TP2: Críticas Cinematográficas - Grupo 5

# Introducción

En el presente trabajo práctico abordamos el problema de clasificación automática de sentimientos a partir de críticas cinematográficas redactadas en idioma español. El objetivo principal es desarrollar modelos de machine learning capaces de predecir si una crítica determinada expresa una opinión positiva o negativa.

El dataset provisto para el entrenamiento contiene únicamente tres columnas: el id que es único, review_es que almacena el texto completo de la crítica, y sentimiento, que indica la polaridad de dicha crítica, pudiendo ser positiva o negativa. Este conjunto de datos cuenta con una considerable cantidad de registros, lo que permite aplicar técnicas de aprendizaje supervisado con buenos resultados. Su principal particularidad radica en la necesidad de aplicar un procesamiento específico sobre texto en lenguaje natural. Luego contamos con el dataset de test que solo contamos con los id y el review_es, a lo que nosotros con los distintos modelos realizados debíamos agregar la columna sentimiento.

A lo largo del desarrollo del trabajo, se exploraron múltiples enfoques y técnicas de modelado. En primer lugar, se aplicaron distintas estrategias de preprocesamiento sobre el texto, incluyendo: normalización de mayúsculas, eliminación de signos de puntuación, eliminación de palabras vacías (stopwords), lematización y vectorización mediante Bag of Words y TF-IDF. Posteriormente, se entrenaron y evaluaron los siguientes modelos de clasificación:

- Naïve Bayes
- Random Forest
- XGBoost
- Red Neuronal utilizando Keras y TensorFlow
- Modelo de Ensamble

Para cada uno de estos modelos se implementó una búsqueda de hiperparámetros utilizando técnicas como RandomizedSearchCV, priorizando la métrica F1-score debido a su equilibrio entre precisión y exhaustividad.

Entre las hipótesis asumidas por el equipo se encuentra la suposición de que:

- Las críticas contienen suficiente información semántica para ser clasificadas con alta precisión.
- El etiquetado de los datos es correcto y no contiene ruido significativo.
- No existen críticas neutras, por lo que el problema puede ser modelado como una clasificación binaria pura.

Este trabajo se enmarca en una competencia en la plataforma Kaggle, en la cual se realizaron envíos periódicos para validar la generalización de los modelos sobre un conjunto de datos de prueba oculto. Subiendo el csv del dataset de test, que en la misma plataforma se encargaba de sacar un score.

# Cuadro de Resultados

Realizar un cuadro de resultados comparando los modelos que entrenaron seleccionando los que a su criterio obtuvieron la mejor performance en cada caso. Deben indicar cuál es el que seleccionaron como mejor predictor de todo el TP. Confeccionar el siguiente cuadro con esta información:

Medidas de rendimiento en el conjunto de TEST:

- F1
- Precision
- Recall
- Resultado obtenido en Kaggle.

| Modelo | F1-Test | Presicion Test | Recall Test | Kaggle |
|---|---|---|---|---|
| Bayes Naive | 0.859 | 0.880 | 0.880 | 0.750 |
| Random Forest | 0.840 | 0.840 | 0.840 | 0.742 |
| XgBoost | 0.8757 | 0.8759 | 0.8757 | 0.717 |
| Red Neuronal | 0.881 | 0.879 | 0.883 | 0.789 |
| Ensamble | 0.890 | 0.890 | 0.890 | 0.751 |

# Descripción de Modelos

## Naïve Bayes

Para este modelo se utilizó el clasificador Multinomial Naïve Bayes, el cual es especialmente adecuado para tareas de clasificación de texto debido a su simplicidad, eficiencia y buenos resultados en problemas donde las características son representadas como conteos o frecuencias.

Se implementaron técnicas básicas pero efectivas de limpieza y tokenización del texto. Las críticas fueron normalizadas a minúsculas y se eliminaron espacios redundantes. La tokenización se realizó utilizando word_tokenize de NLTK. Además, se eliminaron las stopwords en español para reducir el ruido en los datos.

Para representar los textos como vectores, se utilizó el modelo Bag of Words a través de CountVectorizer, con los siguientes parámetros:

- preprocessor: función personalizada que convierte el texto a minúsculas y elimina espacios.
- tokenizer: función personalizada basada en NLTK.
- min_df=5: se eliminaron términos con muy baja frecuencia.
- stop_words=stoplist: lista de palabras vacías en español provista por NLTK.

