TP2: Críticas Cinematográficas - Grupo 05

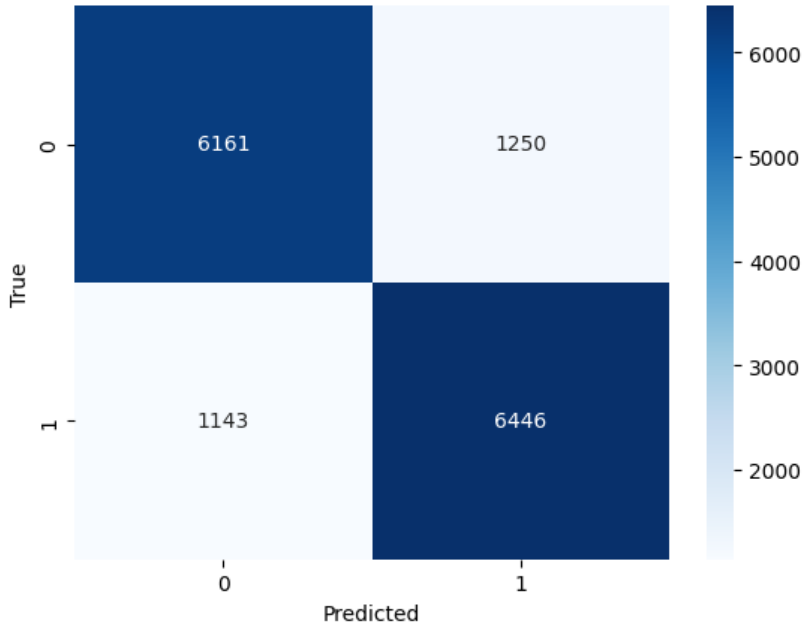
Introducción

En este trabajo práctico se pidió armar al menos 5 modelos de tipo Bayes Naïve, Random Forest, XGBoost, red neuronal y un ensamble de 3 de los modelos anteriores. El objetivo principal fue poder clasificar de un dataset de críticas cinematográficas cuáles son positivas y cuáles negativas.
El dataset test está constituido por 2 columnas y el train por 3 columnas: ID, la reseña de la película y sentimiento (positivo o negativo). A diferencia del anterior trabajo y los datasets que estuvimos viendo en la primera parte del cuatrimestre, el dataset tiene la particularidad de contar con la mayoría de su información en una sola columna de la cual tuvimos que hacer un preprocesamiento para poder empezar a construir modelos.
Elegimos un modelo de TF-IDF para vectorizar el texto de las reseñas para así reducir la cantidad de palabras repetidas con mucha frecuencia y minimizar la complejidad del dataset que creímos no aportaba información relevante. Luego, utilizamos el nuevo dataset procesado y reducido en el resto del trabajo. No tomamos ninguna suposición sobre el dataset ya que no contábamos con las herramientas para predecir el contenido del mismo de antemano.

Cuadro de Resultados

Modelo	F1-Test	Precision Test	Recall Test	Accuracy	Kaggle
Bayes Naive	0.85	0.85	0.845	0.85	0.75479
Random Forest	0.845	0.845	0.85	0.85	0.73735
XgBoost	0.845	0.845	0.845	0.85	0.69257
Red Neuronal	0.86	0.86	0.86	0.86	0.72824
Ensamble	0.835	0.835	0.835	0.84	0.71312

El mejor predictor resultó ser Bayes Naive.



