TP2: Críticas Cinematográficas - Grupo 06

Introducción

El dataset provisto consiste de una lista de críticas de películas compartidas por distintos usuarios, posiblemente recopiladas de algún sitio web de reseñas. El objetivo del trabajo es entrenar múltiples modelos y realizar una clasificación de los textos de cada crítica, determinando si se trata de una crítica positiva o negativa.

La consigna aporta dos datasets, uno de entrenamiento, y otro de pruebas; este último se utilizará para asignarle una puntuación a la calidad de la clasificación del modelo en la competencia de Kaggle.

El dataset de entrenamiento tiene **50.000** registros, cada uno con información distribuida en 3 columnas.

- ***ID***: número único identificatorio de cada crítica (autoincremental).
- ***review_es***: texto principal de la crítica.
- ***sentimiento***: variable categórica para la clasificación. Puede tomar dos valores: "positivo" o "negativo".

El dataset de pruebas tiene **8.599** registros, y contiene las mismas columnas, con excepción de *sentimiento*, la cual justamente es la que se intenta predecir.

Durante la exploración inicial de los registros, surgieron las siguientes cuestiones a considerar:

- La clasificación dependerá exclusivamente del texto de *review_es*. La columna *ID* no presenta información relevante y puede ser descartada del análisis.
- Los textos recolectados presentaban distintos obstáculos para la clasificación. Errores de formato, caracteres especiales, inconsistencias en la puntuación, y faltas de ortografía eran los más comunes, aunque en ocasiones se presentaron críticas en otros idiomas. Estas cuestiones debieron abordarse durante el preprocesamiento de los textos.

En cuanto a las técnicas de preprocesamiento de datos, se realizaron las siguientes:

- **Limpieza del Texto**: Se eliminaron todos los caracteres no alfabéticos usando expresiones regulares, como signos de puntuación, números y otros caracteres especiales. También se pasó todo el texto a minúsculas dado que consideramos que las mayúsculas no agregaban valor a los términos.
- **Tokenización y Lematización**: Identificamos los diferentes 'tokens', o palabras, en los textos, mediante tokenizadores dedicados. Además, definimos los tokens de manera tal que permitan agrupar las distintas apariciones de una palabra, para que puedan ser analizadas como una sola entidad.
- **Eliminación de Stopwords**: Se eliminan palabras comunes, artículos y conectores, sin un significado concreto usando la lista de NLTK; esto ayuda a reducir el ruido en los datos y a enfocar el análisis en las palabras relevantes.
- **Vectorización del Texto**: Se le asigna a cada palabra un valor basado en su frecuencia en el documento ayudando a resaltar las palabras más importantes.

Para probar la calidad de la clasificación durante el entrenamiento del modelo, realizamos las siguientes transformaciones:

- **Data Augmentation**: Utilizamos la función 'make_classification' para generar datos en base a los que ya teníamos.
- **División del Dataset**: Separamos el dataset en un conjunto de entrenamiento y otro de prueba utilizando 'train_test_split', con una relación 80/20.
- **Parámetros de Vectorización**: Utilizamos tanto CountVectorizer como TfidfVectorizer para llevar los textos a una representación numérica basada en la frecuencia de aparición de los términos. Se probó también alternar la cantidad de stopwords, así como también la inclusión de n-gramas y de otros lenguajes.

Descripción de Modelos

Bayes Naive

Un clasificador probabilístico basado en la aplicación del teorema de Bayes con una fuerte suposición de independencia entre características. Se utilizó un MultinomialNB, que es adecuado para datos discretos como los contados de palabras.

Hiperparámetros del mejor modelo:

- alpha: 1.0 (suavizado de Laplace)

Técnicas de Preprocesamiento de Datos:

- **Normalización de Texto**: Se eliminan acentos y caracteres especiales utilizando unicodedata.normalize.
- **Vectorización de Texto**:
    - **CountVectorizer**: min_df: 30, ngram_range: (1,2), analyzer: 'word'
    - **TfidfTransformer**: Transforma los vectores de conteo en vectores de TF-IDF.

