Trabajo Práctico 2 : Críticas cinematográficas

Preprocesamiento

Con el objetivo de encarar un análisis de emociones de un conjunto de críticas cinematográficas, primeramente tuvimos que elegir un método para vectorizarlas y por ende, sean utilizadas por los diferentes modelos. En este caso optamos por una vectorización por Bag of Words mediante CountVectorizer y TfidfVectorizer (frecuencia de término – frecuencia inversa de documento). Para probar el rendimiento de ambos, decidimos utilizar Naive Bayes de referencia, dentro de la notebook se puede ver claramente que CountVectorizer tuvo una peor performance en todas las métricas salvo la precisión. Por esto mismo, decidimos utilizar TfidfVectorizer para el resto de entrenamientos.

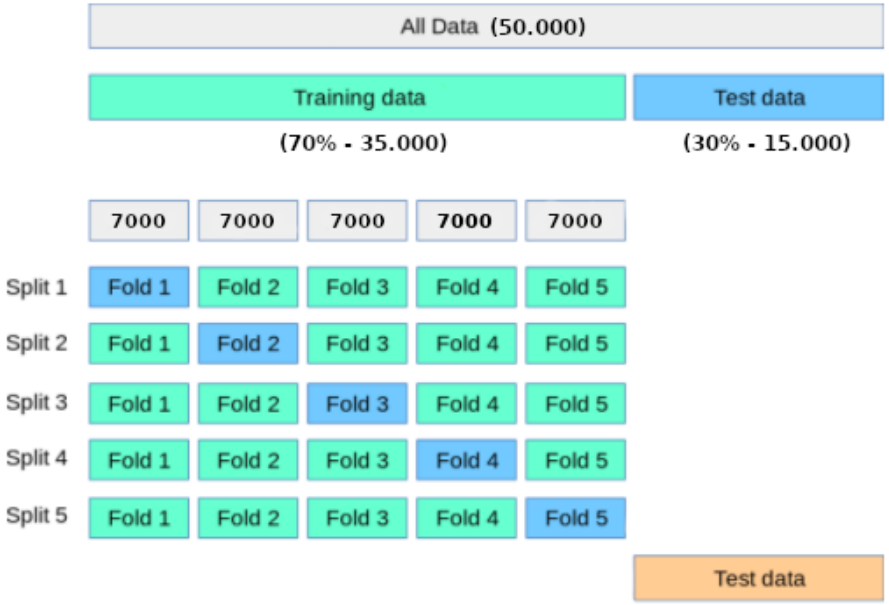
Otro aspecto a tener en cuenta fueron las stopwords, palabras que no aportan información al set sino más bien incorporan ruido al mismo, para esto utilizamos Natural Language Toolkit (NLTK) que nos proporcionaba una lista predefinida de este tipo de palabras en Español y pudimos incorporarlas como parámetro dentro de las funciones de vectorización. Finalmente para realizar los entrenamientos, dividimos al dataset en uno de entrenamiento y testeo con una relación de 70-30, y también modificamos la columna de sentimiento por 0 (negativo) e 1 (positivo), para facilitar el entrenamiento sobre una clasificación.

Entrenamiento

En lo que respecta al entrenamiento de los modelos tuvimos algunos lineamientos generales. Como explicamos anteriormente, elegimos la frecuencia inversa de documento como el vectorizador predilecto para todos los modelos, para esto hicimos utilización de un pipeline que incluía dicho vectorizador y el modelo entrenado, ésto facilitó después poder hacer predicciones con un nuevo conjunto (test para kaggle).

Así mismo a la hora de realizar Cross Validation, utilizamos Randomized Search, el cual tiene una menor complejidad computacional y por lo tanto pudimos armar una grilla de parámetros un poco más exhaustiva. Otros tres aspectos que se mantienen en todos los entrenamientos, es la búsqueda de un buena reducción de la dimensionalidad mediante el parámetro max_features del vectorizador, la focalización de los entrenamientos sobre la métrica F1 (objetivo para la competencia Kaggle) y por último, la estratificación de x_train en 5 folds, decisión que se debió especialmente a que no había una abundancia de datos y consideramos que realizar CV con un test de 15.000

registros era un buen número.



Algunas excepciones a la hora de entrenar se dieron en Naive Bayes y la red neuronal. En el primer caso, decidimos utilizar un Grid Search, debido a que no requiere una gran complejidad y había un único parámetro para optimizar, y vectores de máxima dimensión, ya que nos parecía acorde al modelo. Por otro lado, para la red neuronal decidimos utilizar la métrica Accuracy propia ya que tuvimos mejores resultados que realizado un F1 manual, ya que para el caso de keras no permite utilizar las métricas de scikit learn.

