In [167…
```
import pandas as pd
import numpy as np
import warnings
import re
import nltk
import seaborn as sns
import matplotlib.pyplot as plt
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import sent_tokenize, word_tokenize
import warnings
warnings.filterwarnings(action = 'ignore')
import gensim
from gensim.models import Word2Vec,Phrases
import keras
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, Bidirectional
from keras.layers.embeddings import Embedding
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/molly1998/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/molly1998/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[167… True

In [118…
```
trainset=pd.read_csv('/Users/molly1998/Desktop/python//train.csv')
testset=pd.read_csv('/Users/molly1998/Desktop/python//test.csv')
dataset=pd.concat([trainset,testset],ignore_index=True)
dataset_comm=np.concatenate([trainset['text'],testset['text']],axis=0)
dataset_tar=np.concatenate([trainset['sentiment'],testset['sentiment']],axis=0)
len(dataset_comm)
len(dataset_tar)
```

Out[118… 50000

In [82]:
```
dataset_comm[1]
```

Out[82]:
```
'The saddest thing about this "tribute" is that almost all the singers (includin
g the otherwise incredibly talented Nick Cave) seem to have missed the whole poi
nt where Cohen\'s intensity lies: by delivering his lines in an almost tuneless
poise, Cohen transmits the full extent of his poetry, his irony, his all-round h
umanity, laughter and tears in one.<br /><br />To see some of these singer upsta
rts make convoluted suffering faces, launch their pathetic squeals in the patent
effort to scream "I\'m a singer!," is a true pain. It\'s the same feeling many o
f you probably had listening in to some horrendous operatic versions of simple s
ongs such as Lennon\'s "Imagine." Nothing, simply nothing gets close to the simp
licity and directness of the original. If there is a form of art that doesn\'t n
eed embellishments, it\'s Cohen\'s art. Embellishments cast it in the street loo
king like the tasteless make-up of sex for sale.<br /><br />In this Cohen\'s tri
bute I found myself suffering and suffering through pitiful tributes and awful r
einterpretations, all of them entirely lacking the original irony of the master
```

```
and, if truth be told, several of these singers sounded as if they had been recr
uited at some asylum talent show. It\'s Cohen doing a tribute to them by letting
them sing his material, really, not the other way around: they may have been fri
ends, or his daughter\'s, he could have become very tender-hearted and in the mo
od for a gift. Too bad it didn\'t stay in the family.<br /><br />Fortunately, bu
t only at the very end, Cohen himself performed his majestic "Tower of Song," bu
t even that flower was spoiled by the totally incongruous background of the U2,
all of them carrying the expression that bored kids have when they visit their p
oor grandpa at the nursing home.<br /><br />A sad show, really, and sadder if yo
u truly love Cohen as I do.'
```

In [83]:
```
en_stops = set(stopwords.words('english'))
```

In [104…
```
def clean_review(comments: str) -> str:
    #Remove non-letters
    letters_only = re.compile(r'[^A-Za-z\s]').sub(" ", comments)
    #Convert to lower case
    lowercase_letters = letters_only.lower()
    return lowercase_letters


def lemmatize(tokens: list) -> list:
    #Lemmatize
    tokens = list(map(WordNetLemmatizer().lemmatize, tokens))
    lemmatized_tokens = list(map(lambda x: WordNetLemmatizer().lemmatize(x,"v"),
    #Remove stop words
    meaningful_words = list(filter(lambda x: not x in en_stops, lemmatized_token
    return meaningful_words


def preprocess(review: str, total: int, show_progress: bool = True) -> list:
    if show_progress:
        global counter
        counter += 1
        print('Processing... %6i/%6i'% (counter, total), end='\r')
    review = clean_review(review)
    tokens = word_tokenize(review)
    lemmas = lemmatize(tokens)
    return lemmas
counter=0
all_comments=list(map(lambda x: preprocess(x,len(dataset_comm)),dataset_comm))
```

```
Processing...  50000/ 50000
```

In [107…
```
all_comments[1]
```

Out[107…
```
['saddest',
 'thing',
 'tribute',
 'almost',
 'singer',
 'include',
 'otherwise',
 'incredibly',
 'talented',
 'nick',
 'cave',
 'seem',
 'miss',
 'whole',
 'point',
 'cohen',
```

