RESEÑAS DE MEDICAMENTOS – ANÁLISIS DE SENTIMIENTO DE TEXTOS

¿Cuál es el problema a resolver y el objetivo inicial deseado?

Mi gato estaba teniendo crisis convulsivas severas debido a una epilepsia causada por una lesión cerebral. Hoy en día no existen medicamentos veterinarios para la epilepsia, tenemos que usar medicamentos humanos y queremos saber cuál es el mejor.

Mi objetivo es crear un algoritmo de inteligencia artificial para determinar el sentimiento de las revisiones de medicamentos para que sea más fácil decidir cuál usar.

¿De dónde has extraído los datos?

El conjunto de datos se publicó originalmente en el repositorio UCI Machine Learning. Citación:

Felix Gräßer, Surya Kallumadi, Hagen Malberg y Sebastian Zaunseder. 2018. Análisis de sentimiento basado en aspectos de revisiones de medicamentos aplicando aprendizaje entre dominios y datos cruzados. En Actas de la Conferencia Internacional sobre Salud Digital de 2018 (DH '18). ACM, Nueva York, NY, EE. UU., 121-125.

https://archive.ics.uci.edu/ml/datasets/Drug+Review+Dataset+%28Drugs.com%29

https://www.kaggle.com/datasets/jessicali9530/kuc-hackathon-winter-2018

El conjunto de datos proporciona revisiones de pacientes sobre medicamentos específicos junto con condiciones relacionadas y una calificación de paciente de 10 estrellas que refleja la satisfacción general del paciente. Los datos se obtuvieron rastreando sitios de revisión farmacéutica en línea.

Información de atributos:
1. drugName (categórico): nombre del fármaco.
2. condition (categórica): nombre de la condición.
3. review (texto): revisión del paciente.
4. rating (numérica): calificación del paciente de 10 estrellas.
5. date (fecha): fecha de entrada de revisión.
6. usefulCount (numérico): número de usuarios que encontraron útil la revisión.

¿Cómo se han limpiado y preparado? ¿Has aplicado "feature engineering"? En caso afirmativo, justifica las decisiones.

Búsqueda de valores "NaN" (nulos) para tratar con ellos si los hay (en nuestro caso no había "NaN"), eliminación de columnas innecesarias que no aportan información y creación de columnas nuevas con información que pueda ayudarnos a desarrollar el mejor modelo. Se usan algunas bibliotecas predefinidas para crearlas (SentimentAnalyzer de Vader y TextBlob). Luego, visualizamos los datos en gráficos para tener una mejor visión de ellos.

```
# use the vader library to determine if a review is positive or negative

# Negative : Compound Score <= -0.05
# Neutral : Compound Score >-0.05 and < 0.05
# Positive : Compound Score >= 0.05

analyzer = SentimentIntensityAnalyzer()
drugs['review_sent'] = drugs['review'].apply(analyzer.polarity_scores)
drugs_sent_int = pd.concat([drugs.drop(['review_sent'], axis=1), drugs['review_sent'].apply(pd.Series)], axis=1)
```

El "compound" es una métrica que calcula la suma de todas las calificaciones del léxico que se han normalizado entre -1 (negativo más extremo) y +1 (positivo más extremo).

Sentiment polarity and subjectivity

```
: from textblob import TextBlob

def polarity(review):
    # Sentiment polarity of the reviews
    pol = []
    for i in review:
        analysis = TextBlob(i)
        pol.append(analysis.sentiment.polarity)
    return pol


def subjectivity(review):
    # Sentiment subjectivity of the reviews
    sub = []
    for i in review:
        analysis = TextBlob(i)
        sub.append(analysis.sentiment.subjectivity)
    return sub
```

La polaridad es un valor flotante dentro del rango [-1.0 a 1.0] donde 0 indica neutral, +1 indica un sentimiento muy positivo y -1 representa un sentimiento muy negativo.

La subjetividad es un valor flotante dentro del rango [0.0 a 1.0] donde 0.0 es muy objetivo y 1.0 es muy subjetivo. La oración subjetiva expresa algunos sentimientos personales, puntos de vista, creencias, opiniones, acusaciones, deseos, creencias, sospechas y especulaciones, mientras que las oraciones objetivas son fácticas.

