TP2: Críticas Cinematográficas - Grupo 32

Introducción

Objetivo: El objetivo de este trabajo es el etiquetar como positiva o negativa, a través de diferentes modelos predictivos, críticas cinematográficas realizadas en diferentes tipos de películas.

Descripción del dataset: El dataset contiene 50 mil registros y 3 columnas. Una columna "ID" con la identificación numérica correspondiente, una columna "review_es" que cuenta con las críticas realizadas a diferentes películas y una última columna "sentimiento" la cual indica si la crítica correspondiente es positiva o negativa.

Técnicas exploradas: Como la mayoría de los modelos no pueden trabajar con palabras y necesita valores numéricos, al dataset se le aplicó una vectorización TF-IDF (Frecuencia de Término - Frecuencia Inversa de Documento) la cual asigna pesos a las palabras en función de su frecuencia en un documento específico y su frecuencia en todo el corpus de documentos.
Para el **XGBoost** como primera opción buscamos optimizar diferentes hiperparametros, obteniendo así un f1-score de 0.8056 en test, pero luego al probar con un modelo de XGBoost por default, sin modificar ningún hiperparametro, obtuvimos un f1-score de 0.8477 en test. Probando ambos modelos en la competencia, el XGBoost por default dio un mejor f1-score con 0.71002, por ello se decidió utilizar este último (ambos modelos se encuentran realizados en la notebook como evidencia de lo dicho).
Para el modelo **Bayes Naïve** se utilizó una distribución multinomial y se busco optimizar el hiperparametro 'alpha' a través de cross validation, nuestro mejor modelo obtuvo un f1-score de 0.8692 en test y una puntuación de 0.76584 en Kaggle.
Para el **Random Forest** buscamos hiperparametros a través de cross validation y con el mejor modelo obtuvimos un f1-score de 0.8427 en test y una puntuación de 0.73851 en Kaggle.
En el modelo de ensamble **Stacking** obtuvimos un f1-score de 0.9009 en test y una puntuación de 0.7683 en Kaggle. Utilizamos los modelos de Random Forest, XGBoost y Bayes Naïve, para el metamodelo utilizamos una regresión logística. No utilizamos el modelo de redes neuronales ya que la vectorización es distinta para esta,

intentamos adaptar los otros modelos, lo cual pudimos lograrlo con Pipelines pero no sin quedarnos sin RAM en la notebook.
Para la **Red Neuronal**, tuvimos que re-vectorizar los datos de una forma que sea interpretable por el modelo, pasar a tensores, luego a dataset, redimensionar, etc... Las iteraciones y optimización de hiper parámetros fue de manera manual ya que no pudimos lograr CV, y para crear la predicción final agregamos la capa de vectorización antes del modelo, para que acepte como input una cadena de texto y realize su propia vectorización. Obtuvimos un F1-Score de 0.88 en test y una puntuación de 0.7608 en Kaggle. En la sección pertinente entraremos en más detalle sobre el armado de la red.

Preprocesamiento: Observamos que el dataset no presentaba datos nulos. La columna “sentimiento” no tenía valores inesperados, eran o positivos o negativos. Encontramos que aproximadamente un 4% de las críticas del dataset no estaban en español, dada la poca cantidad que eran se procedió a eliminarlas. Eliminamos la columna “ID” del set de entrenamiento, debido a que no son necesarias.

Cuadro de Resultados

Modelo	F1-Test	Presicion Test	Recall Test	Accuracy test	Kaggle
Bayes Naive	0.8692	0.8722	0.8661	0.8682	0.76584
Random Forest	0.8427	0.8590	0.8270	0.8439	0.73851
XGBoost	0.8477	0.8641	0.8319	0.8489	0.71002
Red Neuronal	0.8703	0.9069	0.8365	0.8740	0.7631
Ensamble Stacking	0.9009	0.9060	0.8959	0.9004	**0.7683**

Descripción de Modelos

- **Bayes Naive:**
  Es un clasificador probabilístico basado en el teorema de Bayes. Asume independencia condicional entre las características, lo cual simplifica cálculos. También es eficaz en problemas de clasificación.
  Los hiperparámetros incluyen la elección del tipo de Naive Bayes y ajustes específicos según el tipo. Se entrena fácilmente a partir de datos etiquetados, calculando probabilidades condicionales y priors.

- **Random Forest:**
  Algoritmo de conjunto que combina múltiples árboles de decisión.
  Cada árbol se entrena con una muestra aleatoria del conjunto de datos, y luego se combinan la salida de los árboles para tomar decisiones.
  Los hiperparámetros incluyen: el número de árboles, la profundidad máxima del árbol, mínimo de casos para que el nodo se divida, mínimo de casos para que el nodo sea hoja, y otros parámetros específicos del árbol de decisión.
  En este modelo, para las etiquetas de salida o variables objetivo es necesario que sean numéricas para que se pueda entrenar de manera adecuada. Por lo tanto antes de utilizarlas se las codifica numéricamente con label encoder.

- **XGBoost:**
  Es un método de aprendizaje automático supervisado para clasificación y regresión basado en el Boosting, en este caso usado para clasificación. Esto es una forma de aprendizaje de los modelos basada en generar predicciones y asignar un mayor peso a aquellas mal clasificadas, así iterativamente hasta un límite determinado por los hiperparámetros.
  Hiperparámetros: Incluyen la tasa de aprendizaje, la profundidad del árbol, el número de árboles y parámetros de regularización.
  Igualmente a Random Forest, no puede manejar variables categóricas para las etiquetas de salida o variables objetivo, por lo que también se les aplica label encoder antes de utilizarlas en el entrenamiento de este.
  El modelo se entrena iterativamente, ajustando los pesos de las instancias y construyendo árboles que minimizan la función objetivo.

