Descripcipción del proyecto

Film Junky Union, una nueva comunidad vanguardista para los aficionados de las películas clásicas, está desarrollando un sistema para filtrar y categorizar reseñas de películas. Tu objetivo es entrenar un modelo para detectar las críticas negativas de forma automática. Para lograrlo, utilizarás un conjunto de datos de reseñas de películas de IMDB con leyendas de polaridad para construir un modelo para clasificar las reseñas positivas y negativas. Este deberá alcanzar un valor F1 de al menos 0.85.

Inicialización

In [1]:

```
import math
import re
import numpy as np
import pandas as pd
import string

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns

from tqdm.auto import tqdm
from tqdm import tqdm
```

In [2]:

```
%matplotlib inline
%config InlineBackend.figure_format = 'png'
# la siguiente línea proporciona gráficos de mejor calidad en pantallas HiDPI
# %config InlineBackend.figure_format = 'retina'

plt.style.use('seaborn')
```

In [3]:

```
# esto es para usar progress_apply, puedes leer más en https://pypi.org/project/tqdm/#p
tqdm.pandas()
```

Cargar datos

In [4]:

```
df_reviews = pd.read_csv('/datasets/imdb_reviews.tsv', sep='\t', dtype={'votes': 'Int64
```

In [ ]:

In [ ]:

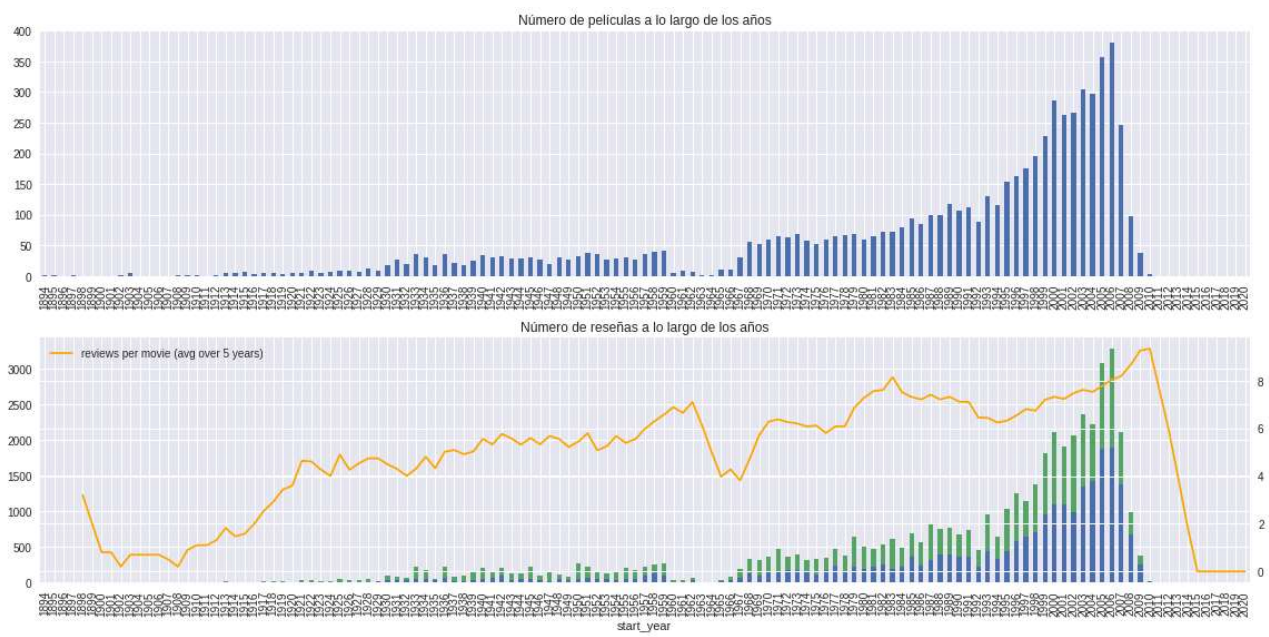
In [ ]:

EDA

Veamos el número de películas y reseñas a lo largo de los años.