No he hecho normalización o estandarización de datos como tal. La “feature” del dataset que nos dará las predicciones es las reseñas (reviews), por lo que estamos tratando es texto. Necesitamos limpiarlo para que los modelos tengan mayor capacidad de predicción:

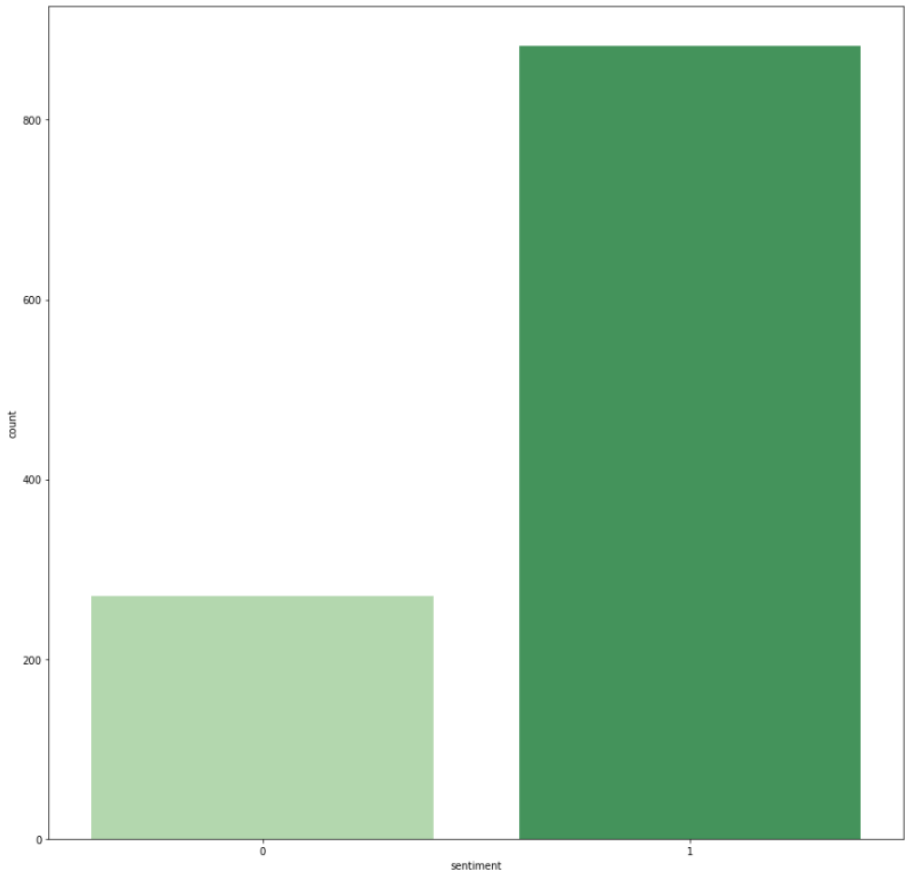
- cambiando a minúsculas.
- reemplazando el patrón repetitivo de # 039.
- eliminando todos los caracteres especiales.
- eliminando todos los caracteres no ASCII.
- eliminando los espacios en blanco iniciales y finales.
- reemplazando múltiples espacios con un solo espacio.
- reemplazando dos o más puntos con uno.
- eliminando las palabras vacías.
- derivando las palabras.
- eliminando las “stopwords” (palabras como conexiones, artículos… que no aportan información semántica).

