TP2: Críticas Cinematográficas - Grupo 11

Introducción

El problema que se abordó en este trabajo práctico consistía en clasificar un listado de críticas cinematográficas como positivas o negativas mediante un análisis de sentimiento. Para esto contábamos con un registro de 50.000 reseñas para utilizarse durante el entrenamiento, que además de contener el texto de la reseña, ya venían etiquetadas; y otro conjunto de 8.599 reseñas que solo contenían el texto y que debían ser correctamente clasificadas.

Al analizar inicialmente el conjunto de datos, la características más particular notada fue que el conjunto contenía críticas en múltiples idiomas, principalmente en español e inglés, con aproximadamente el 96% de las críticas en español. El conjunto de entrenamiento con las críticas ya etiquetadas estaba perfectamente balanceado, con 25.000 críticas positivas y 25.000 negativas.

Previo al entrenamiento de los modelos predictivos, se decidió realizar una tarea de preprocesamiento del texto de las críticas, para limpiar los datos y poder homogeneizar la información de la que los modelos iban a aprender. Para esto se utilizaron técnicas como regex para mejorar la separación entre enunciados (por ejemplo, habían bastantes enunciados en los que la última palabra de uno estaba separada de la primera palabra de otro mediante símbolos como `!!!`), esto con la intención de mejorar el proceso de tokenización. Se removieron las tildes y caracteres especiales, y se reemplazaron las mayúsculas por minúsculas.

Finalmente se hizo uso de varias utilidades de la librería `spaCy`, de la cual se utilizaron dos modelos diferentes, uno para trabajar con las críticas en inglés y otro para las críticas en español. Con estos modelos se filtraron stopwords, signos de puntuación, números, etc., además de que se extrajeron características de cada token para enriquecer el análisis de los textos, como por ejemplo, se hizo uso del tipo de part-of-speech de cada token para distinguir usos de una misma palabra en diferentes contextos así como también reconocimiento de entidades, para diferenciar los nombres de las mismas del resto de palabras en cada texto. También se utilizó lematización para reducir las diferentes variantes de una misma palabra a una base común. Finalmente, también se llegó a realizar un manejo de las negaciones, anteponiendo el prefijo “NO_” a aquellas palabras que estuvieran después de una negación en un enunciado.

Cuadro de Resultados

Modelo	F1-Score	Precision	Recall	Accuracy	Kaggle
Bayes Naive	0,86	0,85	0,87	0,8609	0,7567
Random Forest	0,82	0,82	0,82	0,8230	0,7317
XGBoost	0,85	0,85	0,85	0,8486	0,7294
Red Neuronal	0,86	0,86	0,86	0,8579	0,6924
Ensamble	0,87	0,87	0,87	0,8699	0,7379

Descripción de Modelos

Naive Bayes

A pesar de que este era el modelo más básico, terminó siendo el que arrojó mejores resultados. Este modelo no tiene una amplia variedad de hiperparámetros, por lo que la versión final terminó siendo la versión por defecto. Atribuímos las mejoras en el desempeño de este modelo a las mejores que logramos conseguir durante el proceso de preprocesamiento. En este caso el modelo utiliza la versión más compleja del preprocesamiento de texto, que incluye regex, lematización, filtrado de stop words, manejo de negaciones, IOB encoding y NER.

Aún sin la posibilidad de configurar los hiperparámetros y la complejidad del dataset, el modelo no presentó muchas señales de overfitting, como sí sucedió con los demás modelos. Esta es posiblemente una de las razones por la cual terminó arrojando resultados tan buenos en la competencia también.

