20 / 02 / 2025 Universidad Pública de Navarra

TRABAJO 1 SIA

Grupo 9

ÍNDICE

1. [INTRODUCCIÓN](#_1._INTRODUCCIÓN) -------------------------------------------------------------------- 3
2. [RECOPILACIÓN Y LECTURA DE DATOS](#_2._RECOPILACIÓN_Y) ---------------------------------------- 4
3. [PREPROCESAMIENTOS BÁSICOS Y ELECCIÓN DEL MODELO](#_3. _PREPROCESAMIENTOS_BÁSICOS) ------------ 5
4. [MEJORAS PREPROCESAMIENTO Y MODELO](#_4._MEJORAS_PREPROCESAMIENTO) ---------------------------------- 6
5. [CONCLUSIONES](#_5._CONCLUSIONES) ------------------------------------------------------------------ 11

## 1. INTRODUCCIÓN

En la actualidad, los comercios online son una de las mayores fuentes de venta de bienes y servicios, siendo Amazon una de las más importantes. En este contexto, las reseñas forman un papel crucial en la toma de decisión de compra, ya que expresan la experiencia y opinión de los usuarios que han obtenido los productos previamente. Estas reseñas incluyen tanto una valoración cuantitativa, representada por estrellas en una escala del 1 al 5, como una valoración cualitativa en forma de texto libre.

Debido a esta relación entre la puntuación otorgada y el contenido textual de las reseñas, es posible desarrollar un sistema de predicción que, a partir del análisis del texto, determine la calificación en estrellas que probablemente haya asignado un usuario. Este proyecto tiene como objetivo **predecir la puntuación de un objeto en Amazon en función de la reseña escrita por el usuario**, aplicando técnicas de **Procesamiento de Lenguaje Natural (PLN)** y **aprendizaje automático**.

Este proceso sigue una metodología que abarca desde la recopilación de datos mediante web *scraping* con *BeautifulSoup* hasta la aplicación de modelos de clasificación. Se obtiene un conjunto de reseñas en español, que luego se preprocesan eliminando ruido, normalizando el lenguaje y vectorizando los textos con TF-IDF. Posteriormente, se entrenan modelos de aprendizaje automático como Regresión Logística, Naive Bayes, Random Forest y Redes Neuronales (MLP) para evaluar su desempeño en la predicción de puntuaciones. Dado el desbalance de clases, se aplican técnicas como ajuste de pesos para mejorar la equidad en el entrenamiento. Finalmente, se comparan los resultados y se proponen mejoras para optimizar el sistema.

## 2. RECOPILACIÓN Y LECTURA RESEÑAS

En primer lugar, tuvimos que decidir qué productos queríamos usar para entrenar y evaluar el modelo. Al principio dudamos entre usar uno o varios productos, pero finalmente nos decantamos por esta segunda opción, de forma que seleccionamos productos variados con puntuaciones variadas.

Tratamos el problema de recopilar y almacenar las reseñas de Amazon en un formato apropiado para poder preprocesarlas posteriormente. Para poder llevarlo a cabo, nos ayudamos de la librería de Python *BeautifulSoup*, utilizada para extraer y procesar datos de páginas web mediante ***web scraping***. Permite analizar el código *HTML* o *XML* de un sitio web y facilita la navegación y búsqueda de elementos dentro de la estructura del documento.

De esta forma, implementamos un código para abrir en el navegador la web de *Amazon* y buscar las reseñas de los productos deseados. Al principio tuvimos problemas al entrar en *Amazon* porque se nos detectaba como *bots* y se nos bloqueó el acceso. Eventualmente, descubrimos que para que no sucediera esto, debíamos iniciar sesión en Amazon manualmente y realizar las interacciones en intervalos de 5 segundos para evitar el bloqueo.

Finalmente, conseguimos implementar un código que accediera a la página web, recopilara las reseñas y las almacenara.

## 3. PREPROCESAMIENTOS BÁSICOS Y ELECCIÓN DEL MODELO

El segundo notebook hace uso del *csv* extraído del anterior. Primeramente, decidimos aplicar unos preprocesamientos básicos, como aplicar *minúsculas*a todas las palabras de los textos. Seguidamente, se *eliminan los signos de puntuación* y caracteres especiales con la función *re.sub()*. A su vez, hemos importado una lista de *stop words* en castellano desde *spacy.lang* para terminar de limpiar las *reviews*.

