Reporte Trabajo Práctico 2: Críticas cinematográficas

En este trabajo práctico se buscó construir varios modelos de procesamiento de lenguaje natural para clasificar una colección de críticas cinematográficas en críticas positivas y negativas. A continuación se explicarán dichos modelos y las conclusiones obtenidas.

Feature engineering

Durante el análisis de nuestro conjunto de datos, se identificó la presencia de revisiones redactadas en un idioma diferente al objetivo. Como medida, se llevó a cabo la eliminación de estas revisiones, las cuales representaban aproximadamente 2000 instancias.

Una vez obtenido un conjunto de datos libre de revisiones en otros idiomas, se procedió a realizar una limpieza adicional de las revisiones restantes, considerando aspectos como conectores lingüísticos, adverbios, verbos, pronombres y adjetivos. Con el propósito de mantener la relevancia de los datos, se decidió conservar únicamente aquellos adverbios, verbos, pronombres y adjetivos presentes en las revisiones.

Posteriormente, se realizó un análisis de las palabras cortas más frecuentes. En base a este análisis, se tomó la decisión de seleccionar las palabras de tres letras o menos que contengan una connotación positiva o negativa, descartando el resto.

Asimismo, se repitió el proceso de selección para las palabras más frecuentes en general. Aquellas palabras que poseían una carga connotativa positiva o negativa fueron preservadas, mientras que las demás fueron excluidas.

Por último, se llevó a cabo una identificación de palabras generadoras de ruido. Utilizando las palabras más comunes como referencia, se filtraron de manera sistemática de todas las revisiones previamente limpiadas.

Bayes Naive

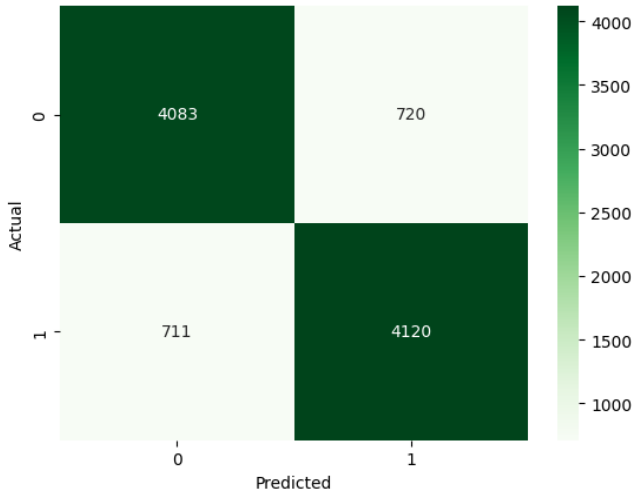
Para la búsqueda de hiper-parámetros, utilizamos GridSearch.

- Parámetros a buscar para Bayes Naive:
    - 'multinomialnb__alpha': [0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.8, 1.0],
    - 'multinomialnb__fit_prior': [True, False],
    - 'multinomialnb__class_prior': [None, [0.3, 0.7], [0.5, 0.5], [0.7, 0.3], [0.5, 0.4], [0.2, 0.7]]

- Parámetros obtenidos:
    - 'multinomialnb__alpha': 1.0,
    - 'multinomialnb__fit_prior': None,
    - 'multinomialnb__class_prior': False

- Métricas:
    - Precision: 0.85
    - Recall: 0.85
    - F1-score: 0.85

- Puntuación en Kaggle: 0.73793
- Matriz de confusión:



Random Forest

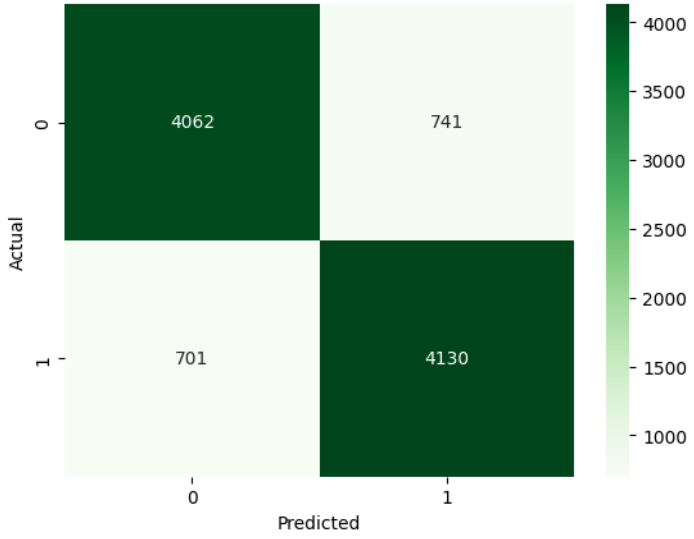
Para la búsqueda de hiper-parámetros, utilizamos GridSearch.