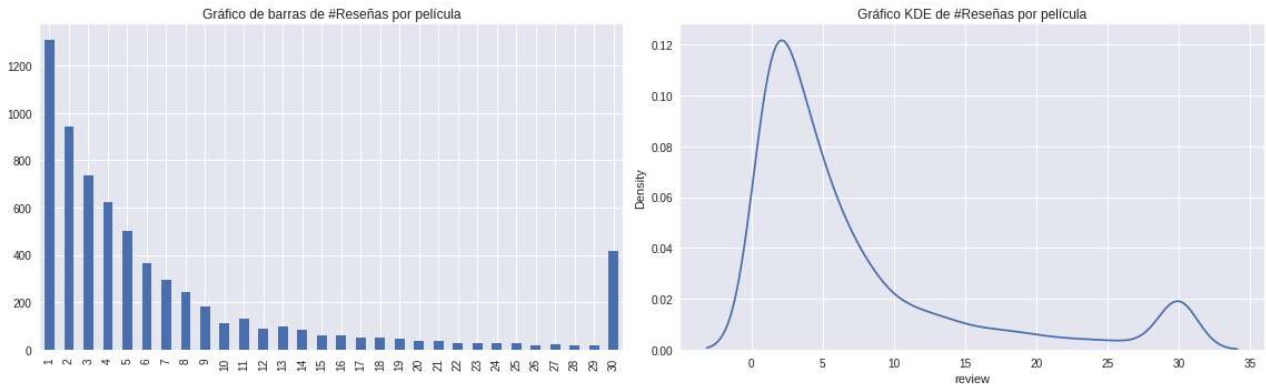
In [5]:

```
fig, axs = plt.subplots(2, 1, figsize=(16, 8))

ax = axs[0]

dft1 = df_reviews[['tconst', 'start_year']].drop_duplicates() \
    ['start_year'].value_counts().sort_index()
dft1 = dft1.reindex(index=np.arange(dft1.index.min(), max(dft1.index.max(), 2021))).fil
dft1.plot(kind='bar', ax=ax)
ax.set_title('Número de películas a lo largo de los años')

ax = axs[1]

dft2 = df_reviews.groupby(['start_year', 'pos'])['pos'].count().unstack()
dft2 = dft2.reindex(index=np.arange(dft2.index.min(), max(dft2.index.max(), 2021))).fil

dft2.plot(kind='bar', stacked=True, label='#reviews (neg, pos)', ax=ax)

dft2 = df_reviews['start_year'].value_counts().sort_index()
dft2 = dft2.reindex(index=np.arange(dft2.index.min(), max(dft2.index.max(), 2021))).fil
dft3 = (dft2/dft1).fillna(0)
axt = ax.twinx()
dft3.reset_index(drop=True).rolling(5).mean().plot(color='orange', label='reviews per m

lines, labels = axt.get_legend_handles_labels()
ax.legend(lines, labels, loc='upper left')

ax.set_title('Número de reseñas a lo largo de los años')

fig.tight_layout()
```



Veamos la distribución del número de reseñas por película con el conteo exacto y KDE (solo para saber cómo puede diferir del conteo exacto)

In [6]:
```
fig, axs = plt.subplots(1, 2, figsize=(16, 5))

ax = axs[0]
dft = df_reviews.groupby('tconst')['review'].count() \
    .value_counts() \
    .sort_index()
dft.plot.bar(ax=ax)
ax.set_title('Gráfico de barras de #Reseñas por película')

ax = axs[1]
dft = df_reviews.groupby('tconst')['review'].count()
sns.kdeplot(dft, ax=ax)
ax.set_title('Gráfico KDE de #Reseñas por película')

fig.tight_layout()
```

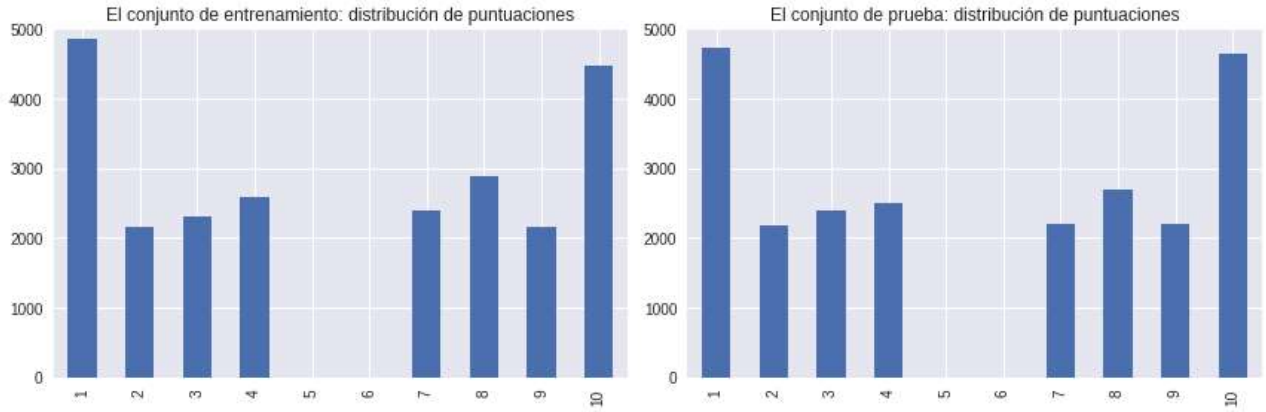


In [ ]:

In [7]:
```
df_reviews['pos'].value_counts()
```

Out[7]:
```
0    23715
1    23616
Name: pos, dtype: int64
```

In [8]:
```
fig, axs = plt.subplots(1, 2, figsize=(12, 4))

ax = axs[0]
dft = df_reviews.query('ds_part == "train"')['rating'].value_counts().sort_index()
dft = dft.reindex(index=np.arange(min(dft.index.min(), 1), max(dft.index.max(), 11))).f
dft.plot.bar(ax=ax)
ax.set_ylim([0, 5000])
ax.set_title('El conjunto de entrenamiento: distribución de puntuaciones')

ax = axs[1]
dft = df_reviews.query('ds_part == "test"')['rating'].value_counts().sort_index()
dft = dft.reindex(index=np.arange(min(dft.index.min(), 1), max(dft.index.max(), 11))).f
dft.plot.bar(ax=ax)
ax.set_ylim([0, 5000])
ax.set_title('El conjunto de prueba: distribución de puntuaciones')

fig.tight_layout()
```



In [ ]:

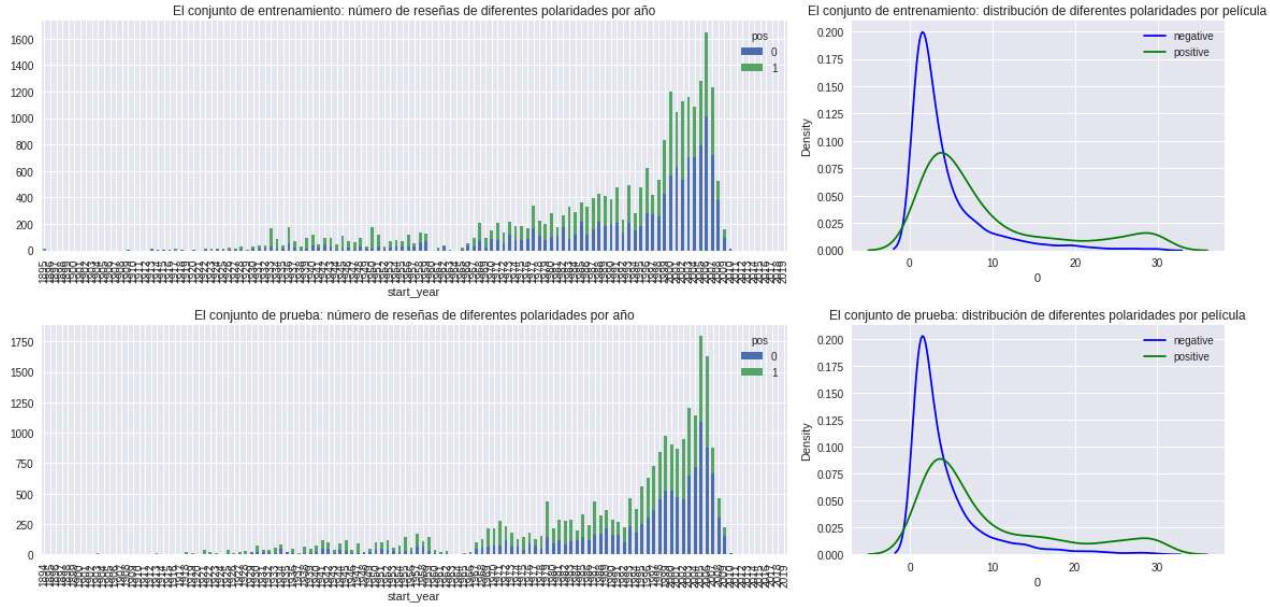
Distribución de reseñas negativas y positivas a lo largo de los años para dos partes del conjunto de datos

