TP2: Críticas Cinematográficas - Grupo 26

Introducción

El problema a resolver implica la clasificación de críticas cinematográficas en español como positivas o negativas. Se buscará entrenar modelos que puedan analizar el contenido de las reseñas y asignarles una etiqueta de sentimiento, indicando si la crítica es positiva o negativa.
Contamos con 2 dataset, uno de entrenamiento (50000, 3) y otro de test (8599, 2). Ambos datasets contienen las mismas columnas (id, review_es y sentimiento), exceptuando el test, que no contará con el sentimiento, ya que es el target a predecir.
Para el procesamiento de texto, decidimos usar los metodos bag of words (CountVectorizer) y otra alternativa (TfidfVectorizer). CountVectorizer simplemente cuenta la frecuencia de las palabras en un documento, mientras que TfidfVectorizer considera no sólo la frecuencia de la palabra, sino también su importancia en el conjunto de documentos.
Para cada modelo, usamos make_pipeline de scikit-learn para crear un pipeline que incluye el modelo del preprocesamiento y el modelo a entrenar. Con esto, creamos un pipeline de procesamiento de texto y clasificación de manera más concisa.

Cuadro de Resultados

Modelo	F1-Test	Precision Test	Recall Test	Accuracy	Kaggle
Bayes Naive (BoW)	**0.87700**	**0.86602**	**0.88825**	**0.87706**	**0.76100**
Random Forest (BoW)	0.72222	0.71903	0.72544	0.72466	0.63636
XgBoost (BoW)	0.83889	0.82062	0.85799	0.83740	0.69994
Red Neuronal	0.84477	0.88489	0.80813	0.85346	0.7356

Ensamble	0.81682	0.82998	0.80408	0.82206	0.72688

Descripción de Modelos

Para cada modelo, hemos decidido variar algunos valores del modelo de preprocesamiento, pertenecientes a las siguiente parámetros:

- **ngram_range**: Determina qué tipo de secuencias de palabras se tendrán en cuenta (palabras individuales, de a pares, etc).
- **max_features**: Limita cuántas palabras diferentes se considerarán al construir la representación del texto.
- **stop_words**: Decide si eliminar palabras comunes pero poco informativas.
- **max_df**: Ignora palabras que aparecen en más documentos que el umbral especificado.
- **min_df**: Ignora palabras que aparecen en menos documentos que el umbral especificado.

Bayes Naive: Método de clasificación probabilístico basado en el teorema de Bayes. Calcula la probabilidad de que una instancia pertenezca a una clase dada sus características.

- **alpha:** Este parámetro controla la suavización de Laplace. Es útil para evitar que el modelo sea demasiado sensible a frecuencias de palabras específicas
- **fit_prior:** Este parámetro determina si se deben aprender o no las probabilidades a priori de las clases.
- **class_prior:** Este parámetro permite especificar las probabilidades a priori de las clases manualmente.

Random Forest: Algoritmo de aprendizaje supervisado que crea un "bosque" de árboles de decisión durante el entrenamiento. Los fusiona para obtener predicciones más precisas y estables. Cada árbol se entrena de forma independiente y luego se combina para tomar una decisión conjunta. Usamos RandomizedSearchCV para buscar aleatoria de los mejores hiperparámetros.

- Ngram_range de (1,1) y (1,2), es decir, palabras individuales y bigramas
- Stop_word [None] para saltar palabras vacías
- Búsqueda aleatoria para encontrar el número óptimo de estimadores dentro del rango entre 1 y 8

XgBoost: Implementación eficiente de Gradient Boosting, un método de ensamblado que construye árboles secuencialmente, corrigiendo los errores de los árboles anteriores.

- Ngram_range de (1,1) y (1,2), es decir, palabras individuales y bigramas
- [50, 100] estimadores, un número reducido para acelerar y agilizar el entrenamiento del modelo pese a que sacrificamos precisión.
- Una profundidad de [3, 4] para cada árbol individual.

Red Neuronal: Implementación de una red neuronal de 3 capas densas y una de dropout utilizando Count Vectorizer. Para las capas utilizamos los activadores relu y sigmund, y al compilar el modelo el optimizador Adam con una tasa de aprendizaje del 0.001. Se optimizaron el learning_rate y la binary_crossentropy.

