Datasets:

```
https://www.kaggle.com/utathya/imdb-review-dataset
http://ai.stanford.edu/~amaas/data/sentiment/ (better)
```

In [ ]:

```
import pandas as pd
# Reading the csv dataset into a pandas dataframe
df = pd.read_csv('E:/Internships/TCS-iON/Code/MyCode/IMDB/imdb_master.csv', encoding = 'ISO
# Adding a column representing 1 for 'pos' and 0 for 'neg' sentiments
df['senti'] = df.apply(lambda x: 1 if x['label'] == 'pos' else 0, axis = 1)
# Deleting unnecessary columns
df = df.drop(['Unnamed: 0', 'type', 'file'], axis = 1)
# Converting the data type to string
df['review'] = df["review"].astype("str")
# Converting all text to lowercase for use
df['review'] = df['review'].str.lower()
df.head()
```

review	label	senti
once again mr. costner has dragged out a movie...	neg	0
this is an example of why the majority of acti...	neg	0
first of all i hate those moronic rappers, who...	neg	0
not even the beatles could write songs everyon...	neg	0
brass pictures (movies is not a fitting word f...	neg	0

once again mr. costner has dragged out a movie for far longer than necessary. aside from the terrific sea rescue sequences, of which there are very few i just did not care about any of the characters. most of us have ghosts in the closet, and costner's character are realized early on, and then forgotten until much later, by which time i did not care. the character we should really care about is a very cocky, overconfident ashton kutcher. the problem is he comes off as kid who thinks he's better than anyone else around him and shows no signs of a cluttered closet. his only obstacle appears to be winning over costner. finally when we are well past the half way point of this stinker, costner tells us all about kutcher's ghosts. we are told why kutcher is driven to be the best with no prior inkling or foreshadowing. no magic here, it was all i could do to keep from turning it off an hour in.

In [ ]:

```
import re
import string
from nltk import WordNetLemmatizer
from nltk.stem.snowball import SnowballStemmer
from nltk.corpus import stopwords
# Initialising the nltk stop_words, stemmer and lemmatizer functions
stop_words = set(stopwords.words("english"))
lemmatizer = WordNetLemmatizer()
stemmer = SnowballStemmer("english")
# Creating a function for text cleaning
def textCleanser(myText):
    # Removing the name titles and the period symbols after it
    myText = re.sub(r'[mdsr]r(s)?\.', '', myText)
    # Removing punctuation
    myPunct = string.punctuation
    punctToSpace = str.maketrans(myPunct, len(myPunct)*' ')
    myText = myText.translate(punctToSpace)
    # Removing the '@username' mentions
    myText = re.sub(r'@\w+', '', myText)
    # Removing urls
    myText = re.sub(r'(http(s)?://)?(www\.)?.+\.com', '', myText)
    # Removing numbers
    myText = re.sub(r'\d+', '', myText)
    # Removing stopwords
    myText = [word for word in myText.split(' ') if not word in stop_words]
    myText = [word for word in myText if word != '']
    # Lemmatizing the text
    myText = [lemmatizer.lemmatize(token) for token in myText]
    # Stemming the text
    # myText = [stemmer.stem(token) for token in myText]
    return myText
for i in range(len(df['review'])):
    df['review'][i] = textCleanser(df['review'][i])
df.head()
```

	review	label	senti
	[costner, dragged, movie, far, longer, necessa...	neg	0
	[example, majority, action, film, generic, bor...	neg	0
	[first, hate, moronic, rapper, could, nt, act,...	neg	0
	[even, beatles, could, write, song, everyone, ...	neg	0
	[brass, picture, movie, fitting, word, really,...	neg	0

['costner', 'dragged', 'movie', 'far', 'longer', 'necessary', 'aside', 'terrific', 'sea', 'rescue', 'sequence', 'care', 'character', 'u', 'ghost', 'closet', 'costner', 'character', 'realized', 'early', 'forgotten', 'much', 'later', 'time', 'care', 'character', 'really', 'care', 'cocky', 'overconfident', 'ashton', 'kutcher', 'problem', 'come', 'kid', 'think', 'better', 'anyone', 'else', 'around', 'show', 'sign', 'cluttered', 'closet', 'obstacle', 'appears', 'winning', 'costner', 'finally', 'well', 'past', 'half', 'way', 'point', 'stinker', 'costner', 'tell', 'u', 'kutcher', 'ghost', 'told', 'kutcher', 'driven', 'best', 'prior', 'inkling', 'foreshadowing', 'magic', 'could', 'keep', 'turning', 'hour']