- Parámetros a buscar para Random Forest:
    - ‘randomforestclassifier__criterion’ : ["gini", "entropy"]
    - 'randomforestclassifier__max_depth': [10, 20, 25, 30, 45, 60]
    - 'randomforestclassifier__max_features': ['sqrt']
    - 'randomforestclassifier__n_estimators': [300, 400, 500, 600, 800, 1000]

- Parámetros obtenidos:
    - ‘randomforestclassifier__criterion’ : “gini”
    - 'randomforestclassifier__max_depth': 60
    - 'randomforestclassifier__max_features': 'sqrt’
    - 'randomforestclassifier__n_estimators': 800

- Métricas:
    - Precision: 0.85
    - Recall: 0.85
    - F1-score: 0.85

- Puntuación en Kaggle: 0.7263
- Matriz de confusión:



XGBoost

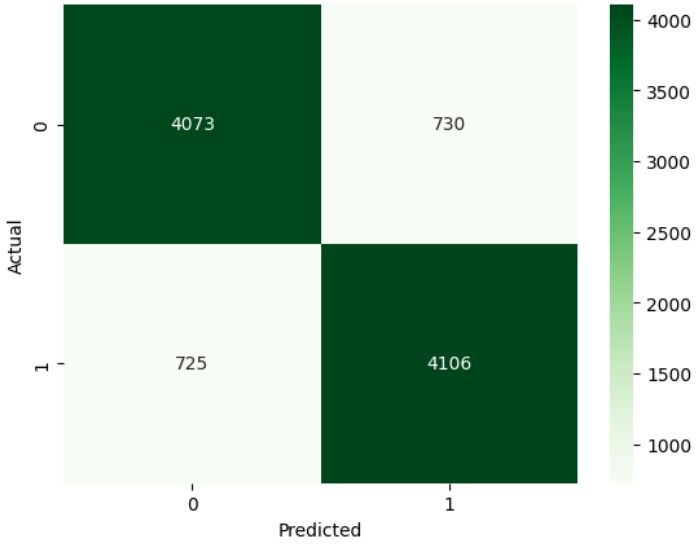
Para la búsqueda de hiper-parámetros, utilizamos RandomizedSearch.

- Parámetros a buscar para XGBoost:
    - 'xgbclassifier__n_estimators': np.array([300,400,500,600,700,800]),
    - 'xgbclassifier__subsample': np.array([0.5,0.6,0.7,0.8,0.9,1]),
    - 'xgbclassifier__max_depth': np.array([10,11,12,13,15,16,17,18,19,20]),
    - 'xgbclassifier__reg_lambda': np.array([0.1, 0.3, 0.5, 0.9, 1]),
    - 'xgbclassifier__gamma': np.array([0, 1 , 10]),
    - 'xgbclassifier__reg_alpha': np.array([0, 10, 25 ,50, 100])

- Parámetros obtenidos:
    - 'Xgbclassifier__n_estimators': `0.9`,
    - 'Xgbclassifier__subsample' `0.1`,
    - 'Xgbclassifier__max_depth': 20 ,
    - 'Xgbclassifier__reg_lambda': 0.1 ,
    - 'Xgbclassifier__gamma': 0 ,
    - 'Xgbclassifier__reg_alpha': 0

- Métricas:
    - Precision: 0.85
    - Recall: 0.85
    - F1-score: 0.85

- Puntuación en Kaggle: 0.70885
- Matriz de confusión:



Red neuronal

Arquitectura:

```
model.add(layers.Embedding(88335, 32, input_length=max_len))
model.add(layers.Conv1D(32, 3, padding='same',
activation='relu'))
model.add(layers.MaxPooling1D())
model.add(layers.Flatten())
model.add(layers.Dense(250, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=[f1_m, precision_m, recall_m])
```

Explicación de la arquitectura:
Esta arquitectura utiliza una combinación de capas de incrustación, convolución y densidad para extraer características y realizar la clasificación de sentimientos en críticas de películas. La arquitectura aprovecha la capacidad de aprendizaje de las redes neuronales para captar patrones complejos en datos de texto y proporciona una representación semántica de las palabras para comprender mejor el contexto.