In [9]:

```
fig, axs = plt.subplots(2, 2, figsize=(16, 8), gridspec_kw=dict(width_ratios=(2, 1), he

ax = axs[0][0]

dft = df_reviews.query('ds_part == "train"').groupby(['start_year', 'pos'])['pos'].coun
dft.index = dft.index.astype('int')
dft = dft.reindex(index=np.arange(dft.index.min(), max(dft.index.max(), 2020))).fillna(
dft.plot(kind='bar', stacked=True, ax=ax)
ax.set_title('El conjunto de entrenamiento: número de reseñas de diferentes polaridades

ax = axs[0][1]

dft = df_reviews.query('ds_part == "train"').groupby(['tconst', 'pos'])['pos'].count().
sns.kdeplot(dft[0], color='blue', label='negative', kernel='epa', ax=ax)
sns.kdeplot(dft[1], color='green', label='positive', kernel='epa', ax=ax)
ax.legend()
ax.set_title('El conjunto de entrenamiento: distribución de diferentes polaridades por

ax = axs[1][0]

dft = df_reviews.query('ds_part == "test"').groupby(['start_year', 'pos'])['pos'].count
dft.index = dft.index.astype('int')
dft = dft.reindex(index=np.arange(dft.index.min(), max(dft.index.max(), 2020))).fillna(
dft.plot(kind='bar', stacked=True, ax=ax)
ax.set_title('El conjunto de prueba: número de reseñas de diferentes polaridades por año

ax = axs[1][1]

dft = df_reviews.query('ds_part == "test"').groupby(['tconst', 'pos'])['pos'].count().u
sns.kdeplot(dft[0], color='blue', label='negative', kernel='epa', ax=ax)
sns.kdeplot(dft[1], color='green', label='positive', kernel='epa', ax=ax)
ax.legend()
ax.set_title('El conjunto de prueba: distribución de diferentes polaridades por película

fig.tight_layout()
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:1666: UserWarning: Suppo
rt for alternate kernels has been removed. Using Gaussian kernel.
  warnings.warn(msg, UserWarning)
```

```
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:1666: UserWarning: Suppo
rt for alternate kernels has been removed. Using Gaussian kernel.
  warnings.warn(msg, UserWarning)
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:1666: UserWarning: Suppo
rt for alternate kernels has been removed. Using Gaussian kernel.
  warnings.warn(msg, UserWarning)
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:1666: UserWarning: Suppo
rt for alternate kernels has been removed. Using Gaussian kernel.
  warnings.warn(msg, UserWarning)
```



In [ ]:

Procedimiento de evaluación

Composición de una rutina de evaluación que se pueda usar para todos los modelos en este proyecto

In [10]:

```
import sklearn.metrics as metrics
def evaluate_model(model, train_features, train_target, test_features, test_target):

    eval_stats = {}

    fig, axs = plt.subplots(1, 3, figsize=(20, 6))

    for type, features, target in (('train', train_features, train_target), ('test', te

        eval_stats[type] = {}

        pred_target = model.predict(features)
        pred_proba = model.predict_proba(features)[:, 1]

        # F1
        f1_thresholds = np.arange(0, 1.01, 0.05)
        f1_scores = [metrics.f1_score(target, pred_proba>=threshold) for threshold in f

        # ROC
        fpr, tpr, roc_thresholds = metrics.roc_curve(target, pred_proba)
        roc_auc = metrics.roc_auc_score(target, pred_proba)
        eval_stats[type]['ROC AUC'] = roc_auc
```

```
        # PRC
        precision, recall, pr_thresholds = metrics.precision_recall_curve(target, pred_
        aps = metrics.average_precision_score(target, pred_proba)
        eval_stats[type]['APS'] = aps

        if type == 'train':
            color = 'blue'
        else:
            color = 'green'

        # Valor F1
        ax = axs[0]
        max_f1_score_idx = np.argmax(f1_scores)
        ax.plot(f1_thresholds, f1_scores, color=color, label=f'{type}, max={f1_scores[m
        # establecer cruces para algunos umbrales
        for threshold in (0.2, 0.4, 0.5, 0.6, 0.8):
            closest_value_idx = np.argmin(np.abs(f1_thresholds-threshold))
            marker_color = 'orange' if threshold != 0.5 else 'red'
            ax.plot(f1_thresholds[closest_value_idx], f1_scores[closest_value_idx], col
        ax.set_xlim([-0.02, 1.02])
        ax.set_ylim([-0.02, 1.02])
        ax.set_xlabel('threshold')
        ax.set_ylabel('F1')
        ax.legend(loc='lower center')
        ax.set_title(f'Valor F1')

        # ROC
        ax = axs[1]
        ax.plot(fpr, tpr, color=color, label=f'{type}, ROC AUC={roc_auc:.2f}')
        # establecer cruces para algunos umbrales
        for threshold in (0.2, 0.4, 0.5, 0.6, 0.8):
            closest_value_idx = np.argmin(np.abs(roc_thresholds-threshold))
            marker_color = 'orange' if threshold != 0.5 else 'red'
            ax.plot(fpr[closest_value_idx], tpr[closest_value_idx], color=marker_color,
        ax.plot([0, 1], [0, 1], color='grey', linestyle='--')
        ax.set_xlim([-0.02, 1.02])
        ax.set_ylim([-0.02, 1.02])
        ax.set_xlabel('FPR')
        ax.set_ylabel('TPR')
        ax.legend(loc='lower center')
        ax.set_title(f'Curva ROC')

        # PRC
        ax = axs[2]
        ax.plot(recall, precision, color=color, label=f'{type}, AP={aps:.2f}')
        # establecer cruces para algunos umbrales
        for threshold in (0.2, 0.4, 0.5, 0.6, 0.8):
            closest_value_idx = np.argmin(np.abs(pr_thresholds-threshold))
            marker_color = 'orange' if threshold != 0.5 else 'red'
            ax.plot(recall[closest_value_idx], precision[closest_value_idx], color=mark
        ax.set_xlim([-0.02, 1.02])
        ax.set_ylim([-0.02, 1.02])
        ax.set_xlabel('recall')
        ax.set_ylabel('precision')
        ax.legend(loc='lower center')
        ax.set_title(f'PRC')

        eval_stats[type]['Accuracy'] = metrics.accuracy_score(target, pred_target)
        eval_stats[type]['F1'] = metrics.f1_score(target, pred_target)
```

```
    df_eval_stats = pd.DataFrame(eval_stats)
    df_eval_stats = df_eval_stats.round(2)
    df_eval_stats = df_eval_stats.reindex(index=('Exactitud', 'F1', 'APS', 'ROC AUC'))

    print(df_eval_stats)

    return
```