Entrenamiento:

- **División de Datos**: Los datos se dividen en conjuntos de entrenamiento y prueba utilizando train_test_split.
- **Vectorización y Transformación**:
    - Los textos de entrenamiento se vectorizan utilizando CountVectorizer y se transforman a TF-IDF utilizando TfidfTransformer.
    - Los textos de prueba se transforman utilizando el mismo CountVectorizer y TfidfTransformer ajustados con los datos de entrenamiento.
- **Entrenamiento del Modelo**: El modelo MultinomialNB se entrena con los vectores TF-IDF de entrenamiento.
- **Validación Cruzada**: Se utiliza cross_val_score para evaluar el modelo utilizando la métrica f-1.

Random Forest

Un clasificador de ensamble que utiliza múltiples árboles de decisión entrenados con diferentes subconjuntos del dataset y de características, y promedia los resultados para mejorar la precisión y controlar el sobreajuste.

Hiperparámetros del mejor modelo:

- criterion: entropy
- min_samples_leaf: 5
- min_samples_split: 2
- n_estimators: 10
- oob_score: True
- random_state: 42
- n_jobs: -1

Preprocesamiento de Datos:

- **Normalización y Limpieza de Texto**: Tokenización, eliminación signos de puntuación y eliminación stopwords
- **Vectorización de Texto**: CountVectorizer y TfidfTransofrmer

Entrenamiento:

- **División de Datos**: Los datos se dividen en conjuntos de entrenamiento y prueba utilizando train_test_split.
- **Vectorización y Transformación**: Los textos de entrenamiento se vectorizan utilizando CountVectorizer y se transforman a TF-IDF utilizando TfidfTransformer. Los textos de prueba se transforman utilizando el mismo CountVectorizer y TfidfTransformer ajustados con los datos de entrenamiento.
- **Búsqueda Aleatoria de Hiperparámetros**: Se utiliza RandomizedSearchCV para encontrar los mejores hiperparámetros con validación cruzada de 15 pliegues y 1 iteración.
- **Entrenamiento del Modelo**: El mejor modelo encontrado por RandomizedSearchCV se entrena con los vectores TF-IDF de entrenamiento.

XGBoost

Este algoritmo de aprendizaje supervisado emplea árboles de decisión junto a la técnica de boosting para realizar las predicciones. Esto es, se instancian sucesivos modelos de ajuste débil, y se ejecutan en secuencia, cada uno de ellos intentando disminuir el error cometido por el anterior.

Su configuración resultó sumamente sencilla y ofrecía distintas formas de entrenarlo, entre los cuales se encontraba el formato de dato DMatrix, cuya implementación está optimizada para este modelo en particular.

Hiperparámetros del mejor modelo:

- eval_metric: logloss
- learning_rate: 0.1
- max_depth: 6
- subsample: 0.8
- colsample_bytree: 0.8
- num_boost_round: 1000
- early_stopping_rounds: 10

Preprocesamiento de Datos:

- **Normalización y Limpieza de Texto**: Tokenización, eliminación de signos de puntuación mediante RegEx y eliminación de stopwords.
- **Vectorización de Texto**: Bag of Words empleando CountVectorizer, donde se eliminan las stopwords. Se incluyeron bigramas y una frecuencia mínima de 20 apariciones.

Entrenamiento:

- **Vectorización y Transformación**: Los textos de entrenamiento se vectorizan utilizando CountVectorizer.
- **Instancia de DMatrix**: Se crean los objetos dtrain y dval a partir de los conjuntos resultantes luego del split. Este formato permite realizar algunas operaciones con un tiempo de ejecución optimizado.
- **Codificación de Etiquetas:** Mediante LabelEncoder se llevan los valores de la columna 'sentiment' a los valores numéricos 0 y 1, para poder ser trabajados por el modelo.
- **Entrenamiento del Modelo**: Se entrena repetidas veces el modelo mediante el método train de XGBoost, con dtrain y un conjunto único de parámetros, con tal de obtener los mejores resultados.