Cleaning text

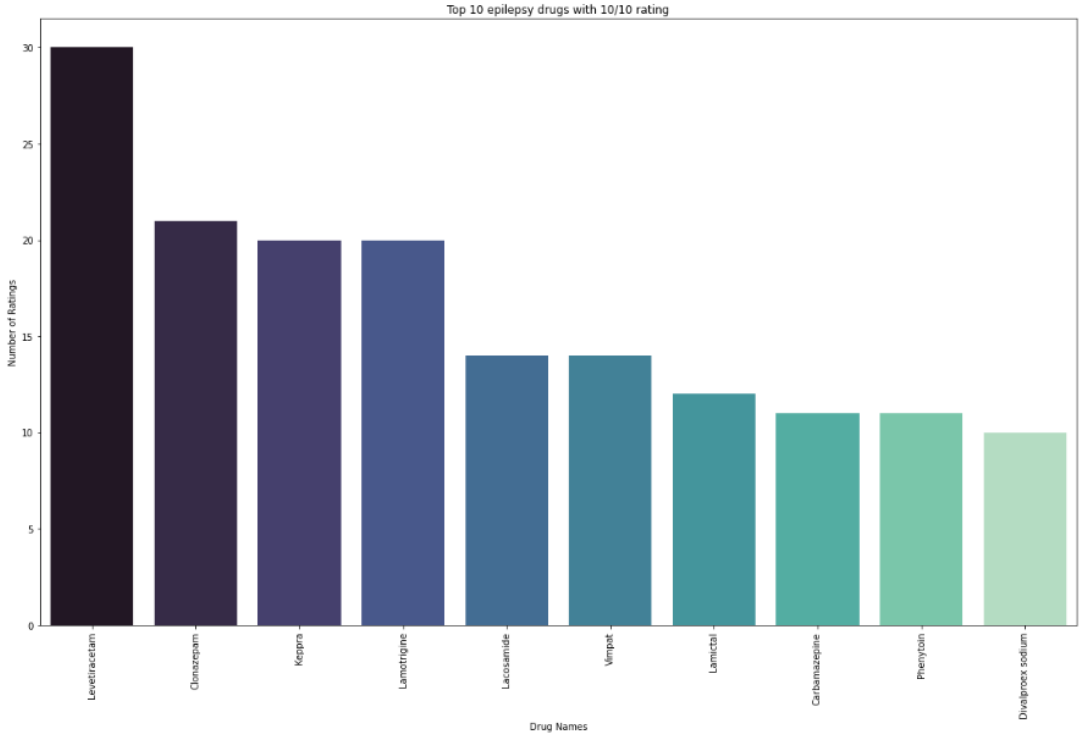
```
def review_clean(review):
    # changing to lower case
    lower = review.str.lower()

    # Replacing the repeating pattern of &#039;
    pattern_remove = lower.str.replace("&#039;", "")

    # Removing all the special Characters
    special_remove = pattern_remove.str.replace(r'[^\w\d\s]',' ')

    # Removing all the non ASCII characters
    ascii_remove = special_remove.str.replace(r'[^\x00-\x7F]+',' ')

    # Removing the leading and trailing Whitespaces
    whitespace_remove = ascii_remove.str.replace(r'^\s+|\s+?$','')

    # Replacing multiple Spaces with Single Space
    multiw_remove = whitespace_remove.str.replace(r'\s+',' ')

    # Replacing Two or more dots with one
    dataframe = multiw_remove.str.replace(r'\.{2,}', ' ')

    return dataframe
```

Incluye algunos gráficos relevantes de la exploración de los datos.

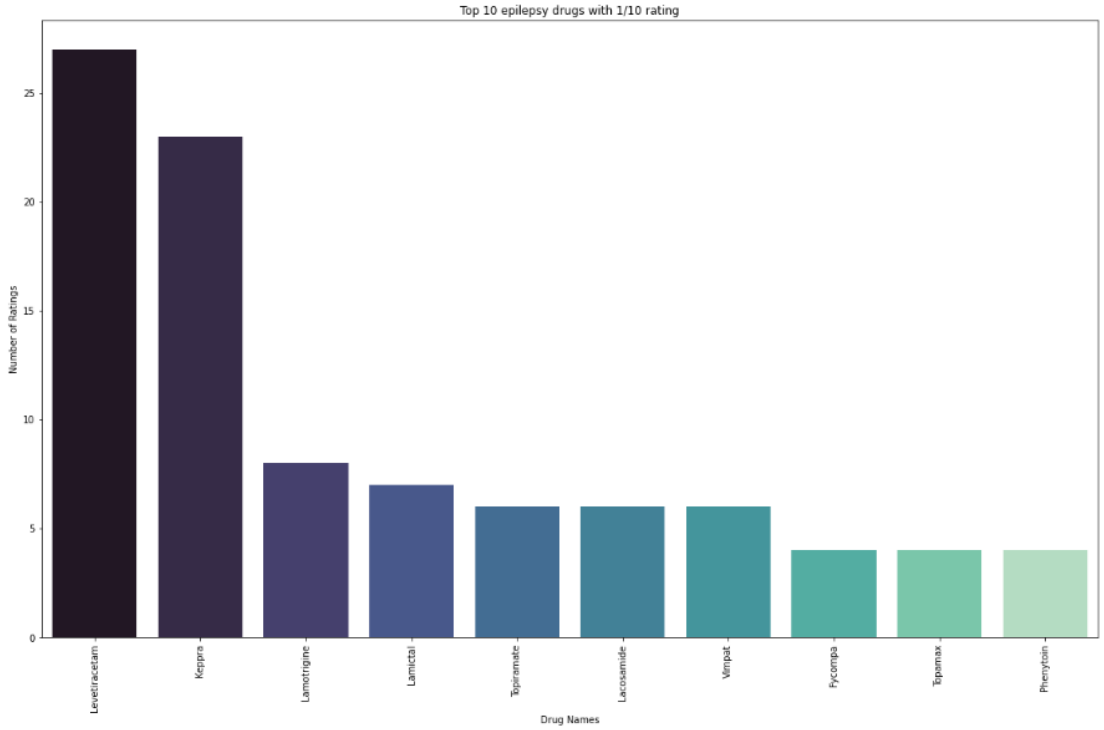
Distribución de las clases (sentimiento negativo: 0, sentimiento positivo: 1)