Descripción de Modelos

- **Bayes Naïve:**

El modelo de Bayes Naïve basa su algoritmo en el teorema de Bayes para realizar clasificación. El modelo calcula la probabilidad de que una instancia pertenezca a una clase determinada mediante la multiplicación de las probabilidades condicionales de cada característica dada la clase, y selecciona la clase con la mayor probabilidad.
En nuestra implementación, utilizamos el clasificador Bayes Naive, específicamente el clasificador Multinomial Naive Bayes, para predecir la polaridad de reseñas de cine, ya sean positivas o negativas. Aplicamos dos enfoques de vectorización distintos: CountVectorizer y TfidfTransformer. Además, exploramos variantes del modelo, como la reducción de vocabulario y la optimización de hiperparámetros. Específicamente, ajustamos los hiperparámetros alpha (valor que representa el suavizado de las estimaciones de probabilidad condicional) y fit_prior (determina si se deben aprender o asignar probabilidades a priori para las clases) mediante un proceso de Grid

Search. Probamos diferentes valores para alpha, incluyendo 0.1, 0.5, 1.0, 1.5 y 2.0, así como dos opciones para fit_prior: true y false. Tras realizar la búsqueda en un espacio de hiperparámetros definido, encontramos que los mejores valores fueron {'alpha': 1.0, 'fit_prior': False}, con una puntuación óptima de 0.8502571428571428. Este fue el mejor modelo al subir la predicción en Kaggle.

● **Random Forest:**

Un modelo de Random Forest se basa en la combinación de múltiples árboles de decisión, donde cada árbol se construye utilizando una muestra aleatoria con reemplazo de los datos de entrenamiento y una selección aleatoria de características.
Se trata de un modelo capaz de manejar grandes conjuntos de datos, es resistente al overfitting y puede tomar relaciones no lineales y de interacción entre características.
En nuestro caso probamos entrenando los RF con el set completo, con el set reducido y con el set preprocesado. Al mismo tiempo probando diferentes representaciones de texto como BoW, TF-IDF y Word Embedding.

Evaluamos su desempeño con el conjunto de datos completo y filtrado, observando que el filtrado resultaba en métricas menos favorables. Posteriormente, procedimos a la optimización de hiperparámetros mediante Random Search, focalizando en los parámetros n_estimators y max_depth. Después de la búsqueda, encontramos que los mejores hiperparámetros fueron{'randomforestclassifier__max_depth':30,
'randomforestclassifier__n_estimators': 448}.

- ‘n_estimators’, puesto que indica el número de árboles en el bosque y aumentar su valor mejora el rendimiento del modelo. Se trata de encontrar un número que se encuentre cerca del punto de rendimiento óptimo donde se obtiene un muy buen rendimiento sin agregarle costo computacional adicional.

- ‘max_depth’, dado que determina la profundidad de cada árbol en el bosque, con éste parámetro podemos controlar la complejidad y capacidad de generalización del modelo. El valor justo nos permite equilibrar la capacidad de capturar patrones con la prevención del overfitting.

Además, extendimos el modelo Random Forest al espacio TF-IDF y a embeddings de palabras (Word Embeddings). En el caso de TF-IDF, exploramos diferentes configuraciones y, mediante Random Search, encontramos que {'bootstrap': True, 'max_depth': 43, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 700} optimizaban el desempeño, con una mejor puntuación de 0.840703204707964. En cuanto a los Word Embeddings, utilizamos el modelo Word2Vec para representar las reseñas de cine como vectores y aplicamos un clasificador Random Forest.

- **XGBoost:**

XGBoost es un modelo que usa boosting para aprender secuencialmente de los distintos modelos creados. Está diseñado para mejorar la velocidad y el rendimiento del algoritmo, utilizando técnicas como el descenso por gradiente y la poda de árboles. Utiliza un conjunto de árboles de decisión débiles como modelos base y cada árbol corrige los errores del anterior, siendo el resultado final una combinación ponderada de estos árboles. Su principal utilidad es manejar grandes conjuntos de datos.
Los hiperparámetros que utilizamos fueron:

- 'n_estimators': [50, 100, 150, 200], que representa los árboles a construir.
- 'max_depth': [3, 5, 7], que representa la profundidad máxima de cada árbol.

- **Red neuronal:**

Utilizamos una red neuronal eligiendo capas e hiperparámetros para optimizar los datos del dataset.
Los utilizados fueron:

- max_words: Número máximo de palabras a considerar en el vocabulario.
- max_sequence_length: Longitud máxima de las secuencias de entrada.
- embedding_dim: Dimensión de los vectores de palabras resultantes de la capa de embedding.