Después de esta selección de preprocesamientos como modelo base, se vectoriza el texto con un vector *TF-IDF* base, sin parámetros y se añaden a los conjuntos de entrenamiento y validación para el entrenamiento de modelos.

Se ha elegido *TF-IDF* en lugar de *BoW*, *TF* e *IDF* porque combina la frecuencia de las palabras en cada documento (TF) con la importancia relativa de esas palabras en todo el corpus (IDF), lo que permite destacar términos clave mientras se minimiza el impacto de palabras comunes. A diferencia de *BoW*, que solo cuenta palabras sin diferenciar su relevancia, *TF-IDF* resalta aquellas que influyen en el significado del texto. Además, mientras que *TF* solo mide frecuencia e *IDF* solo penaliza palabras comunes, *TF-IDF* consigue un equilibrio entre ambos, capturando mejor la información crítica de cada reseña sin añadir ruido innecesario. Esto lo hace ideal para los modelos de machine learning clásicos, permitiendo una representación efectiva del texto sin necesidad de grandes volúmenes de datos ni entrenamientos costosos, a diferencia de métodos más avanzados como Word Embeddings.

Dividimos el conjunto de entrenamiento y el conjunto train lo usamos para entrenar los modelos. En cuanto a los modelos seleccionados, hemos probado con los mínimos requeridos por el proyecto, *Naive Bayes Multinomial* y *Regresión logística.*

Una vez realizados todos estos procesos, obtuvimos los siguientes resultados:

|  |  |
| --- | --- |
| Modelo | Precisión |
| Naive Bayes Multinomial | 50% |
| Regresión Logística | 58% |

## 4. MEJORAS PREPROCESAMIENTO Y MODELO

Para comenzar, probamos a añadir otros preprocesamientos con *lematización* o *stemming*. Se aplica a la vez que la tokenización y la eliminación de los *stop* *words*. Discutiendo sobre el tema, llegamos a la conclusión de que la técnica de lematización iba a brindar una mayor precisión, ya que se trata de una técnica algo más sofisticada y en la que se mantiene algo más de información. Sin embargo, en la siguiente tabla se refleja las diferencias en media del *accuracy* de los distintos modelos respecto a no añadirlos:

|  |  |  |
| --- | --- | --- |
|  | *Lemma* | *Stemming* |
| *Accuracy* | +2% | +3% |

Vemos que funciona mejor *stemming* a pesar de ser a priori una mayor simplificación y por lo tanto perdida de información. En este caso arroja consistentemente en todos los modelos de predicción un mayor *accuracy*. Hemos podido comprobar que a los modelos de aprendizaje no siempre les conviene unos datos más informativos y por ende complejos. Técnicas que simplifican el proceso de aprendizaje acaban resultando en un mejor resultado a pesar de aportar menos información.

Una vez pudimos comprobar esto, basamos el resto de las pruebas y teorías en base a esas premisas. Tratar de simplificar los datos de entrada para los distintos modelos.

El siguiente añadido con el que hicimos pruebas fue con los distintos posibles parámetros de la vectorización TF-IDF. El primero que nos llamó la atención fue en “ngram\_range”, que consigue extraer todas las secuencias de n-gramas siendo la N un rango que le indicas (x,y). Un n-grama con una n de un alto valor conseguiría simplificar una vez más el trabajo del modelo, aportando a su vez un mayo contexto. Por eso probamos con distintas combinaciones, al principio con valores altos ya que el tiempo de computación no estaba siendo alto y podíamos permitir un aumento de la dimensionalidad de los datos.

|  |  |
| --- | --- |
| Rango *n\_gramas* | Aumento de *Accuracy* |
| (1,20) | -3% |
| (1,10) | -2% |
| (1,5) | 0% |

El aumento está medido respecto a no poner ese parámetro en la función *TfidfVectorizer()* en los distintos modelos de aprendizaje. Rápidamente observamos que no estábamos consiguiendo mejoras, más bien veíamos una mejora según nos acercábamos a rangos más pequeños, incluso siendo el mejor resultado posible sin poner nada. Concluimos que ver cada palabra por separado nos estaba arrojando mejores resultados y que agrupar las palabras solo creaba confusión a nuestro modelo. En cuanto al tiempo de ejecución, sí que aumentaba según el rango era más amplio, pero nunca superaba los 30 segundos así que no tuvimos en cuenta este dato para decidir el valor del parámetro.