Se entrenó el modelo utilizando un pipeline con MultinomialNB y se aplicó una búsqueda de hiperparámetros con RandomizedSearchCV, evaluando 50 combinaciones distintas en validación cruzada de 10 folds (cv=10). Los hiperparámetros optimizados fueron:

- alpha: parámetro de suavizado de Laplace, explorado en un rango logarítmico entre 0.001 y 5.0.
- fit_prior: se evaluó tanto con True como False para considerar o no la distribución previa de clases.

El criterio de evaluación utilizado fue la métrica macro F1-score, para balancear el rendimiento entre ambas clases.

## Random Forest

El segundo modelo implementado fue Random Forest, un algoritmo de ensamblado basado en árboles de decisión que construye múltiples árboles durante el entrenamiento y devuelve la clase más frecuente entre ellos. Este enfoque permite reducir la varianza del modelo y mejorar su capacidad de generalización, siendo especialmente útil en contextos con alta dimensionalidad como el análisis de texto vectorizado.

Se utilizaron exactamente las mismas técnicas de preprocesamiento que en el modelo anterior, manteniendo la coherencia en el pipeline general:

- Limpieza del texto: conversión a minúsculas y eliminación de espacios innecesarios.
- Tokenización con word_tokenize de NLTK.
- Eliminación de stopwords en español.
- Vectorización mediante Bag of Words utilizando CountVectorizer, con min_df=5.

Esto permitió transformar las críticas textuales en vectores de ocurrencias de palabras, adecuados para el entrenamiento del modelo.

Se entrenó un modelo RandomForestClassifier con búsqueda de hiperparámetros utilizando RandomizedSearchCV (50 iteraciones, validación cruzada de 5 folds). Se exploraron los siguientes parámetros:

- n_estimators: cantidad de árboles en el bosque (entre 50 y 150).
- max_depth: profundidad máxima de los árboles (entre 5 y 30).
- min_samples_split: número mínimo de muestras requeridas para dividir un nodo interno (entre 2 y 10).
- min_samples_leaf: mínimo número de muestras requeridas en una hoja (entre 1 y 10).
- max_features: número de características consideradas en cada división ("sqrt" o "log2").
- bootstrap: si se utiliza muestreo con reemplazo (True o False).

La métrica utilizada fue nuevamente el F1 macro, con el objetivo de capturar de forma equilibrada el desempeño en ambas clases (positiva y negativa), especialmente considerando que podrían no estar perfectamente balanceadas.

## XGBoost

El modelo XGBoost (Extreme Gradient Boosting) es un algoritmo de boosting que ha demostrado ser altamente eficiente y competitivo en tareas de clasificación, especialmente en entornos de alto rendimiento como competiciones de Kaggle. XGBoost construye de forma secuencial árboles de decisión, donde cada nuevo árbol intenta corregir los errores cometidos por los anteriores, optimizando una función objetivo con regularización para reducir el sobreajuste.

A diferencia de los modelos anteriores, para XGBoost se optó por una representación del texto más rica mediante TF-IDF (Term Frequency - Inverse Document Frequency). Esta técnica permite ponderar las palabras no sólo por su frecuencia en una crítica individual, sino también considerando su rareza global en el corpus. Las decisiones de vectorización fueron:

- Utilización de bigrams además de unigrams (ngram_range=(1, 2)), para capturar relaciones más complejas entre palabras.
- Eliminación de stopwords en español utilizando la lista provista por NLTK.
- Límite de 10.000 features como máximo, para mantener la eficiencia sin perder información relevante.
- Entrenamiento y búsqueda de hiperparámetros

El modelo XGBClassifier se entrenó con una búsqueda aleatoria de hiperparámetros (RandomizedSearchCV), utilizando validación cruzada de 5 folds y 10 combinaciones evaluadas. Los hiperparámetros explorados fueron:

- n_estimators: número de árboles en el modelo (entre 50 y 150).
- max_depth: profundidad máxima de cada árbol (entre 3 y 6).
- learning_rate: tasa de aprendizaje (valores aleatorios entre 0.05 y 0.25).
- subsample: fracción de muestras utilizadas para entrenar cada árbol (entre 0.7 y 1.0).
- colsample_bytree: fracción de características utilizadas en cada árbol (entre 0.7 y 1.0).