- epochs: Número de épocas de entrenamiento.
- batch_size: Tamaño del lote utilizado durante el entrenamiento.
- optimizer: Algoritmo de optimización, en este caso, 'adam'.
- loss: Función de pérdida utilizada, en este caso, 'binary_crossentropy'.
- metrics: Métrica utilizada para evaluar el rendimiento, en este caso, 'accuracy'.

También utilizamos Tokenizer para convertir texto en secuencias de números y pad_sequences para asegurar que todas las secuencias tengan la misma longitud.

La arquitectura de la red incluye:

- Una capa de embedding para representar palabras como vectores densos.
- Dos capas de unidades de red neuronal recurrente (GRU).
- Capas de Dropout para regularización.
- Capas totalmente conectadas (Dense) con funciones de activación 'relu' y 'sigmoid' para clasificación binaria.

• Ensamble (Hard Voting)
El ensamble se compone de tres modelos base: Naive Bayes, XGBoost (XGB), y Random Forest. El objetivo principal es mejorar las predicciones individuales de estos modelos mediante la combinación de sus decisiones a través de un proceso de votación.
El ensamble se crea utilizando la técnica de Hard Voting, donde las predicciones individuales de los modelos base se combinan por mayoría de votos. Se utiliza la clase VotingClassifier de scikit-learn para implementar esta estrategia.
El ensamble se entrena utilizando datos de entrenamiento (conjunto x_train_local_tfidf_redvoc_preprocesado e y_train_local_encoded). El objetivo es que el ensamble aprenda de los patrones capturados por los modelos base.
En cuanto a si probamos técnicas o algoritmos adicionales a las pedidas en la consigna del trabajo, la respuesta es no.

Conclusiones generales

A lo largo del desarrollo de nuestro trabajo, el análisis exploratorio inicial se reveló como un paso fundamental. Este análisis nos permitió comprender la estructura del conjunto de datos y, de esta manera, abordar eficientemente la etapa de preprocesamiento. La ausencia de esta exploración previa habría dificultado la determinación del modelo de preprocesamiento más útil para nuestras necesidades.
El preprocesamiento de datos, nuestra primera tarea, resultó crucial para optimizar la performance de los modelos. Basándonos en experiencias anteriores con diferentes conjuntos de datos, asumimos que este paso contribuyó significativamente al tiempo de ejecución relativamente corto y al puntaje destacado que lograron nuestros modelos en relación a los demás en el leaderboard.
En las pruebas de TEST, la red neuronal se destacó como el modelo de mejor rendimiento. Sin embargo, en el entorno de Kaggle, fue el modelo Bayes Naive el que obtuvo el mejor desempeño. Este resultado fue especialmente notable considerando que Bayes Naive resultó ser el modelo más sencillo de entrenar, destacándose por su eficiencia y rapidez en comparación con otros modelos más complejos.
Creemos que Bayes Naive podría funcionar en un entorno productivo, ya que no solo demostró ser de bajo costo en términos de entrenamiento, sino que también logró la performance más alta en comparación con el resto de nuestros modelos.
Sin embargo, reconocemos que siempre existe margen para mejorar los resultados. La optimización de hiperparámetros de cada modelo podría ser un enfoque valioso, a pesar de que esto implique un aumento en el tiempo de ejecución en el entrenamiento. Esta estrategia podría resultar en mejoras significativas en métricas clave como precisión, recall, accuracy y F1 score.
En conclusión, creemos que este trabajo nos permitió darle un enfoque más real al uso de los temas que fuimos aprendiendo a lo largo del cuatrimestre.

Tareas Realizadas

Integrante	**Promedio Semanal (hs)**
Martín Abramovich	6 hs.
Iara Jolodovsky	6 hs.
Tomás Vainstein	5 hs.