Analisis de comentarios de peliculas

Descripcion del problema

Para este proyecto se busca hacer un análisis de sentimientos sobre los comentarios en español de una película y se espera poder clasificarlos en 'positivo' o negativo'. El conjunto de datos suministrados es un csv con dos columnas, la primera es el comentario en sí, y la segunda es la clasificación que ya se le dio.

Entendimiento del negocio y enfoque analitico

- *Oportunidad / Problema de negocio*
  La oportunidad que se presenta para este caso es poder definir según un comentario, si se está haciendo una crítica positiva o negativa acerca de la película.

- *Enfoque analitico*
  El requerimiento desde el punto de vista del aprendizaje automático es clasificar cada comentario en una de dos categorías, positivo o negativo. Para ello se debe utilizar un enfoque de aprendizaje supervisado puesto que los datos ya vienen etiquetados con sus respectivas categorías. De esta forma, se puede entrenar con dichas etiquetas al modelo. Con el fin de lograr este objetivo se limpian los datos, se entrena el modelo y se prueba.

- *Organización y rol dentro de ella que se beneficia con la oportunidad definida*
  La organización que se beneficia con la oportunidad definida es la industria del entretenimiento, particularmente las empresas dedicadas a hacer producciones cinematográficas. El rol de los productores es quien más gana en el asunto, pues les será más fácil filtrar aquellos comentarios negativos y entender qué se podría mejorar o mirar los comentarios positivos para ver qué aspectos fueron los que más le gustaron al público y replicarlos para siguientes producciones. En general, el poder separar los comentarios positivos de los negativos permite un mejor entendimiento de que aspectos de le película fueron los más impactantes para replicarlos o no en próximas películas.

- *Tecnicas y algoritmos utilizados*
  Primeramente, se utilizará el algoritmo de procesamiento de lenguaje natural Naive Bayes, puesto que es el más común para este tipo de tareas. Asimismo, se emplearán otros dos algoritmos de clasificación, Random Forest y KNN.

1. Importacion de librerias

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

# Librerias para el preprocesamiento
import spacy
```

```
nlp = spacy.load('es_core_news_sm')

# Librerias para transformar los datos
from sklearn.preprocessing import FunctionTransformer
from sklearn.base import BaseEstimator, TransformerMixin

from sklearn.datasets import load_iris
from sklearn.decomposition import PCA

# Librerias para vectorizar
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

# Librerias para el modelo
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

# Librerias para la evaluacion
from sklearn.metrics import make_scorer, accuracy_score,
precision_score, recall_score, f1_score
from sklearn.model_selection import cross_val_score

# Librerias para la busqueda de hiperparametros
from sklearn.model_selection import GridSearchCV

# Librerias para el pipeline
from sklearn.pipeline import Pipeline

# Librerias para exportar el modelo
from joblib import dump, load

from sklearn.decomposition import TruncatedSVD
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.datasets import fetch_20newsgroups
```

2. Carga de datos

```
# Carga del csv entrenamiento
file_name = './data/MovieReviews.csv'
raw = pd.read_csv(file_name, sep=',')
reviews = raw.copy()

# Carga del csv prueba
file_name = './data/MovieReviewsPruebas.csv'
raw = pd.read_csv(file_name, sep=',')
reviews_test = raw.copy()
```

3. Entendimiento de los datos

Lo primero es ver la cantidad de datos que hay para entrenar el modelo, puesto que es importante comprender que entre mas datos se tengan, mejor sera la aproximacion que se obtenga. Para este caso se tiene una cantidad considerablemente buena de datos.

```
reviews.shape
```

```
(5000, 3)
```

Se debe revisar el tipo de datos que hay para cada columna.

```
# Tipo de variables
reviews.dtypes
```

```
Unnamed: 0      int64
review_es      object
sentimiento    object
dtype: object
```

Para rectificar que efectivamente la clasificacion sea de 'positivo' y 'negativo' se revisan los valores que hay en la columna categorica.

