Dado el siguiente procedimiento para clasificar sentimientos usando el conjunto de datos imdb.

1. Ejecute el procedimiento y compare el resultado de las variables accuracy_lstm y accuracy_cnn_lstm.
2. Replique el procedimiento para el conjunto de datos enviado en anexo para crear un clasificador de sentimientos en espaniol (Big_AHR.csv.zip).
3. Compare y muestre los resultados obtenidos usando solo LSTM y CNN + LSTM de sus clasificador en espaniol.

(*) En caso de problema de ejecución por falta de recursos. puede crear un subconjunto del archivo Big_AHR.csv.zip

(*) Use los siguientes links como referencia.

1. https://github.com/anandsarank/cnn-lstm-text-classification/blob/main/CNN%20with%20LSTM%20for%20Text%20Classification.ipynb
2. https://colab.research.google.com/github/alvinntnu/python-notes/blob/master/nlp/sentiment-analysis-lstm-v1.ipynb
3. https://www.kaggle.com/code/chizhikchi/lstm-binary-sentiment-classification-for-spanish/notebook
4. https://www.kaggle.com/code/chizhikchi/ahr-corpus-presentation

```
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Conv1D,MaxPooling1D
from tensorflow.keras.layers import LSTM,Dropout
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.callbacks import ModelCheckpoint
np.random.seed(7)
from prettytable import PrettyTable
import warnings
warnings.filterwarnings('ignore')
```

```
# load the dataset but only keep the top n words, zero the rest
top_words = 10000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top_words)
X_train,X_cv,y_train,y_cv = train_test_split(X_train,y_train,test_size = 0.2)
print("Shape of train data:", X_train.shape)
print("Shape of Test data:", X_test.shape)
print("Shape of CV data:", X_cv.shape)

# truncate and pad input sequences
max_review_length = 600
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)
X_cv = sequence.pad_sequences(X_cv,maxlen=max_review_length)
```

```
Shape of train data: (20000,)
Shape of Test data: (25000,)
Shape of CV data: (5000,)
```

LSTM

```
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
filepath="weights_best.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max',save_weights_only=True)
callbacks_list = [checkpoint]
model.fit(X_train, y_train, epochs=5, batch_size=256,verbose = 1,callbacks = callbacks_list,validation_data=(X_cv,y_cv))
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 600, 32)           320000

 lstm_1 (LSTM)               (None, 100)               53200

 dense_1 (Dense)             (None, 1)                 101

=================================================================
Total params: 373,301
```

```
Trainable params: 373,301
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
79/79 [==============================] - ETA: 0s - loss: 0.6667 - accuracy: 0.6374
Epoch 1: val_accuracy improved from -inf to 0.68740, saving model to weights_best.hdf5
79/79 [==============================] - 443s 6s/step - loss: 0.6667 - accuracy: 0.6374 - val_loss: 0.6536 - val_accuracy: 0.6874
Epoch 2/5
79/79 [==============================] - ETA: 0s - loss: 0.6041 - accuracy: 0.7469
Epoch 2: val_accuracy improved from 0.68740 to 0.81200, saving model to weights_best.hdf5
79/79 [==============================] - 428s 5s/step - loss: 0.6041 - accuracy: 0.7469 - val_loss: 0.4494 - val_accuracy: 0.8120
Epoch 3/5
79/79 [==============================] - ETA: 0s - loss: 0.3440 - accuracy: 0.8550
Epoch 3: val_accuracy improved from 0.81200 to 0.85620, saving model to weights_best.hdf5
79/79 [==============================] - 383s 5s/step - loss: 0.3440 - accuracy: 0.8550 - val_loss: 0.3381 - val_accuracy: 0.8562
Epoch 4/5
79/79 [==============================] - ETA: 0s - loss: 0.2549 - accuracy: 0.9017
Epoch 4: val_accuracy improved from 0.85620 to 0.86100, saving model to weights_best.hdf5
79/79 [==============================] - 382s 5s/step - loss: 0.2549 - accuracy: 0.9017 - val_loss: 0.3264 - val_accuracy: 0.8610
Epoch 5/5
79/79 [==============================] - ETA: 0s - loss: 0.2010 - accuracy: 0.9251
Epoch 5: val_accuracy improved from 0.86100 to 0.87560, saving model to weights_best.hdf5
79/79 [==============================] - 382s 5s/step - loss: 0.2010 - accuracy: 0.9251 - val_loss: 0.3145 - val_accuracy: 0.8756
<keras.callbacks.History at 0x7ece1cf270a0>
```

```
# Final evaluation of the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
model.load_weights("weights_best.hdf5")
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
scores = model.evaluate(X_test, y_test, verbose=1,batch_size = 256)
accuracy_lstm = scores[1]*100
print("Accuracy using LSTM: %.2f%%" % (accuracy_lstm))
```

```
98/98 [==============================] - 98s 990ms/step - loss: 0.3209 - accuracy: 0.8713
Accuracy using LSTM: 87.13%
```