```
 'intensity',
 'lie',
 'deliver',
 'line',
 'almost',
 'tuneless',
 'poise',
 'cohen',
 'transmit',
 'full',
 'extent',
 'poetry',
 'irony',
 'round',
 'humanity',
 'laughter',
 'tear',
 'one',
 'br',
 'br',
 'see',
 'singer',
 'upstart',
 'make',
 'convolute',
 'suffer',
 'face',
 'launch',
 'pathetic',
 'squeal',
 'patent',
 'effort',
 'scream',
 'singer',
 'true',
 'pain',
 'feel',
 'many',
 'probably',
 'listen',
 'horrendous',
 'operatic',
 'version',
 'simple',
 'song',
 'lennon',
 'imagine',
 'nothing',
 'simply',
 'nothing',
 'get',
 'close',
 'simplicity',
 'directness',
 'original',
 'form',
 'art',
 'need',
 'embellishment',
 'cohen',
 'art',
 'embellishment',
 'cast',
 'street',
 'look',
```

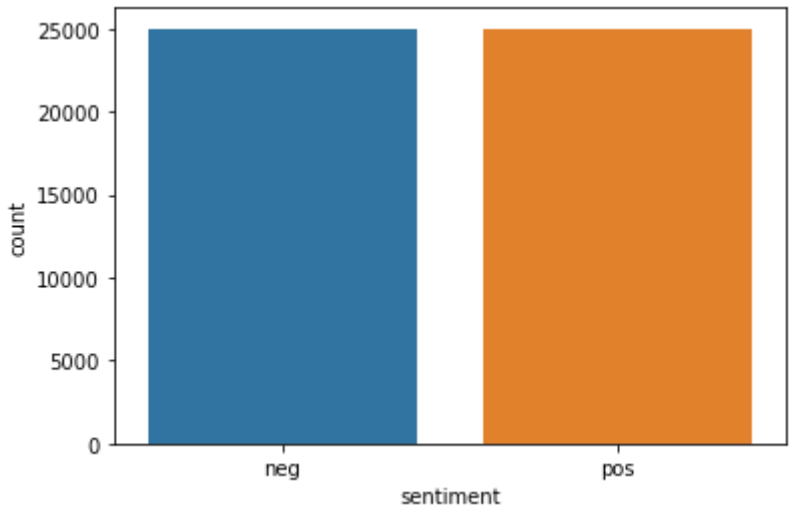
```
'like',
'tasteless',
'make',
'sex',
'sale',
'br',
'br',
'cohen',
'tribute',
'find',
'suffer',
'suffer',
'pitiful',
'tribute',
'awful',
'reinterpretation',
'entirely',
'lack',
'original',
'irony',
'master',
'truth',
'tell',
'several',
'singer',
'sound',
'recruit',
'asylum',
'talent',
'show',
'cohen',
'tribute',
'let',
'sing',
'material',
'really',
'way',
'around',
'may',
'friend',
'daughter',
'could',
'become',
'tender',
'hearted',
'mood',
'gift',
'bad',
'stay',
'family',
'br',
'br',
'fortunately',
'end',
'cohen',
'perform',
'majestic',
'tower',
'song',
'even',
'flower',
'wa',
'spoil',
'totally',
'incongruous',
```

```
'background',
'u',
'carry',
'expression',
'bore',
'kid',
'visit',
'poor',
'grandpa',
'nurse',
'home',
'br',
'br',
'sad',
'show',
'really',
'sadder',
'truly',
'love',
'cohen']
```

In [121…
```
dataset_tar[dataset_tar=="neg"]=0
dataset_tar[dataset_tar=="pos"]=1

##check if data is balanced
sns.countplot(x='sentiment', data=dataset)
```

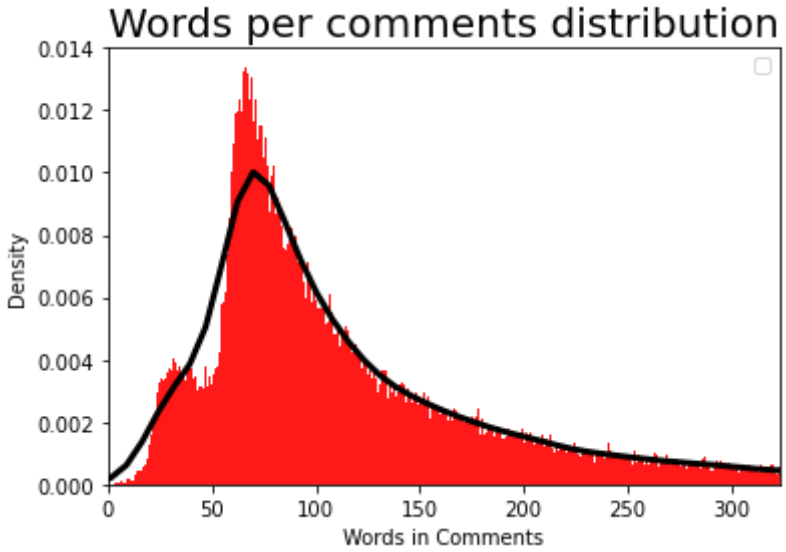
Out[121… <AxesSubplot:xlabel='sentiment', ylabel='count'>



In [141…
```
##count words in each comment
dataset['count_words']=list(map(lambda x: len(x),all_comments))
dataset['count_words']
fig, ax = plt.subplots()
sns.distplot(dataset['count_words'], bins=dataset['count_words'].max(),
             hist_kws={"alpha": 0.9, "color": "red"}, ax=ax,
             kde_kws={"color": "black", 'linewidth': 3})
ax.set_xlim(left=0, right=np.percentile(dataset['count_words'], 95))
ax.set_xlabel('Words in Comments')
ymax = 0.014
plt.ylim(0, ymax)

