SISTEMAS INTELIGENTES

Trabajo 1 – Predecir la puntuación de un objeto en Amazon en base a la reseña

Igor Vons, Aitor Uranga, Adrián Reviriego

Indice

1. Elección de productos

De cara a la realización de la práctica decidimos varios puntos para seleccionar nuestros productos.

-tener un número notable de comentarios

-tener la menor relación posible

-su "originalidad"

Tras recorrer una larga lista de candidatos, terminamos eligiendo 4, de los cuales 3 estarían destinados al entrenamiento y el último, para realizar la evaluación. Esto se estableció así para evitar posibles "infecciones" en los resultados. Esto es, para evitar que el posible sesgo de los modelos que relacione el producto con su calificación acabe influyendo en la precisión de las predicciones, se utiliza un producto distinto a los usados en el entrenamiento, dejando que se clasifiquen únicamente por el resto de las características aprendidas. Los productos seleccionados fueron:

1. October Elf PANtuflas de otoño e invierno para adultos, zapatos cálidos:

   https://www.amazon.com/October-Elf-Autumn-Slippers-baguette/dp/B01N0L5YZ9/?tag=readerwp-20

2. Máscara de caballo para fiesta de disfraces de cabeza de caballo:

   https://www.amazon.com/-/es/Máscara-caballo-disfraces-adultos-hombres/dp/B0107XRQ7E/ref=cm_cr_arp_d_pdt_img_top?ie=UTF8

3. Estatua de gnomo de jardín para interiores y exteriores, escultura de gnomo:

   https://www.amazon.com/Funny-Guy-Mugs-Dachshund-Sculpture/dp/B07G7FN1CV/?tag=readerwp-20

4. Amazon Basics - Pilas alcalinas AA de 1,5 voltios, gama Performance, paquete de 20:

   https://www.amazon.es/AmazonBasics-alcalinas-Performance-Paquete-variable/dp/B00NTCH52W/ref=cm_cr_arp_d_product_top?ie=UTF8

El primero nos proporciona 253 ejemplos, el segundo 697, el tercero 287 y el último 92.204. En conjunto, los 3 primeros ejemplos nos acaban ofreciendo en torno a los 1100 comentarios, descartando algunos que por bien estar vacíos o por tener caracteres aleatorios (hubo algunos cuyo único contenido consistía en guiones) no nos servían. Con respecto a los del cuarto producto, a pesar de que en la página se indique una cifra tan alta, a efectos técnicos no es del todo cierto. Esto se debe principalmente a un factor, y es que, con una cantidad de comentarios tan grande, llega un momento que la web de Amazon bloquea al usuario, de este modo, a partir de la página 500 no se le permite ver más reseñas, mostrando en su lugar el siguiente mensaje "No hay ninguna opinión con los criterios seleccionados. Pruebe a borrar o cambiar algún filtro. Borrar filtro". Esto significa que a lo sumo se pueden extraer 5000 comentarios (500 páginas con 10 comentarios por página). En realidad, debido a otros fallos y comentarios que debido a su contenido se descartan, eventualmente nos quedamos con unas 3500 reseñas. Una vez hemos obtenido estos textos pasaremos a procesarlos y tratarlos para poderlos usar en nuestro entrenamiento.

2. Preprocesamiento

Para poder usar los comentarios en nuestro entrenamiento primero tenemos que tratar los textos.

El primer problema encontrado fue que los comentarios no necesariamente estaban todos en el mismo idioma, lo que hacía que la comparación de vocabularios no tuviera consistencia. Para solucionarlo usamos la librería de *GoogleTrans* de Python, que funciona un poco como si fuese una API de *Google Translator* en realidad, y pasamos todos los comentarios al inglés, dado que es la lengua que mejor infraestructura en cuanto a tratamiento de lenguaje se refiere tiene.

Una vez todos los mensajes están en el mismo idioma, podemos modificarlos para facilitar la codificación de las palabras. Esto consistiría en 4 apartados principalmente: minúsculas, "*stop words*", '*PorterStemmer*', '*WordNetLemmatizer*'.

La parte de las minúsculas es bastante simple, y es que, si de dos palabras iguales una tiene una de las letras en mayúscula, está ya se interpreta como otra palabra distinta. Lo mismo para las "*stop words*", las cuales simplemente las obtenemos de la librería *NLTK* para el inglés y las eliminamos de nuestros mensajes.

El tercer paso, la aplicación de '*PorterStemmer*', se aplica para eliminar prefijos y sufijos de las palabras. En principio permite que luego '*WordNetLemmatizer*' tenga más fácil el trabajar con las palabras. Hasta donde hemos podido ver, se trata de un paso que arroja resultados mixtos y dependen mucho del ejemplo, dado que en casos concretos se puede perder valor semántico, aunque en general se observó cierta mejora.

El último paso, donde aplicamos '*WordNetLemmatizer*', permite reducir el número de palabras distintas convirtiéndolas en raíces comunes, permitiendo así al mismo tiempo reducir el número de tokens distintos y relacionar semánticamente palabras distintas que se tratarían independientemente unas de otras.