Random Forest

En el caso de este modelo, es el que peor rendimiento tuvo entre todos. Tuvimos que recurrir a agregar features adicionales, como el conteo de negaciones y adjetivos, para que pudiera mejorar su rendimiento. Pese a sus pobres resultados, al menos fue el modelo que más consistencia mostró entre las métricas del conjunto de entrenamiento y el de test. Respecto al preprocesamiento de texto, utiliza las mismas técnicas que Naive Bayes. Finalmente el mejor

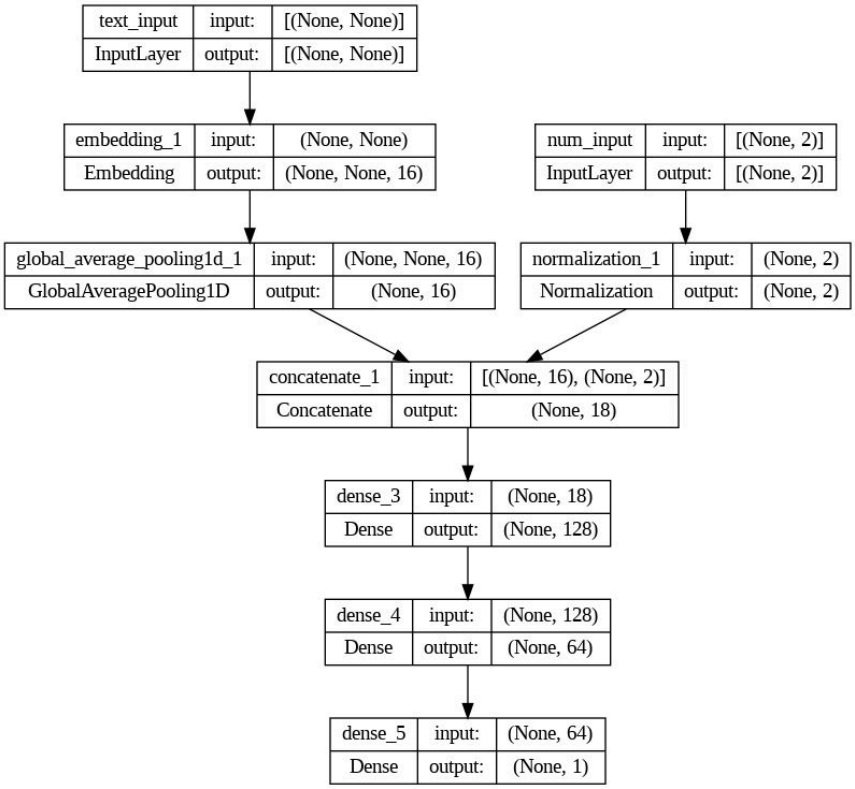
modelo tiene los siguientes híperparametros: *{n_estimators=1000, criterion="gini", max_depth=15, min_samples_split=10, min_samples_leaf=8, oob_score=True, n_jobs=-1}.*

XGBoost

De entre los modelos clásicos este fue el que más overfitting presentó, y por lo tanto, el que más nos costó configurar, especialmente considerando la extensiva cantidad de hiperparámetros que pueden ser configurados para este modelo. Finalmente se optó por utilizar una versión simplificada del preprocesamiento que ayudara al modelo a generalizar mejor. Esta incluye solamente lematización, eliminación de stop words, regex y eliminación de acentos y reemplazo de mayúsculas. Los mejores híperparametros son: *{n_estimators=500, learning_rate=0.3, reg_alpha=0.2, reg_lambda=0.2, gamma=4, max_depth=2, n_jobs=-1, objective='binary:logistic'}.*

Red Neuronal

Para la construcción de las redes neuronales se consideraron varios modelos de red posibles, pero los tres más destacados fueron un modelo de múltiples entradas, que tomaba los features numéricos que también fueron utilizados para el random forest, además del texto de los comentarios.



Otro de los modelos que arrojó buenos resultados fue una red neuronal convolucional, que tomaba el resultado de los vectores de embeddings y aplicaba diferentes filtros y reducía la dimensionalidad del problema para luego alimentar a múltiples redes densas sencillas. También se probó utilizando métodos de pooling máximo o promedio en vez y junto con las capas convolucionales.