```
# Ver aproximadamente la cantidad de datos por la columna sentimiento
reviews['sentimiento'].value_counts()
```

```
negativo    2500
positivo    2500
Name: sentimiento, dtype: int64
```

Tambien se revisa la cantidad de nulos que pueda haber en el data set para ver cuantos datos se pierden al tener que eliminarlos. Para este caso no hay nulos.

```
# Verificar la cantidad de datos nulos
reviews.isnull().sum()
```

```
Unnamed: 0     0
review_es      0
sentimiento    0
dtype: int64
```

Finalmente se echa un vistazo a como se ven los datos.

```
reviews.head(3)
```

```
   Unnamed: 0                                          review_es
sentimiento
0           0  Si está buscando una película de guerra típica...
positivo
1           1  Supongo que algunos directores de películas de...
positivo
2           2  Es difícil contarle más sobre esta película si...
positivo
```

4. Creacion de los pipelines para cada algoritmo

Transformadores

Lo primero que se crea es un transformador para el preprocesamiento de los datos. Debido a que estamos tratando con frases en español se quiere estandarizar todos los comentarios aplicandoles ciertos cambios. De esta forma, se pasa todo a minusculas, se eliminan las tildes para evitar discrimanaciones incorrectas por errores de ortografia, se eliminan los numeros, signos de puntuacion y se eliminan las palabras vacias. Estas son palabras que no aportan significado al contexto, como lo son conectores, articulos y demas. Para ello se utilizo la libreria spacy.

```
# Funcion para limpiar el texto utilizando spacy
def clean_text(text):
    # Pasar a minusculas
    text = text.lower()

    # Eliminar las tildes
    text = re.sub(r'[á]', 'a', text)
    text = re.sub(r'[é]', 'e', text)
    text = re.sub(r'[í]', 'i', text)
    text = re.sub(r'[ó]', 'o', text)
    text = re.sub(r'[ú]', 'u', text)

    # Eliminar los numeros
    text = re.sub(r'\d+', ' ', text)

    # Eliminar los signos de puntuacion
    text = re.sub(r'[^\w\s]', ' ', text)

    # Tokenizar
    tokens = nlp(text)

    # Eliminar stopwords
    tokens = [token.text for token in tokens if not token.is_stop]

    # Unir los tokens
    text = ' '.join(tokens)

    return text
```

4.1 Naive Bayes (Juan Diego Calixto)

Pipeline

La creacion del pipeline se hace unicamente con dos procesos.

El primero es el transformador de los datos que estandariza los comentarios como se especifico anteriormente, utilizando un TFIDFVECTORIZER. Este transformador lo que

hace es tokenizar cada comentario separándolo por palabras y asignándole un puntaje a cada palabra según su número de apariciones en todos los comentarios, pero aquellas palabras que estén presentes en todos los comentarios y no sirvan para identificar si son positivos o negativos el TFIDF les baja al puntaje para que no afecten el entrenamiento del modelo. Asimismo, se delimitó el número de palabras significantes a 1800 puesto que luego de hacer pruebas fue el número máximo que dio mejor resultado.

El segundo proceso ya es la creación del modelo utilizando Naive Bayes. Se le paso por parametro un alpha de 1 para sumarle a todos los puntajes +1. De esta forma aquellas palabras que tenían un puntaje de 0 y que podian afectar la probabilidad, ya no sean un problema.

```
pipe = Pipeline([
                ('tfidf', TfidfVectorizer(preprocessor=clean_text,
max_features=1800)),
                ('model', MultinomialNB(alpha=1.0))
])
```

5.1 Ejecucion y análisis Naive Bayes

Entrenamiento del modelo

Se hace una separacion de los datos en entrenamiento y prueba. El X son las variables de decisión y el Y lo que se busca predecir.

```
# Separacion de las variables de decision y de prediccion
X = reviews['review_es']
Y = reviews['sentimiento']
```

La separación se hace 80% para etrenamiento y 20% para prueba.