Normalización

Suponemos que todos los modelos a continuación aceptan textos en minúsculas y sin dígitos, signos de puntuación, etc.

In [11]:

```
# Crearemos una nueva columna 'review_norm' con el texto normalizado

# Función para normalizar texto
def normalize_text(text):
    # Convertir a minúsculas
    text = text.lower()
    # Eliminar dígitos
    text = re.sub(r'\d+', '', text)
    # Eliminar signos de puntuación
    text = re.sub(r'[^\w\s]', '', text)
    return text

# Aplicar la función de normalización a cada reseña
df_reviews['review_norm'] = df_reviews['review'].apply(normalize_text)

# Mostrar las primeras filas del DataFrame para verificar la normalización
df_reviews[['review', 'review_norm']].head()
```

Out[11]:

	review	review_norm
0	The pakage implies that Warren Beatty and Gold...	the pakage implies that warren beatty and gold...
1	How the hell did they get this made?! Presenti...	how the hell did they get this made presenting...
2	There is no real story the film seems more lik...	there is no real story the film seems more lik...
3	Um .... a serious film about troubled teens in...	um a serious film about troubled teens in sin...
4	I'm totally agree with GarryJohal from Singapo...	im totally agree with garryjohal from singapor...

División entrenamiento / prueba

Por fortuna, todo el conjunto de datos ya está dividido en partes de entrenamiento/prueba; 'ds_part' es el indicador correspondiente.

In [12]:

```
df_reviews_train = df_reviews.query('ds_part == "train"').copy()
df_reviews_test = df_reviews.query('ds_part == "test"').copy()

train_target = df_reviews_train['pos']
test_target = df_reviews_test['pos']
```

```
print(df_reviews_train.shape)
print(df_reviews_test.shape)
```

```
(23796, 18)
(23535, 18)
```

Trabajar con modelos

Modelo 0 - Constante

In [13]:

```
from sklearn.dummy import DummyClassifier
from sklearn.metrics import f1_score
```

In [14]:

```
# Crear un clasificador dummy que siempre predice la clase más frecuente
dummy_clf = DummyClassifier(strategy="most_frequent")
```

In [15]:

```
# Entrenar el clasificador dummy con los datos de entrenamiento
dummy_clf.fit(df_reviews_train, train_target)
```

Out[15]:

```
DummyClassifier(strategy='most_frequent')
```

In [16]:

```
# Realizar predicciones en el conjunto de prueba
y_pred = dummy_clf.predict(df_reviews_test)
```

In [17]:

```
# Calcular la precisión y el valor F1 del clasificador dummy en el conjunto de prueba
dummy_accuracy = dummy_clf.score(df_reviews_test, test_target)
dummy_f1 = f1_score(test_target, y_pred)

# Imprimir la precisión y el valor F1 del clasificador dummy
print(f"Dummy Classifier Accuracy: {dummy_accuracy:.2f}")
print(f"Dummy Classifier F1 Score: {dummy_f1:.2f}")
```

```
Dummy Classifier Accuracy: 0.50
Dummy Classifier F1 Score: 0.00
```