Sin embargo, el diseño que terminó dando mejor resultado fue una red neuronal recurrente con dos capas del tipo GRU, con 16 neuronas ambas, la primera retornando las secuencias a la segunda, que es la que finalmente se encargaba de terminar el proceso de transformación de secuencia a vector para alimentar una capa densa final antes de la capa con una sola neurona para la decisión final. Este tipo de redes son de utilidad para tareas como el procesamiento de lenguaje natural, ya que la capacidad de “recordar” de las neuronas les permite tener un contexto adicional cuando se analizan secuencias de texto, en las que la utilización de determinadas palabras puede cambiar en dependencia del contexto y de las palabras junto las cuales es utilizada. Se probaron varias variantes de redes recurrentes, sin embargo si se aumentaba mucho la complejidad de la misma siempre se termina tendiendo a overfitting.

Cabe destacar que también se probó con la utilización de embeddings preentrenados como BERT o DistilBERT, sin embargo, los altos tiempos de entrenamiento con estos embeddings y

problemas técnicos con la utilización de las métricas elegidas para la evaluación del modelo durante el entrenamiento impidieron la correcta implementación de estos modelos.

Ensamble

Para este modelo se hizo un ensamble con los mejores modelos de XGBoost, RandomForest y NaiveBayes entrenados previamente. El tipo de ensamble que se utilizó fue stacking (de la libreria de scikit-learn) ya que fue el que dio la mejor performance, luego de probar también stacking y voting con distintos modelos.
Si bien se entrenó el ensamble con cross validation, este tipo de ensamble en particular no tiene híperparametros por lo que no hizo falta un RandomSearch o GridSearch para buscar los mismos. Finalmente, los híperparametros para XGBoost resultaron ser: *{n_estimators=500, learning_rate=0.3, reg_alpha=0.4, reg_lambda=0.3, gamma=6, max_depth=2}.*
Por otro lado, se tuvo que volver a encontrar híperparametros para XGBoost para poder unificar el dataset utilizado para el entrenamiento de los modelos ensamblados, ya que XGBoost resultó mejor con otro preprocesamiento distinto al de RandomForest y NaiveBayes. El ensamble que mejor resultados dio utiliza la versión más compleja del preprocesamiento de texto, que incluye regex, lematización, filtrado de stop words, manejo de negaciones, IOB encoding y NER.

Métodos Adicionales

Además de los modelos establecidos por la consigna, se probaron otros como SVM y KNN, sin embargo arrojaron resultados pobres con los rangos de hiperparámetros probados, y los extensos tiempos de entrenamiento de ambos modelos impidieron que se pudiera seguir experimentando mucho con los mismos para ajustarlos y agregarlos al ensamble.

Por otra parte, también se utilizaron diferentes métodos para la búsqueda de los hiperparámetros de los modelos convencionales, novedosos para nosotros y distintos a los aprendidos en clases. Además del clásico random search y grid search, que fueron los métodos que finalmente nos terminaron arrojando los mejores resultados, probamos métodos bayesianos para la búsqueda de estos valores, así como métodos “evolutivos” como los sugeridos por el libro de la bibliografía oficial, como por ejemplo, el que provee esta librería.

Conclusiones generales

A lo largo de este trabajo logramos comparar el proceso de desarrollo y resultados de una gran cantidad de versiones de diferentes algoritmos de clasificación utilizados para análisis de sentimientos como algoritmos de boosting, redes neuronales y ensambles. Durante el proceso

de elaboración el principal obstáculo fue constantemente lidiar con el overfitting, que hacía que nuestros modelos arrojaran buenos resultados (en cuanto a métricas) en los sets de los que disponíamos para el entrenamiento, pero que a la hora de evaluar los resultados reales en la competencia de Kaggle no daban los resultados esperados.

A pesar de que intentamos con más de 60 variantes diferentes de los modelos, nos sorprende que ninguno haya logrado superar el desempeño del algoritmo más “básico” de todos, Naive Bayes, un modelo que asume proposiciones que en realidad son falsas, como por ejemplo, la independencia entre las palabras de una crítica y su orden, no mostró señales claras de overfitting, y logró generalizar mejor que los demás modelos. Ni siquiera a través del uso de técnicas de vectorización como las embeddings en redes neuronales, que fueron diseñadas específicamente para enriquecer el proceso de conversión de texto a vectores encontrando relaciones entre las diferentes palabras, logramos mejorar este rendimiento.

Atribuímos gran parte de la mejora en los resultados de la competencia a la labor de las tareas de preprocesamiento, cambio que se vió reflejado en los resultados de todos los modelos, pero principalmente en los de Naive Bayes.

Consideramos que haber podido solucionar de los problemas de overfitting con anterioridad nos hubiese permitido destinar más tiempo a optimizar la búsqueda de mejores rangos de hiperparámetros en los modelos convencionales, y probar con diferentes arquitecturas en los modelos de redes neuronales, incluso haber persistido en el intento por incorporar nuevas técnicas con el potencial de mejorar significativamente los resultados como lo eran los embeddings preentrenados. Sin embargo, los altos tiempos de entrenamiento de la mayoría de los modelos, así como los límites de uso de los ambientes de Colab, restringieron en cierta capacidad los esfuerzos por buscar nuevas soluciones para lidiar con el overfitting.

Fue justamente el modelo más simple y rápido de entrenar, Naive Bayes, el que resultó más útil en relación con el desempeño obtenido, no obstante, nos hubiese gustado persistir un poco más con la exploración de formas para mejorar los demás modelos más complejos para ver si por el diseño de los mismos, y sus capacidades para encontrar los patrones inherentes en el conjunto de entrenamiento, tienen la capacidad de producir mejores resultados de los que finalmente arrojaron. Por ejemplo, ya que el preprocesamiento representó una gran mejora en sí mismo, quizá hubiesemos explorado nuevas formas de preprocesamiento que hubiesen favorecido a los modelos más complejos por sobre el más simple. Consideramos que se pudo haber probado con aún más variantes de los modelos utilizados, ya sea expandiendo el espacio de hiperparámetros buscados, o incluso variaciones de los modelos de ensamble utilizando modelos significativamente diferentes como por ejemplo, un ensamble de las arquitecturas de redes neuronales que mejores resultados dieron.

Si bien Naive Bayes fue el modelo con mejor rendimiento y que tuvo poco overfitting, permanecemos un poco escépticos ante la posibilidad de utilizarlo en producción para una tarea de análisis de sentimiento como esta. Nos sorprendió la dificultad que enfrentamos para mejorar modelos que pensábamos naturalmente mejores en este tipo de tareas, especialmente las redes neuronales recurrentes, que son aclamadas por su desempeño en estas tareas.

En definitiva, lo aprendido durante este trabajo práctico extiende un poco sobre lo aprendido en el anterior sobre el proceso de desarrollo de inteligencia artificial y la constante iteración y experimentación que este conlleva. Nos resultó particularmente interesante el encarar una tarea de procesamiento del lenguaje y comprender la forma en cómo las computadoras pueden "entender" el lenguaje natural y realizar tareas como el análisis de sentimiento. Coincidimos en que la parte del entrenamiento de redes neuronales fue particularmente entretenida, ya que es un tema que habíamos venido escuchando ya con anterioridad y especialmente ahora con el surgir de nuevas arquitecturas como transformers y los large language models con las que nos hubiese gustado experimentar.

Tareas Realizadas

Integrante	Principales Tareas Realizadas	Promedio Semanal (hs)
Carlos Castillo	Redacción de informe Preproceso de datos Entrenamiento XGBoost Entrenamiento RNN Entrenamiento Naive Bayes	16
Juan Pablo Destefanis	Redacción de informe Preproceso de datos Entrenamiento Random Forest Entrenamiento ensambles Entrenamiento Naive Bayes	5
Celeste Gómez	Redacción de informe Preproceso de datos Entrenamiento XGBoost Entrenamiento Naive Bayes	7