1.1.1-.

Ya que lo ideal es vectorizar texto para obtener una gran cantidad de features útiles, el “sentimiento” no nos sirve para generar features. Sin embargo, usar este como etiqueta nos es de utilidad para ir catalogando palabras como “positivas” o “negativas”, y así saber qué tipo de palabras ocupar y su disposición para poder generar críticas que entren en una de estas dos categorías.

1.2.1-.

Es importante eliminar los valores nulos ya que estos no nos entregan información, solo demoran y producen más error al algoritmo.

1.2.2-.

a) Las clases se encuentran balanceadas, podemos dar cuenta de ello por la proporción de positivas y negativas que se muestran en el código de este inciso. Ya que solo tenemos 2 etiquetas, esto implica que las clases tienen igual proporción de negativas que de positivas.

b) Si las clases están desbalanceadas, el algoritmo se le entrenará proporcionalmente a ello, tendrá mejor certeza para categorizar una, pero no para la otra, por lo que podría dar más falsos positivos y negativos, o puede ser que el algoritmo tenga un sesgo hacia la categoría dominante, mientras que para la otra su trabajo sea deficiente, ya que se pierden detalles que caracterizan a esta característica.

c) Para balancear la data, una opción es darle mayor peso a la categoría con menos datos. Para esto, ponderamos más su data de forma que en el entrenamiento se le dé más énfasis (Logistic regression o árboles tienen la opción de weight por ejemplo) [1]. Otra opción es submuestrear la categoría con más datos, quitando algunos de los datos de esta clase, tal que queden balanceados. El poder hacer esto o no dependerá de la cantidad de datos que se tengan y la diferencia de proporción entre ambas etiquetas. Otra opción sería crear datos, algunos algoritmos ayudan a esto. También es posible usar algoritmos con Random Forest o Gradient Boosting que son más robustos ante desequilibrios de data, pues estamos entrenando varios modelos y usando el promedio de estos. [2]

1.2.3-.

Es necesario eliminar las stopwords para reducir el peso del código y la data irrelevante, tal que el algoritmo no se enfrasque con información que no resulta ser de utilidad y que el procesamiento sea mucho menos pesado.

1.3.1-. Bag of words:

Este es una técnica de modelación del lenguaje que se puede implementar el python. Es decir, este método es capaz de extraer las palabras del texto y convertirlas en features, sin importar el orden de las palabras o su estructura gramatical, es decir, solo se preocupa de las palabras. Para esto, se hace una lista de todas las palabras que componen nuestros textos, excluyendo signos de puntuación o carácteres especiales, stopwords y convertir las palabras a minúsculas para que no se cuente una misma palabra como otra debido a ello. Además, hacemos listas de palabras de cada input de texto que tenemos. Con esto, marcamos la frecuencia con que aparecen las palabras en cada texto, y con ello generamos para cada uno de estos un vector distinto, el cual será el que usaremos para el algoritmo. Es por ello también la importancia de eliminar las stopwords de aquí, pues estas se repiten con mucha frecuencia [3].

Como vemos, puede ser problemático que no importe el orden de las palabras en la oración o que no se tenga en cuenta el sentido semántico de estas. Las palabras con poca frecuencia serán ignoradas a menos de que le demos más peso [3].

1.3.2-. TF-IDF:

Este algoritmo funciona con la misma base que bag of words, la diferencia es que este trata de darle la relevancia a la palabra que tiene, y no según su frecuencia en el texto, ya que una palabra que se repite mucho podría bien ser irrelevante [3].

Para hacerlo, se calcula el número de veces que una palabra se repite partido en el número total de palabras en el documento. Tenemos entonces la frecuencia con que la palabra se repite. Esto nos da TF.

Luego, para el cálculo de IDF, se le suma 1 al logaritmo de frases en el csv partido en el número de frases donde la palabra existe, de tal forma que mientras más cercano a 0 en este caso, mayor la frecuencia.

Con esto entonces se tiene una noción de qué tan frecuente es la palabra sin que sea realmente importante [3].

1.3.3-. SBERT:

Aquí ya se tiene una base de datos donde el algoritmo fue capaz de asignarle un significado semántico a las palabras (incluso a palabras que se escriben igual pero según el contexto tienen distinto significado). Básicamente, se transforman estas palabras a otro espacio, donde la cercanía euclidiana de las palabras entre sí indica una conexión semántica entre estas. El problema es que su complejidad lo hace muy difícil de correr en comparación a otros métodos. Es un método llamado “Siames Network”, el cual utiliza dos o más subnetworks para generar vectores de cada palabra (o lo que se esté clasificando, funciona incluso para reconocer rostros) [5].

1.5-.

LIME:

Es un modelo de caja negra entrenado para predecir a través de hacer pequeñas variaciones en la data e ir aprendiendo qué ocurre de ello, haciendo así su propia data de entrenamiento. Algo interesante es que al utilizarlo con texto como en este caso, le podremos hacer preguntas respecto a la base de datos y este podrá responder, como algoritmos tipo chat gpt o Quora. También puede darte las relaciones entre tus datos y otras características de interés [6].

En el caso de palabras, la variación es la presencia o ausencia de palabras, en imágenes, se superponen o quitan píxeles similares a los contiguos, para datos tabulados, se pesan y combinan de distintas formas, etc [6].

La idea es hacer la data de los modelos de machine learning interpretables por humanos y simple de entender.

1.6-.