- **Red Neuronal:**
  A través del entrenamiento, la red aprende patrones complejos, destacándose en tareas como reconocimiento de voz, visión por computadora y procesamiento de lenguaje natural.

  En la red neuronal que construimos incluimos 3 capas, la primera una capa de **Embedding**, cuyo objetivo es convertir los vectores resultantes de la vectorización del texto en vectores densos de tamaño fijo, determinado por **embedding_dim**. Al aumentar este hiper parámetro, hacemos que la capa se ajuste mejor a los datos a riesgo de causar overfitting. Sin embargo, para limitar este factor, agregamos a esta capa un **Dropout**, que si bien por como está definido en Keras parece una capa distinta, lo que hace esto es que un porcentaje de las neuronas de la real capa anterior

se apaguen a medida que se entrena la red, previniendo el overfitting.
Luego, agregamos la capa **GlobalAveragePooling1D**, que es una capa de reducción de dimensionalidad, es útil para extraer las características más relevantes o destacadas de una secuencia. Al seleccionar el valor máximo en cada dimensión, la capa resalta la información más importante presente en la secuencia.
Luego agregamos una **capa Densa de 20 neuronas con Dropout de 20%**
Finalmente, una capa de una única neurona (**Densa**) que determina un valor predicho, la función de activación que usamos es la sigmoidea, que nos dio mejores resultados.
Estamos seguros que si seguimos probando y probando combinaciones de hiper parámetros y capas podremos lograr mejoras de performance, pero tras muchos intentos esta arquitectura es la de mejor resultado
Definimos luego el **umbral de decisión final como 0.5**, el **Batch-size se define automáticamente** según convenga como definido por Tensor Flow, limitamos el **vocabulario a 10000 palabras** para mayor eficiencia, modificamos manualmente la cantidad de **Epochs** hallando que el mejor resultado es con **4**

Técnicas de preprocesamiento de datos: Partiendo de la misma base de los modelos, adaptamos los datos que tenemos a tensores y luego a Datasets para poder ser comprendidos por la red

Entrenamiento: Ajusta los pesos de las conexiones durante el entrenamiento mediante algoritmos de backpropagation.

- **Ensamble Stacking:**
  Técnica de ensamble que combina las predicciones de múltiples modelos base mediante un meta-modelo. Mejora la capacidad predictiva al capturar las fortalezas individuales de los modelos base, siendo útil en problemas complejos de clasificación o regresión.
  Hiperparámetros: En este caso, al usar como meta-modelo una regresión logística, optimizamos solo la cantidad máxima de iteraciones, luego los demás modelos ya se suponen optimizados.
  Fue el modelo que mejor resultado nos dio, con un score de 0.7683 en Kaggle.

Conclusiones generales

En particular, nos resultó útil el análisis exploratorio de los datos, ya que al ver lo contenido en el dataframe de test y estudiando el lenguaje de las reseñas que supuestamente estaban en español, descartamos algunos registros que tal vez no estaban en español o qué contenían información no interpretable o vaga. Al hacer esto logramos una mejor performance en todos los modelos por lo que creemos que fue un gran acierto en este sentido. También habíamos realizado una “limpieza de las críticas” donde convertimos los textos en minúscula, eliminamos caracteres

especiales, eliminamos palabras comunes que no aportaban significado al contexto (como “a”, “el”, “de”, “la”, etc.), pero luego de probar nuestros modelos con esta “limpieza” no dieron los resultados esperados, de hecho empeoraron algunos puntajes tanto en test como el Kaggle, por lo que no fue tenida en cuenta en el trabajo final.
Tras implementar los distintos modelos, observamos que el Bayes Naive, más allá de su baja complejidad y rapidez, obtuvo muy buenos resultados ante el conjunto que utilizamos aun siendo el modelo más sencillo y rápido de entrenar.
En general, observamos que todos los modelos obtienen muy buenos resultados con nuestros datos de test, no así cuando subimos las predicciones a Kaggle. Esto nos hace creer que los textos de validación contienen un vocabulario muy distinto al de train, o que puede haber críticas con ironía o ambiguas. Por esto, creemos que hubiera sido productivo utilizar un léxicon para predecir y observar los resultados o realizar un análisis más exhaustivo del conjunto de test para poder adaptar el armado de modelos. Decidimos no hacer esto para no ajustar los datos específicamente a este problema, sino lograr resultados más genéricos que en un entorno real probablemente tenga mejor performance.
Sin embargo, no consideramos que el modelo pueda estar en una etapa productiva, ya que el conjunto de entrenamiento tiene poca información y por cómo funciona BOW, tener una palabra nueva significa no tener idea alguna de cómo clasificar su polaridad, por lo que los modelos quedan muy ciegos ante estas ocurrencias. Para que el modelo pueda ser productivo deberíamos darle más información para entrenamiento o alguna forma (léxicon) de definir la polaridad inicial de palabras nuevas en el vocabulario.
Obtuvimos el mejor resultado tanto en test como en Kaggle con el **Ensamble Stacking.** Nos parece lógico esto porque como sabemos, en Stacking entrenamos modelos en base a resultados del anterior. Suponiendo que cada uno mejora la performance general del anterior (aunque sabemos que en la práctica no es así por tema de overfitting), al utilizar el segundo mejor modelo en el stacking, mejoramos levemente la performance. Sabemos que probablemente se pueda obtener un incluso mejor score si utilizamos algún otro meta-modelo y optimizamos sus hiper parámetros, pero tras iterar mucho, no encontramos alguno que mejore el resultado.

Tareas Realizadas

Integrante	Promedio Semanal (hs)
Daniel Agustin Marianetti	6
Ezequiel Lazarte	8
Franco Ezequiel Rodriguez	4