- Capa de entrada de 32 neuronas
- Tasa del 30% de dropout
- Capa oculta de 16 neuronas
- Capa de salida de 2 neuronas

Ensamble Voting: El ensamble de votos combina las predicciones de múltiples modelos para tomar una decisión final.
Este ensamble está formado por los siguientes modelos:

- **Bayes Naive (BoW)**
- **Random Forest (BoW)**
- **XgBoost (BoW)**

Conclusiones generales

Luego del arduo trabajo de explorar los datos y entrenar modelos, destacamos varios factores que concluimos que fueron muy importantes al momento de desarrollar el trabajo práctico.
Entre todos ellos, nos pareció muy importante lo siguiente: el cómo la calidad del preprocesamiento del texto tuvo un impacto significativo en el rendimiento de los modelos, a comparación de mejorar los parámetros propios que tenía cada modelo, lo cual estábamos muy acostumbrados a hacer.
En cuanto a las opciones que quedaron fuera del alcance de este trabajo, nos hubiese gustado incorporar alguna información externa al problema, como embeddings de dominio específico o datos adicionales que pudieran enriquecer la representación de texto.

- ¿Fué útil realizar un análisis exploratorio de los datos?

Sí, fue extremadamente útil realizar un análisis exploratorio de datos. Nos proporcionó una visión de la estructura y las características de nuestros datos, lo que resultó fundamental para entender la naturaleza del problema que estamos abordando. Además, nos permitió identificar patrones, detectar posibles problemas en los datos y tener una visión clara de cómo deberíamos entrenar a nuestros modelos.

- ¿Las tareas de preprocesamiento ayudaron a mejorar la performance de los modelos?

Definitivamente, las tareas de preprocesamiento mejoraron el rendimiento de nuestros modelos. Al principio, experimentamos con diversos enfoques, pero fue cuando aplicamos técnicas como la tokenización, la eliminación de stopwords y la lematización que realmente empezamos a ver una diferencia significativa. Estas tareas no solo mejoraron la calidad de los datos de entrada para nuestros modelos, sino que también ayudaron a capturar patrones más relevantes en el texto, lo que se tradujo en un rendimiento mucho mejor durante las evaluaciones.

- ¿Cuál de todos los modelos obtuvo el mejor desempeño en TEST?

Al evaluar el desempeño de nuestros modelos en el conjunto de prueba, observamos que el modelo **Bayes Naïve** superó a los demás en términos de métricas como precisión, recall y F1-score.

- ¿Cuál de todos los modelos obtuvo el mejor desempeño en Kaggle?

En Kaggle, evaluamos el desempeño de nuestros modelos utilizando la métrica F1-score, encontramos que el modelo **Bayes Naïve** fue el que obtuvo el mejor resultado.

- ¿Cuál fue el modelo más sencillo de entrenar y más rápido? ¿Es útil en relación al desempeño obtenido?

El más rápido de entrenar fue la red neuronal, mientras que el más sencillo de entrenar fue el XGBoost. La practicidad al aplicar el GridSearchCV sumado a que utilizamos una menor cantidad de hiperparámetros lograron que disminuyera nuestro márgen de errores a la hora de entrenar el modelo.

- ¿Cree que es posible usar su mejor modelo de forma productiva?

Si, creemos que nuestro mejor modelo (**Bayes Naïve**) tiene potencial de ser utilizado de manera productiva. Demostró mucha capacidad al momento de generalizar los datos. Además, su naturaleza probabilística y simplicidad lo hacen adecuado para su implementación en entornos productivos.

- ¿Cómo podría mejorar los resultados?

Seguramente, haciendo un mejor preprocesamiento del texto, ya que el modelo en sí, tiene muy pocos parámetros que podemos ajustar, en cambio, explorando más

técnicas de preprocesamiento (o modelos que convierten texto en vectores), de seguro logremos mejorar aún más los resultados obtenidos.

Tareas Realizadas

Integrante	**Promedio Semanal (hs)**
Garcia, Nicolas	3-4
Vallcorba, Agustin	3-4
Carbajal Robles, Kevin Emir	3-4