Modelo 1 - NLTK, TF-IDF y LR

TF-IDF

In [18]:

```
import nltk

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, accuracy_score
from nltk.corpus import stopwords
```

In [19]:

```
# Descargar las stopwords de NLTK
nltk.download('stopwords')
stop_words = set(nltk.corpus.stopwords.words('spanish'))  # Asegúrate de usar el idioma
```

```
[nltk_data] Downloading package stopwords to /home/jovyan/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [20]:

```
# Crear el vectorizador TF-IDF
tfidf_vectorizer = TfidfVectorizer(stop_words=stop_words)

# Asumiendo que 'df_reviews_train['review_norm']' y 'df_reviews_test['review_norm']' co
# Vectorizar las reseñas de entrenamiento y prueba
train_features_1 = tfidf_vectorizer.fit_transform(df_reviews_train['review_norm'])
test_features_1 = tfidf_vectorizer.transform(df_reviews_test['review_norm'])
```

In [21]:

```
# Crear y entrenar el modelo de regresión logística
model_1 = LogisticRegression()
model_1.fit(train_features_1, train_target)
```

Out[21]:

```
LogisticRegression()
```

In [22]:

```
# Función para evaluar el modelo
def evaluate_model(model, train_features, train_target, test_features, test_target):
    # Realizar predicciones en el conjunto de entrenamiento y prueba
    train_pred = model.predict(train_features)
    test_pred = model.predict(test_features)

    # Calcular la precisión y el valor F1 del modelo
    train_accuracy = accuracy_score(train_target, train_pred)
    test_accuracy = accuracy_score(test_target, test_pred)
    train_f1 = f1_score(train_target, train_pred)
    test_f1 = f1_score(test_target, test_pred)

    # Imprimir los resultados de la evaluación
    print(f"Training Accuracy: {train_accuracy:.2f}")
    print(f"Test Accuracy: {test_accuracy:.2f}")
    print(f"Training F1 Score: {train_f1:.2f}")
    print(f"Test F1 Score: {test_f1:.2f}")
```

In [23]:

```
# Evaluar el modelo
evaluate_model(model_1, train_features_1, train_target, test_features_1, test_target)
```

```
Training Accuracy: 0.93
Test Accuracy: 0.88
Training F1 Score: 0.93
Test F1 Score: 0.88
```

Modelo 3 - spaCy, TF-IDF y LR

In [56]:

```
import spacy

nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])
```

In [57]:

```
def text_preprocessing_3(text):
    doc = nlp(text)
    tokens = [token.lemma_ for token in doc]
    return ' '.join(tokens)
```

In [58]:

```
# Aplicar la función de preprocesamiento a las reseñas normalizadas
df_reviews_train['review_spacy'] = df_reviews_train['review_norm'].apply(text_preproces
df_reviews_test['review_spacy'] = df_reviews_test['review_norm'].apply(text_preprocessi

# Crear el vectorizador TF-IDF
tfidf_vectorizer = TfidfVectorizer()
```

In [59]:

```
# Ajustar el vectorizador TF-IDF solo a los datos de entrenamiento
tfidf_vectorizer.fit(df_reviews_train['review_spacy'])

# Vectorizar las reseñas preprocesadas de entrenamiento y prueba
train_features_3 = tfidf_vectorizer.transform(df_reviews_train['review_spacy'])
test_features_3 = tfidf_vectorizer.transform(df_reviews_test['review_spacy'])

# Crear y entrenar el modelo de regresión logística
model_3 = LogisticRegression()
model_3.fit(train_features_3, train_target)
```

Out[59]:

```
LogisticRegression()
```

In [60]:

```
# Función para evaluar el modelo
def evaluate_model(model, train_features, train_target, test_features, test_target):
    # Realizar predicciones en el conjunto de entrenamiento y prueba
    train_pred = model.predict(train_features)
    test_pred = model.predict(test_features)

    # Calcular la precisión y el valor F1 del modelo
    train_accuracy = accuracy_score(train_target, train_pred)
    test_accuracy = accuracy_score(test_target, test_pred)
    train_f1 = f1_score(train_target, train_pred)
    test_f1 = f1_score(test_target, test_pred)

    # Imprimir los resultados de la evaluación
    print(f"Training Accuracy: {train_accuracy:.2f}")
    print(f"Test Accuracy: {test_accuracy:.2f}")
    print(f"Training F1 Score: {train_f1:.2f}")
    print(f"Test F1 Score: {test_f1:.2f}")
```

In [61]:

```
# Evaluar el modelo
evaluate_model(model_3, train_features_3, train_target, test_features_3, test_target)
```