```
# Creación de los conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=0)
y_train = y_train.values.ravel()
```

Se le pasa al pipeline los datos de entrenamiento

```
# Entrenamiento del modelo
model = pipe.fit(X_train, y_train)
```

Analisis de metricas

```
# Realizar predicciones en el conjunto de entrenaiento y prueba
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# Calcular las métricas de evaluación
NB_train_accuracy = accuracy_score(y_train, y_train_pred)
NB_train_precision = precision_score(y_train, y_train_pred,
pos_label='positivo')
NB_train_recall = recall_score(y_train, y_train_pred,
```

```
pos_label='positivo')
NB_train_f1 = f1_score(y_train, y_train_pred, pos_label='positivo')

NB_test_accuracy = accuracy_score(y_test, y_test_pred)
NB_test_precision = precision_score(y_test, y_test_pred,
pos_label='positivo')
NB_test_recall = recall_score(y_test, y_test_pred,
pos_label='positivo')
NB_test_f1 = f1_score(y_test, y_test_pred, pos_label='positivo')
```

Según las métricas podemos ver que el modelo se ajustó bastante bien a los datos pero no hizo un sobreajuste. Si bien son un poco más altas las métricas entrenamiento no son exageradas con respecto a las del conjunto de prueba. Entonces, se puede concluir que el modelo se ajustó considerablemente bien a los datos y puede predecir con un porcentaje de acierto de al rededor del 80% la clasificaciín de un comentario para esa película.

```
# Imprimir las métricas de evaluación
print('\nMetricas del conjunto de entrenamiento:')
print("Accuracy:",  NB_train_accuracy)
print("Precision:", NB_train_precision)
print("Recall:",    NB_train_recall)
print("F1 score:",  NB_train_f1)

# Imprimir las métricas de evaluación
print('\nMetricas del conjunto de prueba:')
print("Accuracy:",  NB_test_accuracy)
print("Precision:", NB_test_precision)
print("Recall:",    NB_test_recall)
print("F1 score:",  NB_test_f1)
```

```
Metricas del conjunto de entrenamiento:
Accuracy: 0.869
Precision: 0.8592023065833734
Recall: 0.8855869242199108
F1 score: 0.8721951219512195

Metricas del conjunto de prueba:
Accuracy: 0.81
Precision: 0.7963340122199593
Recall: 0.8128898128898129
F1 score: 0.8045267489711934
```

Búsqueda de hiperparámetros

Quizás con un poder computacional mucho mayor se pueda hacer la busqueda de hiperparámetros

```
'''
# Definir los valores para los hiperparametros
```

```
param_grid = {
    'tfidf__max_features': [100, 500, 1000],
    'model__alpha': [1.0],
}

# Crear un objeto GridSearchCV
grid_search = GridSearchCV(pipe, param_grid, cv=5)

# Ajustar el modelo con los hiperparametros
grid_search.fit(X_train, y_train)

# Obtener los mejores hiperparametros
best_params = grid_search.best_params_
print('Valores óptimos de los hiperparámetros:', best_params)
'''
```

```
"\n# Definir los valores para los hiperparametros\nparam_grid = {\n
'tfidf__max_features': [100, 500, 1000],\n    'model__alpha': [1.0],\
n}\n\n# Crear un objeto GridSearchCV\ngrid_search = GridSearchCV(pipe,
param_grid, cv=5)\n\n# Ajustar el modelo con los hiperparametros\
ngrid_search.fit(X_train, y_train)\n\n# Obtener los mejores
hiperparametros\nbest_params = grid_search.best_params_\
nprint('Valores óptimos de los hiperparámetros:', best_params)\n"
```

6.1 Exportar el modelo Naive Bayes

Finalmente se exporta el modelo para que pueda ser utilizado en otro conjunto de datos

```
# Nombre del archivo donde se guardará el modelo
pipe_file_name = 'pipelineNaiveBayes.joblib'

# Guardar el modelo
dump(model, pipe_file_name)
```

```
['pipelineNaiveBayes.joblib']
```

```
# Cargar el modelo
test_model = load(pipe_file_name)
test_model
```

```
Pipeline(steps=[('tfidf',
                 TfidfVectorizer(max_features=1800,
                                 preprocessor=<function clean_text at
0x7ff3586013a0>)),
                ('model', MultinomialNB())])
```

Se rectifica que efectivamente el modelo este prediciendo correctamente.

