Introducción:

El objetivo de este trabajo práctico es aplicar técnicas de Procesamiento de Lenguaje Natural (PLN) para clasificar usando análisis de sentimientos si una reseña de película dada es positiva o negativa. Para ello, utilizaremos varios modelos de aprendizaje automático y compararemos su rendimiento. Los modelos seleccionados para este estudio son: Bayes Naïve, Random Forest, XGBoost, una red neuronal utilizando Keras y TensorFlow, y algunos ensambles de tipo voting y stacking de 3 modelos cada uno.

Metodología:

Recolección y preprocesamiento de datos:

Se obtuvo un conjunto de datos previamente etiquetados que contienen reseñas de películas clasificados en 2 categorías de sentimientos (positivo o negativo). Las técnicas de preprocesamiento de texto probadas fueron las siguientes:

- Eliminación de stopwords
- Normalización (todo toLower)
- Lematización
- Eliminación de reseñas en otros idiomas (aprox el 3,6% eran reseñas en inglés)
- Eliminación de tildes
- Eliminación de signos de puntuación
- Tokenización (exclusivo para Redes neuronales)
- Reducción de dimensionalidad: Se agregaban al vocabulario solamente palabras que se repitiera más de 2/5/10 veces

Cada una de estas técnicas se hacía **de a 1 a la vez** y hacíamos Cross validation con un modelo de xgboost para ver cómo se comportaba el modelo cuando se aplicaba cada técnica, con esto lograbamos saber si alguna de estas técnicas empeoraba el desempeño del modelo, este fue el caso de la técnica de **reducción de dimensionalidad**, que por más que **era bueno reduciendo el tamaño del vocabulario** (de 150k a 45k palabras) esto empeoraba bastante las métricas (f1 score de 0.85 a 0.75), **por ello al final decidimos no usar esta técnica**.

Cabe destacar que el tamaño del vocabulario del dataset original era de 175.853 y luego del preprocesamiento quedó de 153.273, representando una reducción de dimensionalidad de aprox el 13%, el cual reduce el ruido para obtener un mejor performance del modelo.

Extracción de características:

Se utilizó la representación de bolsa de palabras (Bag of Words) para convertir los textos en vectores numéricos mediante *CountVectorizer()*.

Implementación y entrenamiento de modelos:

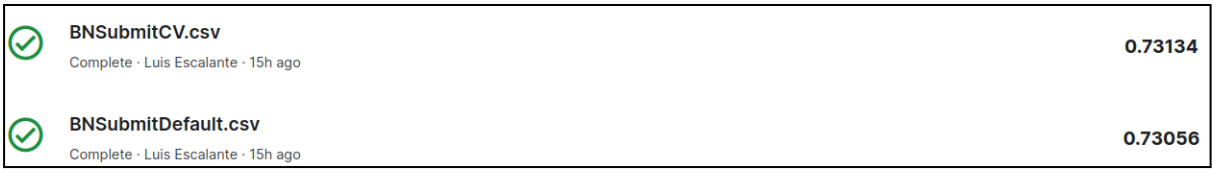
Bayes Naïve:

Este se implementó utilizando el modelo *MultinomialNB()* de la biblioteca scikit-learn en Python. Este **por lejos fue el mejor modelo en cuanto a velocidad**, entrenandose en menos de 1 segundo y consiguiendo resultados muy buenos (f1 de 0.85 en entrenamiento y 0.73 en kaggle).

En cuanto a los hiperparametros se optimizaron: *alpha y fit_prior.* el performance del default con el optimizado fue casi idéntico tanto en entrenamiento como en kaggle (métricas mencionadas arriba). (Default - CV)

	precision	recall	f1-score	support
0	0.85	0.87	0.86	7283
1	0.86	0.85	0.85	7172
accuracy			0.86	14455
macro avg	0.86	0.86	0.86	14455
weighted avg	0.86	0.86	0.86	14455

	precision	recall	f1-score	support
0	0.87	0.85	0.86	7432
1	0.85	0.86	0.85	7023
accuracy			0.86	14455
macro avg	0.86	0.86	0.86	14455
weighted avg	0.86	0.86	0.86	14455