In [ ]:

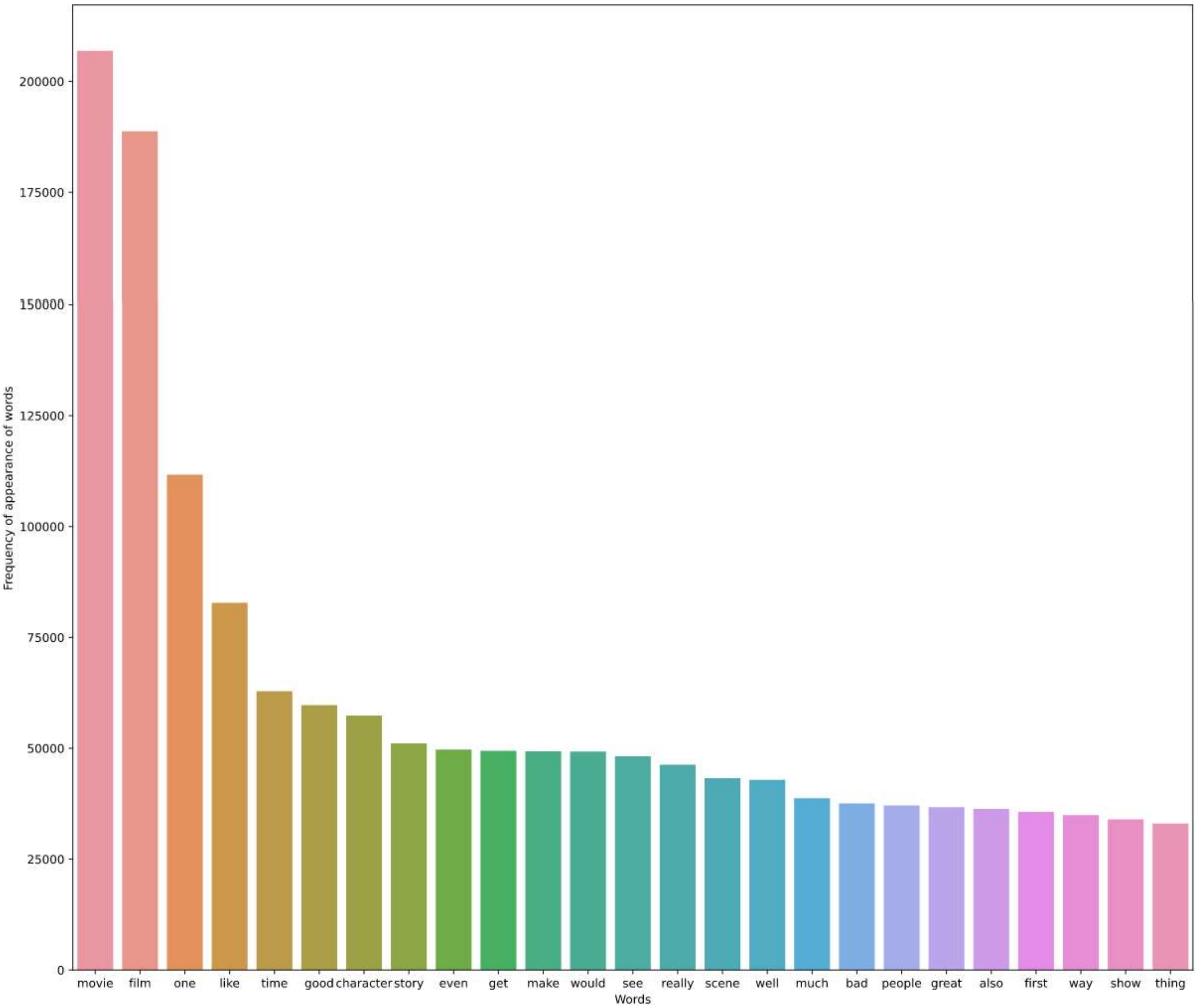
```
myReviews = []
for i in range(len(df['review'])):
    for j in df['review'][i]:
        if j != 'br':
            myReviews.append(j)
```