CNN + LSTM

```
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
filepath="weights_best_cnn.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max',save_weights_only=True)
callbacks_list = [checkpoint]
model.fit(X_train, y_train, epochs=5, batch_size=256,verbose = 1,callbacks = callbacks_list,validation_data=(X_cv,y_cv))
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 600, 32)           320000

 conv1d (Conv1D)             (None, 600, 32)           3104

 max_pooling1d (MaxPooling1D  (None, 300, 32)          0
 )

 lstm_3 (LSTM)               (None, 100)               53200

 dense_3 (Dense)             (None, 1)                 101

=================================================================
Total params: 376,405
Trainable params: 376,405
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
79/79 [==============================] - ETA: 0s - loss: 0.5901 - accuracy: 0.6581
Epoch 1: val_accuracy improved from -inf to 0.81040, saving model to weights_best_cnn.hdf5
79/79 [==============================] - 115s 1s/step - loss: 0.5901 - accuracy: 0.6581 - val_loss: 0.4061 - val_accuracy: 0.8104
```

```
Epoch 2/5
79/79 [==============================] - ETA: 0s - loss: 0.2627 - accuracy: 0.8960
Epoch 2: val_accuracy improved from 0.81040 to 0.88440, saving model to weights_best_cnn.hdf5
79/79 [==============================] - 113s 1s/step - loss: 0.2627 - accuracy: 0.8960 - val_loss: 0.2789 - val_accuracy: 0.8844
Epoch 3/5
79/79 [==============================] - ETA: 0s - loss: 0.1779 - accuracy: 0.9370
Epoch 3: val_accuracy improved from 0.88440 to 0.88460, saving model to weights_best_cnn.hdf5
79/79 [==============================] - 113s 1s/step - loss: 0.1779 - accuracy: 0.9370 - val_loss: 0.2868 - val_accuracy: 0.8846
Epoch 4/5
79/79 [==============================] - ETA: 0s - loss: 0.1304 - accuracy: 0.9546
Epoch 4: val_accuracy did not improve from 0.88460
79/79 [==============================] - 113s 1s/step - loss: 0.1304 - accuracy: 0.9546 - val_loss: 0.3079 - val_accuracy: 0.8710
Epoch 5/5
79/79 [==============================] - ETA: 0s - loss: 0.1024 - accuracy: 0.9670
Epoch 5: val_accuracy did not improve from 0.88460
79/79 [==============================] - 111s 1s/step - loss: 0.1024 - accuracy: 0.9670 - val_loss: 0.3403 - val_accuracy: 0.8630
<keras.callbacks.History at 0x7ece1ea26e30>
```

```
# Final evaluation of the model
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
model.load_weights("weights_best_cnn.hdf5")
scores = model.evaluate(X_test, y_test, verbose=0)
accuracy_cnn_lstm = scores[1]*100
print("Accuracy CNN using LSTM: %.2f%%" % (accuracy_cnn_lstm))
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 600, 32)           320000

 conv1d_1 (Conv1D)           (None, 600, 32)           3104

 max_pooling1d_1 (MaxPooling  (None, 300, 32)          0
 1D)

 lstm_4 (LSTM)               (None, 100)               53200

 dense_4 (Dense)             (None, 1)                 101

=================================================================
Total params: 376,405
Trainable params: 376,405
Non-trainable params: 0
_________________________________________________________________
None
Accuracy CNN using LSTM: 87.69%
```

```
table = PrettyTable()
table.field_names = ['Model', 'Accuracy']
table.add_row(['LSTM', accuracy_lstm])
table.add_row(['CNN using LSTM', accuracy_cnn_lstm])
print(table)
```

```
+----------------+-------------------+
|     Model      |      Accuracy     |
+----------------+-------------------+
|      LSTM      | 87.68799901008606 |
| CNN using LSTM | 87.12800145149231 |
+----------------+-------------------+
```

En el conjunto de datos de IMDb, tanto el modelo LSTM como el modelo CNN + LSTM presentan un desempeño muy similar en cuanto a precisión. El modelo LSTM logra un nivel de precisión de aproximadamente el 87.68%, mientras que el modelo CNN + LSTM alcanza una precisión ligeramente inferior, alrededor del 87.12%. Hay una pequeña diferencia, ambos modelos posee la capacidad sólida para capturar características relevantes en las críticas de películas.