A pesar de los resultados poco satisfactorios, insistimos en probar con otro tipo de rangos que no empezaran en 1. Esto podría ser de utilidad al reducir la dimensionalidad de los datos y volver a esa idea de simplificar los datos que obtuvimos al principio:

|  |  |
| --- | --- |
| Rango *n\_gramas* | Aumento de *Accuracy* |
| (10,20) | -15% |
| (10,15) | -16% |
| (3,6) | -14% |

Los resultados fueron desastrosos por lo que paramos las distintas pruebas sobre el parámetro. La precisión en *naive bayes* descendió drásticamente ofreciendo valores entre 10-15%, esto nos dimos cuenta de que tenía mucho sentido ya que la probabilidad de encontrar la misma secuencia de 10 palabras en las *reviews* es prácticamente nula así que funcionaba como un clasificador aleatorio.

Por otro lado, seguimos investigando por otros parámetros de la función. Pasamos a probar la normalización que ofrece, siendo los posibles valores la “l1”y “l2”. Con la primera opción empeoraban ligeramente los resultados, pero es con la segunda con la que se mantenían muy parecidos incluso anecdóticamente superiores. Sin embargo, decidimos que este “l2” podría verse beneficiado en otros modelos de aprendizaje posteriores que requieran de ello. Por eso, no descartamos por completo y lo tuvimos en cuenta más adelante.

Revisando el conjunto de datos de entrenamiento, nos dimos cuenta de que las clases estaban muy desbalanceadas. La mayoría de las puntuaciones caían en 1 o 5 estrellas por lo que la cantidad de ejemplos en las otras tres posibilidades era claramente inferior, lo que podía comprometer los resultados. Para ello implementamos un método para balancear los datos desde la librería *imb-learn*. A la hora de realizar pruebas los resultados no fueron los esperados:

|  |  |
| --- | --- |
| Modelo | Diferencia en Accuracy |
| Regresión logística | + 1% |
| Naive Bayes | - 10% |

Nos dimos cuenta analizando los resultados que el problema venia de la naturaleza de los datos de validación. Los conjuntos de datos con los que medimos el rendimiento son una selección aleatoria de los datos de entrenamiento, por lo que siguen la misma distribución desbalanceada. Esto hace que sea mejor mantener el desbalance ya que eso es con lo que se encontrará el modelo a la hora de la verdad.

De todas maneras, como es Naive Bayes el que peor parado parecía quedar, dimos una oportunidad a Naive Bayes con un balanceo. Para ello calculamos el número de veces que aparecen reseñas de cada clase. Una vez obtenidos estos resultados, los normalizamos para trabajar con los datos entre 0 y 1. Para conseguir darle más pesos a la clase minoritaria se calcula el peso de forma inversa. Provocando que la importancia de las clases más frecuentes sea mucho menor. Estos nuevos pesos se los añadimos a Naive Bayes para ver su comportamiento con esta modificación. La clasificación obtenida fue una mucho más diversa en comparación a al otra, pero con una precisión final más pobre. Ya que acertaba más las clases minoritarias, pero con un mayor fallo en las más comunes. Por lo que el grueso de los ejemplos estaba peor clasificado.

Por último, probamos otro tipo de clasificadores. A parte del Naive Bayes y la Regresión logística añadimos y probamos los siguientes. El *accuracy* está medido frente a la media de los dos originales con los que trabajamos.

|  |  |
| --- | --- |
| Modelo Aprendizaje | *Accuracy* |
| *SVM* | + 3% |
| *Random* *Forest* | + 7% |
| *Naive Bayes* Balanceado | - 10 % |
| *OVO* | + 8% |
| *OVA* | + 7% |

Exceptuando el *Naive Bayes* balanceado, que ya hemos visto su situación. Vimos que todos los demás ofrecían mejor resultado que el *Naive bayes*, por eso al comparar la precisión como la media del mencionado con la regresión logística salen todos más beneficiados. Por ello, tratamos de mejorar los resultados del clasificador haciendo pruebas con sus distintos parámetros. La manera más eficaz para ello fue utilizar un *grid\_search* con una gran variedad de posibles parámetros como un Alpha de 0.01 a 3, o posibles combinaciones de prioridad de clases. El resultado final fue un Alpha de 0.1 sin prioridades de clase, pero con prioridad de *fit* que arrojó un 59% (aumentándolo en 15%), justamente lo que buscábamos.

Otros clasificadores de los que esperábamos mejores resultados eran *OVO* y *OVA*, y habiendo tenido tan buen resultado con el *grid\_search* para *Naive Bayes* seguimos con esta filosofía. Sin embargo, no obtuvimos los resultados buscados y el porcentaje de acierto estaba estancado.

