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 [ ]:

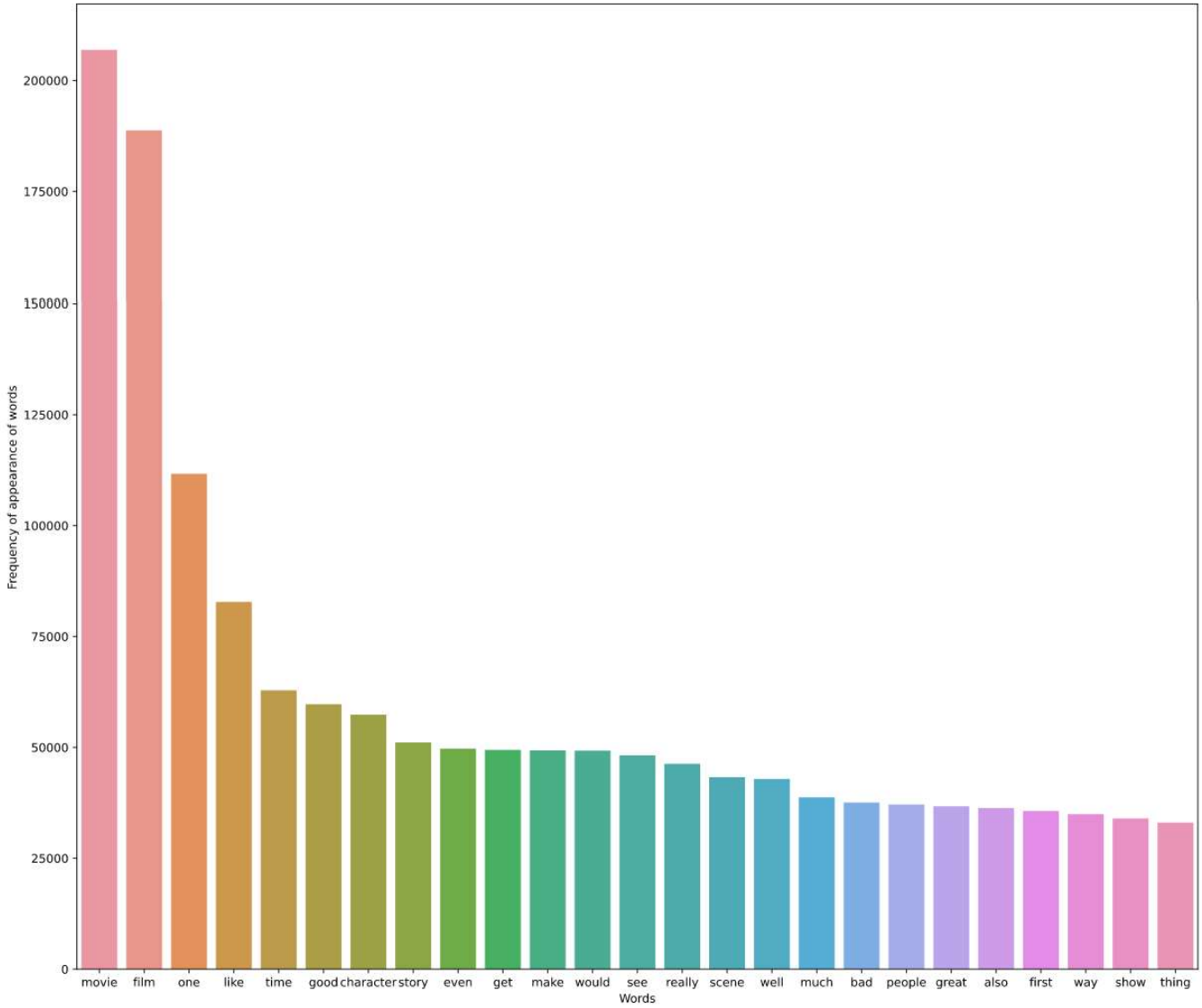
```
myReviews = []
for i in range(len(df['review'])):
    for j in df['review'][i]:
        if j != 'br':
            myReviews.append(j)
```

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()
```



In [ ]:

```
# Defining a dummy function for the tokenizer and preprocessor inputs of the vectorizers so
def dummy_fun(doc):
    return doc
tfidf = TfidfVectorizer(sublinear_tf=True, max_df = 85000, min_df = 2000, norm='l2', encodi
features = []
features = tfidf.fit_transform(df.review).toarray()
labels = df.senti
print(features.shape)
```

Features Shape:

```
(100000, 788)
```

In [ ]:

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.pipeline import Pipeline
# Splitting the dataframe to training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(df["review"], df['senti'],test_size=0.2
vectorizerCount = CountVectorizer(tokenizer = dummy_fun, preprocessor = dummy_fun)
X_train_counts = vectorizerCount.fit_transform(X_train)
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
nBayes = MultinomialNB().fit(X_train_tfidf, y_train)
```

In [ ]:

```
y_= nBayes.predict(vectorizerCount.transform(X_test))
```

In [ ]:

```
# Testing the data
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_,normalize=True))
```

Accuracy:

```
74.98%
```