```
# Clasificar los datos de prueba
reviews_test['sentimiento_NaiveBayes'] =
test_model.predict(reviews_test['review_es'])
```

```
# Visualizar los resultados
reviews_test
```

```
     Unnamed: 0                                          review_es  \
0             0  La saga medieval alemana de Fritz Lang continú...
1             1  Este anime es una visita obligada para los fan...
2             2  Esta es una de las mejores películas para masc...
3             3  Cuando se lanzó su DVD, llegué al mercado y lo...
4             4  No me he reído tan duro en una película en muc...
..          ...                                                ...
295         295  1er vimos 2/18/2007 - 4 de 10 (Dir-Leon Leonar...
296         296  Desde su título no inspirado a las actuaciones...
297         297  Encontré a esta buena película para pasar su t...
298         298  Solía trabajar en la compañía que originalme...
299         299  La dama en cemento es un verdadero curso sobre...

    sentimiento_NaiveBayes
0                 positivo
1                 positivo
2                 positivo
3                 positivo
4                 positivo
..                     ...
295               negativo
296               negativo
297               positivo
298               negativo
299               positivo

[300 rows x 3 columns]
```

4.2 Random Forest (Sergio Pardo)

Pipeline

La creacion del pipeline se hace unicamente con dos procesos.

El primero es el transformador que estandariza los comentarios con el mismo proceso que el modelo de Naive Bayes con la funcion 'clean_text'. Sin embargo, este utiliza el metodo de COUNTVECTORIZER que únicamente crea una tabla con las palabras tokenizadas y sus repeticiones a lo largo de todo el conjunto de datos, aquí no se asignan puntajes a las palabras, únicamente se pone la cantidad de veces que aparece.

El segundo proceso ya es la creacion del modelo utilizando Random Forest. Se emplea este algoritmo pues es una tarea de clasificación y por experiencia consideramos que resultaría adecuada para este proyecto.

```
pipe = Pipeline([
                ('tfidf', CountVectorizer(preprocessor=clean_text)),
```

```
                ('model', RandomForestClassifier())
])
```

5.2 Ejecución y análisis Random Forest

Entrenamiento del modelo

Se hace una separación de los datos en entrenamiento y prueba. El X son las variables de decision y el Y lo que se busca predecir.

```
# Separacion de las variables de decision y de prediccion
X = reviews['review_es']
Y = reviews['sentimiento']
```

La separación se hace 80% para etrenamiento y 20% para prueba.

```
# Creacion de los conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=0)
y_train = y_train.values.ravel()
```

Se le pasa al pipeline los datos de entrenamiento

```
# Entrenamiento del modelo
model = pipe.fit(X_train, y_train)
```

Análisis de metricas

```
# Realizar predicciones en el conjunto de entrenaiento y prueba
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# Calcular las metricas de evaluacion
RF_train_accuracy = accuracy_score(y_train, y_train_pred)
RF_train_precision = precision_score(y_train, y_train_pred,
pos_label='positivo')
RF_train_recall = recall_score(y_train, y_train_pred,
pos_label='positivo')
RF_train_f1 = f1_score(y_train, y_train_pred, pos_label='positivo')

RF_test_accuracy = accuracy_score(y_test, y_test_pred)
RF_test_precision = precision_score(y_test, y_test_pred,
pos_label='positivo')
RF_test_recall = recall_score(y_test, y_test_pred,
pos_label='positivo')
RF_test_f1 = f1_score(y_test, y_test_pred, pos_label='positivo')
```

Según las étricas podemos ver que el modelo se ajustó bastante bien a los datos de entrenamiento al punto de generar un sobreajuste. No obstante, los resultados de evaluación son buenos (se acercan al 80%) Entonces se puede concluir que el modelo se ajustó considerablemente bien a los datos y puede predecir con un porcentaje de acierto de al rededor del 80% la clasificaion de un comentario para esa pelicula.

