5 estrellas. Además la regresión logística la hemos aplicado en conjunto con OVA (One Versus All) porque es un problema multiclase.

**ANÁLISIS DE RESULTADOS :**

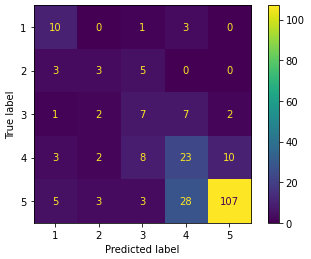
* **MATRIZ DE CONFUSIÓN:**

Para ver los resultados el primer paso que hemos tomado es visualizar la matriz de confusión obtenida, puesto que nos interesaba saber que estamos acertando ejemplos de todas las clases y no únicamente ejemplos de la clase mayoritaria, y con la matriz de confusión íbamos a poder comprobarlo.

La **matriz de confusión** obtenida en **train** es la siguiente:



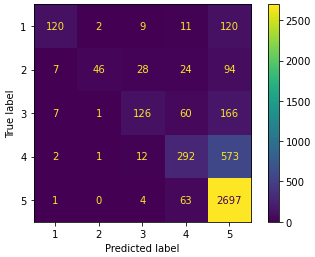
La **matriz de confusión** obtenida en **test** es la siguiente:



Como podemos observar gracias a que hemos dado más peso a las clases minoritarias acertamos la gran mayoría de los ejemplos que pertenecen a estas clases, es por eso que los valores más altos se concentran en la diagonal, además vemos que en la clase en la que más fallos cometemos es en la de 5 estrellas puesto que es la clase a la que menos peso le hemos asignado por el gran número de ejemplos que pero a pesar de todo se aciertan una gran cantidad de los ejemplos de esta clase gracias a que ha podido entrenar con más ejemplos que el resto de clases.

Otro aspecto a destacar es que de no haber ponderado los pesos de las clases hubiéramos tenido un gran problema con la clase de 5 estrellas pues el modelo tendía a predecir clase 5 en la mayoría de ejemplos, y aunque eso diera buenos resultados en precisión no estaba realmente prediciendo de forma correcta.

Aquí podemos ver un ejemplo de la **matriz de confusión** **sin ponderar los pesos** de las clases:



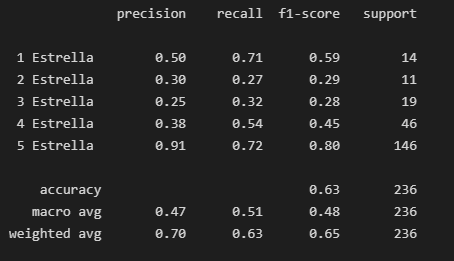
* **PRECISION Y RECALL:**

La precisión es la medida que nos transmite qué porcentaje de todos los ejemplos predichos como clase minoritaria es de verdad de esa clase. El recall por otro lado nos indica de todos los ejemplos de las clases minoritarias, ¿qué porcentaje se han clasificado correctamente? Tenemos menos recall qué precisión porque hemos asignado más peso a las clases minoritarias, lo cual hace que como efecto secundario clasifiquemos mal más ejemplos de cinco estrellas, pero acertemos más comentarios con valoraciones bajas.

* **F SCORE :**

Como sabemos el F Score calcula una media armónica de la precisión y el recall, esto hace que este tipo de métrica penalice resultados muy bajos o bien en la precisión o bien en el recall es por ello que un buen F score representa un buen equilibrio entre precisión y recall.

Con nuestro modelo hemos conseguido un F score del 63.13%.

A continuación mostramos los F scores por clases además de otros datos: 

Como podemos ver en clases de los extremos como la de 1 estrella , 4 o 5 se mantiene un mayor equilibrio entre precisión y recall pues son clases más diferenciadas del resto y por tanto algo más fáciles de predecir, eso se refleja en un F score más alto en estas clases, mientras que el F score desciende en las clases intermedias puesto que son más parecidas entre ellas y más fáciles de confundir unas con otras.

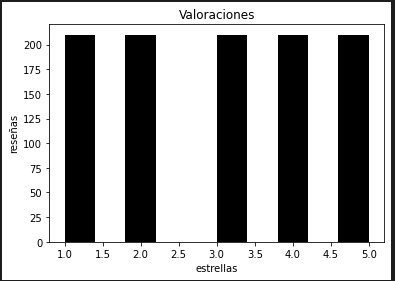
* **AUC ROC:**

El AUC ROC (Area Under the Curve ROC) nos sale bastante más alto esto se debe a que es la relación entre verdaderos y falsos positivos. Seguimos acertando muchos ejemplos de 5 estrellas ya que tenemos muchos para entrenar, así que tenemos un alto True Positive Rate.

**IDEAS DESCARTADAS :**

* **PODA DE DATOS :**

Una primera idea que tuvimos para evitar el desbalance de las clases era dejar para cada clase solo la cantidad de ejemplos que tuviese la clase minoritaria, consiguiendo así una cantidad equilibrada de ejemplos de cada clase como podemos ver en la siguiente gráfica :



Aunque a primera vista pudiese parecer una buena idea más tarde resultó dar muy malos resultados puesto que el modelo no tenía suficientes ejemplos de cada clase como para realizar un buen entrenamiento y eso se reflejo en muy malos resultados para predecir las clases.

Además los ejemplos sobrantes que quitamos podían contener información importante a la hora de clasificar y con este proceso de poda perdíamos una enorme cantidad de esa información.

* **Establecer min df :**

Lo que inicialmente hicimos fue establecer un min\_df como parámetro del countVectorizer.

La intención que teníamos era eliminar palabras que fuesen muy poco frecuentes porque no considerábamos que tuviesen una gran repercusión, pero al hacerlo obtuvimos un pequeño descenso en la precisión dado que algunas de estas palabras contenían información relevante para clasificar, además estábamos eliminando palabras poco frecuentes que eran extrañas por el efecto de haber aplicado previamente stemming sobre ellas.