Importing dependencies

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import nltk
from sklearn.feature_extraction.text import CountVectorizer
from wordcloud import WordCloud,STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import re,string,unicodedata
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,f1_score
from sklearn.model_selection import train_test_split
from string import punctuation
from nltk import pos_tag
from nltk.corpus import wordnet
import re
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
```

Loading the data

```
df=pd.read_csv('/content/drive/MyDrive/Sentiment Analysis/Data/IMDB-Dataset.csv', encoding='latin-1')
```

Data Cleaning and Preprocessing

```
#Customize stopword as per data
nltk.download('stopwords')
stop_words = stopwords.words('english')
new_stopwords = ["would","shall","could","might"]
stop_words.extend(new_stopwords)
stop_words.remove("not")
stop_words=set(stop_words)
print(stop_words)
```

```
{'under', 'my', 'an', "that'll", 'been', "didn't", 'aren', 'the', "hasn't", 'most', "haven't", 'yourself', 'herself', 'didn', 'wasn
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
#Removing special character
def remove_special_character(content):
    return re.sub('\W+',' ', content )#re.sub('\[[^&@#!]]*\]', '', content)

# Removing URL's
def remove_url(content):
    return re.sub(r'http\S+', '', content)

#Removing the stopwords from text
def remove_stopwords(content):
    clean_data = []
    for i in content.split():
        if i.strip().lower() not in stop_words and i.strip().lower().isalpha():
            clean_data.append(i.strip().lower())
    return " ".join(clean_data)

# Expansion of english contractions
def contraction_expansion(content):
    content = re.sub(r"won\'t", "would not", content)
    content = re.sub(r"can\'t", "can not", content)
    content = re.sub(r"don\'t", "do not", content)
    content = re.sub(r"shouldn\'t", "should not", content)
```

```
    content = re.sub(r"needn\'t", "need not", content)
    content = re.sub(r"hasn\'t", "has not", content)
    content = re.sub(r"haven\'t", "have not", content)
    content = re.sub(r"weren\'t", "were not", content)
    content = re.sub(r"mightn\'t", "might not", content)
    content = re.sub(r"didn\'t", "did not", content)
    content = re.sub(r"n\'t", " not", content)
    '''content = re.sub(r"\'re", " are", content)
    content = re.sub(r"\'s", " is", content)
    content = re.sub(r"\'d", " would", content)
    content = re.sub(r"\'ll", " will", content)
    content = re.sub(r"\'t", " not", content)
    content = re.sub(r"\'ve", " have", content)
    content = re.sub(r"\'m", " am", content)'''
    return content

#Data preprocessing
def data_cleaning(content):
    content = contraction_expansion(content)
    content = remove_special_character(content)
    content = remove_url(content)