Red Neuronal

Es un modelo matemático inspirado en el cerebro humano que se utiliza para resolver problemas de aprendizaje automático. Consiste en un conjunto de nodos interconectados, llamados neuronas.

Preprocesamiento:

- **Normalización y Limpieza de Texto:** Tokenización, eliminación de signos de puntuación y eliminación de stopwords.
- **Vectorización de Texto:** CountVectorizer.

Hiperparámetros del mejor modelo:

- Optimizador: Adam con tasa de aprendizaje entre 1e-2, 1e-3 y 1e-4 (con búsqueda aleatoria).
- Función de pérdida: “binary_crossentropy”.
- Métricas: “accuracy”.
- Número de epochs: 50.
- Tamaño del batch: 50.

Entrenamiento:

- **División de Datos**: Los datos se dividen en conjuntos de entrenamiento y prueba utilizando train_test_split.
- **Vectorización y Transformación**: Los textos de entrenamiento se vectorizan utilizando CountVectorizer
- **Codificación de Etiquetas**: Las etiquetas se codifican en formato numérico utilizando LabelEncoder.
- **Entrenamiento del Modelo Inicial**: El modelo se entrena con los vectores de características y las etiquetas codificadas utilizando el optimizador SGD, función de pérdida binary_crossentropy, métrica AUC, y 100 epochs.
- **Optimización de Hiperparámetros**: Se utiliza Keras Tuner para encontrar los mejores hiperparámetros (unidades de la capa densa y tasa de aprendizaje del optimizador Adam).
- **Entrenamiento del Modelo Ajustado**: El mejor modelo encontrado por Keras Tuner se entrena con los vectores de características y las etiquetas codificadas utilizando el optimizador Adam, binary_crossentropy, accuracy, y 50 epochs.

Arquitectura de la red neuronal

Luego de probar diferentes cantidades de capas optamos por dos capas densas, la última con solamente una salida, esta fue la configuración con la que obtuvimos un mejor score que con otras cantidades de capas. En base a ello, realizamos la búsqueda de hiper parámetros, buscamos la cantidad de neuronas óptimas para la primer capa, como también el learning_rate que nos brindara una mejor performance.

Para la función de activación, en cambio, utilizamos sigmoid ya que al probar con otras funciones tales como relu, softmax, tahn los valores no llegaban a ser cercanos al accuracy esperado para el modelo. La cantidad de epochs, la cantidad de veces que los datos de entradas pasaron por la neurona, lo fijamos en 100 mientras que el batch_size, el número de ejemplos que se le brinda a la red para que entrene, en 50.

Ensamble

Combina varios clasificadores individuales para mejorar la precisión y robustez de las predicciones. Se usó un VotingClassifier con soft voting. Los modelos ensamblados fueron: Bayes naive, Random Forest, XGBoost y Red neuronal.

Preprocesamiento:

- **Vectorización de Texto**: Usando TfidfVectorizer para convertir los textos en vectores de características.
- **Stopwords**: Lista de palabras comunes en español que se eliminan.
- **lemmatizer**: Se utiliza para reducir las palabras a su forma base.

Hiperparámetros:

- **Bayes Naive**: alpha: 1.0.
- **Random Forest**: n_estimators: 200, max_depth: 20, min_samples_split: 2, min_samples_leaf: 1.
- **XGBoost**: n_estimators: 100, max_depth: 5, learning_rate: 0.1, subsample: 0.8, colsample_bytree: 0.8.
- **Red neuronal**: epochs: 10, batch_size: 32, model__input_dim: 5000.