Random Forest:

Se implementó el modelo de Random Forest utilizando la biblioteca scikit-learn en Python. Los hiperparametros optimizados fueron: *'criterion', 'min_samples_leaf', 'min_samples_split', 'ccp_alpha', 'max_depth', 'n_estimators'.* En este **lo principal a resaltar es la mejora en las métricas luego de la optimización de hiper parámetros**. ya que el default tenía métricas de (f1 de 0.84 en entrenamiento y 0.70 en kaggle) y luego el optimizado subió el f1 a 0.85 en entrenamiento y 0.73 en kaggle. (Default-CV)

	precision	recall	f1-score	support
0	0.85	0.84	0.85	7369
1	0.84	0.85	0.84	7086
accuracy			0.85	14455
macro avg	0.85	0.85	0.85	14455
weighted avg	0.85	0.85	0.85	14455

	precision	recall	f1-score	support
0	0.82	0.87	0.85	6862
1	0.88	0.83	0.85	7593
accuracy			0.85	14455
macro avg	0.85	0.85	0.85	14455
weighted avg	0.85	0.85	0.85	14455

XGBoost:

Se implementó el modelo de XGBoost utilizando la biblioteca xgboost en Python.
Se entrenó el modelo utilizando los datos de entrenamiento y se implementaron 2 modelos, uno con hiperparametros y otro sin. Los hiperparametros optimizados fueron: *'max_depth', 'min_child_weight', 'gamma', 'reg_alpha', 'reg_lambda'.* (Default-CV)

	precision	recall	f1-score	support
0	0.83	0.86	0.85	7034
1	0.86	0.83	0.85	7421
accuracy			0.85	14455
macro avg	0.85	0.85	0.85	14455
weighted avg	0.85	0.85	0.85	14455

	precision	recall	f1-score	support
0	0.83	0.86	0.84	6963
1	0.87	0.83	0.85	7492
accuracy			0.85	14455
macro avg	0.85	0.85	0.85	14455
weighted avg	0.85	0.85	0.85	14455

Red neuronal con Keras y TensorFlow:

En esta etapa, al seleccionar un modelo de redes neuronales, decidimos utilizar tres modelos diferentes con el fin de obtener conclusiones sobre su rendimiento variado. Para evitar el sobreajuste, aplicamos el método de Dropout. Además, para abordar el problema del desvanecimiento o explosión del gradiente, utilizamos la arquitectura LSTM.

RNN: Este modelo fue el que mejores resultados obtuvo en la competencia de kaggle en comparación con los otros dos modelos de redes neuronales. Para los hiper-parámetros, decidimos agregar una capa de Dropout para evitar el sobre ajuste del modelo, de esta forma, obtuvimos buenos resultados en las métricas de f1-score para los datos de entrenamiento pero las primeros hiper-parámetros que probamos nos generaron tiempos de entrenamiento de más de una hora, pero ajustando los valores del batch_size desde 32 a 128 pudimos reducir los tiempos de entrenamiento de 1hr 20 min a 15-20 min.

```
302/302 [==============================] - 3s 9ms/step
              precision    recall  f1-score   support

           0       0.87      0.84      0.86      4837
           1       0.85      0.87      0.86      4799

    accuracy                           0.86      9636
   macro avg       0.86      0.86      0.86      9636
weighted avg       0.86      0.86      0.86      9636
```

RNN Bidireccional: Realizando un research, encontramos que existe un modelo de bidireccional para RNN que podría performar mejor por su diseño de arquitectura. Si bien con los datos de entrenamiento obtuvimos aproximadamente las mismas métricas que el modelo de RNN, para la competencia de kaggle bajó en relación al modelo anterior y el tiempo .

```
313/313 [==============================] - 11s 31ms/step
              precision    recall  f1-score   support

           0       0.82      0.88      0.85      4961
           1       0.87      0.80      0.84      5039

    accuracy                           0.84     10000
   macro avg       0.84      0.84      0.84     10000
weighted avg       0.84      0.84      0.84     10000
```