Métricas:

- F1-score: 0.9411
- Precision: 0.9396
- Recall: 0.9439

Puntuación en Kaggle: 0.49311

Arquitectura optimizada:

```
model.add(Embedding(vocab_size, output_dim =
hp.Choice('embed_size',values=[60, 80, 100]), input_length =
max_len))
model.add(Conv1D(filters =hp.Choice('conv_filters',values=[50,
60, 80]), kernel_size =
hp.Choice('conv_kernel',values=[10,15,20,30]), padding='valid',
activation='relu', strides=1))
model.add(layers.MaxPooling1D())
model.add(layers.Flatten())
model.add(layers.Dense(hp.Choice('dense_size',values=[65, 125,
250, 300]), activation='relu'))
model.add(Dropout(hp.Choice('dropout',values=[0.3, 0.4, 0.5,
0.6])))
model.add(Dense(1, activation='sigmoid'))
```

Explicación de la arquitectura:
Para optimizar nuestra red neuronal utilizamos keras-tuner para poder construir una arquitectura dinámica que nos permita evaluar las distintas combinaciones posibles ya sea de hiperparametros o arquitectura.

Y los valores que se obtuvieron para cada layer son:

- 'loss': 'binary_crossentropy',
- 'learning_rate': 0.0001,
- 'embed_size': 80,
- 'conv_filters': 50, 'conv_kernel': 15,
- 'dense_size': 125,
- 'dropout': 0.3,
- 'optimizer': 'RMSprop'

Métricas:

- F1-score: 0.9140316247940063
- Recall: 0.9194295406341553
- Precision: 0.9114036560058594

Puntuación en Kaggle: 0.49563

Stacking (Bayes Naive, Random Forest y XGBoost)

- Parámetros a buscar con Bayes Naive:
    - 'multinomialnb__alpha': 0.4,
    - 'multinomialnb__class_prior': [0.5, 0.5],
    - 'multinomialnb__fit_prior': True,
    - 'tfidfvectorizer__max_features': 5500

- Parámetros a buscar con Random Forest:
    - 'randomforestclassifier__criterion': 'entropy',
    - 'randomforestclassifier__max_depth': 60,
    - 'randomforestclassifier__max_features': 'sqrt',
    - 'randomforestclassifier__n_estimators': 1000

- Parámetros a buscar con XGBoost:
    - 'xgbclassifier__subsample': 0.5,
    - 'xgbclassifier__reg_lambda': 0.3,
    - 'xgbclassifier__reg_alpha': 0,
    - 'xgbclassifier__n_estimators': 400,
    - 'xgbclassifier__max_depth': 11,
    - 'xgbclassifier__gamma': 10

- Parámetros obtenidos con Bayes Naive:
    - "multinomialnb__alpha": 0.4,
    - "multinomialnb__class_prior": null,
    - "multinomialnb__fit_prior": true

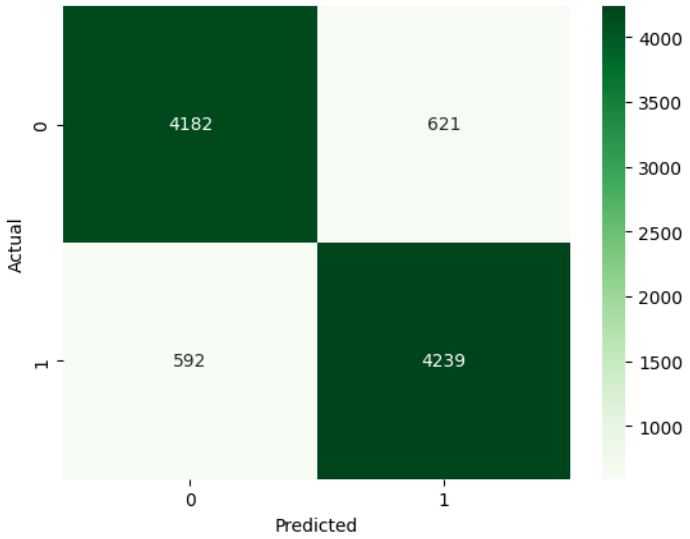
- Parámetros obtenidos con Random Forest:
    - 'randomforestclassifier__criterion': 'entropy',
    - 'randomforestclassifier__max_depth': 60,
    - 'randomforestclassifier__max_features': 'sqrt'
    - 'randomforestclassifier__n_estimators': 1000