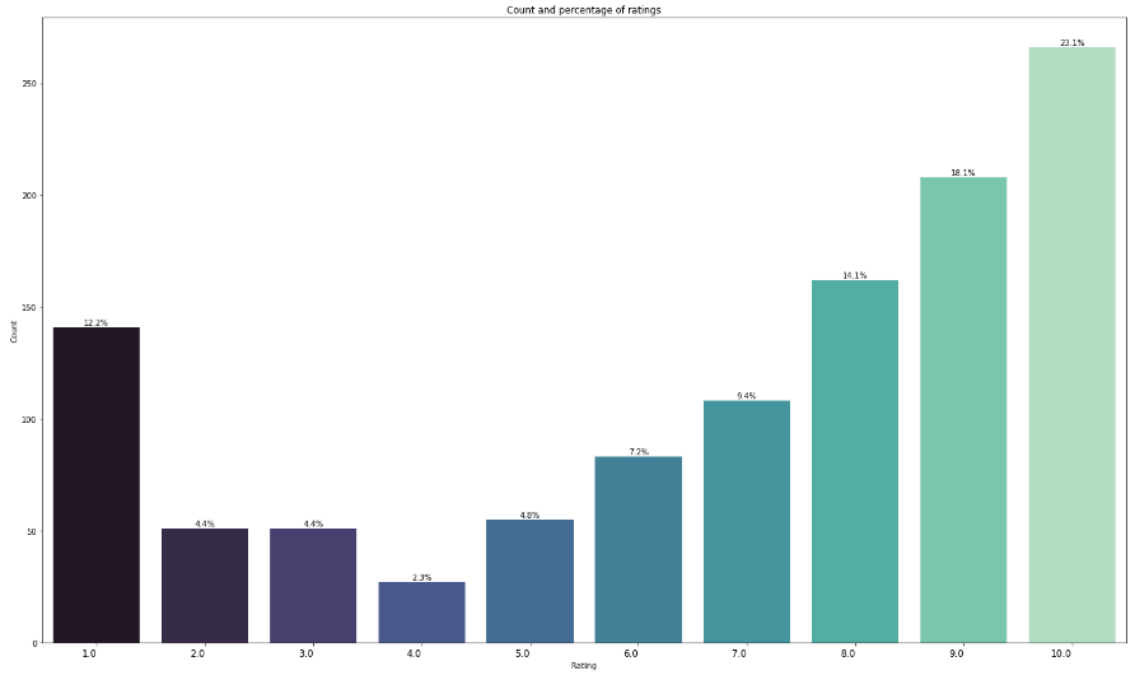
Los 10 medicamentos con más valoraciones de 10.



Los 10 medicamentos con más valoraciones de 1.



La distribución de las valoraciones.



Wordcloud de las reseñas sobre epilepsia y convulsiones.

¿Qué algoritmos de ML has utilizado? ¿Por qué?

Primeramente, he probado algoritmos de ML tradicional:

- Random Forest
- LGBM
- Cat Boost

Por último, utilicé algoritmos de DL:

- Una red neuronal con LSTM customizada
- Una red neuronal preentrenada – GloVe

Antes de entrenar cualquier modelo, al estar trabajando con texto, debemos vectorizar este texto ya que los algoritmos solo pueden trabajar con datos numéricos. Usamos el Tfidf (del inglés Term frequency – Inverse document frequency), (o sea, la frecuencia de ocurrencia del término en la colección de documentos), es una medida numérica que expresa cuán relevante es una palabra para un documento en una colección. Esta medida se utiliza a menudo como un factor de ponderación en la recuperación de información y la minería de texto. El valor tf-idf aumenta proporcionalmente al número de veces que una palabra aparece en el documento, pero es compensada por la frecuencia de la palabra en la colección de documentos, lo que permite manejar el hecho de que algunas palabras son generalmente más comunes que otras.

```
from sklearn.feature_extraction.text import TfidfVectorizer

def vectorize(data,tfidf_vect_fit):
    X_tfidf = tfidf_vect_fit.transform(data)
    words = tfidf_vect_fit.get_feature_names()
    X_tfidf_df = pd.DataFrame(X_tfidf.toarray())
    X_tfidf_df.columns = words
    return(X_tfidf_df)
```

```
tfidf_vect_ep = TfidfVectorizer(analyzer=clean)
tfidf_vect_fit_ep = tfidf_vect_ep.fit(X_ep_train['review'])
X_ep_train = vectorize(X_ep_train['review'],tfidf_vect_fit_ep)
```

Al tener datos no balanceados, mi idea era empezar por un algoritmo de Balanced Random Forest pero tuve algunos problemas con el Bootstrap que no me dejaban implementarlo, así que finalmente opté por un **Random Forest** normal (que finalmente ha sido el modelo que ha dado mejores resultados).

Teniendo en cuenta las capacidades de **Light Gradient Boosting Machine (LGBM)** en el manejo de datos con desequilibrio, fue mi segunda opción. Funcionó casi tan bien como RF, de hecho.

La aplicación de **Cat bBosting** es totalmente anecdótica: no lo había probado nunca y, además, tenía "cat" en el nombre.

En un principio, creía que las redes neuronales específicas para texto como **LSTM con embeddings**, iban a funcionar mejor que los algoritmos de ML tradicionales. Después de probar varias redes con diferente número de capas, dropouts, loss-functions… el resultado ha sido algo peor que con el RF y el LGBM.

Por último, probé una red neuronal con **embeddings pre-entrenados (GloVe**). GloVe, Global Vectors, es un modelo para la representación de palabras distribuidas. El modelo es un algoritmo de aprendizaje no supervisado para obtener representaciones vectoriales de palabras. Esto se logra asignando palabras a un espacio significativo donde la distancia entre las palabras está relacionada con la similitud semántica. Es el segundo modelo con mejores predicciones de los que he probado.

¿Cómo has realizado el entrenamiento (se ha aplicado algún tipo de cross-validation, hyperparameter tuning, etc.)?

He aplicado tanto cross-validation como hyperparameter tunning con Grid Search para encontrar los mejores parámetros.

```
rf_scores_ep = cross_val_score(rf,X_ep_train,y_ep_train.values.ravel(),cv=5)
```

```
print(rf_scores_ep)
rf_scores_ep.mean()
```

```
[0.8972973  0.88586957 0.92934783 0.89673913 0.9076087 ]

0.9033725029377203
```

```
parameters = {
    'n_estimators': [100, 1000, 2000],
    'criterion': ['gini', 'entropy'],
    'max_depth': [5, 10, 20],
    'max_features':['sqrt', 'log2', None]
}

cv = GridSearchCV(rf,parameters)
cv.fit(X_ep_train,y_ep_train.values.ravel())
print_results(cv)
```