ax.set_title('Words per comments distribution', fontsize=20)
plt.legend()
plt.show()
```

```
No handles with labels found to put in legend.
```



data=data_senti.copy(deep=True) X_train=data.loc[:35000,'text'] y_train=data.loc[:35000,'sentiment'] X_test=data.loc[35000:,'text'] y_test=data.loc[35000:,'sentiment'] x_word2_train=X_train.copy() x_word2_test=X_test.copy()

In [146…
```
bigrams = Phrases(sentences=all_comments)
trigrams = Phrases(sentences=bigrams[all_comments])
```

In [147…
```
print(trigrams[bigrams[all_comments[1]]])
```

```
['saddest_thing', 'tribute', 'almost', 'singer', 'include', 'otherwise', 'incred
ibly_talented', 'nick_cave', 'seem', 'miss', 'whole', 'point', 'cohen', 'intensi
ty', 'lie', 'deliver_line', 'almost', 'tuneless', 'poise', 'cohen', 'transmit',
'full_extent', 'poetry', 'irony', 'round', 'humanity', 'laughter_tear', 'one',
'br', 'br', 'see', 'singer', 'upstart', 'make', 'convolute', 'suffer', 'face',
'launch', 'pathetic', 'squeal', 'patent', 'effort', 'scream', 'singer', 'true',
'pain', 'feel', 'many', 'probably', 'listen', 'horrendous', 'operatic', 'versio
n', 'simple', 'song', 'lennon', 'imagine', 'nothing', 'simply', 'nothing', 'ge
t', 'close', 'simplicity', 'directness', 'original', 'form', 'art', 'need', 'emb
ellishment', 'cohen', 'art', 'embellishment', 'cast', 'street', 'look_like', 'ta
steless', 'make', 'sex', 'sale', 'br', 'br', 'cohen', 'tribute', 'find', 'suffe
r', 'suffer', 'pitiful', 'tribute', 'awful', 'reinterpretation', 'entirely', 'la
ck', 'original', 'irony', 'master', 'truth_tell', 'several', 'singer', 'sound',
'recruit', 'asylum', 'talent', 'show', 'cohen', 'tribute', 'let', 'sing', 'mater
ial', 'really', 'way', 'around', 'may', 'friend', 'daughter', 'could', 'become',
'tender', 'hearted', 'mood', 'gift', 'bad', 'stay', 'family', 'br', 'br', 'fortu
nately', 'end', 'cohen', 'perform', 'majestic', 'tower', 'song', 'even', 'flowe
r', 'wa', 'spoil', 'totally', 'incongruous', 'background', 'u', 'carry', 'expres
sion', 'bore', 'kid', 'visit', 'poor', 'grandpa', 'nurse_home', 'br', 'br', 'sa
d', 'show', 'really', 'sadder', 'truly', 'love', 'cohen']
```

In [148…
```
w2v_model = word2vec.Word2Vec(sentences = trigrams[bigrams[all_comments]], min_c
                                        window = 8, workers=5, sample=1e-3)
w2v_model.init_sims(replace=True)
```

In [153…
```
w2v_model.wv.most_similar("love")
```

Out[153…
```
[('adore', 0.590673565864563),
 ('heart_warm', 0.5380239486694336),
 ('friendship', 0.5373594760894775),
 ('touch', 0.5295887589454651),
 ('fell_love', 0.5051866769790649),
```

```
('heart', 0.5048468112945557),
('chick_flick', 0.5030962228775024),
('inspirational', 0.5007826089859009),
('heartwarming', 0.49950939416885376),
('absolutely_love', 0.48973602056503296)]
```

In [163…
```
print(list(w2v_model.wv.vocab.keys()).index("love"))
print(list(w2v_model.wv.vocab.keys()).index("fell_love"))
```