Parámetros de los modelos

Naive Bayes

Para el caso de Naive Bayes se optó por un GridSearch con los vectores de tamaño máximo, debido a que no es un modelo computacionalmente exigente y que no encontraría ventajas en reducir la dimensión de sus datos.

Hiperparámetros conseguidos por Cross Validation
'clf__alpha': 0.55

- **'alpha'**: para controlar el sobreajuste, ajustar la importancia de las características y optimizar el rendimiento promedio mediante validación cruzada, logrando un equilibrio entre capacidad de ajuste y generalización del modelo.

XGBoost

Mejor reducción conseguida por Cross Validation
'preprocess__max_features': 75000

Hiperparámetros conseguidos por Cross Validation	
'clf__colsample_bytree'	0.7906733956233418
'clf__gamma'	4.861702447101156
'clf__learning_rate'	0.24257500795548073
'clf__max_depth'	5
'clf__n_estimators'	500
'clf__subsample'	0.8796686986457685

- **'colsample_bytree'**: para manejar la proporción de columnas que se utilizarán en cada árbol durante el entrenamiento.
- **'n_estimators'**: para determinar el número de árboles en el ensamble, para rinda y no tarde demasiado tiempo.
- **'gamma'**: para regular la complejidad del modelo. Un valor más alto de 'gamma' impone una partición más conservadora, lo que ayuda a evitar divisiones que no aporten.
- **'learning_rate**:' para controlar la tasa de aprendizaje del modelo.
- **'subsample'**: para controlar la fracción de muestras utilizadas para entrenar cada árbol individual.
- **'max_depth'**: para limitar la profundidad máxima de los árboles en el ensamble. Al controlar la complejidad de los árboles, 'max_depth' nos ayuda a evitar un sobreajuste excesivo al conjunto de entrenamiento.

Random Forest

Mejor reducción conseguida por Cross Validation
'preprocess__max_features': 10000,

Hiperparámetros conseguidos por Cross Validation	
'clf__n_estimators'	200
'clf__min_samples_split'	3
'clf__min_samples_leaf'	4
'clf__max_features'	'log2'
'clf__max_depth'	None

- **'max_features'**: para controlar la cantidad de características consideradas al dividir cada nodo del árbol. Al limitar la cantidad de características disponibles en cada división, 'max_features' nos permite reducir la dimensionalidad y la complejidad del modelo, evitando así el sobreajuste.
- **'min_samples_leaf'**: para establecer el número mínimo de muestras requeridas en cada hoja del árbol. Al establecer un valor más alto, podemos evitar particiones que generen hojas con un número muy bajo de muestras, lo que podría llevar a un sobreajuste.
- **'min_samples_split'**: para especificar el número mínimo de muestras requeridas para realizar una partición en un nodo del árbol. Al establecer un valor más alto, nos aseguramos de que las particiones solo se realicen cuando haya suficientes muestras para obtener una estimación confiable.
- **'n_estimators'**: para determinar el número de árboles en el ensamble, para rinda y no tarde demasiado tiempo.
- **'max_depth'**: para limitar la profundidad máxima de los árboles en el ensamble. Al controlar la complejidad de los árboles, 'max_depth' nos ayuda a evitar un sobreajuste excesivo al conjunto de entrenamiento.

Ensamble Voting

Mejor reducción conseguida por Cross Validation
'preprocess__max_features': 20000,

Hiperparámetros conseguidos por Cross Validation (Random Forest)	
'clf__rfc__n_estimators'	150
'clf__rfc__min_samples_split'	3
'clf__rfc__min_samples_leaf'	7
'clf__rfc__max_features'	'sqrt'
'clf__rfc__max_depth'	20
'clf__rfc__criterion'	'entropy'

- **'criterion'**: para determinar la función de calidad utilizada para evaluar la calidad de una división en los árboles de decisión.Hemos seleccionado 'entropy' como el criterio para evaluar la calidad de las divisiones en los árboles de decisión debido a su capacidad para reducir la incertidumbre y su sensibilidad a las distribuciones desequilibradas.
- **'min_samples_leaf'**: para establecer el número mínimo de muestras requeridas en cada hoja del árbol. Al establecer un valor más alto, podemos evitar particiones que generen hojas con un número muy bajo de muestras, lo que podría llevar a un sobreajuste.
- **'min_samples_split'**: para especificar el número mínimo de muestras requeridas para realizar una partición en un nodo del árbol. Al establecer un valor más alto, nos aseguramos de que las particiones solo se realicen cuando haya suficientes muestras para obtener una estimación confiable.
- **'max_depth'**: para limitar la profundidad máxima de los árboles en el ensamble. Al controlar la complejidad de los árboles, 'max_depth' nos ayuda a evitar un sobreajuste excesivo al conjunto de entrenamiento.
- **'n_estimators'**: para determinar el número de árboles en el ensamble, para rinda y no tarde demasiado tiempo.
- **'max_features'**: para controlar la cantidad de características consideradas al dividir cada nodo del árbol. Al limitar la cantidad de características disponibles en cada división, 'max_features' nos permite reducir la dimensionalidad y la complejidad del modelo, evitando así el sobreajuste.