CLASIFICADOR EN ESPAÑOL

```
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.text import Tokenizer
from sklearn.model_selection import train_test_split
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Conv1D,MaxPooling1D
from tensorflow.keras.layers import LSTM,Dropout
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.callbacks import ModelCheckpoint
np.random.seed(7)
from prettytable import PrettyTable
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.preprocessing.text import Tokenizer
```

```
dataset = pd.read_csv('/content/Big_AHR.csv')
dataset
```

	Unnamed: 0	title	rating	review_text	location
0	0	Excelente y personal amable	5	Un hotel muy bueno. El personal fue muy amabl...	Seville_Province_of_Seville_Andalucia
1	1	Céntrico	4	Muy buen hotel al nivel de lo esperado, habita...	Seville_Province_of_Seville_Andalucia
2	2	Hotel excepcional	5	Magnífico hotel. La verdad es que todo perfect...	Seville_Province_of_Seville_Andalucia
				Hotel hermoso,	

```
X = dataset['review_text']
y = dataset['label']

top_words = 10000

# Tokenize the text data
tokenizer = Tokenizer(num_words=top_words)
tokenizer.fit_on_texts(X)
sequences = tokenizer.texts_to_sequences(X)


# Split the data into training, testing, and validation datasets
X_train, X_test, y_train, y_test = train_test_split(sequences,y, test_size=0.2)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,
test_size=0.2)
# Truncate and pad input sequences
max_review_length = 600
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)
X_cv = sequence.pad_sequences(X_cv, maxlen=max_review_length)
print("Shape of train data:", X_train.shape)
print("Shape of Test data:", X_test.shape)
print("Shape of CV data:", X_cv.shape)
```

```
Shape of train data: (11629, 600)
Shape of Test data: (3635, 600)
Shape of CV data: (2908, 600)
```

LSTM

```
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
filepath="weights_best.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy',verbose=1, save_best_only=True, mode='max',save_weights_only=True)
callbacks_list = [checkpoint]
model.fit(X_train, y_train, epochs=5, batch_size=256,verbose = 1,callbacks = callbacks_list,validation_data=(X_cv,y_cv))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 600, 32)           320000    
                                                                 
 lstm (LSTM)                 (None, 100)               53200     
                                                                 
 dense (Dense)               (None, 1)                 101       
                                                                 
=================================================================
Total params: 373,301
Trainable params: 373,301
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
46/46 [==============================] - ETA: 0s - loss: -0.6711 - accuracy: 0.7145
Epoch 1: val_accuracy improved from -inf to 0.74725, saving model to weights_best.hdf5
46/46 [==============================] - 250s 5s/step - loss: -0.6711 - accuracy: 0.7145 - val_loss: -1.2691 - val_accuracy: 0.7472
Epoch 2/5
46/46 [==============================] - ETA: 0s - loss: -1.7086 - accuracy: 0.7235
Epoch 2: val_accuracy did not improve from 0.74725
46/46 [==============================] - 239s 5s/step - loss: -1.7086 - accuracy: 0.7235 - val_loss: -1.7368 - val_accuracy: 0.7472
Epoch 3/5
46/46 [==============================] - ETA: 0s - loss: -2.1713 - accuracy: 0.7235
Epoch 3: val_accuracy did not improve from 0.74725
46/46 [==============================] - 233s 5s/step - loss: -2.1713 - accuracy: 0.7235 - val_loss: -2.1222 - val_accuracy: 0.7472
Epoch 4/5
46/46 [==============================] - ETA: 0s - loss: -2.5970 - accuracy: 0.7235
Epoch 4: val_accuracy did not improve from 0.74725
46/46 [==============================] - 237s 5s/step - loss: -2.5970 - accuracy: 0.7235 - val_loss: -2.5025 - val_accuracy: 0.7472
Epoch 5/5
46/46 [==============================] - ETA: 0s - loss: -3.0152 - accuracy: 0.7235
Epoch 5: val_accuracy did not improve from 0.74725
46/46 [==============================] - 241s 5s/step - loss: -3.0152 - accuracy: 0.7235 - val_loss: -2.8628 - val_accuracy: 0.7472
<keras.callbacks.History at 0x78db7c29a800>
```

```
# Final evaluation of the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length,
input_length=max_review_length))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
model.load_weights("weights_best.hdf5")
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
scores = model.evaluate(X_test, y_test, verbose=1,batch_size = 256)
accuracy_lstm = scores[1]*100
print("Accuracy using LSTM: %.2f%%" % (accuracy_lstm))
```

```
15/15 [==============================] - 15s 936ms/step - loss: -1.9780 - accuracy: 0.7265
Accuracy using LSTM: 72.65%
```