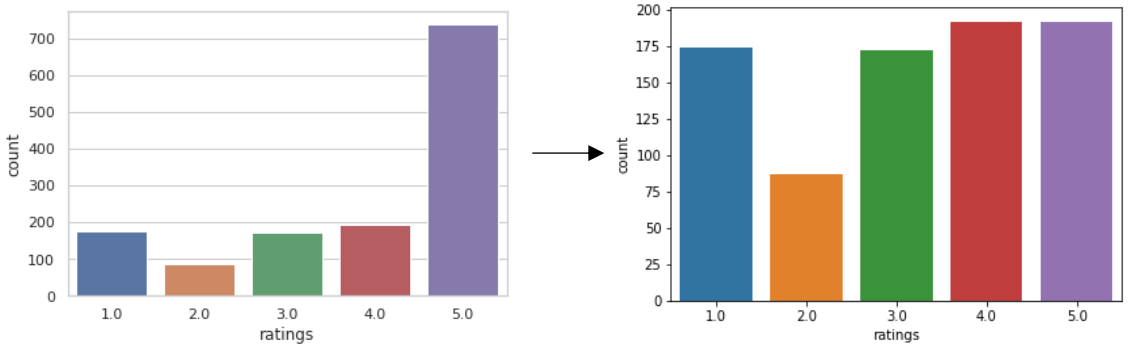
I am ChatGPT, a conversational AI language model developed by OpenAI. I am designed to generate human-like responses to text-based conversations with users. My purpose is to assist and provide helpful responses to the best of my knowledge and ability.

'chatgpt', ',', 'convers', 'ai', 'languag', 'model', 'develop', 'openai', '.', 'design', 'gener', 'human-lik', 'respons', 'text-bas', 'convers', 'user', '.', 'purpos', 'assist', 'provid', 'help', 'respons', 'best', 'knowledg', 'abil', '.'

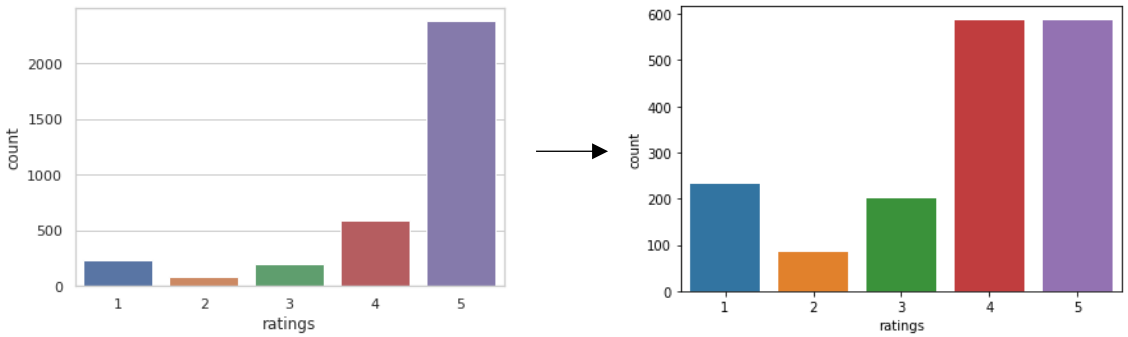
En término de tamaño de *datasets*, un problema que hemos encontrado con todos los productos es que hay muchas más *reviews* de 5 estrellas que de cualquiera de las otras 4 clases, lo cual es especialmente problemático para el entrenamiento, dado que podría hacer que los modelos predigan muchos más ejemplos a la clase 5, lo que supondría mucha menor penalización, ya que desviaría la precisión en su favor. Para mitigar este sesgo lo primero que hacemos es aumentar el tamaño de las primeras 4 clases, traduciendo los mensajes al español y luego de vuelta al inglés, creando algo de variación, lo que nos permite obtener ½ más de datos. Después, dado que sigue habiendo muchos más ejemplos de la

clase 5, hemos elegido aleatoriamente comentarios de esta hasta llegar al mismo número que la segunda clase con más ejemplos.

Dataset entrenamiento



Dataset evaluación

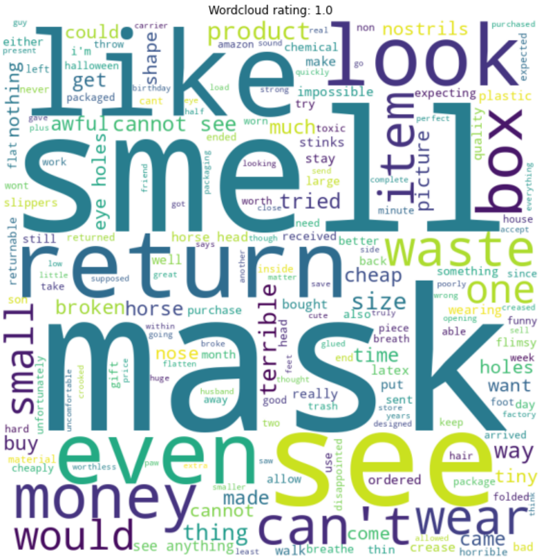


Una vez estos procesos se han aplicado a todo nuestro conjunto de datos, ya podemos codificarlos numéricamente. Para ello contamos con 2 métodos, siendo estos '*CountVectorizer*' y '*TfidfVectorizer*'. La diferencia entre ambos es que '*TfidfVectorizer*', como sugiere su nombre, también tiene en cuenta la relevancia de las palabras, además de su frecuencia de aparición, que sería el caso de '*CountVectorizer*'. En nuestro caso, hemos optado por realizar la codificación como último pasa antes de del entrenamiento, lo que nos permite usar los dos sobre cada modelo.