    content = remove_stopwords(content)
    return content
```

```
#Applying data cleaning
pd.options.display.max_colwidth = 1000
#Data cleaning
df['Reviews_clean']=df['Reviews'].apply(data_cleaning)
df.head(5)
```

		there in the...		Agora eu sei que esses filmes nÃ£o devem ser sÃ©rios, mas vamos lÃ¡, Ã© o cinema 101 que se alguÃ©m f...	many cameos sorry ass excuses films taking away jobs actors writers directors truly deserv...
1	1	I am writing this in hopes that this gets put over the previous review of this "film". How anyone can find this slop entertaining is completely beyond me. First of all a spoof film entitled "Disaster Movie", should indeed be a spoof on disaster films. Now I have seen 1 (yes count them, 1) disaster film being spoofed, that being "Twister". How does Juno, Iron Man, Batman, The Hulk, Alvin and the Chipmunks, Amy Winehouse, or Hancock register as Disaster films? Selzterwater and Failburg once again have shown that they lack any sort of writing skill and humor. Having unfortunately been tortured with Date Movie and Epic Movie I know exactly what to expect from these two...no plot, no jokes just bad references and cheaply remade scenes from other films. Someone should have informed them that satire is more than just copy and paste from one film to another, though I shouldn't say that because some of these actually just seem to be taken from trailers.There is nothing clever or witty or re...	Disaster Movie	Estou escrevendo isso na esperanÃ§a de que isso seja colocado sobre a revisÃ£o anterior deste "filme". Como alguÃ©m pode achar divertido esse desleixo estÃ¡ completamente alÃ©m de mim. Antes de mais nada, um filme de parÃ³dia intitulado "Filme de desastre" deveria ser, de fato, uma parÃ³dia de filmes de desastre. Agora eu jÃ¡ vi 1 (sim, conte-os, 1) filme de desastre sendo falsificado, sendo "Twister". Como Juno, Homem de Ferro, Batman, O Hulk, Alvin e os Esquilos, Amy Winehouse ou Hancock se registram como filmes de Desastre? Selzterwater e Failburg mostraram mais uma vez que nÃ£o possuem nenhum tipo de habilidade e humor de escrita. Infelizmente, tendo sido torturado com Date Movie e Epic Movie, sei exatamente o que esperar desses dois ... nenhum enredo, nenhuma piada, apenas mÃ¡s referÃªncias e cenas refeitas de outros filmes. AlguÃ©m deveria ter informado a eles que a sÃ¡tira Ã© mais do que apenas copiar e colar de um filme para outro, embora eu nÃ£o deva dizer isso porque algu...	writing hopes gets put previous review film anyone find slop entertaining completely beyond first spoof film entitled disaster movie indeed spoof disaster films seen yes count disaster film spoofed twister juno iron man batman hulk alvin chipmunks amy winehouse hancock register disaster films selzterwater failburg shown lack sort writing skill humor unfortunately tortured date movie epic movie know exactly expect two plot jokes bad references cheaply remade scenes films someone informed satire copy paste one film another though not say actually seem taken trailers nothing clever witty remotely smart way two write not believe people still pay see travesties insult audience though enjoy films doubt smart enough realize rating unfortunately not number low enough yes includes negatives rate deserves top worst films time right date movie epic faliure mean movie meet spartans rather forced hour manos hands fate marathon watch slop
		Really, I could write a scathing review of this turd sandwich, but		Realmente, eu poderia escrever uma crÃtica contundente sobre esse	

Feature Engineering

```
#Mapping rating data to Binary label 1 (+ve) if rating >=7 and 0 (-ve) if rating <=4 and 2 (neutral) if ra
df['Label'] = df['Ratings'].apply(lambda x: '1' if x >= 7 else ('0' if x<=4 else '2'))
#Removing
df=df[df.Label<'2']
data=df[['Reviews_clean','Label']]
print(data['Label'].value_counts())
```

```
0    60000
1    60000
Name: Label, dtype: int64
```

```
#Importing dependencies for feature engineering
import sys
import os
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
from prettytable import PrettyTable
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
```

Lemmatization

```
# lemmatization of word
class LemmaTokenizer(object):
    def __init__(self):
```

```
        self.wordnetlemma = WordNetLemmatizer()
    def __call__(self, reviews):
        return [self.wordnetlemma.lemmatize(word) for word in word_tokenize(reviews)]
```

Vectoization with Count Vectorizer and TDIDF Vectorizer with Unigram

```
import nltk
nltk.download('punkt')
nltk.download('wordnet')
train,test=train_test_split(data,test_size=.3,random_state=42, shuffle=True)
countvect = CountVectorizer(analyzer = "word", tokenizer = LemmaTokenizer(), ngram_range=(1,1), min_df=10,m
tfidfvect = TfidfVectorizer(analyzer = "word", tokenizer = LemmaTokenizer(), ngram_range=(1,1),min_df=10,ma
x_train_count = countvect.fit_transform(train['Reviews_clean']).toarray()
x_test_count = countvect.transform(test['Reviews_clean']).toarray()
x_train_tfidf = tfidfvect.fit_transform(train['Reviews_clean']).toarray()
x_test_tfidf = tfidfvect.transform(test['Reviews_clean']).toarray()
y_train = train['Label']
y_test = test['Label']
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

Feature Importance with Logistic Regression and Count Vectorizer with unigram

```
lgr = LogisticRegression()
lgr.fit(x_train_count,y_train)
lgr.score(x_test_count,y_test)
lgr.coef_[0]
i=0
importantfeature = PrettyTable(["Feature", "Score"])
for feature, importance in zip(countvect.get_feature_names_out(), lgr.coef_[0]):
    if i<=200:
        importantfeature.add_row([feature, importance])
        i=i+1
print(importantfeature)
```

```
+----------------+------------------------+
|    Feature     |         Score          |
+----------------+------------------------+
|      able      |  0.09948021729583935   |
|   absolutely   |  0.04928181139341904   |
|      act       |  -0.15698758988629533  |
|     acting     |  -0.16554860358454662  |
|     action     |   0.2686694865338257   |
|     actor      |  -0.12933328612077247  |
|    actress     |  -0.1489168697925077   |
|    actually    | -0.052070130929095626  |
|      add       |  0.027070353095046004  |
|     adult      |  0.05954646965772197   |
|      age       |  0.09771190095691933   |
|     alien      |  -0.0721752881106565   |
|     almost     | -0.013852048588960835  |
|     along      |  0.12262111059815697   |
|    already     |  -0.20171237379314463  |
|      also      |  0.09685115820655628   |
|    although    |  0.19629354744205255   |
|     always     |   0.2018578524201237   |
|    amazing     |   0.8421518297897805   |
|    american    |  0.05490888090162389   |
|    annoying    |  -0.7415920897196361   |
|    another     |  -0.15261540989210784  |
|     anyone     | -0.018801393011886016  |
|    anything    |  -0.29270079300578816  |
|     anyway     |  0.039398139882690836  |
|     around     |  -0.09786505565645186  |
|      art       | -0.002824546854484584  |
|    attempt     |  -0.6738581933334602   |
|   attention    |   0.3022833083553928   |
|    audience    | -0.009222501642938315  |
|    average     |  -0.27552710116327334  |
|      away      |   0.0662190754650192   |
|     awful      |   -1.1933746190676038  |
|       b        | -0.017925197253893986  |
|      back      |  0.022902845995928983  |
|      bad       |  -0.34492919739614175  |
|     based      |  0.09379241960512272   |
```

beautiful	0.4260586136785764
become	0.09103497829707141
becomes	-0.1130423217825764
begin	-0.05036128319201348
beginning	0.169964075188926
behind	-0.039263205837075384
believe	-0.0005587913653516917
best	0.41066001589332657
better	-0.05624599552071519
big	-0.01987374992124096
bit	0.19924970223693375
black	-0.023156394685004994
blood	-0.06723193016262485
body	-0.03681562140985229
book	-0.044476933676119594
boring	-1.0915821995627517
boy	0.013725984800562233

Feature Importance with TFIDF vectorizer and Logistic Regression with Unigram

```
lgr = LogisticRegression()
lgr.fit(x_train_tfidf,y_train)
lgr.score(x_test_tfidf,y_test)
lgr.coef_[0]
i=0
importantfeature = PrettyTable(["Feature", "Score"])
for feature, importance in zip(tfidfvect.get_feature_names_out(), lgr.coef_[0]):
    if i<=100:
        importantfeature.add_row([feature, importance])
        i=i+1
print(importantfeature)
```

Feature	Score
able	0.39700755873095384
absolutely	0.48054639840990787
act	-1.0488078209705092
acting	-1.6529223210920863
action	2.661498410649932
actor	-1.3612120300887194
actress	-1.0076219456773845
actually	-0.4711678536905498
add	0.1548901563324264
adult	0.6882768024887284
age	0.7577095272499967
alien	-0.3362334252148682
almost	-0.2930875737610137
along	0.7374473907997262
already	-1.1274303477568333
also	1.2142852251221006
although	1.2214036573549865
always	1.7060017688032059
amazing	4.793428511865544
american	0.461586618743471
annoying	-3.8025854084696022
another	-1.328014670919997
anyone	-0.21234706315571122
anything	-2.1229963168743273
anyway	0.13215048299969895
around	-0.7966348090365658
art	0.3632181231416922
attempt	-4.047793034178131
attention	1.6863904729767933
audience	0.42474496077471147
average	-1.0920591134290885
away	0.5886410680859387
awful	-6.970830611496972
b	-0.06081964613796297
back	0.17153588391190128
bad	-3.747434546237462
based	0.5968891662547068
beautiful	2.766550374408251
become	0.7464562458889129
becomes	-0.7759953047462301
begin	-0.3778164648278749
beginning	0.8735361482302675
behind	0.014016943320300752
believe	-0.005037036386458905
best	3.792732964739248
better	-0.46679086055047947
big	-0.06518111858919383
bit	1.818312976952132
black	0.02061498655822086

```
|     blood      |  -0.28435723025867005 |
|      body      |  -0.24305079174047806 |
|      book      |  0.14493265537053093  |
|     boring     |  -6.6150764736277985  |
|      boy       |   0.2518472727011538  |
```

Feature Importance with Logistic Regression and TFIDF Vectorizer with Bigram

```
lgr.fit(x_train_tfidf,y_train)
lgr.score(x_test_tfidf,y_test)
lgr.coef_[0]
i=0
importantfeature = PrettyTable(["Feature", "Score"])
for feature, importance in zip(tfidfvect.get_feature_names_out(), lgr.coef_[0]):
    if i<=50:
        importantfeature.add_row([feature, importance])
        i=i+1
print(importantfeature)
```

```
+------------+----------------------+
|  Feature   |        Score         |
+------------+----------------------+
|    able    |  0.39700755873095384 |
| absolutely |  0.48054639840990787 |
|    act     |  -1.0488078209705092 |
|   acting   |  -1.6529223210920863 |
|   action   |   2.661498410649932  |
|   actor    |  -1.3612120300887194 |
|  actress   |  -1.0076219456773845 |
|  actually  |  -0.4711678536905498 |
|    add     |  0.1548901563324264  |
|   adult    |  0.6882768024887284  |
|    age     |  0.7577095272499967  |
|   alien    |  -0.3362334252148682 |
|   almost   |  -0.2930875737610137 |
|   along    |  0.7374473907997262  |
|  already   |  -1.1274303477568333 |
|    also    |  1.2142852251221006  |
|  although  |  1.2214036573549865  |
|   always   |  1.7060017688032059  |
|  amazing   |   4.793428511865544  |
|  american  |   0.461586618743471  |
|  annoying  |  -3.8025854084696022 |
|  another   |  -1.328014670919997  |
|   anyone   | -0.21234706315571122 |
|  anything  |  -2.1229963168743273 |
|   anyway   |  0.13215048299969895 |
|   around   |  -0.7966348090365658 |
|    art     |  0.3632181231416922  |
|  attempt   |  -4.047793034178131  |
| attention  |  1.6863904729767933  |
|  audience  |  0.42474496077471147 |
|  average   |  -1.0920591134290885 |
|    away    |  0.5886410680859387  |
|   awful    |  -6.970830611496972  |
|     b      | -0.06081964613796297 |
|    back    |  0.17153588391190128 |
|    bad     |  -3.747434546237462  |
|   based    |  0.5968891662547068  |
| beautiful  |   2.766550374408251  |
|   become   |  0.7464562458889129  |
|  becomes   |  -0.7759953047462301 |
|   begin    |  -0.3778164648278749 |
| beginning  |  0.8735361482302675  |
|   behind   | 0.014016943320300752 |
|  believe   | -0.005037036386458905 |
|    best    |   3.792732964739248  |
|   better   | -0.46679086055047947 |
|    big     | -0.06518111858919383 |
|    bit     |   1.818312976952132  |
|   black    | 0.02061498655822086  |
|   blood    | -0.28435723025867005 |
|    body    | -0.24305079174047806 |
+------------+----------------------+
```

```
pd.options.display.max_colwidth = 1000
df[["Reviews","Ratings","Movies"]][(df['Ratings']>=9)&(df['Reviews_clean'].str.contains("bad review"))].he
```

	Reviews	Ratings	Movies
120047	While I wouldn't call this the greatest movie ever made, it's not anywhere near as bad as other reviewers have made it out to be. An average rating of 5 or 6 stars would be fair, but 1.5 is harsh and totally undeserved.Ring of Terror feels like an episode of The Twilight Zone stretched to an hour. In fact, it's so much like a TV show that one wonders if it might not have been originally created as a pilot.If you're a fan of 1950s horror/suspense series like Thriller, The Veil, One Step Beyond, Tales of Tomorrow, and Alfred Hitchcock Presents, you'll likely find this a pleasant way to spend an hour, as I did.Normally I would only give this film 6 out of 10 stars, but because others have been panning it so unmercifully, I'm giving it a 9.	9	Ring of Terror
120211	This movie was a blast for my little guys, they loved every minute of it, I have read all of the bad reviews, and could not disagree more. This movie, is pure and good. There is just enough action to keep the kids interested, and not so much that you leave the theater with them bouncing off the walls either. It is funny with jokes that everyone can appreciate. I think people have gotten used to so much violence and adult content in our kids movies that they are disappointed when it is missing, like the movie wasn't entertaining enough for the parents. Well, NEWS FLASH.....It's a kids movie, and a perfect one at that. Kids need these kinds of movies, not Spongebob and the like which are more to entertain the parents.	9	Doogal
120238	I am a huge horror buff and prefer pieces that delve into the characters psychological issues. This film was awesome on so many levels, the acting and writing were fantastic and creepy and I was afraid or and empathetic with the murderer the whole time. What an interesting study on the line between sick and a danger to others, and the line between being a mean girl and being psychotic. Set in a great location, a house full of creepy art, in the winter in Conneticut and with amazing performances from many of my favorite actresses. It actually shocks me that others have given this such a bad review, I loved this movie, I guess it goes to show you everyone will have a different opinion but I say don't miss this film!	9	#Horror
120239	No idea why there are so many bad reviews here? I loved it; I thought it was a very advanced thoughtful film. The graphic were #killer. The comparison of video game culture and young girl culture was spot on. This film makes connections that I've never seen on the big screen but, do see in every day life.The casting was spot on, Hello 12 year-old girls are supposed to be a little annoying. I do wish that more directors would take color into more consideration the way this film does. T The highly stylized sets make the murder scenes more believable because everything is so unbelievable. How can you live in 2016 and not "get"a film about social media and accelerationism. #duh Someone explain this to me.	9	#Horror
120273	What do you get when you cross Love Story with Star Wars with Blade Runner with Back to the Future with MTV? Love Story 2050, that's what. What a fun movie for the entire family. This fantasy of epic proportions is much, much better than AI, a similar sci-fi classic. The thrills are non-stop in this blockbuster, from its lead off car chase to bike racing stunts to the vantage point of a moving roller coaster to speeding hover-craft--you will be on the edge of your seat from beginning to end. The version I saw was only partially in English and I still was glued to the screen. I can't wait to see a version with subtitles. The mega budget special effects are out of this world and highly convincing. The future vision of XBox was hilarious. Those who are complaining about how long this movie is simply don't understand Bollywood. The three hours went by quickly; it seemed to be only an hour. There could have been a better twist with the Darth Vader character. For example, I suspected tha...	9	Love Story 2050
...	...	...	...
	Being a film watcher that looks for great acting, i surprised myself when i enjoyed this film.Being more a film fan than a martial arts fan i was expecting to be writing a bad review for this flick. No one can deny that van damme is a great martial artist, but his acting is so so the opposite.You have to look at it as a martial arts film (which it is) and accept the stunning fight sequences (especially the final fight scene which was ... are just the way he can express his great skills to an audience.I recommend this film		

Vectorization with Count Vectorizer and TFIDF Vectorizer with Trigram

```
import nltk
nltk.download('punkt')
nltk.download('wordnet')
train,test=train_test_split(data,test_size=.3,random_state=42, shuffle=True)
countvect = CountVectorizer(analyzer = "word", tokenizer = LemmaTokenizer(), ngram_range=(3,3), min_df=10,
tfidfvect = TfidfVectorizer(analyzer = "word", tokenizer = LemmaTokenizer(), ngram_range=(3,3),min_df=10,m
x_train_count = countvect.fit_transform(train['Reviews_clean']).toarray()
x_test_count = countvect.transform(test['Reviews_clean']).toarray()
x_train_tfidf = tfidfvect.fit_transform(train['Reviews_clean']).toarray()
x_test_tfidf = tfidfvect.transform(test['Reviews_clean']).toarray()
y_train = train['Label']
y_test = test['Label']
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Feature Importance with Logistic Regression and Count Vectorizer with Trigram

```
lgr = LogisticRegression()
lgr.fit(x_train_count,y_train)
lgr.score(x_test_count,y_test)
lgr.coef_[0]
i=0
importantfeature = PrettyTable(["Feature", "Score"])
for feature, importance in zip(countvect.get_feature_names_out(), lgr.coef_[0]):
    if i<=200:
        importantfeature.add_row([feature, importance])
        i=i+1
print(importantfeature)
```

Feature	Score
acting not bad	-0.41582443344090864
acting not good	-1.0614028411735406
acting not great	-0.46707395630784376
acting pretty good	0.9624699253760244
acting top notch	1.8034277974499235
action movie not	1.0284974648276257
action set piece	1.2171146811632634
action take place	0.04259251365109098
actor good job	0.596764841910835
actually pretty good	0.3083612569919975
actually quite good	-0.16354119748827142
almost every scene	-0.31966198722624384
b movie not	-0.04716888748028589
bad acting bad	-2.3287081585210254
bad bad bad	-1.7666009469066026
bad guy not	0.16629388224472788
bad horror movie	-1.025351468087346
bad movie not	-0.6539660886638583
bad not even	-1.6309771354996798
bad special effect	-0.7680710806837042
based true story	0.49087040504086865
best film ever	1.0597205763936839
best movie ever	1.7335209232441549
best movie seen	1.8058104341454444
best movie year	1.4959704711901778
best part film	-0.004016531359990099
best part movie	-0.34403424997231924
best thing film	-0.4869700947142451
best thing movie	-0.3083868653248926
billy bob thornton	0.2884307029202007
blah blah blah	0.007095611048698205
blair witch project	-0.08529108091698837
breath fresh air	1.7939117056463305
budget horror film	0.10992199248210421
ca not wait	1.400551992472707
can not believe	-0.6265339467832575
can not say	-0.0684256438974244
can not understand	0.3379726050622202
can not wait	1.9630279451434467
character one dimensional	-1.3001005883590102
claude van damme	0.42246204356876227
comic book movie	1.2236084630256583
complete waste time	-2.134455149165938
cuba gooding jr	-0.008237196651769706
cutting room floor	-0.6996884736419002
definitely worth watch	3.047267493628777
definitely worth watching	2.602296157932083
die hard fan	-0.20540327619197413
done much better	-1.2610545026473492
effect story comedy	-0.34626285732134815
eight title brazil	1.9244976057248782
even come close	-0.3927793208702641
even though movie	0.09907622167408738
even though not	0.3986668259650988
ever made not	0.04364907312319447

Feature Importance with Logistic Regression and TFIDF Vectorizer with Trigram

```
lgr = LogisticRegression()
lgr.fit(x_train_tfidf,y_train)
lgr.score(x_test_tfidf,y_test)
lgr.coef_[0]
i=0
importantfeature = PrettyTable(["Feature", "Score"])
for feature, importance in zip(countvect.get_feature_names_out(), lgr.coef_[0]):
    if i<=200:
        importantfeature.add_row([feature, importance])
```

```
        importantfeature.add_row([feature, importance])
        i=i+1
print(importantfeature)
```

```
|       last not least        |   0.5342101052691458   |
|      laugh loud funny       |   1.1992626797244383   |
|      laugh loud moment      |   1.579982771508334    |
|       let not forget        |   0.2806708247087335   |
|      let start saying       |  0.22303161177052175   |
|       life never get        |   -1.414401390247894   |
|       like first one        |  0.21391598082108368   |
|      like horror movie      |   0.3416935127070429   |
|       like low budget       |  -0.7363102353575469   |
|       like movie not        |   0.4337066177177099   |
|       like year old         |  -0.9046879860746775   |
|       long long time        |   0.890723950778871    |
|       long time ago         |  -0.4400006101536415   |
|      long time since        |   1.0371833424687407   |
|    look like something      |  -1.4612283968567106   |
|  looking forward seeing     |   0.3420201866127275   |
|        low budget b         |  -0.3493683725161391   |
|      low budget film        |  -0.05025750828091194  |
|     low budget horror       |  0.16691606375730828   |
|      low budget movie       |  -0.3777817886509993   |
|       low budget not        |  -0.7954959319061509   |
|       made look like        |  -0.8329059692183841   |
|       made tv movie         |  -1.1992791868008246   |
|     main character not      |  -0.35667938989585646  |
|       make bad movie        |  -0.8343606236371462   |
|       make feel like        |  0.49557670809661575   |
|      make good movie        |  -0.5957815838327223   |
|     make little sense       |   -2.170742317406258   |
|       make look like        |  -1.3731999730172837   |
|     make matter worse       |  -1.4266417661074398   |
|      make movie like        | -0.053224269636337765  |
|      make much sense        |  -0.9564228130327596   |
|       make sense not        |  -0.8238547074105317   |
|      manos hand fate        |  -1.8363526093917364   |
|      many people not        |   1.6192825842119478   |
|       many year ago         |  0.09841002968252952   |
|      martial art movie      |   0.2835682066326356   |
|        may may not          |  0.18103770787015513   |
|    minute running time      |  0.03571516987396411   |
|       movie bad not         |  -1.9659211720089467   |
|       movie can not         |  -0.4317660295502344   |
|   movie definitely not      |  0.11276141296999115   |
|     movie even though       |   0.345095789847708    |
|      movie ever made        | -0.020178030075973068  |
|      movie ever seen        |  -0.22637931237367434  |
|    movie ever watched       |  -0.3111217065248724   |
|      movie feel like        |  -0.6180978288345567   |
|      movie felt like        |  -0.7818066093279746   |
|      movie first time       |   0.6565489516266568   |
|        movie go see         |   1.663096145465962    |
|      movie last night       |   0.7952411643836983   |
|       movie like not        |   0.2901139773726702   |
|       movie like one        |  0.10774419034086645   |
|      movie long time        |   1.676402379746924    |
|       movie look like       |  -1.1848619442151127   |
|      movie low budget       |  0.14194063408040242   |
|       movie may not         |   1.3229603233644274   |
+-----------------------------+------------------------+
```

✓ 9s completed at 8:25 PM