- Parámetros obtenidos con XGBoost:
    - 'xgbclassifier__subsample': 0.5,
    - 'xgbclassifier__reg_lambda': 0.3,
    - 'xgbclassifier__reg_alpha': 0,
    - 'xgbclassifier__n_estimators': 400,
    - 'xgbclassifier__max_depth': 11,
    - 'xgbclassifier__gamma': 10}

- Métricas:
    - Precision: 0.87

    - Recall: 0.88
    - F1-score: 0.87

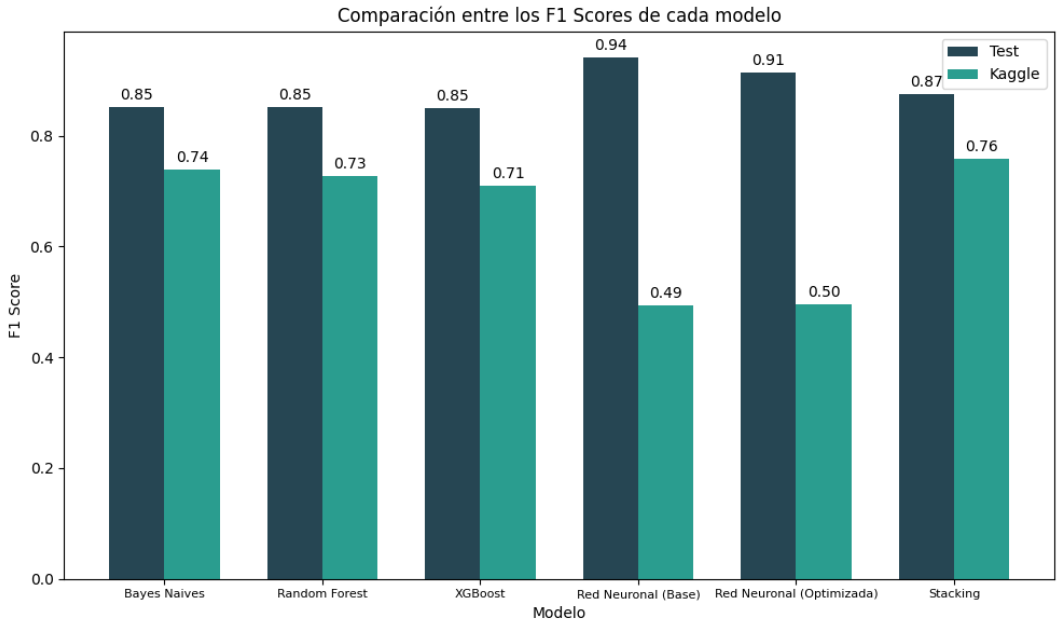
- Puntuación en Kaggle: 0.75828
- Matriz de confusión:



Conclusiones

Luego de haber procesado los datos y entrenado los modelos, podemos ver que el mejor fue Stacking, el cual obtuvo un F1-score de 0.87 y un puntaje de 0.75828 en Kaggle. También podemos notar que hubo un diferencia grande entre el F1-score local y el que se observa en Kaggle, tanto con la red base como con la red optimizada. Sin embargo, se probaron varios modelos de distintas complejidades y nunca logramos reducir esta diferencia.

Por último, en el siguiente gráfico se pueden observar los F1-scores de cada modelo en comparación con los de Kaggle:


Comparación entre los F1 Scores de cada modelo
Test
Kaggle
F1 Score
0.0
0.2
0.4
0.6
0.8
0.85
0.74
0.85
0.73
0.85
0.71
0.94
0.49
0.91
0.50
0.87
0.76
Bayes Naives
Random Forest
XGBoost
Red Neuronal (Base)
Red Neuronal (Optimizada)
Stacking
Modelo