A continuación, mostramos la *wordcloud* del *dataset* de entrenamiento, así como la de cada clase. Se puede observar que algunas de las palabras que mas aparecen como "mask" o "gnome" son palabras que preferiríamos que fueran evitadas como tokens, dado que relacionan la puntuación de la *review* con un producto en lugar de con su contenido. Por ello en el próximo apartado se comenta como eliminamos los términos y bajo que criterios lo hacemos.

Wordcloud para todos los ratings

Wordcloud rating: 1.0



Wordcloud rating: 2.0

Wordcloud rating: 3.0

Wordcloud rating: 4.0

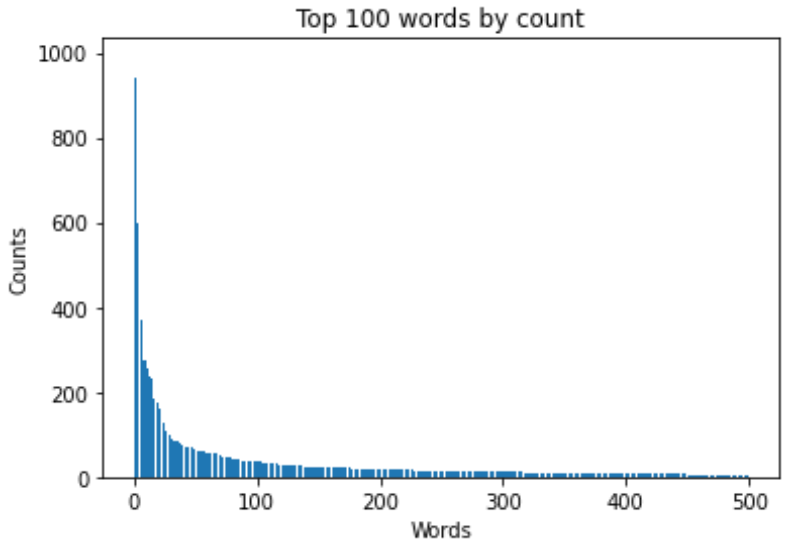
Wordcloud rating: 5.0

3. Entrenamiento de modelos

Para realizar el entrenamiento y predicción elegimos un buen abanico de modelos: *Regresión Lineal, Regresión Logística, Naïve Bayes Multinomial, Red Neuronal, Máquinas de Vector Soporte, OVO* y *OVA ambos con clasificador base SVC,* y *Ensembles (DecisionTree, Bagging, Boosting* y *Random Forest).*

Aquí todos los modelos son de clasificación, exceptuando únicamente la regresión lineal. Este último se ha añadido como perspectiva alterna a la de un clasificador, lo que tiene sentido porque al fin y al cabo lo que se trata es de determinar la opinión que representa un comentario, lo cual no tiene por qué estar representado necesariamente en clases discretas, sino más bien por un gradiente.

Para realizar el entrenamiento aplicaremos los dos vectorizadores presentados antes, con lo que se itera sobre una lista que contenga ambos. Una vez se han codificado ambos *datasets*, se puede pasar al *'GridSearchCV'*, que se encargara de calcular los mejores hiperparámetros para cada modelo. '*GridSearchCV'* lo que hace realmente es ejecutarse sobre una '*Pipeline'*, compuesta por el vectorizador y el modelo. Este se incluye en la 'Pipeline' para que '*GridSearchCV'* trabaje con los parámetros que determinan que tokens se usaran en el corpus. Antes de llegar a este apartado se ploteó la gráfica del número de apariciones de cada token a partir de la cual se pueden establecer umbrales aproximados de cuáles tokens elegir.



Introducimos estos posibles umbrales en el diccionario de hiperparámetros que utilizara '*GridSearchCV'* para realizar ejecuciones. Dado que nos decidimos centrar en el intervalo de palabras que tienen aproximadamente entre 300 y 50 apariciones, elegimos las posiciones 5, 7, 9, y 15, por un lado, y 70, 100, 120, 150 y 250 por otro. Así evitamos tokens que carecen de significado semántico por contar con excesiva presencia y de aquello que por aparecer muy pocas veces acaban por no aportar demasiada información.

Con respecto a los modelos, los parámetros que hemos estudiado serían:

- Regresión logística:
  - C: cuanto mayor sea mayor será el *Bias*
  - solver: algoritmo usado para la optimización. Elegimos *'newton-cg', 'saga'* y *'lbfgs'* por ser los que soportan perdida multinomial.