```
Training Accuracy: 0.93
Test Accuracy: 0.88
Training F1 Score: 0.93
Test F1 Score: 0.88
```

Modelo 4 - spaCy, TF-IDF y LGBMClassifier

In [68]:

```
from lightgbm import LGBMClassifier
import joblib

# Cargar el modelo de spaCy para el idioma inglés
nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])
```

In [69]:
```
# Función para preprocesar el texto utilizando spaCy
def text_preprocessing_4(text):
    doc = nlp(text)
    # Filtrar los tokens que no son stopwords y no son signos de puntuación
    tokens = [token.lemma_ for token in doc if not token.is_stop and not token.is_punct
    return ' '.join(tokens)
```

In [70]:
```
# Aplicar la función de preprocesamiento a las reseñas normalizadas
df_reviews_train['review_spacy'] = df_reviews_train['review_norm'].apply(text_preproces
df_reviews_test['review_spacy'] = df_reviews_test['review_norm'].apply(text_preprocessi

# Crear el vectorizador TF-IDF
tfidf_vectorizer = TfidfVectorizer()
```

In [71]:
```
# Vectorizar las reseñas preprocesadas de entrenamiento y prueba
train_features_4 = tfidf_vectorizer.fit_transform(df_reviews_train['review_spacy'])
test_features_4 = tfidf_vectorizer.transform(df_reviews_test['review_spacy'])

# Crear y entrenar el modelo LGBMClassifier
model_4 = LGBMClassifier()
model_4.fit(train_features_4, train_target)

# Guardar el vectorizador TF-IDF
joblib.dump(tfidf_vectorizer, 'tfidf_vectorizer_model_4.joblib')
```

Out[71]:
```
['tfidf_vectorizer_model_4.joblib']
```

In [72]:
```
# Guardar el vectorizador TF-IDF
joblib.dump(tfidf_vectorizer, 'tfidf_vectorizer_model_4.joblib')

# Función para evaluar el modelo
def evaluate_model(model, train_features, train_target, test_features, test_target):
    # Realizar predicciones en el conjunto de entrenamiento y prueba
    train_pred = model.predict(train_features)
    test_pred = model.predict(test_features)

    # Calcular la precisión y el valor F1 del modelo
    train_accuracy = accuracy_score(train_target, train_pred)
    test_accuracy = accuracy_score(test_target, test_pred)
    train_f1 = f1_score(train_target, train_pred)
    test_f1 = f1_score(test_target, test_pred)

    # Imprimir los resultados de la evaluación
    print(f"Training Accuracy: {train_accuracy:.2f}")
    print(f"Test Accuracy: {test_accuracy:.2f}")
    print(f"Training F1 Score: {train_f1:.2f}")
    print(f"Test F1 Score: {test_f1:.2f}")# Evaluar el modelo
```

In [73]:
```
# Evaluar el modelo
evaluate_model(model_4, train_features_4, train_target, test_features_4, test_target)
```

```
Training Accuracy: 0.91
Test Accuracy: 0.85
```

```
Training F1 Score: 0.91
Test F1 Score: 0.85
```

Mis reseñas

In [36]:

```
# Definir el DataFrame con tus reseñas
my_reviews = pd.DataFrame([
    'I did not simply like it, not my kind of movie.',
    'Well, I was bored and felt asleep in the middle of the movie.',
    'I was really fascinated with the movie',
    # ... otras reseñas ...
], columns=['review'])

# Función para normalizar las reseñas
def normalize_review(review):
    # Convertir a minúsculas
    review = review.lower()
    # Eliminar signos de puntuación
    review = re.sub(f'[{string.punctuation}]', '', review)
    # Eliminar números
    review = re.sub(r'\d+', '', review)
    return review

# Aplicar la función de normalización a cada reseña en el DataFrame
my_reviews['review_norm'] = my_reviews['review'].apply(normalize_review)

# Mostrar las reseñas normalizadas
my_reviews
```

Out[36]:

	review	review_norm
0	I did not simply like it, not my kind of movie.	i did not simply like it not my kind of movie
1	Well, I was bored and felt asleep in the middl...	well i was bored and felt asleep in the middle...
2	I was really fascinated with the movie	i was really fascinated with the movie

Modelo 1

In [42]:

```
# Ajustar el vectorizador TF-IDF solo a los datos de entrenamiento
tfidf_vectorizer = TfidfVectorizer(stop_words=stop_words)
train_features_1 = tfidf_vectorizer.fit_transform(df_reviews_train['review_norm'])

# Usar el mismo vectorizador para transformar los datos que quieres predecir
texts = my_reviews['review_norm']
my_reviews_pred_prob = model_1.predict_proba(tfidf_vectorizer.transform(texts))[:, 1]