# Ensamble

Para explotar de manera complementaria las fortalezas de los diferentes modelos individuales (**XGBoost**, **Random Forest** y **Naïve Bayes**) y mitigar sus debilidades, implementamos 2 esquemas de ensamblado: **Voting Soft** y **Stacking**. Ambos enfoques combinan las predicciones de los modelos base para obtener una solución final más robusta.

### Voting Soft

Este método consiste en promediar las probabilidades de cada clase que entregan los modelos base y elegir la etiqueta con mayor probabilidad promedio (soft voting).

- **Modelos incluidos**:
  - `pipe_xgb` (XGBoost con vectorización TF-IDF, ngramas (1,2) y parámetros ajustados)
  - `pipeline_rf` (RandomForest con Bag-of-Words unigrama y parámetros ajustados)
  - `pipe_nb` (MultinomialNB con Bag-of-Words hasta trigramas y parámetros ajustados)
- **Configuración principal**:
  - `voting='soft'` para usar probabilidades en lugar de votos directos.
  - `n_jobs=-1` para paralelizar el cómputo en todos los núcleos disponibles.
- **Proceso**:
  - Ajuste del ensamblador (`voting_soft.fit`) sobre el set de entrenamiento.
  - Cálculo de las probabilidades de clase positiva de cada estimador y promedio.

- Conversión de probabilidades agregadas a etiquetas finales.
- Evaluación en el conjunto de test mediante reporte de clasificación.

---

**Stacking**

En stacking, las predicciones de los modelos base se utilizan como nuevas características de entrada para un **meta-modelo** que aprende a combinarlas de forma óptima.

**Modelos base:** (los mismos que en voting)

- `pipe_xgb`
- `pipeline_rf`
- `pipe_nb`

**Meta-modelo:**

- `LogisticRegression()`

**Parámetros clave:**

- `cv=5`: validación cruzada interna para generar predicciones out-of-fold.
- `passthrough=False`: solo se usan las predicciones de los modelos base (no las características originales).
- `n_jobs=-1`: paraleliza el ajuste tanto de los modelos base como del meta-modelo.

**Flujo de entrenamiento:**

1. Cada modelo base entrena sobre el 80 % de los datos y genera predicciones para el 20 % restante (out-of-fold), repitiendo esto en los 5 folds.
2. Se construye un nuevo dataset en el que cada muestra está representada por las predicciones de los 3 modelos base.
3. Se entrena la regresión logística sobre ese nuevo dataset.

**Predicción final:**

- Para datos nuevos, cada modelo base produce su probabilidad o etiqueta.
- El meta-modelo recibe esas 3 salidas y genera la decisión final.

**Salida de test:**

- Se crea el archivo `predicciones_stacking.csv` con las etiquetas para el dataset de prueba de Kaggle.
- Se guarda el objeto stack como pickle (`stacking_ensemble.pkl`).

Ambos enfoques de ensamblado permitieron mejorar la generalización respecto a los modelos individuales.

# Red Neuronal:

Para la resolución del problema de análisis de sentimientos en textos en español, se optó por utilizar una arquitectura basada en redes neuronales profundas preentrenadas. En particular, se empleó el modelo BERT (Bidirectional Encoder Representations from Transformers), en su variante entrenada específicamente para español.

La arquitectura empleada se compone de los siguientes componentes principales:

- **Tokenizer y Embedding Layer**:
  - Cada texto es tokenizado y convertido en vectores mediante embeddings aprendidos durante el preentrenamiento de BERT.
  - Se utilizan embeddings de posición y de tipo de segmento para preservar el orden y tipo de las frases.
- **Encoder (Transformers)**:
  - La arquitectura base de BERT cuenta con **12 capas** de tipo Transformer.
  - Cada capa incluye mecanismos de **auto-atención multi-cabeza** y subredes feedforward completamente conectadas.
  - Se aplican técnicas de normalización, residual connections y dropout para estabilizar y regularizar el entrenamiento.
- **Capa de clasificación (Classification Head)**:
  - Se toma la representación del token [CLS], que actúa como resumen del texto completo.
  - Esta representación pasa por una capa **densa lineal con 2 unidades de salida**, correspondiente a las clases "positivo" y "negativo".

El modelo fue elegido por las siguientes razones:

- · **Estado del arte en NLP**: BERT ha demostrado superar a otras arquitecturas en múltiples tareas de procesamiento de lenguaje natural, incluida la clasificación de texto.
- · **Modelo preentrenado en español**: La versión utilizada fue entrenada sobre corpus en español, lo que mejora la comprensión semántica del idioma y minimiza errores asociados a traducción o ambigüedad lingüística.
- · **Escalabilidad**: El modelo puede ajustarse a múltiples tareas simplemente modificando la capa de salida.

La arquitectura basada en BERT preentrenado para español constituye una elección robusta y adecuada para la tarea de análisis de sentimientos. Su capacidad de generalización, combinada con un proceso de fine-tuning eficiente, permitió obtener resultados competitivos manteniendo una implementación clara

## Conclusiones generales

Con el trabajo realizado en este TP, pudimos ver el árduo desarrollo punta a punta de la resolución de un problema de clasificación binaria. Comprendimos lo importante que es realizar un detenido análisis exploratorio y un correcto preprocesamiento para entender mejor la naturaleza del problema y definir más certeramente los pasos a seguir. Este trabajo práctico nos permitió además poder analizar y comparar los diferentes modelos y sus resultados, las ventajas y desventajas de cada uno y cuáles brindan un mejor resultado para nuestro caso puntual teniendo en cuenta las demandas computacionales y de tiempo que traen. Otras posibilidades que nos hubiera gustado explorar y que infelizmente tuvieron que quedar fuera de este trabajo pueden ser: una búsqueda de hiperparámetros más exhaustiva para los modelos que pese al tiempo que consumieron, no dieron los resultados deseados y la exploración con otras y más opciones de modelos en el ensamble, ya que obtuvimos muy buenos resultados en relación al tiempo y complejidad del mismo.

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

Si, fue útil realizar un análisis exploratorio de los datos ya que nos permitió comprobar que no había críticas neutras, detectar outliers como reseñas demasiado cortas o demasiado largas y encontrar hiperparámetros razonables para nuestros modelos.

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

Por supuesto. A través de la tokenización, eliminación de “stopwords” y la vectorización pudimos observar incrementos directos en las métricas que reflejaban la performance de nuestras predicciones. Ciertamente notamos unas diferencias entre el antes y después del aplicado de las tareas recién mencionadas.

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

  El modelo con mejor desempeño en test fue el ensamble, esto se debe posiblemente a que al combinar las fortalezas de los diferentes modelos utilizados, logramos reducir la varianza, el riesgo de overfitting y percibimos una mejor generalización.

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

  El modelo que obtuvo mejor desempeño en Kaggle fue el de Red Neuronal a partir de implementarlo utilizando Bert.

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

El más rápido y sencillo de entrenar fue el de Random Forest, que efectivamente es más veloz y demanda menos recursos para su ejecución, pero se ve perjudicado en términos de la capacidad de predicción frente a modelos como el de Redes o Ensamble que tardaban por lo menos media hora. Antes de usar Bert, el modelo de Red Neuronal se llevaba este podio ya que tenía un tiempo de ejecución realmente bajo para el f1 obtenido, de todas formas, sacrificamos ese bajo tiempo por un mejor resultado implementando Bert.

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

  Creeríamos que sí, aunque como no fue de los modelos más sencillos de entrenar, habría que analizar si con el volumen de datos de la situación productiva vale la pena invertir en el tiempo que demandó.

- ¿Cómo podría mejorar los resultados?

Posiblemente un mejor preprocesamiento hubiera ayudado a mejorar los resultados, aunque no encontramos mayores situaciones en las que poder perfeccionar este proceso. También buscando mejores y más afinados hiperparámetros obtendríamos unos mejores scores.

## Tareas Realizadas

Teniendo en cuenta que el trabajo práctico tuvo una duración de 9 (nueve) semanas, le pedimos a cada integrante que indique cuántas horas (en promedio) considera que dedicó semanalmente al TP

| Integrante | Principales Tareas Realizadas | Promedio Semanal (hs) |
|---|---|---|
| Nicolas Cardone | Modelos Redes Neuronales, XGBoost<br>Armado de reporte | 5 |
| Jonatan Godoy Godoy | Modelos Ensamble , Naive Bayes<br>Armado de reporte | 5 |
| Nicolas Garcia | Modelos XGBoost , Random Forest<br>Armado de reporte | 5 |