- Red neuronal:
    - hidden_layer_sizes: indica cuantas capas tiene la red
    - solver: algoritmo de optimización de pesos. Elegimos *'adam'* por el buen funcionamiento con *datasets* grandes y *'lbfgs'* que converge más rápido que otros.
- SVC:
    - C: cuanto mayor sea mayor será el *Bias*
    - gamma: cuanto mayor sea mayor será la *varianza*
    - kernel: función para el mapeado en más dimensiones. Elegimos *'rbf'* que recurre a la hiperesferas y *'sigmoid'* que usa la función sigmoide.

Conforme se obtienen resultados, algunos de estos hiperparámetros se pueden descartar en caso de ver que los resultados que lo usan son notablemente peores que el resto.

Una vez todos los hiperparámetros han sido filtrados, podemos aplicar *'GridSearchCV'* sobre cada modelo para estudiar su eficacia.

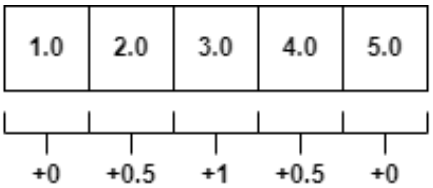
4. Resultados y comparaciones

Antes de exponer los resultados obtenidos con la ejecución de nuestro código, es relevante establecer algunos puntos que harán más fácil interpretarlos.

Para determinar si un modelo está o no haciendo interpretaciones más o menos correctas de las características de los ejemplos, podemos compararlo con lo que sería un clasificador aleatorio. En este caso, dado que tenemos 5 clases, la probabilidad de que un ejemplo caiga en una de ellas es 1/5, o lo que es lo mismo, 20%. Esto debemos tenerlo en cuenta cuando estudiemos la precisión de los modelos.

También hay que recordar cual es el propósito de estos entrenamientos, y este es no es simplemente obtener la mejor precisión posible, sino que haya cierta uniformidad en la precisión de cada clase. Esto puede suceder dado que los mensajes positivos sueles ser mucho más fáciles de identificar que aquellos que se encuentran en puntuaciones intermedias. Por eso no será suficiente con observar un solo número y se recurrirá a las matrices de confusión para ayudar en la selección.

Por último y tal y como ya se ha comentado, esto no es exactamente un problema de clasificación al uso, sino que buscamos una aproximación suficientemente buena de lo que pudiera ser la clase del comentario. Por ello en lugar de usar simplemente la métrica de precisión (cuantos ejemplos se han acertado), daremos algo de relevancia a aquellos que han quedado cerca de clasificarse correctamente. Concretamente en el recuento los aciertos tendrán el valor de 1 y los casi aciertos (la predicción está a la distancia de 1 de la clase real) de 0,5.



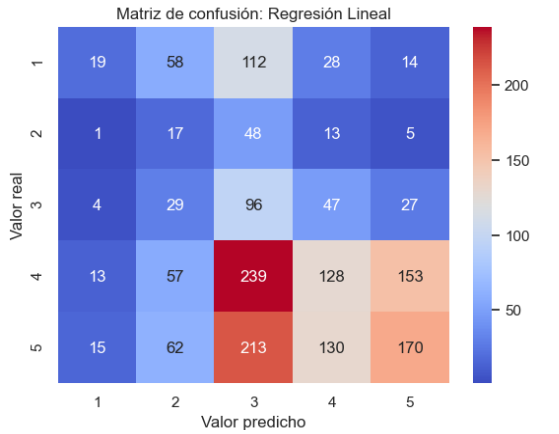
Se suman todas las predicciones y se divide entre el número de ejemplos

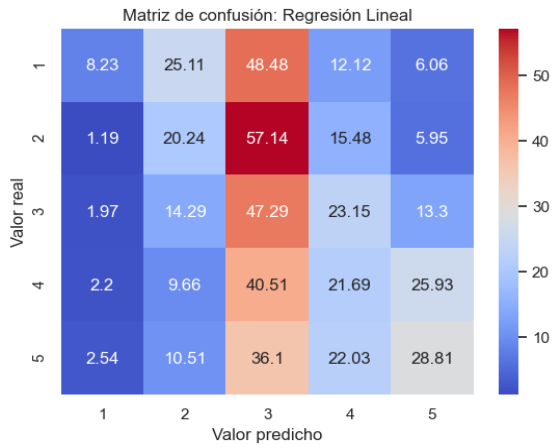
Una vez aclarados estos puntos podemos observar los resultados obtenidos de nuestras ejecuciones. Por cada modelo tenemos una matriz de confusión que representa en valor absoluto el número de ejemplos clasificados, así como otra donde estos valores se representan en porcentajes, mostrando en términos relativos la clasificación de los ejemplos de cada clase.

1. Regresión lineal

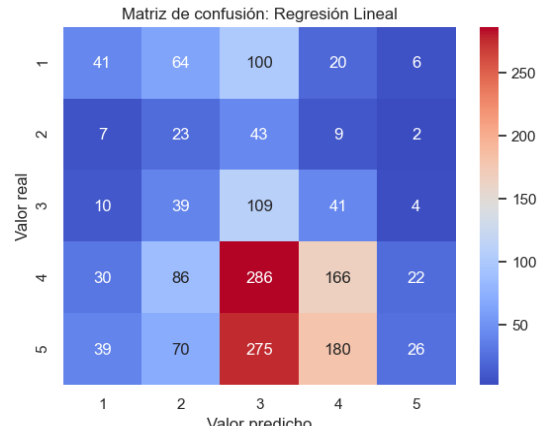
Este es el caso especial dentro de las pruebas debido a que su predicción no son valores concretos. Y aunque durante el entrenamiento esto se puede suplir relacionando el error con la distancia, luego a la hora de interpretarlo sería interesante discretizar un poco más las predicciones. Se ha probado tanto a usar el redondeo como simplemente elegir la parte entrera de la predicción para observar cómo comportaría.

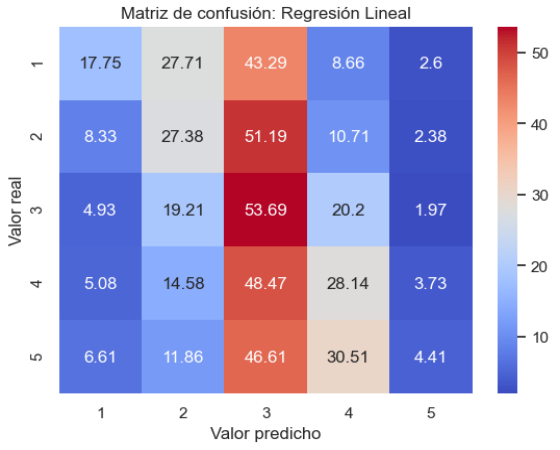
round





trim



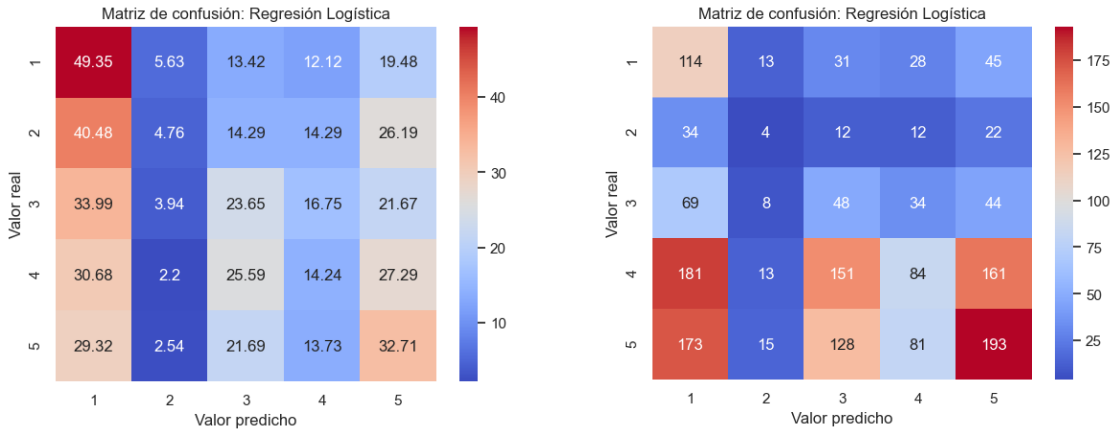


Podemos observar que la clasificación orbita en exceso a la clase 3, lo que se puede deber a la manera en la que se ha elegido el cálculo de error por la distancia. Al estar en el centro permite que los errores se vean menos penalizados de lo que debería ser. La precisión con nuestra medida especial es de 45.55% con redondeo y 41.31% sin él.

2. Regresión logística

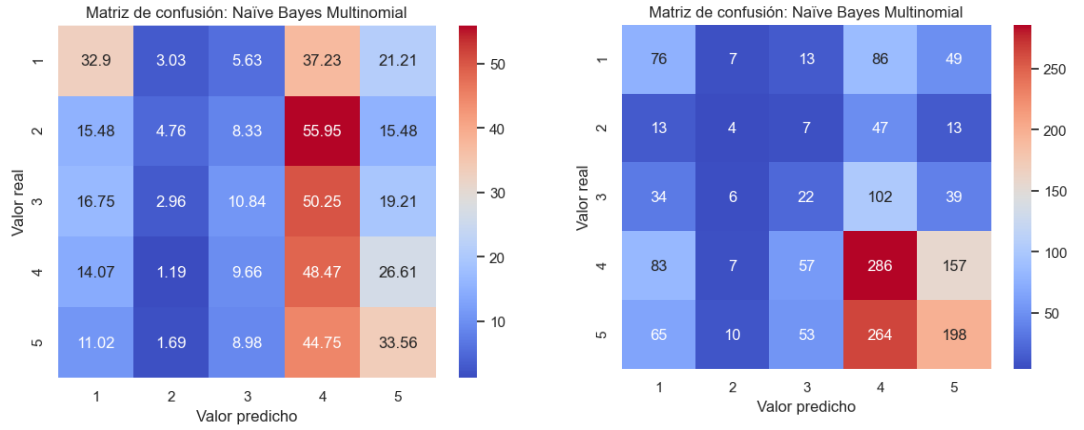
En el caso de la regresión logística hemos trabajado con los distintos algoritmos de optimización que nos ofrece el modelo. En principio el mejor resultado que obtenemos es un

accuracy de 49,41%. Aquí podemos observar que hay un sesgo relativamente pronunciado hacia la clase 1, y un poco menor pero también presente hacia la 5. La clase 2 es las menos poblada.