```
# Imprimir las metricas de evaluacion
print('\nMetricas del conjunto de entrenamiento:')
print("Accuracy:",  RF_train_accuracy)
print("Precision:", RF_train_precision)
print("Recall:",    RF_train_recall)
print("F1 score:",  RF_train_f1)

# Imprimir las metricas de evaluacion
print('\nMetricas del conjunto de prueba:')
print("Accuracy:",  RF_test_accuracy)
print("Precision:", RF_test_precision)
print("Recall:",    RF_test_recall)
print("F1 score:",  RF_test_f1)
```

```
Metricas del conjunto de entrenamiento:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 score: 1.0

Metricas del conjunto de prueba:
Accuracy: 0.775
Precision: 0.7549800796812749
Recall: 0.7879417879417879
F1 score: 0.7711088504577822
```

6.2 Exportar el modelo Random Forest

Finalmente se exporta el modelo para que pueda ser utilizado en otro conjunto de datos

```
# Nombre del archivo donde se guardara el modelo
pipe_file_name = 'pipelineRandomForest.joblib'

# Guardar el modelo
dump(model, pipe_file_name)
```

```
['pipelineRandomForest.joblib']
```

```
# Cargar el modelo
test_model = load(pipe_file_name)
test_model
```

```
Pipeline(steps=[('tfidf',
                 CountVectorizer(preprocessor=<function clean_text at
0x7ff3586013a0>)),
                ('model', RandomForestClassifier())])
```

```
# Clasificar los datos de prueba
reviews_test['sentimiento_RandomForest'] =
test_model.predict(reviews_test['review_es'])
```

```
# Visualizar los resultados
reviews_test
```

```
     Unnamed: 0                                          review_es  \
0             0  La saga medieval alemana de Fritz Lang continú...
1             1  Este anime es una visita obligada para los fan...
2             2  Esta es una de las mejores películas para masc...
3             3  Cuando se lanzó su DVD, llegué al mercado y lo...
4             4  No me he reído tan duro en una película en muc...
..          ...                                                ...
295         295  1er vimos 2/18/2007 - 4 de 10 (Dir-Leon Leonar...
296         296  Desde su título no inspirado a las actuaciones...
297         297  Encontré a esta buena película para pasar su t...
298         298  Solía trabajar en la compañía que originalme...
299         299  La dama en cemento es un verdadero curso sobre...

    sentimiento_NaiveBayes sentimiento_RandomForest
0                 positivo                positivo
1                 positivo                positivo
2                 positivo                positivo
3                 positivo                positivo
4                 positivo                positivo
..                     ...                     ...
295               negativo                negativo
296               negativo                negativo
297               positivo                negativo
298               negativo                negativo
299               positivo                positivo

[300 rows x 4 columns]
```

4.2 KNN (Nathalia Quiroga)

Pipeline

La creación del pipeline se hace únicamente con dos procesos.

El primero, es el transformador que estandariza los comentarios con el mismo proceso que los dos modelos anteriores con la funcion 'clean_text'. Sin embargo, este al igual que Bayes utiliza el metodo de TFIDVECTORIZER ya que es una técnica de vectorización en el procesamiento de lenguaje natural que tiene en cuenta tanto la frecuencia de ocurrencia de una palabra en un documento como su frecuencia de ocurrencia en todo el conjunto de documentos. Esto permite que las palabras que aparecen con frecuencia en un documento específico pero raramente en el resto de los documentos tengan una puntuación más alta, lo que las hace más significativas para la representación de ese documento. TF-IDF Vectorizer también tiene un mejor rendimiento que CountVectorizer en la tarea de clasificación de documentos, ya que las palabras únicas en un documento pueden ser más importantes para la clasificación.

El segundo proceso, como se dijo anteriormente, ya es la creación del modelo y el pipeline usando KNN, además de esto, más adelante se explica cómo se reduce la diemensionalidad del KNN para unas mejores métricas. Se usó este algoritmo ya que puede ser utilizado para la clasificación de texto, donde cada documento (o conjuno de palabras que forman una unidad de texto) se representa como un vector de características y se compara con los documentos de entrenamiento existentes. El documento se clasifica según la clase predominante de los vecinos más cercanos.

TruncatedSVD es una técnica que reduce la dimensionalidad de los datos de alta dimensión al comprimirlos en un espacio de menor dimensión, al mismo tiempo que conserva la información relevante. Esta técnica se utiliza especialmente para manejar entradas dispersas o "sparse input". En el procesamiento de lenguaje natural, TruncatedSVD se utiliza comúnmente para reducir la dimensionalidad de los vectores de características que representan palabras o documentos, como aquellos obtenidos con CountVectorizer o TfidfVectorizer. Al reducir la dimensionalidad de los datos, TruncatedSVD puede mejorar la eficiencia computacional y disminuir el tiempo de entrenamiento de los modelos de aprendizaje automático, y además puede ayudar a prevenir el sobreajuste y reducir el ruido en los datos.

Asimismo, se delimitó el numero de palabras significantes a 1800 puesto que luego de hacer pruebas fue el numero máximo que dio mejor resultado.

Por último, "n_components" es un parámetro que especifica el número de dimensiones que se deben mantener después de aplicar la técnica de reducción de dimensionalidad. Es decir, "n_components" determina el tamaño de la matriz resultante después de aplicar TruncatedSVD.

Este parámetro se utiliza para controlar el equilibrio entre la cantidad de información que se mantiene en los datos y la cantidad de reducción de dimensionalidad deseada.

```
pipe = Pipeline([
                ('tfidf', TfidfVectorizer(preprocessor=clean_text,
max_features=1800)),
                ('svd', TruncatedSVD(n_components=28)),
                ('model', KNeighborsClassifier())
])
```

Para escoger el "n_components" se realizó un tanteo de valores entre 5-50 para evaluar el cambio de las métricas en cada caso y así lograr el más acertado.

5.2 Ejecución y análisis KNN

Entrenamiento del modelo

Se hace una separación de los datos en entrenamiento y prueba. El X son las variables de decisión y el Y lo que se busca predecir.

```
# Separacion de las variables de decision y de prediccion
X = reviews['review_es']
Y = reviews['sentimiento']
```

La separación se hace 80% para etrenamiento y 20% para prueba.

```
# Creacion de los conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=0)
y_train = y_train.values.ravel()
```

Se le pasa al pipeline los datos de entrenamiento

```
# Entrenamiento del modelo
model = pipe.fit(X_train, y_train)
```

Análisis de métricas

```
# Realizar predicciones en el conjunto de entrenaiento y prueba
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# Calcular las metricas de evaluacion
KNN_train_accuracy = accuracy_score(y_train, y_train_pred)
KNN_train_precision = precision_score(y_train, y_train_pred,
pos_label='positivo')
KNN_train_recall = recall_score(y_train, y_train_pred,
pos_label='positivo')
KNN_train_f1 = f1_score(y_train, y_train_pred, pos_label='positivo')

KNN_test_accuracy = accuracy_score(y_test, y_test_pred)
KNN_test_precision = precision_score(y_test, y_test_pred,
pos_label='positivo')
KNN_test_recall = recall_score(y_test, y_test_pred,
pos_label='positivo')
KNN_test_f1 = f1_score(y_test, y_test_pred, pos_label='positivo')
```

Según las métricas podemos ver que el modelo se ajustó bien a los datos de entrenamiento y el sobreajuste no es exagerado. Sin embargo, los resultados de evaluación son considerablemente inferiores respecto a los otros modelos implementados (no superan el 70%). Por lo que, puede afirmarse que este no es un buen modelo para resolver el problema actual

```
# Imprimir las metricas de evaluacion
print('\nMetricas del conjunto de entrenamiento:')
print("Accuracy:",  KNN_train_accuracy)
print("Precision:", KNN_train_precision)
print("Recall:",    KNN_train_recall)
print("F1 score:",  KNN_train_f1)

# Imprimir las metricas de evaluacion
print('\nMetricas del conjunto de prueba:')
print("Accuracy:",  KNN_test_accuracy)
print("Precision:", KNN_test_precision)
print("Recall:",    KNN_test_recall)
print("F1 score:",  KNN_test_f1)
```

```
Metricas del conjunto de entrenamiento:
Accuracy: 0.80525
Precision: 0.7938388625592417
Recall: 0.8296186230807331
F1 score: 0.8113344635504965

Metricas del conjunto de prueba:
Accuracy: 0.691
Precision: 0.6641221374045801
Recall: 0.7234927234927235
F1 score: 0.6925373134328358
```

6.2 Exportar el modelo KNN

Finalmente se exporta el modelo para que pueda ser utilizado en otro conjunto de datos

```
# Nombre del archivo donde se guardara el modelo
pipe_file_name = 'pipelineKNN.joblib'

# Guardar el modelo
dump(model, pipe_file_name)
```

```
['pipelineKNN.joblib']
```

```
# Cargar el modelo
test_model = load(pipe_file_name)
test_model
```

```
Pipeline(steps=[('tfidf',
                 TfidfVectorizer(max_features=1800,
                                 preprocessor=<function clean_text at
0x7ff3586013a0>)),
                ('svd', TruncatedSVD(n_components=28)),
                ('model', KNeighborsClassifier())])
```

```
# Clasificar los datos de prueba
reviews_test['sentimiento_KNN'] =
test_model.predict(reviews_test['review_es'])

# Visualizar los resultados
reviews_test
```

```
     Unnamed: 0                                          review_es  \
0             0  La saga medieval alemana de Fritz Lang continú...
1             1  Este anime es una visita obligada para los fan...
2             2  Esta es una de las mejores películas para masc...
3             3  Cuando se lanzó su DVD, llegué al mercado y lo...
4             4  No me he reído tan duro en una película en muc...
..          ...                                                ...
295         295  1er vimos 2/18/2007 - 4 de 10 (Dir-Leon Leonar...
296         296  Desde su título no inspirado a las actuaciones...
```

```
297         297  Encontré a esta buena película para pasar su t...
298         298  Solía trabajar en la compañía que originalme...
299         299  La dama en cemento es un verdadero curso sobre...

    sentimiento_NaiveBayes sentimiento_RandomForest sentimiento_KNN
0                 positivo                positivo        positivo
1                 positivo                positivo        positivo
2                 positivo                positivo        positivo
3                 positivo                positivo        positivo
4                 positivo                positivo        negativo
..                     ...                     ...             ...
295               negativo                negativo        negativo
296               negativo                negativo        positivo
297               positivo                negativo        positivo
298               negativo                negativo        negativo
299               positivo                positivo        positivo

[300 rows x 5 columns]
```

7. Comparación de los modelos

```
# Creacion del dataframe con las metricas de evaluacion para cada
algoritmo utilizado
metricas = {
    'Algoritmo': ['Naive Bayes', 'Random Forest', 'KNN'],
    'Accuracy': [NB_test_accuracy, RF_test_accuracy,
KNN_test_accuracy],
    'Precision': [NB_test_precision, RF_test_precision,
KNN_test_precision],
    'Recall': [NB_test_recall, RF_test_recall, KNN_test_recall],
    'F1 score': [NB_test_f1, RF_test_f1, KNN_test_f1]
}

pruebas = {
    'Algoritmo': ['Naive Bayes', 'Random Forest', 'KNN'],
    'Accuracy': [NB_train_accuracy, RF_train_accuracy,
KNN_train_accuracy],
    'Precision': [NB_train_precision, RF_train_precision,
KNN_train_precision],
    'Recall': [NB_train_recall, RF_train_recall, KNN_train_recall],
    'F1 score': [NB_train_f1, RF_train_f1, KNN_train_f1]
}