Entrenamiento:

- **Naive Bayes:** Entrenado con el conjunto de datos vectorizado.
- **Random Forest:** Entrenado con RandomizedSearchCV para optimizar los hiperparámetros.
- **XGBoost:** Entrenado con RandomizedSearchCV para optimizar los hiperparámetros.
- **Red Neuronal:** Entrenada con EarlyStopping para evitar el sobreajuste.
- **Ensamble:** El VotingClassifier combina las predicciones de todos los modelos individuales mediante soft voting..

Cuadro de Resultados

Se muestran a continuación las métricas de los mejores puntajes de las submissions realizadas en Kaggle.

Modelo	F1-Test	Precision Test	Recall Test	Accuracy	Kaggle
Bayes Naive	0.8621	0.86	0.86	0.86	0.7334
Random Forest	0.80	0.80	0.80	0.80	0.7319
XGBoost	0.8722	0.8603	0.8845	0.8694	0.7266
Red Neuronal	0.8826	0.8733	0.8922	0.8805	0.7171
Ensamble	0.8907	0.8924	0.8890	0.8901	0.7358

Conclusiones generales

El resultado de la predicción dependía en gran medida no sólo de los hiperparámetros obtenidos, sino de las técnicas de preprocesamiento empleadas para normalizar y tokenizar los datos. Observamos que para una combinación de técnicas (por ejemplo, eliminar signos de puntuación, y caracteres especiales), dependiendo de la implementación o librerías que se utilizaran, la predicción podía empeorar o mejorar drásticamente.

Sumado a este hecho, lo que podía verse como una mejora general de las métricas internas del conjunto train, no siempre lograba reflejarse como un aumento en la puntuación de la competencia, lo que nos empujaba constantemente a buscar alternativas.

La estructura particular de estos datasets significaba que debíamos enfocarnos más en cómo íbamos a normalizar, tokenizar y predecir los resultados. Esto es, sin darle demasiada importancia a ciertas etapas del análisis exploratorio de los datos. Más allá de realizar conteos de registros, buscar datos nulos, o verificar que los datos estén balanceados, por ejemplo, no resultó necesario realizar un análisis de correlaciones analítica y gráficamente para determinar comportamientos extraños en nuestros datos.

Entre los modelos entrenados, el ensamble con VotingClassifier fue el que consistentemente obtuvo los mejores resultados, tanto en las métricas internas como en la competencia de Kaggle. Por otro lado, Naive Bayes fue el modelo que requirió menos configuraciones a la hora de entrenar. Su desempeño fue muy bueno respecto a los demás modelos en este uso puntual, aunque entendiendo que el rendimiento no siempre es superior al que pueden ofrecer los otros modelos.

En cuanto al potencial de mejora de los modelos, reconocemos que aplicar técnicas más abarcativas de normalización de los tokens nos permitirá reducir los sesgos y las predicciones erróneas. Sumado a esto, linealizar el proceso de búsqueda de hiperparámetros, tanto del Bag of Words como de los propios modelos, de forma de poder dimensionar los distintos resultados obtenidos en cada iteración para establecer comparaciones, podría aportar una mejora significativa en las predicciones. También consideraríamos usar otras herramientas y recursos como SentiWordNet, para obtener una impresión más exacta del valor sentimental de cada palabra.

Este ejercicio práctico de análisis de sentimientos podría muy tranquilamente ser aplicada en contextos productivos más amplios. Un modelo como el realizado podría ser aplicado y aprovechado en múltiples sitios y plataformas de streaming. Incluso más allá de los límites de las reseñas cinematográficas, la opinión pública tiene el potencial de ofrecer retroalimentación valiosa acerca de un determinado producto, ya sea para incentivar el desarrollo y la competencia, y para ofrecer recomendaciones que resuenen en mayor medida con el usuario.

Tareas Realizadas

Integrante	**Principales Tareas Realizadas**	**Promedio Semanal (hs)**
Veronica Leguizamón	Modelo Red Neuronal e informe	6
Henry Maldonado	Modelos Bayes Naive, Random Forest e informe	6
Augusto Rivera Lofrano	Modelo Ensamble e informe	6
Alejandro Vargas	Modelo XGBoost e informe	6