3. Naïve Bayes Multinomial

En cuanto al clasificador *Naïve Bayes* probamos su variación para modelos multinomiales con el cual conseguimos un *accuracy* del 54,24%.

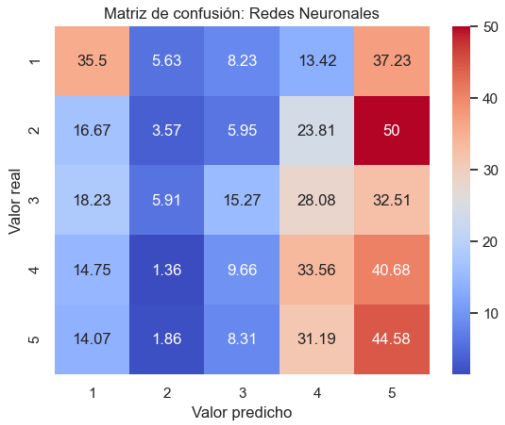


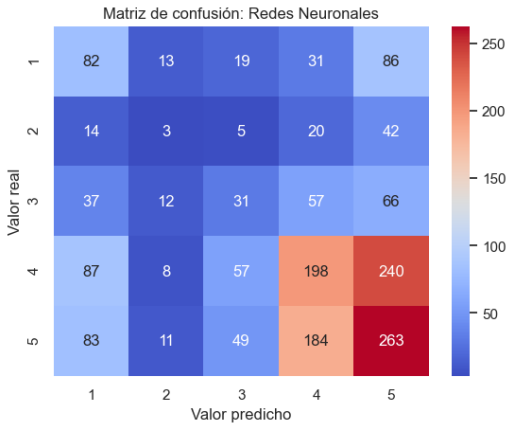
Podemos ver como la clase 4 pasa a ser a la que más ejemplos le son asignados y como la clase 2 sigue siendo la más discriminada.

4. Redes Neuronales

En las Redes Neuronales, vemos una distribución un tanto más uniforme de los aciertos de las predicciones. Ahora se orbita más hacia las clases 5 y 4 principalmente. La 2 continúa siendo la peor clasificada. De hecho, la mitad cae a la clase 5.

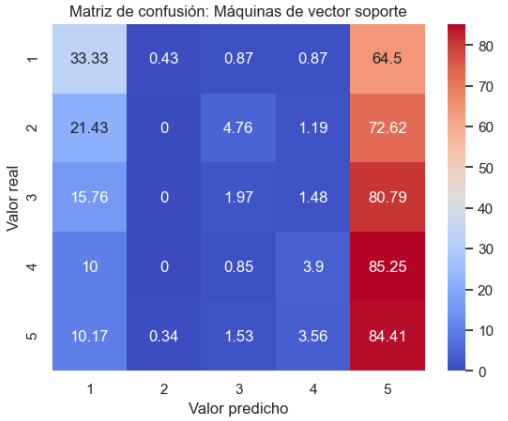
El porcentaje de acierto en este caso es de 51,12%. Cabe destacar que para este resultado se han utilizado 20 capas ocultas, lo que nos indica lo enrevesado que puede ser el problema de separar las clases.

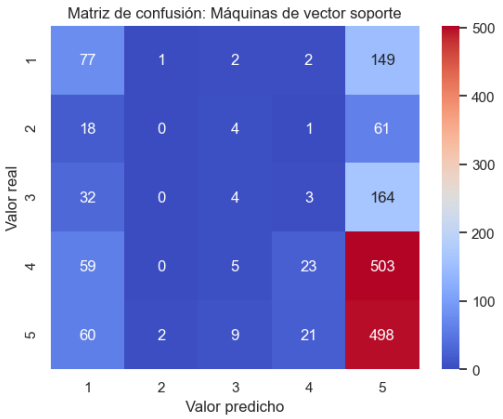




5. Máquinas de vector soporte

Respecto al modelo de Máquinas de vector soporte, podemos observar básicamente se establece una dicotomía entre la clase 5 y la 1, teniendo mucho más peso la 5. Se consigue un *accuracy* del 55,89% pero esto se debe a que gran parte de los ejemplos son de la clase 5 y 4 (que por cómo hemos definido nuestra medida tiene bastante relevancia en el resultado), pero a pesar de ello es obvio que el resultado está mucho más lejos de lo deseado que en otros clasificadores.