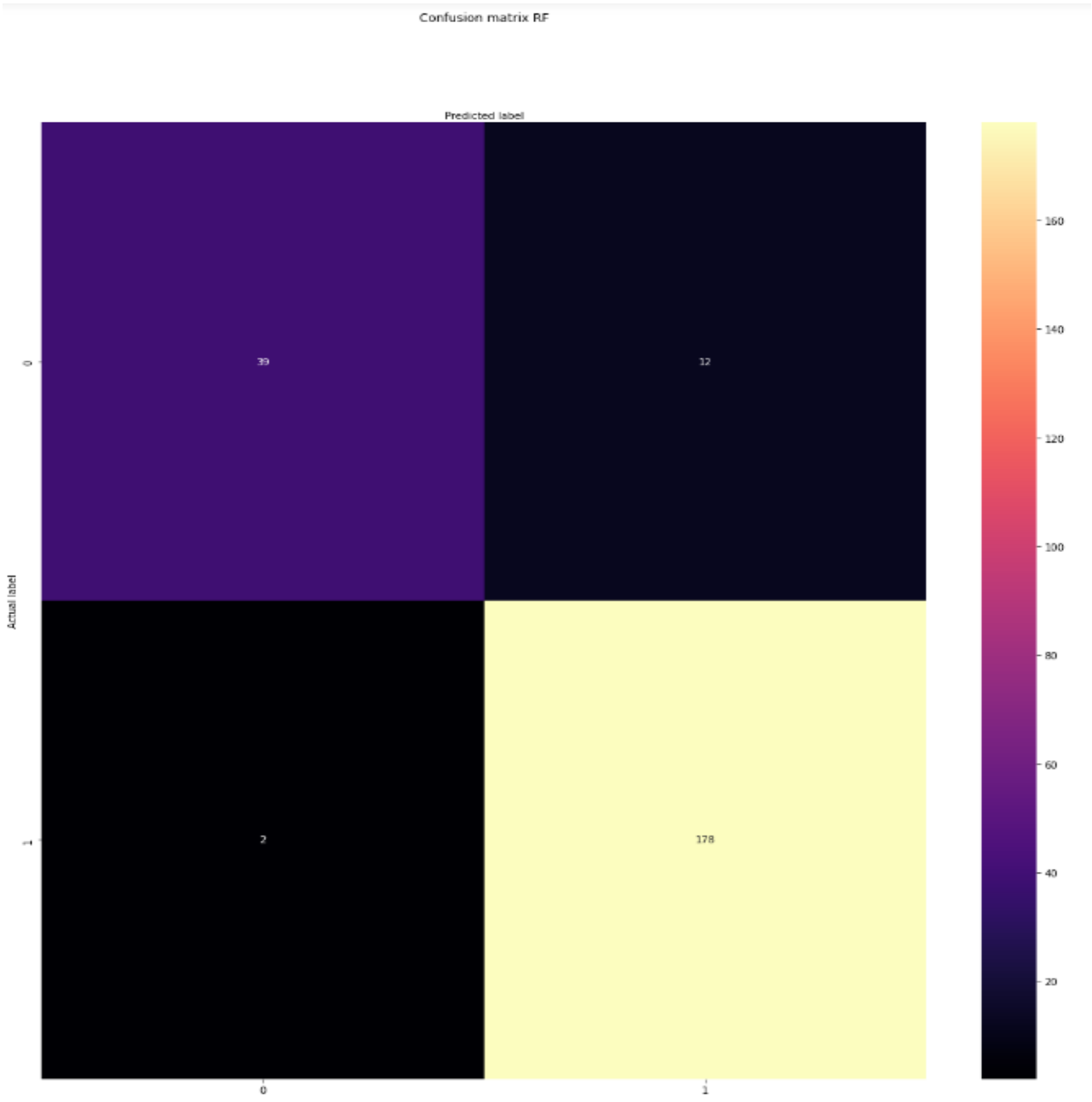
```
BEST PARAMS: {'criterion': 'entropy', 'max_depth': 20, 'max_features': None, 'n_estimators': 1000}
```

¿Qué resultados has obtenido? Incluye algunos gráficos representativos.