Hiperparámetros conseguidos por Cross Validation (Naive Bayes)	
'clf__nb__force_alpha'	True
'clf__nb__alpha'	0.26

- **‘alpha’**: para controlar el sobreajuste, ajustar la importancia de las características y optimizar el rendimiento promedio mediante validación cruzada, logrando un equilibrio entre capacidad de ajuste y generalización del modelo.

Hiperparámetros conseguidos por Cross Validation (Logistic Regression)	
'clf__lr__penalty'	'l2'
'clf__lr__max_iter'	400
'clf__lr__C'	4.900000000000004

- **‘penalty’**: para evitar el sobreajuste y ajustar la relevancia de las características. Mediante la validación cruzada, se busca un equilibrio entre el ajuste del modelo y su capacidad de generalización, asegurando un rendimiento óptimo al considerar la importancia relativa de las características.
- **‘max_iter’**: para determinar un número máximo de iteraciones permitidas durante el entrenamiento del modelo.
- **'C'**: para determinar el grado de tolerancia a los errores en el modelo. Un valor más alto de 'C' permite un ajuste más ajustado a los datos de entrenamiento, mientras que un valor más bajo permite una mayor regularización y un modelo más simplificado.

Red Neuronal

La arquitectura elegida para la implementación de la red neuronal surge a partir de la investigación y análisis de diversas fuentes como papers, artículos de blogs, foros, etc. donde hicimos especial énfasis en que funciones de pérdida, optimizadores, tipo de capas, cantidad de capas y cantidad de neuronas eran las más eficientes para el análisis de emociones planteado.

A partir de esta búsqueda e investigación surgieron variedad de posibilidades en cuanto arquitectura de la red, en particular, dos nos llamaron particularmente la atención

1 - Bidireccional LTSM con capa embedding, la capa “embedding” lo que permite es vectorizar el lenguaje y ltsm bidireccional hace, además, un análisis inverso de la entrada por lo que permite detectar otros tipos de relaciones.

2- Capas densas ocultas con dropout. Lo que nos permite esto es tener nuestra propia vectorización por fuera de la capa embedding y a la vez insertar capas dropout las cuales desactivan neuronal aleatoriamente, esto nos permite regularizar nuestro modelo y evitar el sobreajuste.

Tras evaluar estos modelos entrenándolos con diferentes hiperparámetros decidimos descartar el modelo 1 ya que su complejidad hacía imposible conseguir un modelo consistente, sucedía que por cuestiones de hardware no podíamos entrenarlo con una cantidad de features que logre sacar ventajas en cuanto a su competencia.

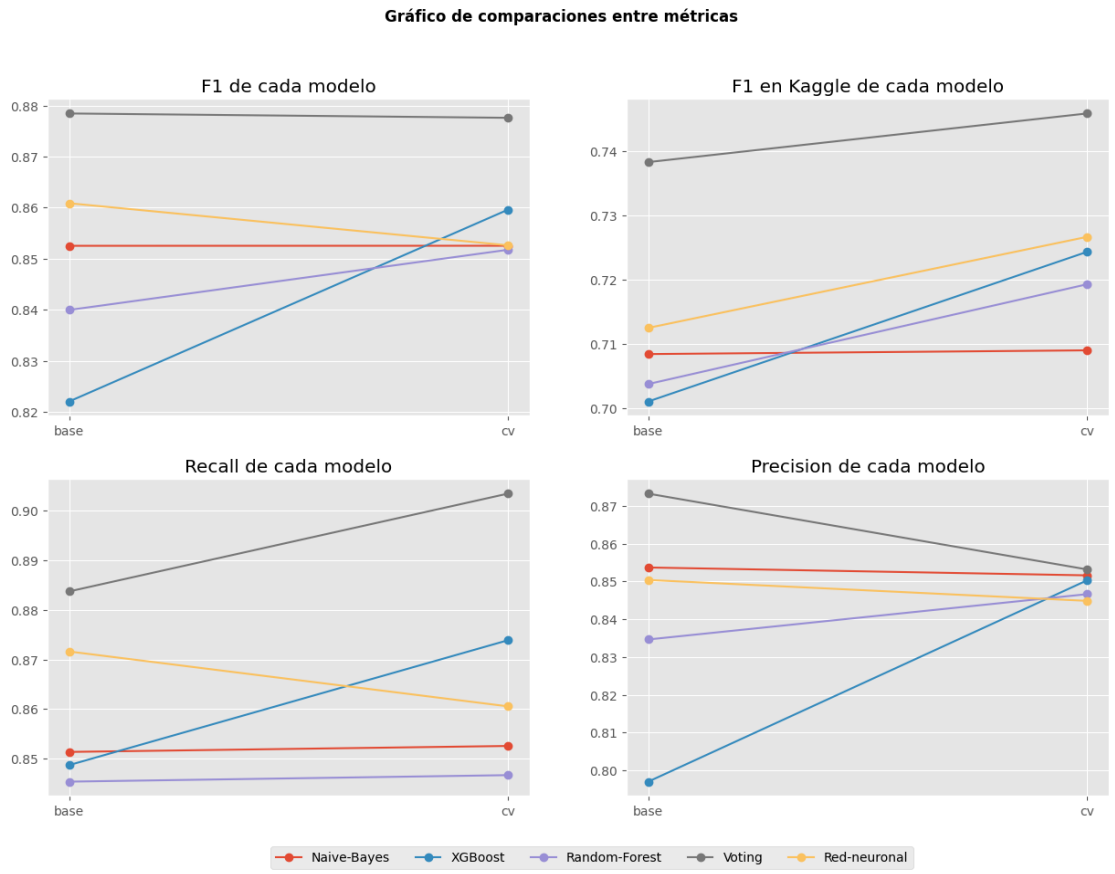
En profundidad sobre el modelo 2, este resultó ser mucho más simple que sus antecesores por lo que se logró entrenar con una gran cantidad de features, lo que nos permite encontrar muchas más relaciones entre ellos, y así conseguir una mejor predicción. A la vez, esta simpleza facilitó la búsqueda de hiperparámetros através de cross validation, por lo que se optimizó el tamaño de batch y las épocas de la red neuronal.