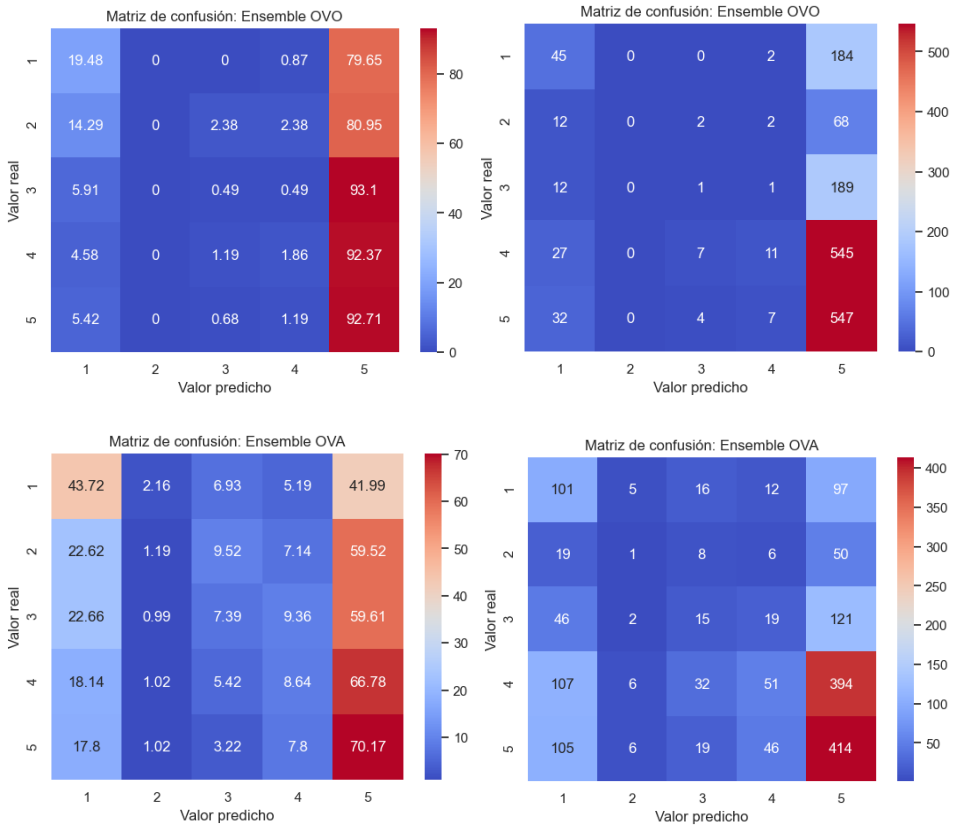




Esto se puede deber a que las máquinas de vector soporte (SVM) tienen dificultades para clasificar correctamente la clase minoritaria en un problema de clasificación desbalanceado porque se sobreentrenan en la clase mayoritaria y maximizan la distancia entre las clases, ignorando los ejemplos de la clase minoritaria

6. Ensembles OVO | OVA

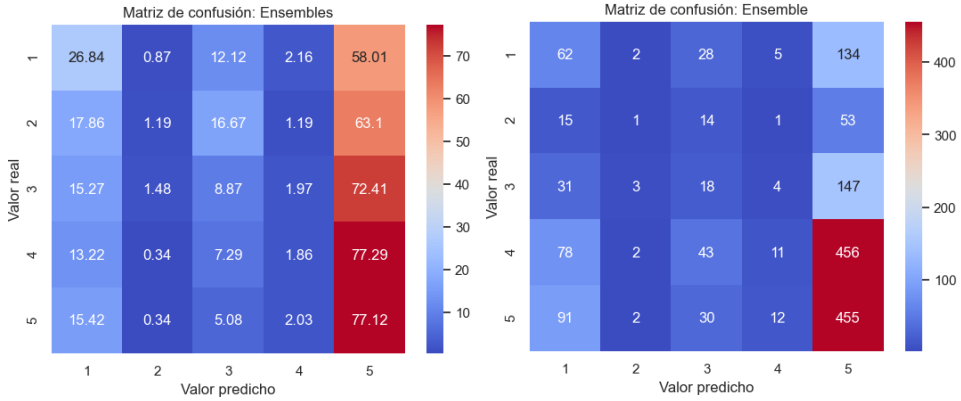
En consideración a los *ensembles* OVO y OVA, conseguimos un *accuracy* del 52,47% y del 50,41% respectivamente



Se repite un comportamiento similar al de las SVM, lo que hasta cierto punto tiene sentido teniendo en cuenta que es el clasificador base usado. Cabe destacar que el método de OVO acentúa más aun el sesgo hacia la clase 5, mientras que el OVA consigue paliarlo ligeramente, haciendo que más ejemplos caigan a sus clases correspondientes.

7. Ensembles DecisionTree, Bagging, Boosting, RandomForest

En este último modelo volvemos a encontrarnos con la misma situación que en los dos anteriores, y es que se prioriza la predicción de la clase 5 a la de todas las demás lo que nos deja con un 48.38% en *accuracy*, pero a efectos prácticos es mucho menos eficaz de lo que esta medida sugiere.