LSTM + CNN

```
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length,input_length=max_review_length))
model.add(Conv1D(filters=32, kernel_size=3, padding='same',activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
print(model.summary())
filepath="weights_best_cnn.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max',save_weights_only=True)
callbacks_list = [checkpoint]
model.fit(X_train, y_train, epochs=5, batch_size=256,verbose =1,callbacks = callbacks_list,validation_data=(X_cv,y_cv))
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 600, 32)           320000

 conv1d (Conv1D)             (None, 600, 32)           3104

 max_pooling1d (MaxPooling1D  (None, 300, 32)          0
 )

 lstm_2 (LSTM)               (None, 100)               53200

 dense_2 (Dense)             (None, 1)                 101

=================================================================
Total params: 376,405
Trainable params: 376,405
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
46/46 [==============================] - ETA: 0s - loss: -0.6516 - accuracy: 0.7140
Epoch 1: val_accuracy improved from -inf to 0.74725, saving model to weights_best_cnn.hdf5
46/46 [==============================] - 70s 1s/step - loss: -0.6516 - accuracy: 0.7140 - val_loss: -1.2510 - val_accuracy: 0.7472
Epoch 2/5
46/46 [==============================] - ETA: 0s - loss: -1.6929 - accuracy: 0.7235
Epoch 2: val_accuracy did not improve from 0.74725
46/46 [==============================] - 70s 2s/step - loss: -1.6929 - accuracy: 0.7235 - val_loss: -1.7348 - val_accuracy: 0.7472
Epoch 3/5
46/46 [==============================] - ETA: 0s - loss: -2.1754 - accuracy: 0.7235
Epoch 3: val_accuracy did not improve from 0.74725
46/46 [==============================] - 66s 1s/step - loss: -2.1754 - accuracy: 0.7235 - val_loss: -2.1329 - val_accuracy: 0.7472
Epoch 4/5
46/46 [==============================] - ETA: 0s - loss: -2.5965 - accuracy: 0.7235
Epoch 4: val_accuracy did not improve from 0.74725
46/46 [==============================] - 66s 1s/step - loss: -2.5965 - accuracy: 0.7235 - val_loss: -2.4889 - val_accuracy: 0.7472
Epoch 5/5
46/46 [==============================] - ETA: 0s - loss: -2.9989 - accuracy: 0.7235
Epoch 5: val_accuracy did not improve from 0.74725
46/46 [==============================] - 66s 1s/step - loss: -2.9989 - accuracy: 0.7235 - val_loss: -2.8477 - val_accuracy: 0.7472
<keras.callbacks.History at 0x78db792a2290>
```

```
# Final evaluation of the model
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length,
input_length=max_review_length))
model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
print(model.summary())
model.load_weights("weights_best_cnn.hdf5")
scores = model.evaluate(X_test, y_test, verbose=0)
accuracy_cnn_lstm = scores[1]*100
print("Accuracy CNN using LSTM: %.2f%%" % (accuracy_cnn_lstm))
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 600, 32)           320000
 conv1d_1 (Conv1D)           (None, 600, 32)           3104

 max_pooling1d_1 (MaxPooling  (None, 300, 32)          0
 1D)

 lstm_3 (LSTM)               (None, 100)               53200

 dense_3 (Dense)             (None, 1)                 101

=================================================================
Total params: 376,405
Trainable params: 376,405
Non-trainable params: 0
_________________________________________________________________
None
Accuracy CNN using LSTM: 72.65%
```

```
table = PrettyTable()
table.field_names = ['Model', 'Accuracy']
table.add_row(['LSTM', accuracy_lstm])
table.add_row(['CNN using LSTM', accuracy_cnn_lstm])
print(table)
```

```
+----------------+-------------------+
|     Model      |      Accuracy     |
+----------------+-------------------+
|      LSTM      | 72.65474796295166 |
| CNN using LSTM | 72.65474796295166 |
+----------------+-------------------+
```

En el conjunto de datos Big_AHR.csv. Tanto el modelo LSTM como el modelo CNN + LSTM logran una precisión de aproximadamente el 72.65%. Estos resultados son más bajos a comparación del conjunto de datos de IMDB, aunque estos resultados son más elevados y realizar clasificaciones de sentimientos con un nivel aceptable de precisión.