1 Packages

In [1]:

```
#Standard python libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import warnings
warnings.filterwarnings(action='ignore')

# Preprocessing tools
from sklearn.model_selection import train_test_split,cross_val_predict,cross
from sklearn.preprocessing import MinMaxScaler,StandardScaler,OneHotEncoder
scaler = StandardScaler()
from sklearn import metrics

# # Models & Utilities
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression,LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import accuracy_score, confusion_matrix, classification

# Warnings
import warnings
warnings.filterwarnings(action='ignore')

# NLP Libraries
import nltk
import collections
nltk.download('punkt')
from sklearn.manifold import TSNE
from nltk.tokenize import word_tokenize
np.random.seed(0)
import re
from nltk.corpus import stopwords
from nltk.collocations import *
from nltk import FreqDist
from nltk import word_tokenize
from nltk import ngrams
import string
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score
from sklearn.datasets import fetch_20newsgroups
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB

# Added
nltk.download('stopwords')

```

executed in 3.06s, finished 12:22:37 2021-06-21

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Johnny\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Johnny\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[1]: True

In [52]:
```
df = pd.read_csv('data/IMDB Dataset.csv')

df.head()
```

executed in 724ms, finished 13:36:51 2021-06-21

Out[52]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

In [53]:
```
# Taking a look at our columns
print(df.info())

# Checking for NA data
print(df.isna().sum())
```

executed in 44ms, finished 13:36:51 2021-06-21

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
None
review       0
sentiment    0
dtype: int64
```

In [54]:
```
df['sentiment'].value_counts()
```

executed in 21ms, finished 13:36:51 2021-06-21

Out[54]:
```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

In [55]:
```
1 df.describe()
```
executed in 161ms, finished 13:36:53 2021-06-21

Out[55]:

	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety and not...	positive
freq	5	25000

In [61]:
```
1 df[df.duplicated(keep=False)]
```
executed in 265ms, finished 13:39:44 2021-06-21

Out[61]:

	review	sentiment
42	Of all the films I have seen, this one, The Ra...	negative
84	We brought this film as a joke for a friend, a...	negative
140	Before I begin, let me get something off my ch...	negative
219	Ed Wood rides again. The fact that this movie ...	negative
245	I have seen this film at least 100 times and I...	positive
...	...	...
49912	This is an incredible piece of drama and power...	positive
49950	This was a very brief episode that appeared in...	negative
49984	Hello it is I Derrick Cannon and I welcome you...	negative
49986	This movie is a disgrace to the Major League F...	negative
49991	Les Visiteurs, the first movie about the medie...	negative

824 rows × 2 columns

In [63]:
```
1 df = df.drop_duplicates()
```
executed in 253ms, finished 13:40:27 2021-06-21

1.0.1 remove duplicates?

1.0.2 change pos to 1 and neg to 0

1.1 Add features

2 Text Preprocessing

2.1 Remove

In [64]:
```
1 first_review = df['review'][0][0:500]
2 first_review
```

executed in 14ms, finished 13:40:33 2021-06-21

Out[64]:
```
"One of the other reviewers has mentioned that after watching just 1 Oz episode
you'll be hooked. They are right, as this is exactly what happened with me.<br
/><br />The first thing that struck me about Oz was its brutality and unflinchi
ng scenes of violence, which set in right from the word GO. Trust me, this is n
ot a show for the faint hearted or timid. This show pulls no punches with regar
ds to drugs, sex or violence. Its is hardcore, in the classic use of the word.<
br /><br />It is called OZ"
```

In [65]:
```
## From tjhe lessons
from nltk import regexp_tokenize
pattern = r"([a-zA-Z]+(?:'[a-z]+)?)"
regexp_tokenize(first_review,pattern)
```

executed in 19ms, finished 13:40:33 2021-06-21

Out[65]:
```
['One',
 'of',
 'the',
 'other',
 'reviewers',
 'has',
 'mentioned',
 'that',
 'after',
 'watching',
 'just',
 'Oz',
 'episode',
 "you'll",
 'be',
 'hooked',
 'They',
 'are',
 'right',
 'as',
 'this',
 'is',
 'exactly',
 'what',
 'happened',
 'with',
 'me',
 'br',
 'br',
 'The',
 'first',
 'thing',
 'that',
 'struck',
 'me',
 'about',
 'Oz',
 'was',
 'its',
 'brutality',
 'and',
 'unflinching',
 'scenes',
 'of',
 'violence',
 'which',
 'set',
 'in',
 'right',
 'from',
```

```
'the',
'word',
'GO',
'Trust',
'me',
'this',
'is',
'not',
'a',
'show',
'for',
'the',
'faint',
'hearted',
'or',
'timid',
'This',
'show',
'pulls',
'no',
'punches',
'with',
'regards',
'to',
'drugs',
'sex',
'or',
'violence',
'Its',
'is',
'hardcore',
'in',
'the',
'classic',
'use',
'of',
'the',
'word',
'br',
'br',
'It',
'is',
'called',
'OZ']
```

In [66]:

```
1 stop_words_list = stopwords.words('english')
2
3 stop_words_list.append('br')
```

executed in 14ms, finished 13:40:33 2021-06-21

In [67]:
```
tokens = regexp_tokenize(first_review, pattern)

tokens

bad_tags = ['br']

cleaned_tokens = []

for token in tokens:
    if token not in bad_tags:
        cleaned_tokens.append(token)

cleaned_tokens[0:5]

```

executed in 21ms, finished 13:40:33 2021-06-21

Out[67]:
```
['One', 'of', 'the', 'other', 'reviewers']
```

In [68]:
```
stop_words_list = stopwords.words('english')

remove_words = ["i've", "i'm", 'br']

stop_words_list += remove_words

stop_words_list
```

executed in 16ms, finished 13:40:34 2021-06-21

Out[68]:
```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
```

3 Tokenizer

In [69]:
```
def my_tokenizer(first_review, stop_words=False,
                 stop_words_add=[],
                 remove_words=remove_words, show_full=False):

        pattern = r"([a-zA-Z]+(?:'[a-z]+)?)"
#         pattern = r"[a-zA-Z.0-9+#-/]*[^.\s]"

        tokens = regexp_tokenize(first_review, pattern)

        stop_words_list = []

        if stop_words == True:
            stop_words_list = stopwords.words('english')
            stop_words_list += stop_words_add

        stop_words_list += remove_words
        [x for x in stop_words_list]
        cleaned_tokens = []

        for token in tokens:
            if token not in stop_words_list:
                cleaned_tokens.append(token)

        if show_full == False:
            return cleaned_tokens
        else:
            return " ".join(cleaned_tokens)#, stop_words_list

my_tokenizer("i'm going to the store", stop_words=True,
             stop_words_add=[], show_full=True)

```

executed in 27ms, finished 13:40:34 2021-06-21

Out[69]:
```
'going store'
```

In [70]:
```
df['reviews_t'] = df['review'].apply(lambda text: my_tokenizer(text, stop_wo

df['reviews_t']

```

executed in 56.1s, finished 13:41:31 2021-06-21

Out[70]:
```
0        [One, reviewers, mentioned, watching, Oz, epis...
1        [A, wonderful, little, production, The, filmin...
2        [I, thought, wonderful, way, spend, time, hot,...
3        [Basically, there's, family, little, boy, Jake...
4        [Petter, Mattei's, Love, Time, Money, visually...
                               ...
49995    [I, thought, movie, right, good, job, It, crea...
49996    [Bad, plot, bad, dialogue, bad, acting, idioti...
49997    [I, Catholic, taught, parochial, elementary, s...
49998    [I'm, going, disagree, previous, comment, side...
49999    [No, one, expects, Star, Trek, movies, high, a...
Name: reviews_t, Length: 49582, dtype: object
```

4 EDA TO DO:

Break out everything by positive and negative FIRST

visualize pos neg bigrams trigrams etc

Are pos negative review shorter or longer

Are longer words more typical of positive or negative reviews

Code to use from:

https://medium.com/plotly/nlp-visualisations-for-clear-immediate-insights-into-text-data-and-outputs-9ebfab168d5b (https://medium.com/plotly/nlp-visualisations-for-clear-immediate-insights-into-text-data-and-outputs-9ebfab168d5b)

fig = px.bar(long_bigram_df_tidy, title='Comparision: ' + ngrams_list[0] + ' | ' + ngrams_list[1], x='ngram', y='value' , color='variable', template='plotly_white', color_discrete_sequence=px.colors.qualitative.Bold , labels={'variable': 'Company:', 'ngram': 'N-Gram'}) fig.update_layout(legend_orientation="h") fig.update_layout(legend=dict(x=0.1, y=1.1)) fig.update_yaxes(title='', showticklabels=False) fig.show()

4.1 Bag of Words

In [71]:
```
# Tokens

tokens = []

for row in df['reviews_t']:
    tokens.extend(row)

print(f'Number of tokens: {len(tokens)}')

```

executed in 331ms, finished 13:41:31 2021-06-21

```
Number of tokens: 6466732
```

In [72]:
```
# Frequency distribution

review_freqdist = FreqDist(tokens)
most_common = review_freqdist.most_common(50)
```

executed in 6.11s, finished 13:41:37 2021-06-21

In [73]:
```
# Number of unique tokens

len(review_freqdist)
```

executed in 14ms, finished 13:41:37 2021-06-21

Out[73]:
```
138151
```

In [74]:
```
1 print(most_common)
```
executed in 13ms, finished 13:41:37 2021-06-21

```
[('I', 143636), ('The', 89220), ('movie', 85132), ('film', 76206), ('one', 4789
9), ('like', 38334), ('This', 29312), ('good', 28333), ('time', 24336), ('It',
23967), ('would', 23762), ('really', 22171), ('story', 22133), ('see', 22017),
('even', 21591), ('much', 18586), ('well', 17971), ('get', 17773), ('bad', 1727
3), ('people', 16838), ('great', 16680), ('made', 15803), ('first', 15573), ('m
ake', 15543), ('also', 15185), ('way', 15130), ('movies', 14819), ('could', 148
08), ('But', 14141), ('characters', 14106), ('think', 13936), ('films', 13410),
('And', 13317), ('seen', 13179), ('character', 13119), ('A', 13092), ('watch',
12806), ('plot', 12577), ('many', 12509), ('two', 12451), ('acting', 12382),
('know', 12229), ('never', 12113), ('life', 12046), ('love', 11786), ("It's", 1
1783), ('In', 11776), ('show', 11710), ('little', 11569), ('best', 11527)]
```

In [75]:
```
1 total = review_freqdist.N()
2 for word in review_freqdist:
3     review_freqdist[word] /= float(total)
4
5 review_freqdist
```
executed in 675ms, finished 13:41:38 2021-06-21

Out[75]:
```
FreqDist({'I': 0.0222115281721896, 'The': 0.013796767826469382, 'movie': 0.0131
64609264772377, 'film': 0.01178431393167368, 'one': 0.007406987022192972, 'lik
e': 0.005927878254425883, 'This': 0.004532737710484987, 'good': 0.0043813474874
17138, 'time': 0.0037632609484976338, 'It': 0.0037061996693229285, ...})
```

In [76]:
```
1 total_word_count = sum(review_freqdist.values())
2 review_top_10 = review_freqdist.most_common(10)
3 print('Word\t\t\tNormalized Frequency')
4 for word in review_top_10:
5     normalized_frequency = word[1] / total_word_count
6     print('{} \t\t\t {:.4}'.format(word[0], normalized_frequency))
```
executed in 76ms, finished 13:41:38 2021-06-21

```
Word                    Normalized Frequency
I                        0.02221
The                      0.0138
movie                    0.01316
film                     0.01178
one                      0.007407
like                     0.005928
This                     0.004533
good                     0.004381
time                     0.003763
It                       0.003706
```

4.2 Word Clouds

In [77]:
```
# !pip install wordcloud

from wordcloud import WordCloud
```
executed in 9ms, finished 13:41:38 2021-06-21

4.2.1 To Do:

Break out by positive and negative

4.2.2 Positive

In [78]:
```
# Positive tokens

df_positive = df['reviews_t'].loc[df['sentiment'] == 'positive']

tokens_positive = []

for row in df_positive:
    tokens_positive.extend(row)

print(f'Number of tokens: {len(tokens_positive)}')
```
executed in 235ms, finished 13:41:38 2021-06-21

```
Number of tokens: 3278252
```

In [ ]:
```

```
executed in 186ms, finished 16:56:10 2021-06-18

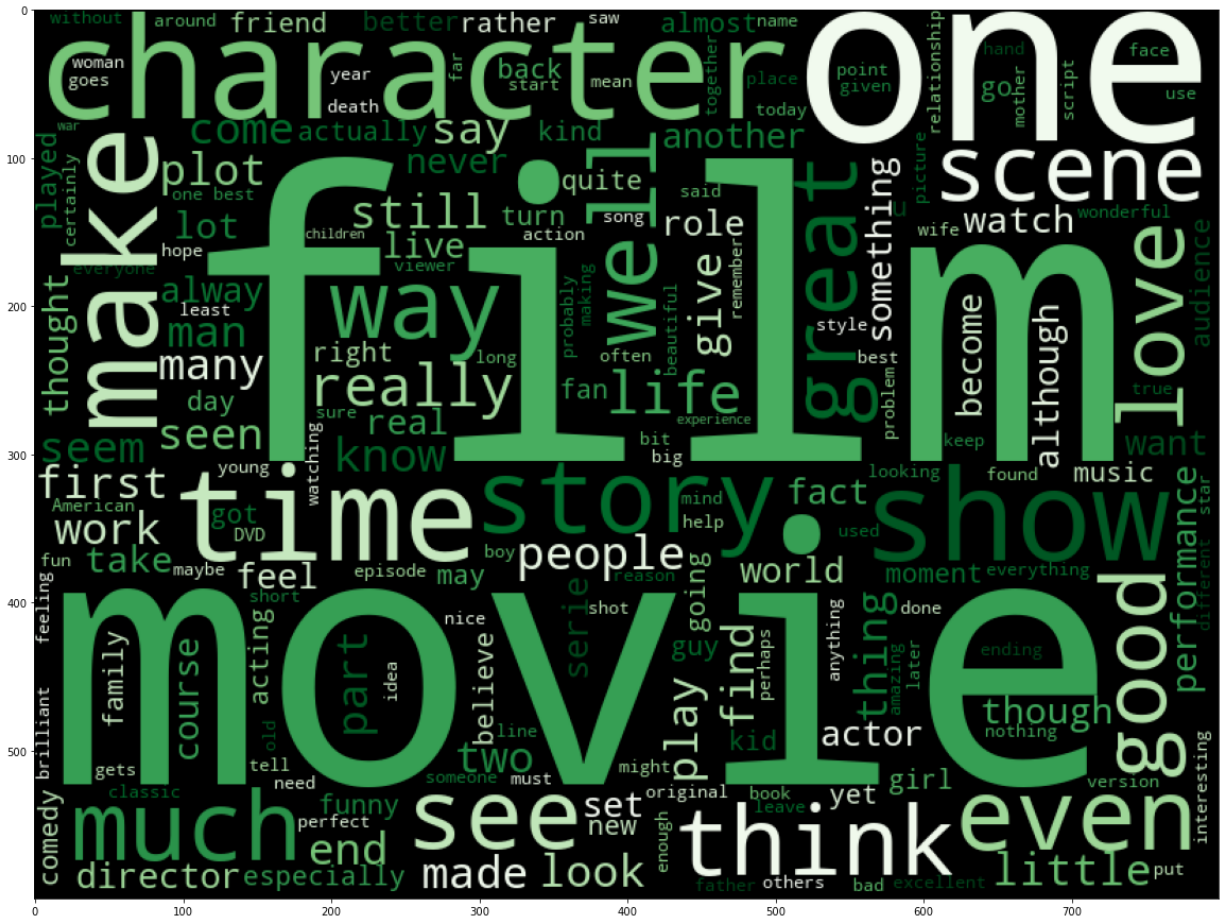
In [79]:
```
#Positive reviews
from wordcloud import WordCloud, STOPWORDS
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 200,
               width = 800, height = 600,
               colormap='Greens',
               stopwords = STOPWORDS).generate(" ".join(tokens_positive))

plt.imshow(wc)

```

executed in 31.2s, finished 13:42:10 2021-06-21

Out[79]: <matplotlib.image.AxesImage at 0x2a1378a1250>



In [80]:
```
1 plt.show(wc)
```
executed in 12ms, finished 13:42:10 2021-06-21

4.2.3 Positive

In [81]:
```
1  # Negative tokens
2
3  df_negative = df['reviews_t'].loc[df['sentiment'] == 'negative']
4
5  tokens_negative = []
6
7  for row in df_negative:
8      tokens_negative.extend(row)
9
10 print(f'Number of tokens: {len(tokens_negative)}')
```
executed in 219ms, finished 13:42:10 2021-06-21

```
Number of tokens: 3188480
```

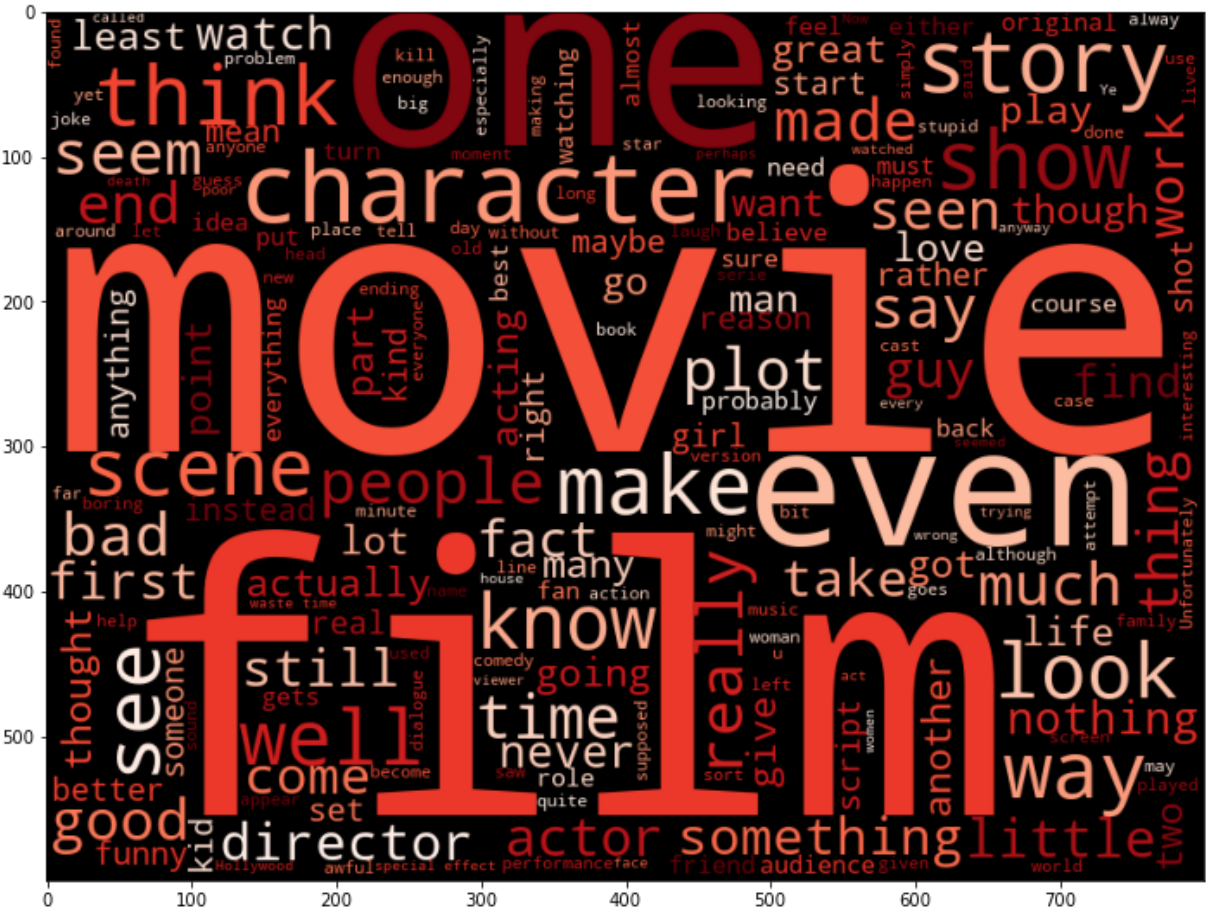
In [82]:
```
#Positive reviews
from wordcloud import WordCloud, STOPWORDS
plt.figure(figsize = (12,12))
wc = WordCloud(max_words = 200,
               width = 800, height = 600,
               colormap='Reds',
               stopwords = STOPWORDS).generate(" ".join(tokens_negative))

plt.imshow(wc)

```

executed in 29.0s, finished 13:42:39 2021-06-21

Out[82]: <matplotlib.figure.AxesImage at 0x2a1384b7460>



In [ ]:
```
1
```

In [ ]:
```
1
```

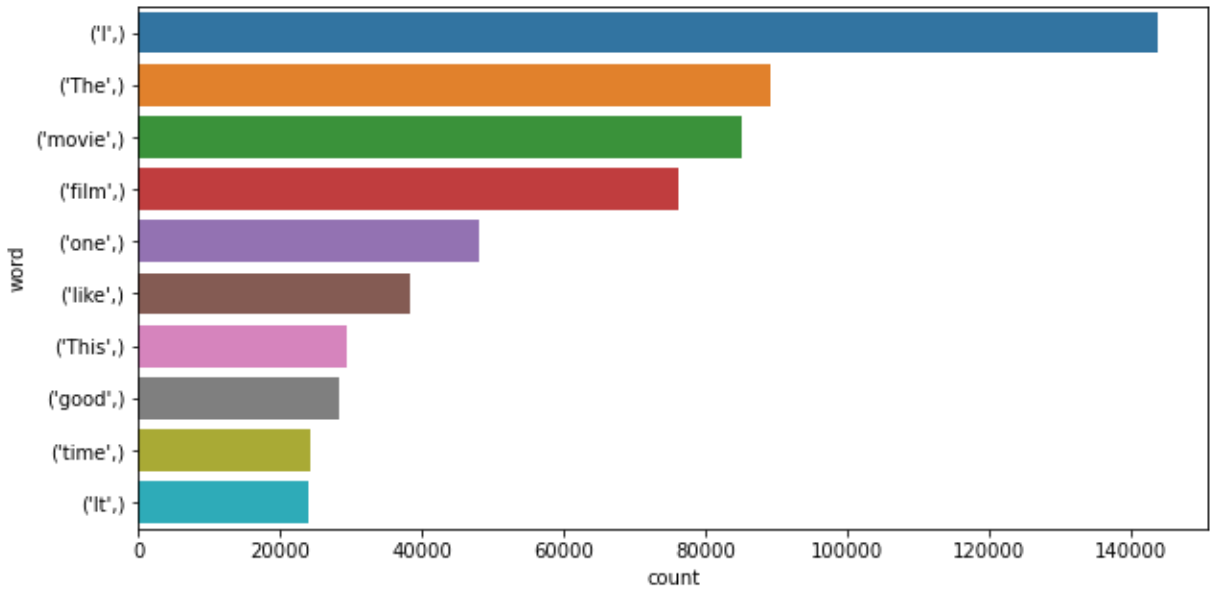
4.3 Ngrams

In [83]:
```
1 n_gram = (pd.Series(nltk.ngrams(tokens, 2)).value_counts())[:7]
```
executed in 13.3s, finished 13:42:52 2021-06-21

In [84]:
```
1  def plot_ngram(tokens, i):
2      n_gram = (pd.Series(nltk.ngrams(tokens, i)).value_counts())[:10]
3      n_gram_df=pd.DataFrame(n_gram)
4      n_gram_df = n_gram_df.reset_index()
5      n_gram_df = n_gram_df.rename(columns={"index": "word", 0: "count"})
6      print(n_gram_df.head(10))
7      plt.figure(figsize = (10,5))
8      return sns.barplot(x='count',y='word', data=n_gram_df)
9
10 plot_ngram(tokens, 1)
```
executed in 6.19s, finished 13:42:58 2021-06-21

```
      word   count
0     (I,)  143636
1    (The,)   89220
2  (movie,)   85132
3   (film,)   76206
4    (one,)   47899
5   (like,)   38334
6   (This,)   29312
7   (good,)   28333
8   (time,)   24336
9     (It,)   23967
```

Out[84]: <AxesSubplot:xlabel='count', ylabel='word'>

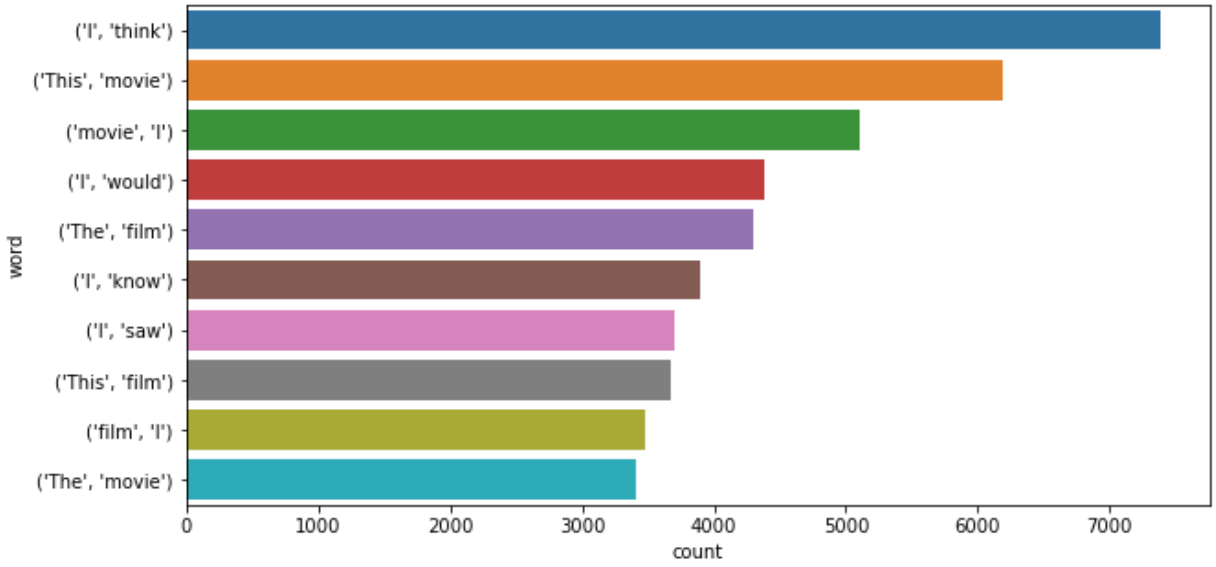


In [85]:
```
1 plot_ngram(tokens, 2)
```

executed in 17.4s, finished 13:43:16 2021-06-21

	word	count
0	(I, think)	7386
1	(This, movie)	6187
2	(movie, I)	5099
3	(I, would)	4387
4	(The, film)	4297
5	(I, know)	3895
6	(I, saw)	3697
7	(This, film)	3670
8	(film, I)	3480
9	(The, movie)	3405

Out[85]: <AxesSubplot:xlabel='count', ylabel='word'>

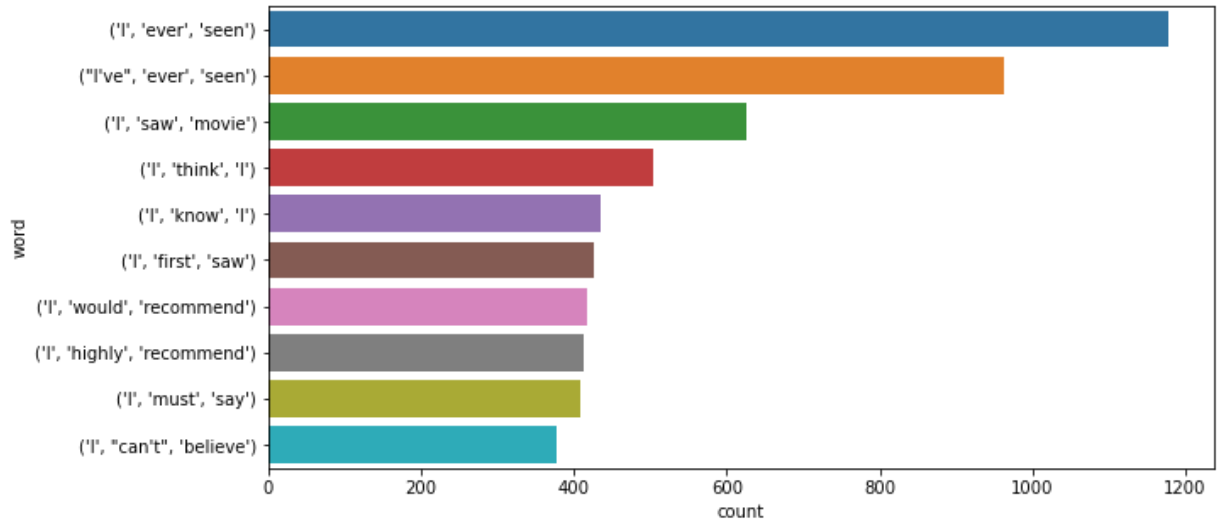


In [86]:
```
1 plot_ngram(tokens, 3)
```

executed in 16.9s, finished 13:43:33 2021-06-21

	word	count
0	(I, ever, seen)	1179
1	(I've, ever, seen)	963
2	(I, saw, movie)	626
3	(I, think, I)	504
4	(I, know, I)	436
5	(I, first, saw)	426
6	(I, would, recommend)	417
7	(I, highly, recommend)	412
8	(I, must, say)	408
9	(I, can't, believe)	377

Out[86]: <AxesSubplot:xlabel='count', ylabel='word'>

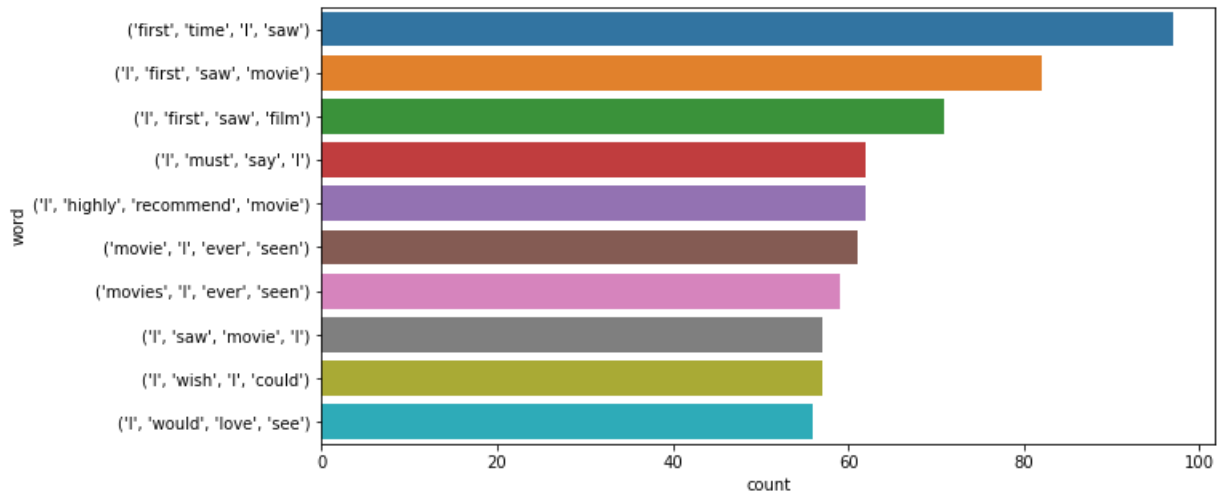


In [87]:
```
1 plot_ngram(tokens_positive, 4)
```

executed in 9.32s, finished 13:43:42 2021-06-21

	word	count
0	(first, time, I, saw)	97
1	(I, first, saw, movie)	82
2	(I, first, saw, film)	71
3	(I, must, say, I)	62
4	(I, highly, recommend, movie)	62
5	(movie, I, ever, seen)	61
6	(movies, I, ever, seen)	59
7	(I, saw, movie, I)	57
8	(I, wish, I, could)	57
9	(I, would, love, see)	56

Out[87]: <AxesSubplot:xlabel='count', ylabel='word'>



In [88]:
```
1 tokens
2
```

executed in 29ms, finished 13:43:42 2021-06-21

Out[88]:
```
['One',
 'reviewers',
 'mentioned',
 'watching',
 'Oz',
 'episode',
 'hooked',
 'They',
 'right',
 'exactly',
 'happened',
 'The',
 'first',
 'thing',
 'struck',
 'Oz',
 'brutality',
 'unflinching',
 'scenes',
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

5 Modeling

5.1 TF - IDF

In [89]:
```
## set up text preprocessing pipeline

tfidf = TfidfVectorizer(strip_accents='unicode',
                        tokenizer=my_tokenizer,
                        stop_words=stop_words_list
                            )

X = df['review']
y = df['sentiment']

# Do train test split here
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,rand

# Fit both x train and test
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
```

executed in 17.9s, finished 13:44:00 2021-06-21

In [90]:
```
# Logistic Regression with TF-IDF Vectoriser
tfidf_log = LogisticRegression(penalty='l2',C=10)
tfidf_log.fit(X_train_tfidf, y_train)
```

executed in 5.20s, finished 13:44:05 2021-06-21

Out[90]:
```
LogisticRegression(C=10)
```

In [91]:

```
def evaluate_model(model, X_train=X_train, X_test=X_test, y_train=y_train,
                   y_test=y_test, cmap='Greens', normalize='true',
                   classes=None,figsize=(10,4)):

    # Print model accuracy
    print(f'Training Accuracy: {model.score(X_train,y_train):.2%}')
    print(f'Test Accuracy: {model.score(X_test,y_test):.2%}')
    print('')

    # Print classification report
    y_test_predict = model.predict(X_test)
    print(metrics.classification_report(y_test, y_test_predict,
                                        target_names=classes))

    # Plot confusion matrix
    fig,ax = plt.subplots(ncols=2,figsize=figsize)
    metrics.plot_confusion_matrix(model, X_test,y_test,cmap=cmap,
                                  normalize=normalize,display_labels=classes
                                  ax=ax[0])

    #Plot ROC curves
    with sns.axes_style("darkgrid"):
        curve = metrics.plot_roc_curve(model,X_train,y_train,ax=ax[1])
        curve2 = metrics.plot_roc_curve(model,X_test,y_test,ax=ax[1])
        curve.ax_.grid()
        curve.ax_.plot([0,1],[0,1],ls=':')
        fig.tight_layout()
        plt.show()
```

executed in 27ms, finished 13:44:05 2021-06-21

In [92]:
```
1 evaluate_model(tfidf_log, X_train=X_train_tfidf, X_test=X_test_tfidf)
```

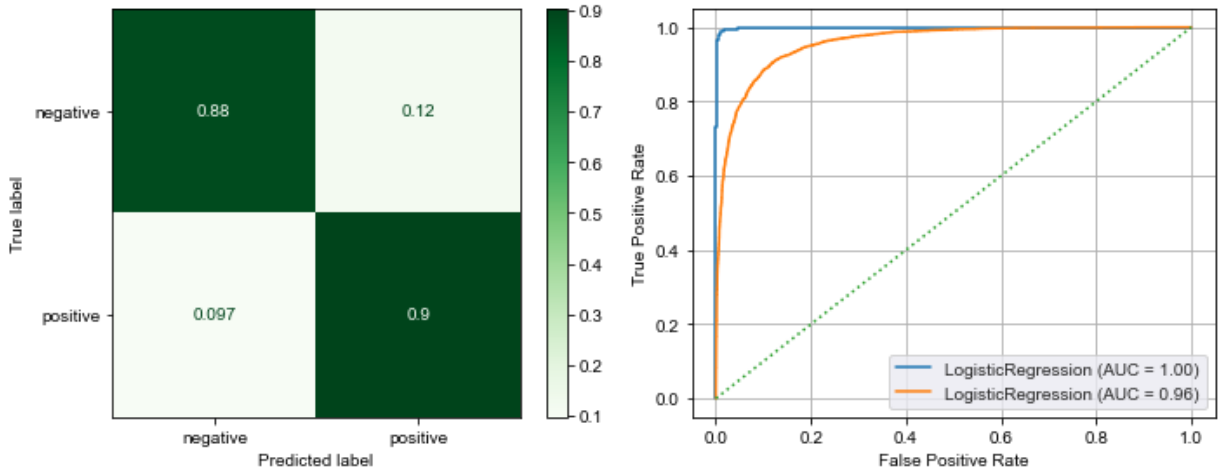
executed in 1.40s, finished 13:44:06 2021-06-21

```
Training Accuracy: 98.81%
Test Accuracy: 89.27%

              precision    recall  f1-score   support

    negative       0.90      0.88      0.89      7422
    positive       0.88      0.90      0.89      7453

    accuracy                           0.89     14875
   macro avg       0.89      0.89      0.89     14875
weighted avg       0.89      0.89      0.89     14875
```



In [93]:
```
feature_names = tfidf.get_feature_names()
feature_names

df_coef = pd.DataFrame()

df_coef['features'] = feature_names
df_coef['coefficients'] = tfidf_log.coef_.flatten()

df_coef.sort_values(by='coefficients', ascending=False)
```

executed in 235ms, finished 13:44:07 2021-06-21

Out[93]:

	features	coefficients
27700	excellent	11.641220
34870	great	11.243282
61781	perfect	10.045060
38009	hilarious	10.004170
2309	amazing	9.765448
...	...	...
82963	terrible	-10.316837
9456	boring	-11.348241
5218	awful	-13.200745
90478	waste	-15.084933
92413	worst	-18.461313

93909 rows × 2 columns

5.2 Model Iteration

In [94]:
```
run = False

# Initiate new model and perform grid search
tfidf_log_hp = LogisticRegression(random_state=8)


if run == True:

# Define lists of parameters to compare
    params = {'C':[0.01,0.1,1,10,100],
             'penalty':['l1','l2','elastic_net'],
             'solver':["liblinear", "newton-cg", "lbfgs", "sag","saga"]
              }

else:
    params = {'C':[100],
             'penalty':['l2'],
             'solver':["newton-cg"]
              }

# 'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'

# Run the grid search with a focus on accuracy
log_grid_search = GridSearchCV(tfidf_log_hp,params,scoring='accuracy',
                               verbose=100,
                               n_jobs=-1)

# Fit grid search to training data and display best parameters
log_grid_search.fit(X_train_tfidf, y_train)

# Print best parameters
log_grid_search.best_params_
```

executed in 23.9s, finished 13:44:31 2021-06-21

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   13.7s
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:   14.2s remaining:   21.3
s
[Parallel(n_jobs=-1)]: Done   3 out of   5 | elapsed:   14.7s remaining:    9.8
s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   18.3s remaining:    0.0
s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   18.3s finished
```

Out[94]:
```
{'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
```

In [95]:
```
# 'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'
```

executed in 12ms, finished 13:44:31 2021-06-21

In [96]:
```
log_grid_search.best_score_
```

executed in 13ms, finished 13:44:31 2021-06-21

Out[96]:
```
0.8879478226386113
```

In [97]:
```
1 evaluate_model(log_grid_search.best_estimator_, X_train=X_train_tfidf, X_tes
```

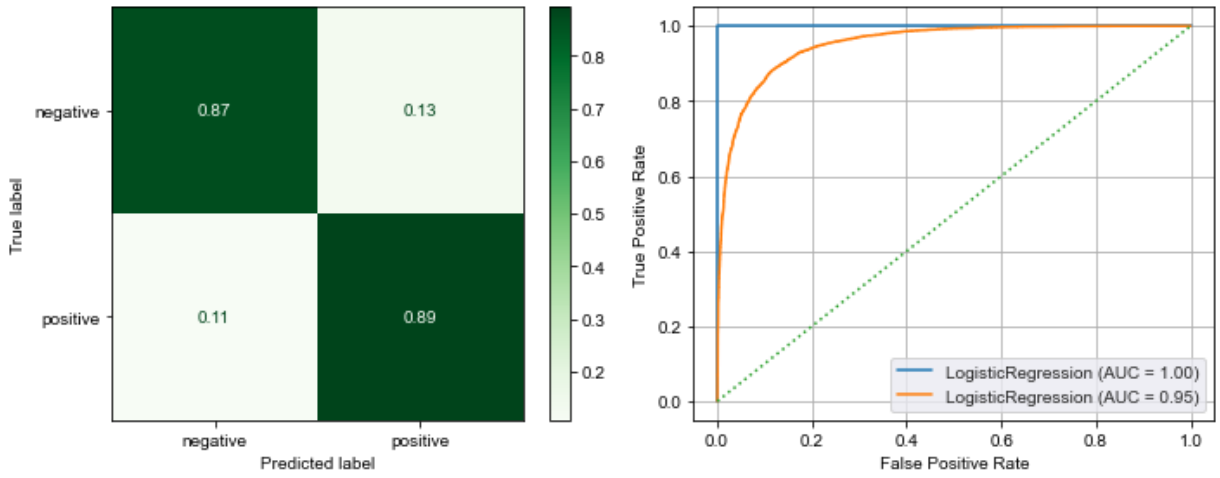
executed in 1.43s, finished 13:44:32 2021-06-21

```
Training Accuracy: 100.00%
Test Accuracy: 88.16%

              precision    recall  f1-score   support

    negative       0.89      0.87      0.88      7422
    positive       0.87      0.89      0.88      7453

    accuracy                           0.88     14875
   macro avg       0.88      0.88      0.88     14875
weighted avg       0.88      0.88      0.88     14875
```



In [ ]:
```
1
```

In [ ]:
```
1
```

5.3 Count Vectorizer

In [98]:
```
## set up text preprocessing pipeline

from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(strip_accents='unicode',
                                tokenizer=my_tokenizer,

                               stop_words=stop_words_list
                              )

# X = df['review']
# y = df['sentiment']

# X

# # Do train test split here
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,ra

# Fit both x train and test
X_train_cv = cv.fit_transform(X_train)
X_test_cv = cv.transform(X_test)



```
executed in 17.5s, finished 13:44:50 2021-06-21

In [99]:
```
# X_train_cv
```
executed in 14ms, finished 13:44:50 2021-06-21

In [100]:
```
# Logistic Regression with Count Vectoriser
cv_log = LogisticRegression(penalty='l2',C=10)
cv_log.fit(X_train_cv, y_train)
```
executed in 5.64s, finished 13:44:55 2021-06-21

Out[100]:
```
LogisticRegression(C=10)
```

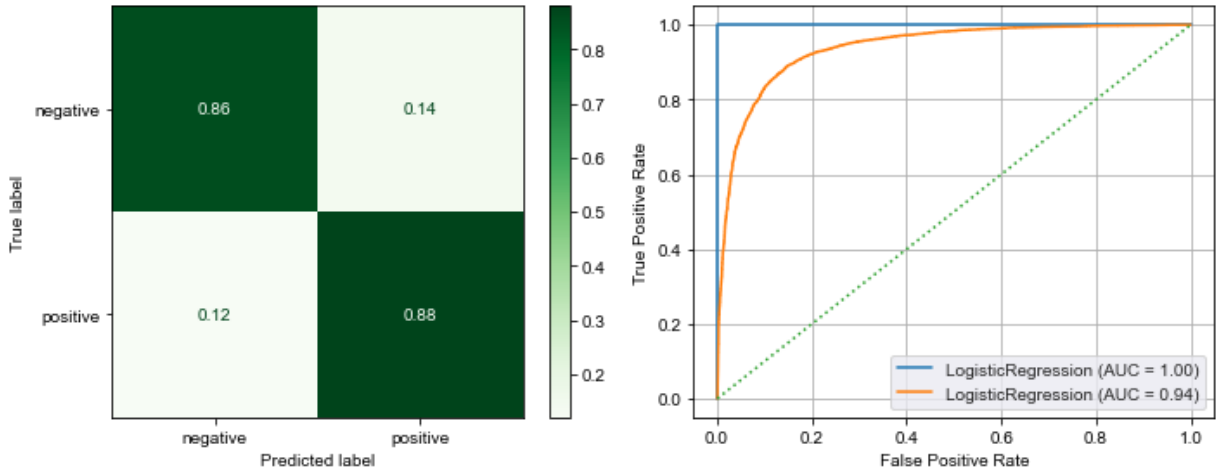
In [101]:
```
1 evaluate_model(cv_log, X_train=X_train_cv, X_test=X_test_cv)
```
executed in 1.61s, finished 13:44:57 2021-06-21

```
Training Accuracy: 99.99%
Test Accuracy: 86.98%

              precision    recall  f1-score   support

    negative       0.88      0.86      0.87      7422
    positive       0.86      0.88      0.87      7453

    accuracy                           0.87     14875
   macro avg       0.87      0.87      0.87     14875
weighted avg       0.87      0.87      0.87     14875
```



5.4 XGBoost

In [ ]:
```
1 model_xgb_tf = XGBClassifier() #Default XGBoost Model with TF_IDF
2 model_xgb_tf.fit(X_train_tfidf, y_train)
3
4 evaluate_model(model_xgb_tf, X_train=X_train_tfidf, X_test=X_test_tfidf)
```
executed in 1m 15.7s, finished 11:00:04 2021-06-21

In [ ]:
```
1
```

In [ ]:
```
1 model_xgb_cv = XGBClassifier() #Default XGBoost Model with TF_IDF
2 model_xgb_cv.fit(X_train_tfidf, y_train)
3
4 evaluate_model(model_xgb_cv, X_train=X_train_cv, X_test=X_test_cv)
```
executed in 1m 12.9s, finished 11:01:17 2021-06-21

In [ ]:

```
# Create compare model function

def model_compare(base_model, grid_search_model):

    # Calculate accuracies
    base_score = base_model.score(X_test, y_test)
    grid_score = grid_search_model.score(X_test, y_test)

    #Print accuracies
    print("--- Base Model ---")
    model_accuracy(base_model)
    print('')
    print("--- Grid Search Model ---")
    model_accuracy(grid_search_model)
    print('')

    # If/else function to display best model and score improvement
    if base_score < grid_score:
        print(f'Our grid search model outperformed our base model by {(grid_
    else:
        print(f'Our base model outperformed our grid search model by {(base_

# model_compare(model_log, log_grid_search.best_estimator_)
```

executed in 12ms, finished 11:01:17 2021-06-21

In [ ]:
```
# If run = True, code will perform full grid search
# If run = False, code will use previously calculated best parameters
run = False

# Instantiate new model for hyperparameter tuning
model_xgb_hp = XGBClassifier(random_state=8)

# Define grid search parameters
if run == True:
    param_grid = {
        'learning_rate': [0.0001, 0.001, 0.01, 0.1],
        'max_depth': [3, 5, 7, 9],
        'min_child_weight': [1, 2],
        'subsample': [0.5, 0.7, 1],
        'n_estimators': [10, 100, 1000]}
else:
    param_grid = {
        'learning_rate': [0.01],
        'max_depth': [5],
        'min_child_weight': [2],
        'subsample': [0.5],
        'n_estimators': [100]}

# Create grid search and train
xgb_grid_search = GridSearchCV(model_xgb_hp, param_grid, scoring='accuracy',
                               cv=5, n_jobs=-1, verbose=100)
xgb_grid_search.fit(X_train_tfidf, y_train)

# # Print metrics
# model_compare(model_xgb_tf, xgb_grid_search.best_estimator_)
# print("")
# print(f"Cross Validated Score: {xgb_grid_search.best_score_ :.2%}")
# print("")
# print(f"Optimal Parameters: {xgb_grid_search.best_params_}")
```

executed in 3m 37s, finished 11:04:54 2021-06-21

In [ ]:
```
print("")
print(f"Cross Validated Score: {xgb_grid_search.best_score_ :.2%}")
print("")
print(f"Optimal Parameters: {xgb_grid_search.best_params_}")
```

executed in 13ms, finished 11:04:54 2021-06-21

In [ ]:
```
# model_compare(model_xgb_tf, xgb_grid_search.best_estimator_)
# print("")
# print(f"Cross Validated Score: {xgb_grid_search.best_score_ :.2%}")
# print("")
# print(f"Optimal Parameters: {xgb_grid_search.best_params_}")
```

executed in 15ms, finished 11:06:26 2021-06-21

In [ ]:
```
X_train_tfidf
```

executed in 10ms, finished 11:06:26 2021-06-21

In [ ]:
```
1
```

6 Word analysis

6.1 Frequency Distribution

6.2 Normalized word frequency

6.3 Bigrams

6.4 Mutual information scores

6.4.1 TF-IDF vectorization

Visualize Vector

6.4.2 bag of words

In [ ]:
```
1
```

In [ ]:
```
1
```

6.5 Models

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

7 Scratch

7.1 Random Forest

In [ ]:
```
# Initiate a random forest model
model_rf = RandomForestClassifier(random_state=8)
model_rf.fit(X_train_tfidf, y_train)

# model_accuracy(model_rf)
```

In [ ]:
```
evaluate_model(model_rf, X_train=X_train_tfidf, X_test=X_test_tfidf)
```

In [ ]:
```
# params = {
#     'n_estimators': [10, 50, 100, 150],
#     'max_depth': [10, 20, 50, None],
#     'min_samples_split': [5, 30]
# }

# clf_rf = RandomForestClassifier()
# gs_rf = GridSearchCV(clf_rf, param_grid=params, scoring='f1_macro', cv=3,
# gs_rf.fit(X_train_tfidf, y_train)
```

In [ ]:
```
# gs_rf.best_score_
```

In [ ]:
```
# gs_rf.best_params_
```

In [ ]:
```

```

7.2 LSTM

In [ ]:
```
embedding_vecor_length = 32
callback = EarlyStopping(monitor='val_loss', patience=2)

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_le
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=['accura

print(model.summary())
```

In [ ]:
```
model.fit(X_train1, y_train, epochs=10, batch_size=256,verbose = 1,validatio
```

In [ ]:
```
accuracy_score(y_test, model.predict_classes(X_test1))
```

In [ ]:
```
plot_history(history)
```

In [ ]:
```

```

In [ ]:

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

sentence_1="This is a good job.I will not miss it for anything"
sentence_2="This is not good at all"



#without smooth IDF
print("Without Smoothing:")
#define tf-idf
tf_idf_vec = TfidfVectorizer(use_idf=True,
                         smooth_idf=False,
                         ngram_range=(1,1),stop_words='english') # to use onl
#transform
tf_idf_data = tf_idf_vec.fit_transform([sentence_1,sentence_2])

#create dataframe
tf_idf_dataframe=pd.DataFrame(tf_idf_data.toarray(),columns=tf_idf_vec.get_f
print(tf_idf_dataframe)
print("\n")

#with smooth
tf_idf_vec_smooth = TfidfVectorizer(use_idf=True,
                         smooth_idf=True,
                         ngram_range=(1,1),stop_words='english')


tf_idf_data_smooth = tf_idf_vec_smooth.fit_transform([sentence_1,sentence_2]

print("With Smoothing:")
tf_idf_dataframe_smooth=pd.DataFrame(tf_idf_data_smooth.toarray(),columns=tf
print(tf_idf_dataframe_smooth)

```

In [ ]:
```
def plot_coefficients(classifier, feature_names, top_features=20):

    # Access the coefficients from classifier
    coef = classifier.coef_

    # Access the classes
    classes = classifier.classes_

    # Iterate the loop for number of classes
    for i in range(len(classes)):

        i = 0
        print(classes[i])

        # Access the row containing the coefficients for this class
        class_coef = coef[i]


        # Below this, I have just replaced 'i' in your code with 'class_coef
        # Pass this to get top and bottom features
        top_positive_coefficients = np.argsort(class_coef)[-top_features:]
        top_negative_coefficients = np.argsort(class_coef)[:top_features]

        # Concatenate the above two
        top_coefficients = np.hstack([top_negative_coefficients,
                                      top_positive_coefficients])
        # create plot
        plt.figure(figsize=(10, 3))

        colors = ["red" if c < 0 else "blue" for c in class_coef[top_coeffic
        plt.bar(np.arange(2 * top_features), class_coef[top_coefficients], c
        feature_names = np.array(feature_names)

        # Here I corrected the start to 0 (Your code has 1, which shifted th
        plt.xticks(np.arange(1, 1 + 2 * top_features),
                   feature_names[top_coefficients], rotation=60, ha="right")
        plt.show()

plot_coefficients(tfidf_log, tfidf.get_feature_names(), top_features=20)
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

7.2.1 25 most common words in positive and negative reviews

In [ ]:
```
1
2 # ax = neg_freq_df.set_index('Word').sort_values('Frequency').plot(kind='bar
3 # ax.set(title="25 Most Common Words in Negative Tweets")
```
executed in 11ms, finished 10:54:37 2021-06-21

7.2.2 Word clouds pos / neg

7.2.3 visualize TSNE

7.2.4 mean word length postive or negative?. bucket and histogram?

7.2.5 Same with word unique percentage

Sentiment across word count

Word count histogram

In [ ]:
```
1 df
```
executed in 56ms, finished 10:54:37 2021-06-21

In [ ]:

```
run = False

if run == True:

    ## Indirect features
    eng_stopwords = set(stopwords.words("english"))

    df['count_sent']=df["review"].apply(lambda x: len(re.findall("\n",str(x)
    #Word count in each comment:
    df['count_word']=df["review"].apply(lambda x: len(str(x).split()))
    #Unique word count
    df['count_unique_word']=df["review"].apply(lambda x: len(set(str(x).spli
    #Letter count
    df['count_letters']=df["review"].apply(lambda x: len(str(x)))
    #punctuation count
    df["count_punctuations"] =df["review"].apply(lambda x: len([c for c in s
    #upper case words count
    df["count_words_upper"] = df["review"].apply(lambda x: len([w for w in s
    #title case words count
    df["count_words_title"] = df["review"].apply(lambda x: len([w for w in s
    #Number of stopwords
    df["count_stopwords"] = df["review"].apply(lambda x: len([w for w in str
    #Average length of the words
    df["mean_word_len"] = df["review"].apply(lambda x: np.mean([len(w) for w
    #Word count percent in each comment:
    df['word_unique_percent']=df['count_unique_word']*100/df['count_word']
    #Punct percent in each comment:
    df['punct_percent']=df['count_punctuations']*100/df['count_word']
    #derived features
    #Word count percent in each comment:
    df['word_unique_percent']=df['count_unique_word']*100/df['count_word']
    #derived features
    #Punct percent in each comment:
    df['punct_percent']=df['count_punctuations']*100/df['count_word']

    df.to_json('data/df_json_store.json')

else:
     df = pd.read_json('data/df_json_store.json')

df.head(2)
```

executed in 1.58s, finished 10:54:37 2021-06-21

7.3 Neural Networks

In [ ]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.callbacks import ModelCheckpoint
from keras.layers import LSTM,Dropout
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.layers import LSTM, Conv1D, MaxPooling1D, Dropout
from keras.callbacks import EarlyStopping
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.metrics import classification_report,confusion_matrix,accuracy_
from keras.wrappers.scikit_learn import KerasClassifier
import string
from keras.preprocessing import text, sequence
from keras import layers, models, optimizers
```

executed in 6.98s, finished 11:06:35 2021-06-21

In [ ]:
```
def plot_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    x = range(1, len(acc) + 1)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(x, acc, 'b', label='Training acc')
    plt.plot(x, val_acc, 'r', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(x, loss, 'b', label='Training loss')
    plt.plot(x, val_loss, 'r', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()
```

executed in 26ms, finished 11:06:35 2021-06-21

In [ ]:
```
df['sentiment'] = df['sentiment'].replace('positive',1)
df['sentiment'] = df['sentiment'].replace('negative',0)
```

executed in 58ms, finished 11:06:35 2021-06-21

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(df['reviews_t'], df['sen

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,test_
```

executed in 43ms, finished 11:06:35 2021-06-21

In [ ]:
```
[x.shape for x in [X_train,X_valid,X_test]]
```

executed in 13ms, finished 11:06:35 2021-06-21

In [ ]:
```
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(df['reviews_t'])

X_train1 = tokenizer.texts_to_sequences(X_train)
X_valid1 = tokenizer.texts_to_sequences(X_valid)
X_test1 = tokenizer.texts_to_sequences(X_test)

vocab_size = len(tokenizer.word_index) + 1  # Adding 1 because of reserved 0

print(X_train[2])
print(X_train1[2])
```
executed in 12.0s, finished 11:06:47 2021-06-21

In [ ]:
```
seq_lens = [len(s) for s in X_train1]
print("average length: %0.1f" % np.mean(seq_lens))
print("max length: %d" % max(seq_lens))
```
executed in 25ms, finished 11:06:47 2021-06-21

In [ ]:
```
maxlen = 150

X_train1 = pad_sequences(X_train1, padding='post', maxlen=maxlen)
X_valid1 = pad_sequences(X_valid1, padding='post', maxlen=maxlen)
X_test1 = pad_sequences(X_test1, padding='post', maxlen=maxlen)

print(X_train1[2, :])
```
executed in 784ms, finished 11:06:48 2021-06-21

In [ ]:
```
vocab_size
```
executed in 12ms, finished 11:06:48 2021-06-21

In [ ]:
```
embedding_dim = 50
callback = EarlyStopping(monitor='val_loss', patience=2)

model = Sequential()
model.add(layers.Embedding(input_dim=vocab_size, output_dim=embedding_dim, i
model.add(layers.Flatten())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy

model.summary()
```
executed in 250ms, finished 11:06:48 2021-06-21

In [ ]:
```
y_train
```
executed in 14ms, finished 11:06:48 2021-06-21

In [ ]:
```
history = model.fit(X_train1, y_train,epochs=10,verbose=True,validation_data
```
executed in 26.5s, finished 11:07:14 2021-06-21

In [ ]:
```
1 accuracy_score(y_test, model.predict_classes(X_test1))
```
executed in 547ms, finished 11:07:15 2021-06-21

In [ ]:
```
1 plot_history(history)
```
executed in 1.07s, finished 11:07:16 2021-06-21

7.4 CNN Model

In [ ]:
```
1  embedding_vecor_length = 32
2  callback = EarlyStopping(monitor='val_loss', patience=2)
3
4  model = Sequential()
5  model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_le
6  model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu
7  model.add(MaxPooling1D(pool_size=2))
8  model.add(LSTM(100))
9  model.add(Dense(1, activation='sigmoid'))
10
11 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accura
12
13 print(model.summary())
```
executed in 617ms, finished 11:07:17 2021-06-21

In [ ]:
```
1 model.fit(X_train1, y_train, epochs=10, batch_size=256,verbose = 1,validatio
```
executed in 4m 9s, finished 11:11:26 2021-06-21

In [ ]:
```
1 accuracy_score(y_test, model.predict_classes(X_test1))
```
executed in 5.61s, finished 11:11:31 2021-06-21

In [ ]:
```
1 plot_history(history)
```
executed in 584ms, finished 11:11:32 2021-06-21

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```