Fuente: https://www.geeksforgeeks.org/bidirectional-recurrent-neural-network/

CNN: Este modelo fue, en comparación con los de redes neuronales, el que peores métricas arrojó en kaggle y peores tiempos de entrenamiento. Al igual que los modelos anteriores, aplicamos una capa de Dropout para evitar el sobreajuste.

```
302/302 [==============================] - 7s 22ms/step
              precision    recall  f1-score   support

           0       0.87      0.80      0.84      4855
           1       0.82      0.88      0.85      4782

    accuracy                           0.84      9637
   macro avg       0.84      0.84      0.84      9637
weighted avg       0.84      0.84      0.84      9637
```

Ensamble de modelos:

Se crearon dos tipos de ensambles híbridos (Voting y stacking), para ambos casos hicimos cross validation con la misma proporción de todos los modelos (70-30) con los modelos entrenados anteriormente (**xgboost, Random Forest y Bayes Naive, todos con params optimizados**). A su vez con el de tipo voting probamos con hard y soft y obtuvimos métricas muy parecidas. Para el de stacking dejamos el estimador final por default (LinearRegressionModel).

En general los 3 ensambles tenían f1 score de 0.87 en entrenamiento y rondaban los 0.74 en Kaggle. Las métricas de entrenamiento fueron:(soft-hard)

	precision	recall	f1-score	support
0	0.87	0.88	0.87	7283
1	0.87	0.87	0.87	7172
accuracy			0.87	14455
macro avg	0.87	0.87	0.87	14455
weighted avg	0.87	0.87	0.87	14455

	precision	recall	f1-score	support
0	0.89	0.84	0.86	7283
1	0.85	0.89	0.87	7172
accuracy			0.86	14455
macro avg	0.87	0.87	0.86	14455
weighted avg	0.87	0.86	0.86	14455

Submission and Description	Public Score	Select
stacking_modelSubmit.csv Complete · Luis Escalante · 4m ago · RF_params = {'n_estimators': 150, 'min_samples_split': 6, 'min_samples_leaf': 1, 'max_depth': 32, 'criterion': 'entropy', ...	0.74685	☐
VotingSoft_RF_XGB_BN_SubmitCV.csv Complete · Luis Escalante · 19h ago	0.74626	☐
stacking_modelSubmitCV.csv Complete · Luis Escalante · 18h ago	0.7451	☐
ensamble_voting_soft.csv Complete · Luis Escalante · 31m ago · RF_params = {'n_estimators': 150, 'min_samples_split': 6, 'min_samples_leaf': 1, 'max_depth': 32, 'criterion': 'entropy'...	0.74355	☐
VotingSoft_RF_XGB_BN_Submit.csv Complete · Luis Escalante · 20h ago	0.74297	☐

Conclusiones:

- Al igual que en el checkpoint 3 del TP1 los ensambles fueron los que mejor performaron tanto en entrenamiento como en test, obteniendo las 5 mejores métricas de kaggle, siendo de estas, 3 del ensamble voting de tipo soft y 2 de stacking.
- Luego le siguió un modelo de RNN con 0.74 en kaggle, y los de Bayes Naive y Random forest optimizado con 0.73
- El modelo que peor performó fue XGBoost con un máximo de 0.71 en kaggle
- En cuanto a los modelos de Redes neuronales, el mejor fue RNN en cuanto a métricas y tiempo de entrenamiento, luego investigamos que el bidireccional podría mejorar las métricas pero en nuestro caso de uso no fue así, por último el CNN fue el peor de los 3.
- Se observó una caída grande de las métricas de entrenamiento en comparación a las de Kaggle, esto creemos que pudo haber sido porque el dataset de entrenamiento y test provenían de fuentes distintas, por lo cual por más que nos encargamos de quedarnos solo con las de español, pudimos observar viendo un poco los datasets que habían distintos dialectos y por ello no pudo generalizar tan bien.
- Se evaluó la posibilidad de aplicar la estrategia mencionada en clases de crear el dialecto solamente con las reseñas del dataset de test para subir las métricas en Kaggle, pero nos pareció poco real y algo muy específico para este problema, porque si esto se dejara funcionando en un servidor productivo y procesara nuevas reseñas de otra fuente va a ser muy pobre el vocabulario y va a generalizar mucho peor.