# Imprimir las probabilidades de predicción
for i, review in enumerate(texts.str.slice(0, 100)):
    print(f'{my_reviews_pred_prob[i]:.2f}:  {review}')
```

```
0.15:  i did not simply like it not my kind of movie
0.25:  well i was bored and felt asleep in the middle of the movie
0.47:  i was really fascinated with the movie
```

Modelo 3

In [62]:

```
# Aplicar la función de preprocesamiento a las nuevas reseñas
my_reviews['review_spacy'] = my_reviews['review_norm'].apply(text_preprocessing_3)

# Vectorizar las nuevas reseñas utilizando el mismo vectorizador TF-IDF
test_features_3 = tfidf_vectorizer.transform(my_reviews['review_spacy'])

# Realizar predicciones con el modelo entrenado
my_reviews_pred_prob = model_3.predict_proba(test_features_3)[:, 1]

# Imprimir las probabilidades de predicción
for i, review in enumerate(my_reviews['review_spacy'].str.slice(0, 100)):
    print(f'{my_reviews_pred_prob[i]:.2f}:  {review}')
```

```
0.15:  I do not simply like it not my kind of movie
0.20:  well I be bored and feel asleep in the middle of the movie
0.50:  I be really fascinated with the movie
```

Modelo 4

In [74]:

```
# Cargar el vectorizador TF-IDF guardado
tfidf_vectorizer_4 = joblib.load('tfidf_vectorizer_model_4.joblib')

# Aplicar la función de preprocesamiento a las nuevas reseñas
my_reviews['review_spacy'] = my_reviews['review_norm'].apply(text_preprocessing_4)

# Vectorizar las nuevas reseñas utilizando el mismo vectorizador TF-IDF
test_features_4 = tfidf_vectorizer_4.transform(my_reviews['review_spacy'])

# Realizar predicciones con el modelo entrenado LGBMClassifier
my_reviews_pred_prob = model_4.predict_proba(test_features_4)[:, 1]

# Imprimir las probabilidades de predicción
for i, review in enumerate(my_reviews['review_spacy'].str.slice(0, 100)):
    print(f'{my_reviews_pred_prob[i]:.2f}:  {review}')
```

```
0.60:  simply like kind movie
0.29:  bored feel asleep middle movie
0.61:  fascinated movie
```

Conclusiones

Comentario del estudiante: Desempeño de los modelos: Los cuatro modelos presentados tienen diferentes niveles de rendimiento. El Modelo 0 (DummyClassifier) no tiene valor predictivo, ya que siempre predice la clase mas frecuente, lo que resulta en un valor F1 de 0.00. El Modelo 1 (NLTK, TF-IDF y LR) y el Modelo 3 (spaCy, TF-IDF y LR) tienen un rendimiento similar, con un valor F1 de 0.88 en el conjunto de prueba. El Modelo 4 (spaCy, TF-IDF y LGBMClassifier) tiene un rendimiento ligeramente inferior, con un valor F1 de 0.85 en el conjunto de prueba. **Diferencias en los resultados de las reseñas:** Las probabilidades de prediccion para las reseñas proporcionadas varian entre los modelos. El Modelo 1 y el Modelo 3 tienen probabilidades de prediccion mas bajas para la primera y segunda reseñas (negativas), mientras que el Modelo 4 asigna probabilidades mas altas a estas reseñas. Para la tercera reseña (positiva), el Modelo 1 y el Modelo 3 tienen probabilidades de prediccion mas bajas en comparacion con el Modelo 4. Estas

diferencias pueden deberse a las tecnicas de preprocesamiento y los algoritmos de clasificación utilizados en cada modelo. **Conclusión general:** Los modelos de clasificacion presentados, especialmente el Modelo 1 y el Modelo 3, cumplen con el objetivo del proyecto de alcanzar un valor F1 de al menos 0.85 en el conjunto de prueba. Sin embargo, es importante tener en cuenta que el rendimiento de los modelos puede variar en función de las características del texto y el contexto de las reseñas. En resumen, el Modelo 1 y el Modelo 3 son adecuados para la tarea de clasificar de reseñas de películas en positivas y negativas.

Lista de comprobación

- [x] Abriste el notebook
- [x] Cargaste y preprocesaste los datos de texto para su vectorización
- [x] Transformaste los datos de texto en vectores
- [x] Entrenaste y probaste los modelos
- [x] Se alcanzó el umbral de la métrica
- [x] Colocaste todas las celdas de código en el orden de su ejecución
- [x] Puedes ejecutar sin errores todas las celdas de código
- [x] Hay conclusiones

In [ ]:

```
```