In [ ]:

```
from collections import Counter
import collections
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.metrics import classification_report
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
import numpy as np
np.random.seed(1234)
# Initialising the Count Vectorizer
cv = CountVectorizer()
myBow = cv.fit_transform(myReviews)
wordFrequency = dict(zip(cv.get_feature_names(), np.asarray(myBow.sum(axis = 0)).ravel()))
wordCounter = collections.Counter(wordFrequency)
# Storing the frequency of appearance of words
dfWordCounter = pd.DataFrame(wordCounter.most_common(25), columns = ['word', 'frequency'])
```

In [ ]:

```
# Plotting the top 25 most frequently occuring words
plt.close('all')
fig, ax = plt.subplots(figsize = (17, 15))
sns.barplot(x = 'word', y = 'frequency', data = dfWordCounter, ax = ax)
sns.set_palette('pastel')
plt.xlabel('Words')
plt.ylabel('Frequency of appearance of words')
plt.show()
```


Frequency of appearance of words
200000
175000
150000
125000
100000
75000
50000
25000
0
movie film one like time good character story even get make would see really scene well much bad people great also first way show thing
Words

In [ ]:

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense , Input , LSTM , Embedding, Dropout , Activation, GRU, Flatt
from keras.layers import Bidirectional, GlobalMaxPool1D
from keras.models import Model, Sequential
from keras.layers import Convolution1D
from keras import initializers, regularizers, constraints, optimizers, layers
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
# Splitting the dataframe to training and testing data
X_train, X_test, y_train, y_test = train_test_split(df["review"], df['senti'],test_size=0.2
# Initialising the tokenizer
max_features = 5000
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(X_train)
list_tokenized_train = tokenizer.texts_to_sequences(X_train)
maxlen = 130
X_t = pad_sequences(list_tokenized_train, maxlen=maxlen)
y = y_train
embed_size = 128
# Initializing a bidirectional sequential LSTM using Adam optimizer, positive activation fu
model = Sequential()
model.add(Embedding(max_features, embed_size))
model.add(Bidirectional(LSTM(32, return_sequences = True)))
model.add(GlobalMaxPool1D())
model.add(Dense(20, activation="relu"))
model.add(Dropout(0.05))
model.add(Dense(1, activation="sigmoid"))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
batch_size = 100
epochs = 5
model.fit(X_t,y, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

Epoch 1:

```
time: 191s
speed: 299ms/step
loss: 0.4645
accuracy: 0.7541
val_loss: 0.4357
val_accuracy: 0.7618
```

Epoch 2:

```
time: 146s
speed: 227ms/step
loss: 0.4051
accuracy: 0.7794
val_loss: 0.4385
val_accuracy: 0.7543
```

Epoch 3:

```
time: 150s
speed: 235ms/step
loss: 0.3785
accuracy: 0.7990
val_loss: 0.4480
val_accuracy: 0.7666
```

Epoch 4:

```
time: 156s
speed: 244ms/step
loss: 0.3473
accuracy: 0.8201
val_loss: 0.4616
val_accuracy: 0.7623
```

Epoch 5:

```
time: 171s
speed: 267ms/step
loss: 0.3076
accuracy: 0.8500
val_loss: 0.5101
val_accuracy: 0.7552
```

In [ ]:

```
# Testing the model
list_sentences_test = X_test
list_tokenized_test = tokenizer.texts_to_sequences(list_sentences_test)
X_te = pad_sequences(list_tokenized_test, maxlen=maxlen)
prediction = model.predict(X_te)
y_pred = (prediction > 0.5)
from sklearn.metrics import f1_score, confusion_matrix
print('F1-score: {0}'.format(f1_score(y_pred, y_test)))
print('Confusion matrix:')
confusion_matrix(y_pred, y_test)
```

F1-score:

```
0.5307570241762346
```

Confusion matrix:

```
[[12130,  2188]
 [ 2839,  2843]]
```

Accuracy:

```
74.86%
```