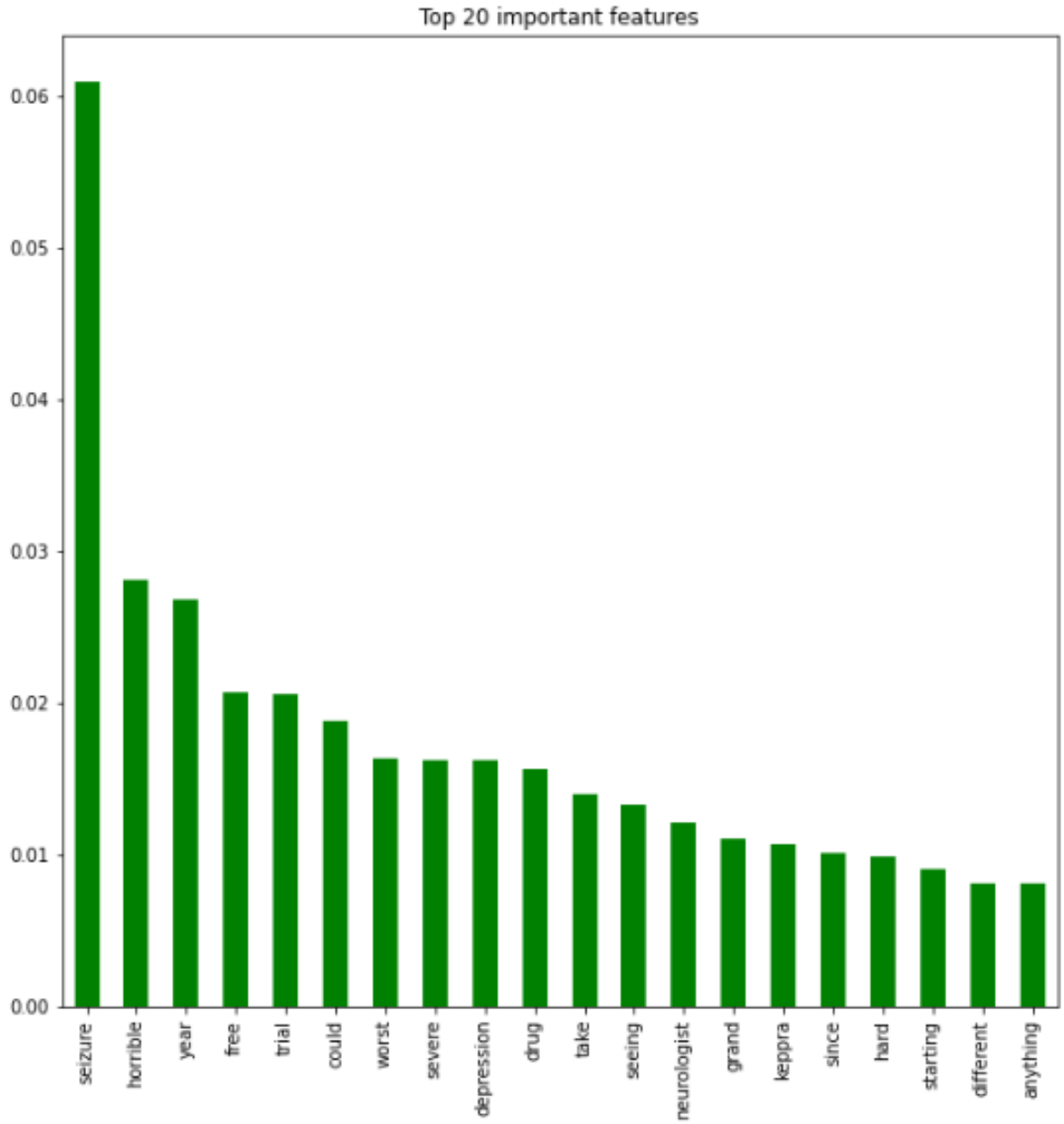
He trabajado con dos dataframes: uno con todos los datos y otro solo con los medicamentos para la epilepsia y las convulsiones. Los modelos han aprendido mejor en el dataframe pequeño y los mejores modelos han sido RF y la red neuronal con LSTM y GloVe.

Model	Dataset	Accuracy
RF	whole	80.4
RF	epilepsy	93.9
LGBM	whole	85.4
LGBM	epilepsy	92.2
CB	whole	81.3
CB	epilepsy	91.7
LSTM custom 1	whole	80.8
LSTM custom 1	epilepsy	77.9
LSTM pretrained	whole	79.2
LSTM pretrained	epilepsy	87