Se decidió implementar un pipeline para poder hacer predicciones y entrenamientos de la red sin pasos intermedios de preparación. También dentro del pipeline se le aplicó Earlystopping para evitar el sobreajuste.

La mayoría de las fuentes consultadas coincidían en dos fundamentos que son, la utilización de “Adam” como función de optimización y binary crossentropy como función de pérdida. Debido a su eficiencia en la convergencia, manejo de gradientes dispersos y regularización incorporada Adam es utilizado en análisis de emociones, en lo que respecta binary crossentropy se decidió utilizar debido al carácter binario del problema, es decir, review positiva o negativa.

Hiperparámetros conseguidos por Cross Validation	
'model__epochs'	20
'model__batch_size'	64

Resultados de los modelos



Método	Base			Cross Validation		
Métrica	**F1**	**Recall**	**Precisión**	**F1**	**Recall**	**Precisión**
Naive Bayes	0.85253	0.85133	0.8537	0.85254	0.85253	0.85162
XG Boost	0.82203	0.84867	0.79702	0.8596	0.87387	0.85029
Random Forest	0.83996	0.84533	0.83465	0.85175	**0.90347**	0.84664
Voting	**0.87846**	**0.88373**	**0.87325**	**0.87761**	0.84664	**0.85319**
Red Neuronal	0.86087	0.8716	0.8504	0.85263	0.86053	0.84487
Puntaje Kaggle	**0.75111**					

Conclusiones

Al analizar el resultado de los modelos se pueden realizar varias conclusiones. Sobre los modelos se puede afirmar que Voting es por lejos el mejor modelo que se adecuó al problema ya que tuvo unas muy buenas métricas en todos los casos, aún así sin la búsqueda de hiperparámetros, los cuales mejoraron todas su métricas exceptuando el recall. Así mismo fue el que mejor generalizó y obtuvo la mejor métrica en la competencia de Kaggle. Otro aspecto a destacar, que se puede ver a simple vista que en el caso de Naive Bayes no hubo mejora alguna entre el modelo base y el hiperparametrizado, probablemente porque no hubo grandes diferencias entre sus parámetros, no es un modelo que admita una gran capacidad de tuning.

Así mismo podemos concluir que al utilizar Randomized Search, los resultados fueron un poco dispersos, incluso algunos empeoraron en alguna de sus métricas, seguramente por la aleatoriedad de buscar puntos que tiene el método. Sin embargo, pudimos constatar por Kaggle, que todos los modelos que pasaron por un cross validation tuvieron una mejor performance. Aún así nos parece que un entrenamiento por Grid Search hubiese tenido unos resultados mejores, al menos en las pruebas locales, pero al utilizar grillas bastante exhaustivas de hiperparámetros y contar con una gran dimensionalidad de los datos, esto fue bastante difícil de conseguir de acuerdo a nuestro nivel de cómputo. Es por esto que creemos que con una mejor preprocesamiento de los datos y por lo tanto, con la posibilidad de utilizar Grid Search, hubiésemos tenido mejores resultados.