De los cálculos de métrica obtuvimos que:

vectorización		precision	recall	f1-score	accuracy
DesitionTree con BoW	negative	0.64	0.59	0.61	
	positive	0.63	0.67	0.65	0.63
SVM con BoW	negative	0	0	0	
	positive	0.51	1	0.67	0.51
RF con BoW	negative	0.74	0.89	0.81	
	positive	0.87	0.69	0.77	0.79
DesitionTree con TIF-IF	negative	0.64	0.59	0.61	
	positive	0.63	0.67	0.65	0.63
SVM con TIF-IF	negative	0	0	0	
	positive	0.51	1	0.67	0.51
RF con TIF-IF	negative	0.74	0.89	0.81	
	positive	0.87	0.69	0.77	0.79
DesitionTree con Sbert	negative	0.62	0.63	0.62	
	positive	0.63	0.62	0.62	0.62
SVM con Sbert	negative	0.82	0.80	0.81	
	positive	0.81	0.83	0.82	0.82
RF con Sbert	negative	0.76	0.77	0.76	
	positive	0.77	0.76	0.76	0.76

Vemos que hubieron ciertos problemas con SVM tanto para BoW como con TIF-IF, pero curiosamente esto no ocurrió en Sbert, y de hecho, en este caso se obtuvieron métricas bastante buenas. Esto podría significar que el entrenamiento para palabras negativas fue insuficiente, mientras que para positivas se sobreajustó.

RandomForest obtuvo en general los mejores resultados, sobre todo con TIF-IF y BoW. DesitionTree tuvo resultados consistentemente mediocres, pues apenas tuvo verdaderos positivos respecto al total.

1.7-.

Se esperaba que Sbert tuviese mejores resultados [7], más estos no sobresalieron por ser muy buenos o muy malos. Esto podría ser porque parte del sentido de las oraciones se fue junto con las stopwords, y no es del todo conveniente eliminar estas para este modelo [8]. Por otro lado, tiene sentido que RandomForest tenga los mejores resultados, pues este clasificador es más robusto. Como vimos en clases, es una combinación de árboles promediados, por tanto, tiene sentido que sea un mejor clasificador que estos. También, como vimos SVM separa datos por hiperplanos, por lo cual, si tenemos una gran cantidad de features como en este caso, incluyendo palabras que pueden tener más de un significado según el contexto, esto puede hacer más complicada la clasificación, por tanto tiene sentido que RandomForest presente mejor rendimiento.

1.8-.

Usar modelos entrenados con más datos, que tengan más información semántica o de la posición de las palabras en el texto. BoW tuvo un buen desempeño, y w2v debería ser mejor [a], pues combina este con la noción de la palabra entre las otras, semántica y gramáticamente, superando también a TF-IDF (no logré instalar ni glove ni fasttext y se me acababa el tiempo como para probarlos u.u). [c]

Probamos luego con ROBERTa, un modelo pre-entrenado, con mayor información semántica que BERT, pues fue entrenado con mucha más data, por tanto, también debería funcionar mejor [7].

Como clasificador utilizaremos L-SVM, pues este ha demostrado muy buenos resultados con datos de texto. [d]

No se obtuvieron buenos resultados :c

1.9-.

Los modelos cada uno trabaja según distintos parámetros, por tanto la importancia de las palabras varía bastante según esto. En general, los emoticones fueron asociados en todos los modelos a algo positivo. Muchos adjetivos también fueron clave, aunque muchos no estuvieron en las categorías que yo esperaba.

Fuentes:

[1] Na. (2019, mayo 16). *Clasificación con datos desbalanceados*. Aprende Machine Learning.
https://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/

[2] Tripathi, H. (2019, Septiembre 24) What Is Balanced And Imbalanced Dataset?.
https://medium.com/analytics-vidhya/what-is-balance-and-imbalance-dataset-89e8d7f46bc5

[3] *An introduction to Bag of Words in NLP using python*. (2020, Sept 11). Great Learning Blog: Free Resources What Matters to Shape Your Career!
https://www.mygreatlearning.com/blog/bag-of-words/

[5] Efimov, V. (2023, septiembre 12). *Large language models: SBERT — sentence-BERT*. Towards Data Science. https://towardsdatascience.com/sbert-deb3d4aef8a4

[6] Smirnov, M. (misha). (2022, febrero 4). *Explain your text classification model with Lime*. Medium.
https://medium.com/@mikeusru/explain-your-text-classification-model-with-lime-1354ff747f15

[7] Makarov, A. (s/f). *Best architecture for your text classification task: Benchmarking your options*. KDnuggets. Recuperado el 15 de noviembre de 2023, de
https://www.kdnuggets.com/2023/04/best-architecture-text-classification-task-benchmarking-options.html

[8] *Is it necessary to do stopwords removal ,Stemming/Lemmatization for text classification while using Spacy,Bert?* (s/f). Stack Overflow. Recuperado el 15 de noviembre de 2023, de
https://stackoverflow.com/questions/63633534/is-it-necessary-to-do-stopwords-removal-stemming-lemmatization-for-text-classif

[9] Van Otten, N. (2023, febrero 15). *Tutorial TF-IDF vs Word2Vec for text classification [how to in python with and without CNN]*. Spot Intelligence.
https://spotintelligence.com/2023/02/15/word2vec-for-text-classification/

[c] Kathrani, K. (2020, mayo 18). *All about embeddings - Word2Vec, Glove, FastText, ELMo, InferSent and sentence-BERT*. Medium.
https://medium.com/@kashyapkathrani/all-about-embeddings-829c8ff0bf5b

[d] Li, S. (2018, septiembre 25). *Multi-class text classification model comparison and selection*. Towards Data Science.
https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568