# Pasar el diccionario a un dataframe
metricas_df = pd.DataFrame(metricas)
pruebas_df = pd.DataFrame(pruebas)

pruebas_df
```

```
       Algoritmo  Accuracy  Precision    Recall  F1 score
0    Naive Bayes   0.86900   0.859202  0.885587  0.872195
```

```
1  Random Forest   1.00000   1.000000  1.000000  1.000000
2            KNN   0.80525   0.793839  0.829619  0.811334
```

```
# Visualizar el dataframe
metricas_df
```

```
       Algoritmo  Accuracy  Precision    Recall  F1 score
0    Naive Bayes     0.810   0.796334  0.812890  0.804527
1  Random Forest     0.775   0.754980  0.787942  0.771109
2            KNN     0.691   0.664122  0.723493  0.692537
```

Conclusiones

Con el fin de comprender cual fue el algoritmo que dio mejores resultados, primero hay que entender que evaluan cada una de las metricas que se calcularon.

- *Accuracy:*
  La exactitud es una medida que indica la proporción de predicciones correctas del modelo en relación con el total de predicciones. Se calcula dividiendo el número de predicciones correctas por el número total de predicciones. Una alta exactitud generalmente indica que el modelo está prediciendo correctamente la mayoría de los ejemplos.

- *Precicion:*
  La precisión es la proporción de predicciones positivas verdaderas con respecto a todas las predicciones positivas (verdaderas y falsas). Se calcula dividiendo el número de predicciones positivas verdaderas por la suma de las predicciones positivas verdaderas y falsas. Una alta precisión indica que el modelo tiene pocos falsos positivos, es decir, pocas instancias negativas se predicen incorrectamente como positivas.

- *Recall:*
  La sensibilidad es la proporción de predicciones positivas verdaderas con respecto a todas las instancias verdaderamente positivas. Se calcula dividiendo el número de predicciones positivas verdaderas por la suma de las predicciones positivas verdaderas y falsas negativas. Una alta sensibilidad indica que el modelo tiene pocos falsos negativos, es decir, pocas instancias positivas se predicen incorrectamente como negativas.

- *F1*
  El valor F1 es una medida que combina la precisión y la sensibilidad en una sola métrica. Se calcula como la media armónica de la precisión y la sensibilidad, y proporciona una medida equilibrada del rendimiento del modelo en términos de tanto los falsos positivos como los falsos negativos. Un valor F1 alto indica un buen equilibrio entre la precisión y la sensibilidad.

Según los resultados, el modelo que tuvo mejores métricas fue el de Naive Bayes. Empezando por la exactitud, se aprecia que empleando este algoritmo se logra predecir correctamente el 80% de los comentarios, en otras palabras, se tiene la certeza de que para

cada 5 comentarios, 4 de ellos serán correctamente clasificados como positivos o negativos. De esta manera, se les asegura a los productores de las películas con un alto grado de confiabilidad que sabrán si la crítica es buena o mala para su respectivo análisis.

Por otro lado, al observar la precisión, se nota que el 80% de las predicciones positivas realizadas por el modelo son verdaderas, en relación con el total de predicciones positivas (verdaderas y falsas). En otras palabras, de cada 100 predicciones positivas hechas por el modelo, aproximadamente 80 son correctas.

En general, al ver que todas las métricas arrojaron valores aproximados del 80%, se puede concluir que el modelo de Naive Bayes resultó bastante adecuado y sus predicciones son considerablemente buenas.