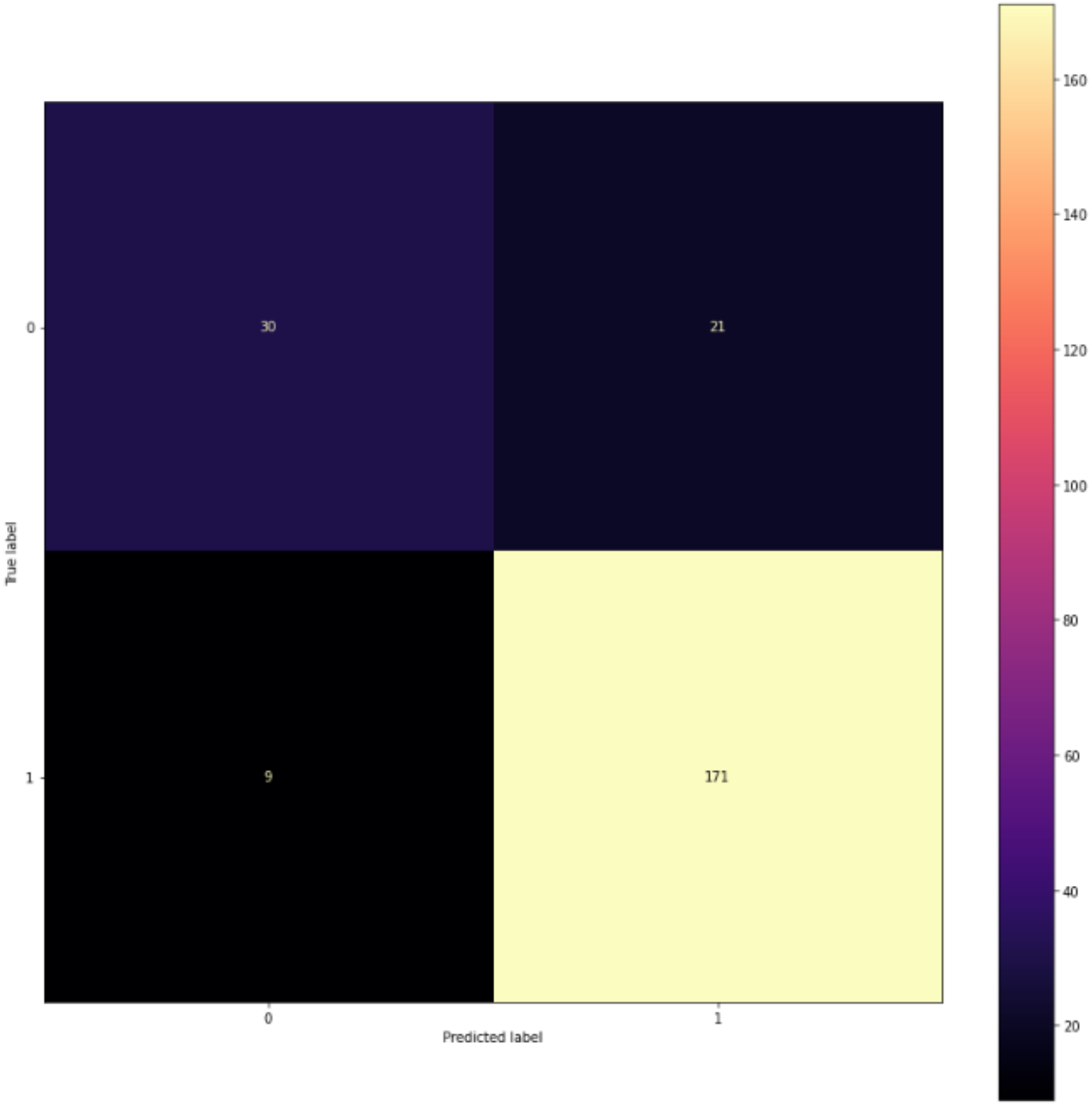
Confusion matrix del RF



Feature importance del RF



Confusion matrix de la red neuronal LSTM con GloVe



Conclusiones.

Al no ser un conjunto de datos complejo y tener bastantes features que pueden ayudar a la predicción, nos ha funcionado mejor un algoritmo de ML tradicional que una red neuronal.

Hemos conseguido mejor rendimiento con un dataframe bastante pequeño (de algo más de 1000 observaciones) que con el dataset general (de casi 200000 observaciones, del cual habíamos cogido una muestra de 9000).

¿Has logrado el objetivo que te habías propuesto? En caso negativo, ¿qué ha fallado?

Me hubiera gustado conseguir valores más altos de predicción, pero haber llegado al 93% de accuracy con un RF está bastante bien al no haber tenido en cuenta el desbalanceo.

Describe brevemente cuáles podrían ser los siguientes pasos a realizar para mejorar el proyecto.

Antes de todo, habría que hacer un análisis más exhaustivo del primer dataset para limpiarlo perfectamente. Me di cuenta que había algunos campos en la feature “condition” que no eran nulos, por lo tanto no se veía en la búsqueda de NaNs, pero que eran incorrectos, pero no había suficiente tiempo para arreglarlo ya que un mismo medicamento se usaba para distintas “conditions”.

También se podrían hacer más gráficos de datos interesantes para tener mejor insight del conjunto.

Al vectorizar las palabras, se podrían usar parejas de palabras en lugar de palabras sueltas para mejorar el entreno del modelo con más contexto, pero era muy lento computacionalmente hablando.

Seguramente, aplicar un Balanced RF o haber hecho undersampling de la clase mayoritaria ayudaría a las predicciones.

Además, me gustaría haber usado Transformers (como BERT) para mejorar las predicciones de las redes neuronales, pero todo lo que probé me dio error y no supe saber el porqué.