```
157
6151
```

In [156…
```
%%time
def vectorize(text,vocabulary):
    keys = list(vocabulary.keys())
    filter_unknown = lambda x: vocabulary.get(x, None) is not None
    encode = lambda x: list(map(keys.index, filter(filter_unknown, x)))
    vectorized = list(map(encode, text))
    return vectorized
padded=pad_sequences(vectorize(trigrams[bigrams[all_comments]],w2v_model.wv.voca
```

```
CPU times: user 3min 33s, sys: 414 ms, total: 3min 34s
Wall time: 3min 34s
```

In [200…
```
X_train, X_test, y_train, y_test = train_test_split(padded,dataset_tar,test_size
```

In [201…
```
X_train=np.asarray(X_train).astype(np.int)
X_test=np.asarray(X_test).astype(np.int)
y_test=np.asarray(y_test).astype(np.int)
y_train=np.asarray(y_train).astype(np.int)
```

In [235…
```
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, Bidirectional,BatchNormalization,
from keras.layers.embeddings import Embedding

def build_model(embedding_matrix: np.ndarray, input_length: int):
    model = Sequential()
    model.add(Embedding(
        input_dim = embedding_matrix.shape[0],
        output_dim = embedding_matrix.shape[1],
        input_length = input_length,
        weights = [embedding_matrix],
        trainable=False))
    #model.add(Bidirectional(LSTM(88, recurrent_dropout=0.1)))
    #model.add(Dense(32))
    #model.add(BatchNormalization())

    model.add(Dense(64))
    model.add(Dropout(0.25))
    model.add(Flatten())
    #model.add(Dense(128))
    #model.add(Dropout(0.15))
    model.add(Dense(1, activation='sigmoid'))
    model.summary()
    return model

model = build_model(
    embedding_matrix=w2v_model.wv.vectors,
    input_length=250)
model.compile(
    loss="binary_crossentropy",
```

```
    optimizer='adam',
    metrics=['accuracy'])
```

```
Model: "sequential_24"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_24 (Embedding)     (None, 250, 256)          3269888
_________________________________________________________________
dense_51 (Dense)             (None, 250, 64)           16448
_________________________________________________________________
dropout_44 (Dropout)         (None, 250, 64)           0
_________________________________________________________________
flatten_2 (Flatten)          (None, 16000)             0
_________________________________________________________________
dense_52 (Dense)             (None, 1)                 16001
=================================================================
Total params: 3,302,337
Trainable params: 32,449
Non-trainable params: 3,269,888
_________________________________________________________________
```

In [236…

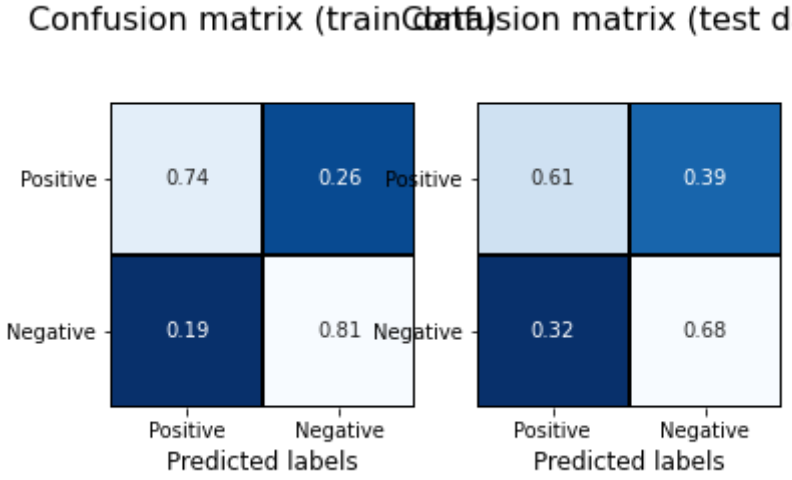
```
mlp_model = model.fit(
    x=X_train,
    y=y_train,
    validation_data=(X_test, y_test),
    batch_size=100,
    epochs=20)
```