Una vez comprobamos estos resultados, tratamos de indagar en otras medidas como la matriz de confusión. Con ello conseguimos ver más en detalle el comportamiento de los distintos clasificadores. Observamos que algunos seguían una estrategia de clasificar más la clase con mayor presencia como la regresión logística. Otros, sin embargo, trataban de ser más precisos al adivinar otras posibilidades como el caso de *Naive Bayes* o los *OVO* y *OVA*. Por ello, se nos ocurrió montar un *esemble* de manera que se consiguiera una combinación de comportamientos y posiblemente un mejor resultado.

Realizamos un *voting* classifier con el *OVO*, la regresión logística y el *naive bayes* con un voto suave mediante probabilidades. La idea inicial era dar un porcentaje de peso con el que prevalezcan los mejores resultados que ofrecían el Ovo y la regresión, pero que en los casos que *naive* predecía los ejemplos de 1 estrella pudiera acertarlo. Para ello un 33% para cada uno o *voting* *hard* con el que surge el mismo efecto hicimos esa primera prueba. También lo acompañamos de otros porcentajes con un mayor para el *OVO* y el *naive* o Logistico y *naive*. En la siguiente tabla reflejamos los resultados:

|  |  |
| --- | --- |
| Porcentajes de Voto | Diferencia de *Accuracy* |
| [0.33, 0.33, 0.33] | + 8% |
| [0.4, 0.2, 0.4] | + 9% |
| [0.4, 0.4, 0.2] | + 7% |

Nota: Los porcentajes están en el orden de [Ovo, Logistic, Naive] y la diferencia de porcentaje medida respecto a la media de los dos modelos originales con los que se compara en el resto del documento

Los resultados son óptimos cuando se le da algo menos de importancia a la regresión logística, cosa que pudimos explicar fácilmente ya que el comportamiento de este último era similar al ovo pero con algo menos de precisión. Por otra parte, confirmamos la teoría de que, al juntar modelos con comportamientos diferentes, se conseguía esa complementariedad que buscábamos con ello.

Para no basarnos solo en el criterio de la precisión, decidimos comprobar en cuantas estrellas se desviaban las reviews previstas respecto a las reales. De esta manera podemos comprobar dos métricas complementarias que nos facilita la selección del mejor clasificador. Para poder escoger la que mayor precisión tiene y la que menos se aleja en caso de fallo. Los totales se han calculado como la suma del número de veces que ha habido una desviación por el número de estrellas desvaídas en cada caso.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Desviación | OVO (63%) | OVA (61%) | Niave Bayes (58%) | Regresión logística (58%) | Ensemble (63%) |
| 1 | 25 | 24 | 28 | 25 | 24 |
| 2 | 10 | 12 | 12 | 10 | 10 |
| 3 | 9 | 11 | 10 | 8 | 9 |
| 4 | 10 | 10 | 11 | 18 | 11 |
| total | 112 | 121 | 126 | 141 | 115 |

Al observar estas distribuciones se puede ver que, aunque algunos clasificadores tengan precisiones muy similares, la desviación de las estrellas es muy distinta. Sobre todo, si nos fijamos en los totales. Por lo que teniendo esto en cuenta podríamos rediseñar el ensemble propuesto anteriormente.

La regresión logística al ser el clasificador que peor desviación tiene de los puestos en el ensemble los descartamos para comprobar de nuevo su comportamiento sin él.

Ajustando de nuevo correctamente el ensemble con los clasificadores de OVA y Naive Bayes y sus pesos para conseguir el mejor resultado. Nos sale que hemos conseguido un 64 % de precisión, pero con una desviación mucho mejor.

|  |  |
| --- | --- |
| Desviación | Ensemble nuevo (64%) |
| 1 | 26 |
| 2 | 13 |
| 3 | 7 |
| 4 | 6 |
| total | 97 |

## 5. CONCLUSIONES

Después de todas las pruebas realizadas y las conclusiones que hemos ido sacando nos quedamos con lo siguiente:

* Preproceso a las palabras de entrada basado en eliminación de *Stop* *Words* y *Stemming*
* Vectorización de las cadenas de manera individual (sin n-gramas) con una normalización L2
* No balancear las clases
* Tener en cuenta métricas que permitan ver la diferencia de clasificación de las distintas clases por separado (Matriz de confusión, Desviación de Puntuación)
* Usar un ensemble como clasificador basado en dos modelos que siguieran patrones de predicción complementarios (*Naive Bayes* y *OVA*)