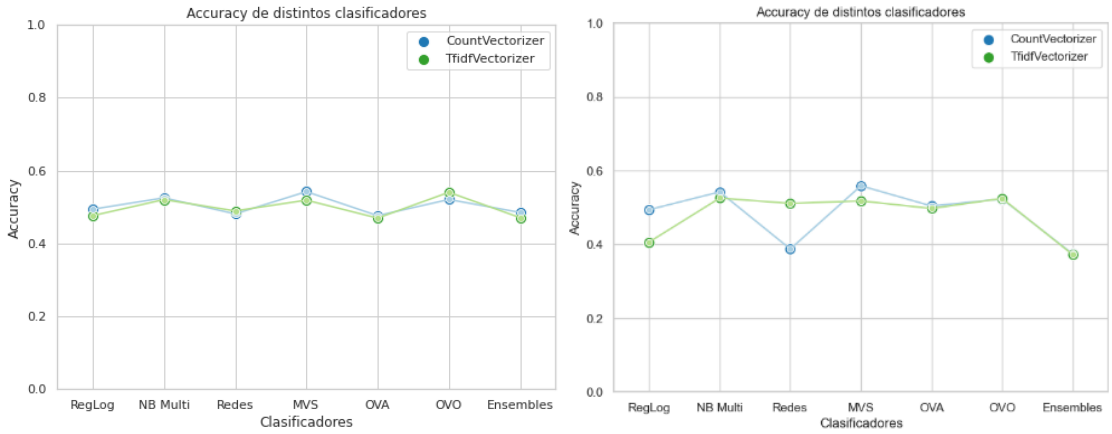
5. Mejoras a futuro

En este caso se seleccionaron productos más bien aleatorios para no tener sesgo con respecto a la expresividad de los compradores. Sin embargo, en caso de querer potenciar la faceta de análisis de sentimiento, sería interesante seleccionar textos que tengan mejor estructura semántica. Se podría conseguir eligiendo, por ejemplo, *reviews* de libros relacionados con áreas muy técnicas o específicas, lo que podría suponer que las personas que compren y opinen sobre dicho producto lo haga expresándose de forma más correcta y analizable.

Otra posible modificación podría consistir en ampliar tanto el tamaño de los *datasets* como el de los productos utilizados. Esto nos permitiría tener menos sesgos sobre la puntuación en función del producto, además de podernos permitir descartar más ejemplos que consideremos insustanciales debido a lo cortos o mal construidos que están.

6. Conclusiones

Como primera reseña cabria comentar la relevancia de los dos métodos de vectorización utilizados y su importancia en el desempeño final. Aunque uno inicialmente uno pudiera pensar que tener en cuenta la relevancia de las palabras debería mejorar bastante el rendimiento de la codificación, lo cierto es que los resultados no parecen soportar esta idea. De hecho a lo largo de las distintas pruebas realizadas hemos llegado a obtener valores invertidos, en el sentido que donde antes uno daba mejor desempeño que el otro ahora era al revés. Se puede concluir de esto que o bien el método de vectorización no tiene tanta relevancia, o lo que sería más plausible, que en el problema que nos atañe y dado nuestro *dataset* de entrenamiento, la forma de vectorizar los tokens no es tan importante.



En general viendo los resultados obtenidos podemos deducir que no se trata de un problema tan trivial como podría parecer a primera vista. Especialmente se ve esto reflejado en que exceptuando las clases 1 y 5 el resto son más bien clasificaciones casi uniformemente distribuidas. Y es que hasta cierto punto tiene sentido. Esto ya se podía intuir a partir de la distribución de los datos obtenidos. ¿Pensemos, cuando alguien puntuaría con un 1? pues realmente cuando estemos descontentos con un producto. Un 5 lo pondremos cuando nos encontremos satisfechos con él. Un 3 cuando tengamos pegas. No quedan por tanto el 2 y el 4. El 2 es la calificación con menor aparición en la mayoría de los productos, y es que rara vez

alguien que se encuentre descontento con un producto va a preferir puntuar con un 2 en lugar de un 1. En esa situación uno, de forma pasional, opta por la mayor penalización generalmente. Por otra parte, la calificación de 4 suele ser un tanto complicada de distinguir del 5 y se debe más a la generosidad del consumidor más que a la satisfacción en si con el producto.

De los resultados observados podemos concluir que las SVM y los *ensembles*, como OVO y OVA, pueden sufrir de sobreajuste en la clase mayoritaria y subajuste en la clase minoritaria, lo que puede llevar a una baja precisión en la clasificación de la clase minoritaria. Por otro lado, las redes neuronales, la regresión logística y el *Naive Bayes multinomial* pueden funcionar mejor en problemas de clasificación desbalanceados debido a su capacidad para adaptarse a diferentes distribuciones de clases, su capacidad para limitar el impacto de los valores atípicos y los datos ruidosos, y su facilidad de implementación y ajuste.

Es importante tener en cuenta que no hay un enfoque único que funcione mejor para todos los problemas de clasificación desbalanceados, y que la elección del modelo y la estrategia de ajuste dependerá en gran medida del conjunto de datos y el problema en cuestión. También es fundamental considerar técnicas específicas para abordar el desbalance en los datos, como técnicas de muestreo y ajuste de parámetros, para obtener un mejor rendimiento en este tipo de problemas.