```
Epoch 1/20
425/425 [==============================] - 20s 47ms/step - loss: 0.6825 - accura
cy: 0.5521 - val_loss: 0.6448 - val_accuracy: 0.6169
Epoch 2/20
425/425 [==============================] - 20s 46ms/step - loss: 0.6171 - accura
cy: 0.6572 - val_loss: 0.6279 - val_accuracy: 0.6497
Epoch 3/20
425/425 [==============================] - 19s 44ms/step - loss: 0.5907 - accura
cy: 0.6858 - val_loss: 0.6406 - val_accuracy: 0.6324
Epoch 4/20
425/425 [==============================] - 19s 44ms/step - loss: 0.5766 - accura
cy: 0.6970 - val_loss: 0.6268 - val_accuracy: 0.6516
Epoch 5/20
425/425 [==============================] - 19s 44ms/step - loss: 0.5605 - accura
cy: 0.7103 - val_loss: 0.6294 - val_accuracy: 0.6540
Epoch 6/20
425/425 [==============================] - 19s 44ms/step - loss: 0.5511 - accura
cy: 0.7150 - val_loss: 0.6547 - val_accuracy: 0.6341
Epoch 7/20
425/425 [==============================] - 19s 44ms/step - loss: 0.5473 - accura
cy: 0.7175 - val_loss: 0.6445 - val_accuracy: 0.6477
Epoch 8/20
425/425 [==============================] - 19s 44ms/step - loss: 0.5411 - accura
cy: 0.7244 - val_loss: 0.6390 - val_accuracy: 0.6532
Epoch 9/20
425/425 [==============================] - 18s 44ms/step - loss: 0.5244 - accura
cy: 0.7366 - val_loss: 0.6510 - val_accuracy: 0.6445
Epoch 10/20
425/425 [==============================] - 18s 43ms/step - loss: 0.5247 - accura
cy: 0.7356 - val_loss: 0.6511 - val_accuracy: 0.6475
Epoch 11/20
425/425 [==============================] - 19s 44ms/step - loss: 0.5232 - accura
cy: 0.7376 - val_loss: 0.6534 - val_accuracy: 0.6447
Epoch 12/20
425/425 [==============================] - 19s 44ms/step - loss: 0.5121 - accura
cy: 0.7480 - val_loss: 0.6571 - val_accuracy: 0.6492
```

```
Epoch 13/20
425/425 [==============================] - 19s 44ms/step - loss: 0.5142 - accura
cy: 0.7404 - val_loss: 0.6620 - val_accuracy: 0.6443
Epoch 14/20
425/425 [==============================] - 19s 44ms/step - loss: 0.5088 - accura
cy: 0.7440 - val_loss: 0.6593 - val_accuracy: 0.6493
Epoch 15/20
425/425 [==============================] - 19s 44ms/step - loss: 0.5036 - accura
cy: 0.7494 - val_loss: 0.6705 - val_accuracy: 0.6495
Epoch 16/20
425/425 [==============================] - 19s 44ms/step - loss: 0.4987 - accura
cy: 0.7492 - val_loss: 0.6792 - val_accuracy: 0.6448
Epoch 17/20
425/425 [==============================] - 19s 44ms/step - loss: 0.4963 - accura
cy: 0.7510 - val_loss: 0.6703 - val_accuracy: 0.6492
Epoch 18/20
425/425 [==============================] - 19s 44ms/step - loss: 0.5032 - accura
cy: 0.7474 - val_loss: 0.6765 - val_accuracy: 0.6469
Epoch 19/20
425/425 [==============================] - 19s 44ms/step - loss: 0.4957 - accura
cy: 0.7543 - val_loss: 0.6767 - val_accuracy: 0.6516
Epoch 20/20
425/425 [==============================] - 19s 44ms/step - loss: 0.4912 - accura
cy: 0.7606 - val_loss: 0.6998 - val_accuracy: 0.6457
```

In [237…

```
y_train_pred = model.predict_classes(X_train)
y_test_pred = model.predict_classes(X_test)
def plot_confusion_matrix(y_true, y_pred, ax, class_names, vmax=None,
                          normed=True, title='Confusion matrix'):
    matrix = confusion_matrix(y_true,y_pred)
    if normed:
        matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]
    sns.heatmap(matrix, vmax=vmax, annot=True, square=True, ax=ax,
                cmap=plt.cm.Blues_r, cbar=False, linecolor='black',
                linewidths=1, xticklabels=class_names)
    ax.set_title(title, y=1.20, fontsize=16)
    #ax.set_ylabel('True labels', fontsize=12)
    ax.set_xlabel('Predicted labels', y=1.10, fontsize=12)
    ax.set_yticklabels(class_names, rotation=0)
fig, (axis1, axis2) = plt.subplots(nrows=1, ncols=2)
plot_confusion_matrix(y_test, y_test_pred, ax=axis2,
                      title='Confusion matrix (test data)',
                      class_names=['Positive', 'Negative'])
plot_confusion_matrix(y_train, y_train_pred, ax=axis1,
                      title='Confusion matrix (train data)',
                      class_names=['Positive', 'Negative'])
```



In [ ]: