MovieReviewSentimentAnalysis

February 20, 2024

[36]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from matplotlib import style
style.use('ggplot')
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
```

[5]:
```
df = pd.read_csv('IMDB Dataset.csv')
```

[6]:
```
df.head()
```

[6]:
```
                                              review sentiment
0  One of the other reviewers has mentioned that …  positive
1  A wonderful little production. <br /><br />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
```

[7]:
```
df.shape
```

[7]:
```
(50000, 2)
```

[8]:
```
df.info()
```

```
<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
```

[9]:
```
sns.countplot(x='sentiment',data=df)
plt.title("Sentiment Distribution")
```

[9]: Text(0.5, 1.0, 'Sentiment Distribution')

[10]:
```
for i in range(5):
    print("Review: ", [i])
    print(df['review'].iloc[i], "\n")
    print("Sentiment: ",df['sentiment'].iloc[i], "\n\n")
```

```
Review:  [0]
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
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.

It is called OZ as that is the nickname given to the Oswald Maximum Security State Penitentary. It focuses mainly on Emerald City, an experimental section of the prison where all the cells have glass fronts and face inwards, so privacy is not high on the agenda. Em City is home to many..Aryans, Muslims, gangstas, Latinos, Christians, Italians, Irish and more…so scuffles, death stares, dodgy dealings and shady agreements are never far away.

I would say the main appeal of the show is due to the fact that it goes where other shows wouldn't dare. Forget pretty pictures painted for mainstream audiences, forget charm, forget romance…OZ doesn't mess around. The first episode I ever saw struck me as so nasty it was surreal, I couldn't say I was ready for it, but as I watched more, I developed a taste for Oz, and got accustomed to the high levels of graphic violence. Not just violence, but injustice (crooked guards who'll be sold out for a nickel, inmates who'll kill on order and get away with it, well mannered, middle class inmates being turned into prison bitches due to their lack of street skills or prison experience) Watching Oz, you may become comfortable with what is uncomfortable viewing…thats if you can get in touch with your darker side.

Sentiment: positive

Review: [1]
A wonderful little production.

The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire piece.

The actors are extremely well chosen- Michael Sheen not only "has got all the polari" but he has all the voices down pat too! You can truly see the seamless editing guided by the references to Williams' diary entries, not only is it well worth the watching but it is a terrificly written and performed piece. A masterful production about one of the great master's of comedy and his life.

The realism really comes home with the little things: the fantasy of the guard which, rather than use the traditional 'dream' techniques remains solid then disappears. It plays on our knowledge and our senses, particularly with the scenes concerning Orton and Halliwell and the sets (particularly of their flat with Halliwell's murals decorating every surface) are terribly well done.

Sentiment: positive

Review: [2]
I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a light-hearted comedy. The plot is simplistic, but the dialogue is witty and the characters are likable (even the well bread suspected serial killer). While some may be disappointed when they realize this is not Match Point 2: Risk Addiction, I thought it was proof that Woody Allen is still fully in control of the style many of us have grown to love.

This was the most I'd laughed at one of Woody's

comedies in years (dare I say a decade?). While I've never been impressed with Scarlet Johanson, in this she managed to tone down her "sexy" image and jumped right into a average, but spirited young woman.

This may not be the crown jewel of his career, but it was wittier than "Devil Wears Prada" and more interesting than "Superman" a great comedy to go see with friends.

Sentiment: positive

Review: [3]
Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fighting all the time.

This movie is slower than a soap opera… and suddenly, Jake decides to become Rambo and kill the zombie.

OK, first of all when you're going to make a film you must Decide if its a thriller or a drama! As a drama the movie is watchable. Parents are divorcing & arguing like in real life. And then we have Jake with his closet which totally ruins all the film! I expected to see a BOOGEYMAN similar movie, and instead i watched a drama with some meaningless thriller spots.

3 out of 10 just for the well playing parents & descent dialogs. As for the shots with Jake: just ignore them.

Sentiment: negative

Review: [4]
Petter Mattei's "Love in the Time of Money" is a visually stunning film to watch. Mr. Mattei offers us a vivid portrait about human relations. This is a movie that seems to be telling us what money, power and success do to people in the different situations we encounter.

This being a variation on the Arthur Schnitzler's play about the same theme, the director transfers the action to the present time New York where all these different characters meet and connect. Each one is connected in one way, or another to the next person, but no one seems to know the previous point of contact. Stylishly, the film has a sophisticated luxurious look. We are taken to see how these people live and the world they live in their own habitat.

The only thing one gets out of all these souls in the picture is the different stages of loneliness each one inhabits. A big city is not exactly the best place in which human relations find sincere fulfillment, as one discerns is the case with most of the people we encounter.

The acting is good under Mr. Mattei's direction. Steve Buscemi, Rosario Dawson, Carol Kane, Michael Imperioli, Adrian Grenier, and the rest of the talented cast, make these characters come alive.

We wish Mr. Mattei good luck and await anxiously for his next work.

Sentiment: positive

[11]:
```python
def no_of_words(text):
    words = text.split()
    word_count = len(words)
    return word_count
```

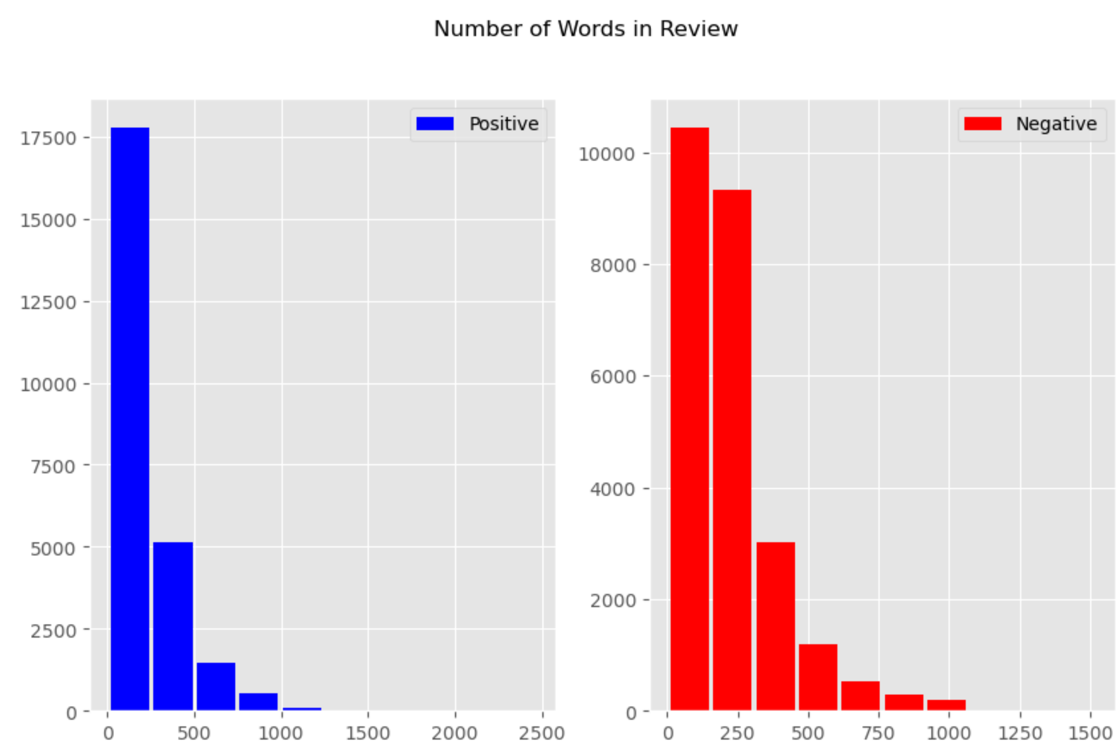
[12]:
```python
df['word count'] = df['review'].apply(no_of_words)
```

[13]:
```python
df.head()
```

[13]:
```
                                              review sentiment  word count
0  One of the other reviewers has mentioned that …  positive         307
1  A wonderful little production. <br /><br />The…  positive         162
2  I thought this was a wonderful way to spend ti…  positive         166
3  Basically there's a family where a little boy …  negative         138
4  Petter Mattei's "Love in the Time of Money" is…  positive         230
```

[14]:
```python
fig, ax = plt.subplots(1,2, figsize = (10,6))
ax[0].hist(df[df['sentiment'] == 'positive']['word count'], label = 'Positive',␣
 ↪color='blue', rwidth=0.9);
ax[0].legend(loc='upper right');
ax[1].hist(df[df['sentiment'] == 'negative']['word count'], label = 'Negative',␣
 ↪color='red', rwidth=0.9);
ax[1].legend(loc='upper right');
fig.suptitle("Number of Words in Review")
```

[14]:
```
Text(0.5, 0.98, 'Number of Words in Review')
```



[18]:
```
df.sentiment.replace("positive",1,inplace=True)
df.sentiment.replace("negative",0,inplace=True)
```

[19]:
```
df.head()
```

[19]:

	review	sentiment	word count
0	One of the other reviewers has mentioned that …	1	307
1	A wonderful little production. The…	1	162
2	I thought this was a wonderful way to spend ti…	1	166
3	Basically there's a family where a little boy …	0	138
4	Petter Mattei's "Love in the Time of Money" is…	1	230

[37]:
```
def data_processing(text):
    text = text.lower()
    text = re.sub('<br />','',text)
    text = re.sub(r'https\S+|http\S+|www\S+','',text,flags = re.MULTILINE)
    text = re.sub(r'\@\w+|\#','',text)
    text = re.sub(r'[^\w\s]','',text)
    text_tokens = word_tokenize(text)
    filtered_text = [w for w in text_tokens if not w in stop_words]
    return " ".join(filtered_text)
```

[38]:
```
df.review = df['review'].apply(data_processing)
```

[40]:
```
duplicated_count = df.duplicated().sum()
print("Number of duplicated entries: ", duplicated_count)
```

```
Number of duplicated entries:  421
```

[41]:
```
df = df.drop_duplicates('review')
```

[42]:
```
stemmer = PorterStemmer()
def stemming(data):
    text = [stemmer.stem(word) for word in data]
    return data
```

[43]:
```
df.review = df['review'].apply(stemming)
```

```
/tmp/ipykernel_284/1414651603.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.review = df['review'].apply(stemming)
```

[46]:
```
df['word count'] = df['review'].apply(no_of_words)
```

```
/tmp/ipykernel_284/2135123054.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['word count'] = df['review'].apply(no_of_words)
```

[51]:
```
df.head()
```

[51]:

	review	sentiment	word count
0	one reviewers mentioned watching 1 oz episode …	1	168
1	wonderful little production filming technique …	1	84
2	thought wonderful way spend time hot summer we…	1	86
3	basically theres family little boy jake thinks…	0	67
4	petter matteis love time money visually stunni…	1	125

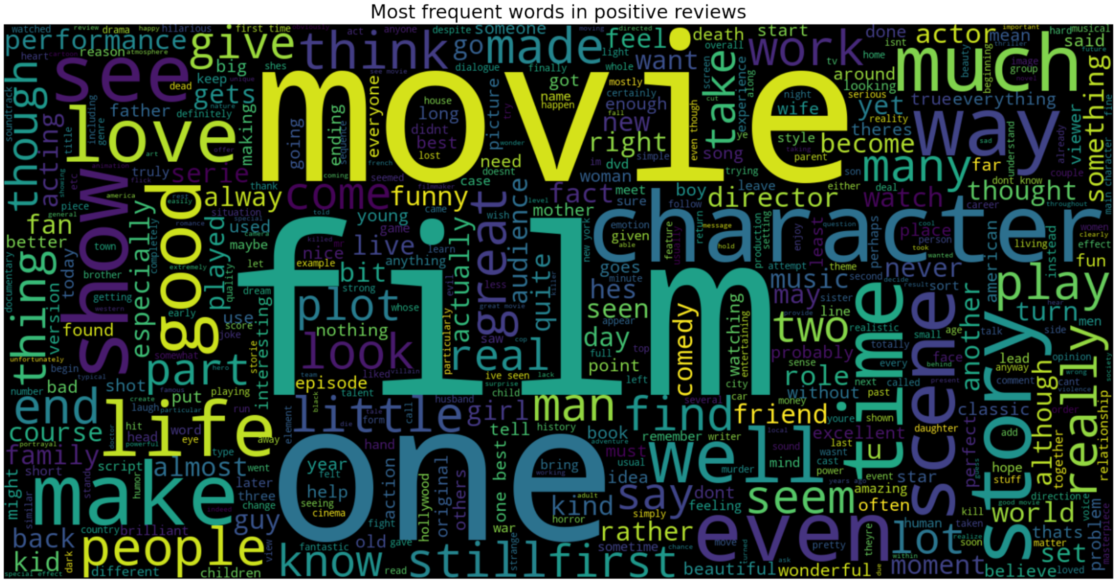
[52]:
```
pos_reviews = df[df.sentiment == 1]
pos_reviews.head()
```

[52]:

	review	sentiment	word count
0	one reviewers mentioned watching 1 oz episode …	1	168
1	wonderful little production filming technique …	1	84
2	thought wonderful way spend time hot summer we…	1	86
4	petter matteis love time money visually stunni…	1	125
5	probably alltime favorite movie story selfless…	1	58

[54]:
```
text = ' '.join([word for word in pos_reviews['review']])
plt.figure(figsize = (20,15), facecolor='None')
wordcloud = WordCloud(max_words=500, width = 1600, height=800).generate(text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Most frequent words in positive reviews', fontsize = 19)
plt.show()
```

Most frequent words in positive reviews




```python
[56]: from collections import Counter
count = Counter()
for text in pos_reviews['review'].values:
    for word in text.split():
        count[word] += 1
count.most_common(15)
```

```
[56]: [('film', 39285),
 ('movie', 35830),
 ('one', 25621),
 ('like', 16998),
 ('good', 14281),
 ('great', 12568),
 ('story', 12338),
 ('see', 11814),
 ('time', 11724),
 ('well', 10930),
 ('really', 10638),
 ('also', 10516),
 ('would', 10320),
 ('even', 9318),
 ('much', 8971)]
```

```python
[60]: pos_words = pd.DataFrame(count.most_common(15))
pos_words.columns = ['word','count']
pos_words.head()
```

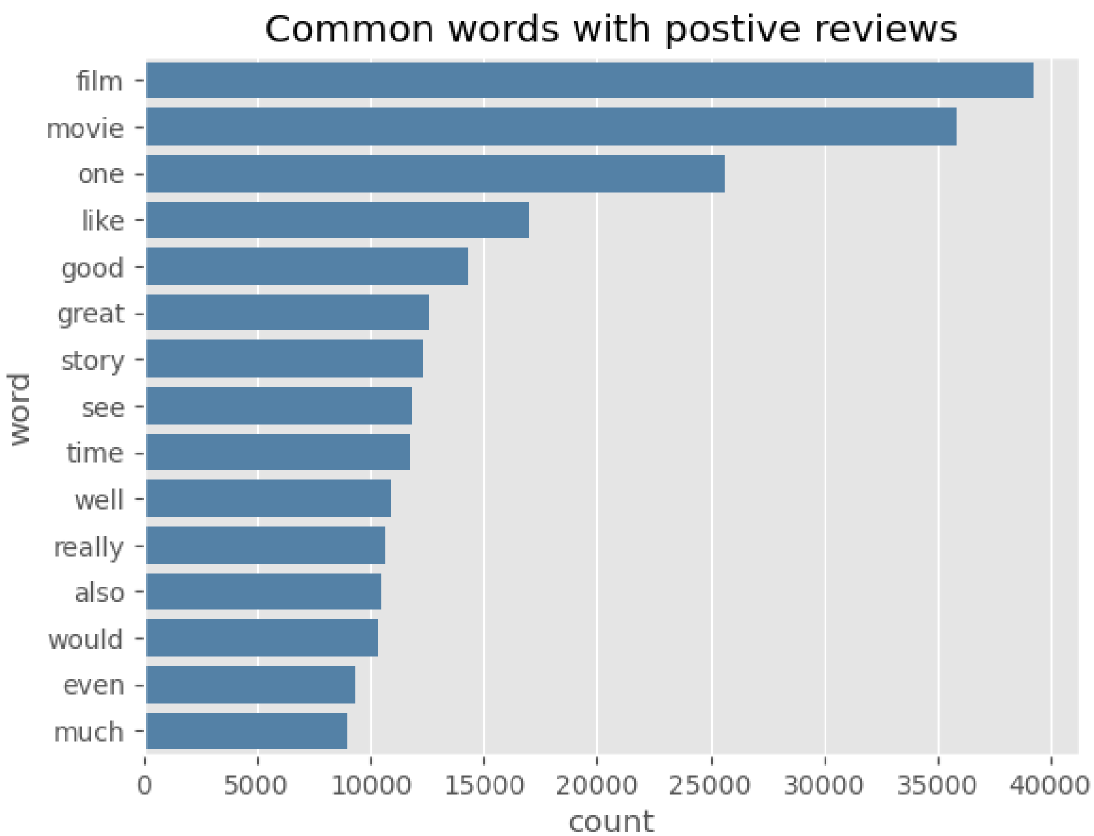
[60]:

	word	count
0	film	39285
1	movie	35830
2	one	25621
3	like	16998
4	good	14281

[69]:

```
sns.barplot(data=pos_words, x='count', y='word',  color = 'steelblue')
plt.title('Common words with postive reviews')
```

[69]: Text(0.5, 1.0, 'Common words with postive reviews')



[70]:

```
neg_reviews = df[df.sentiment == 0]
neg_reviews.head()
```

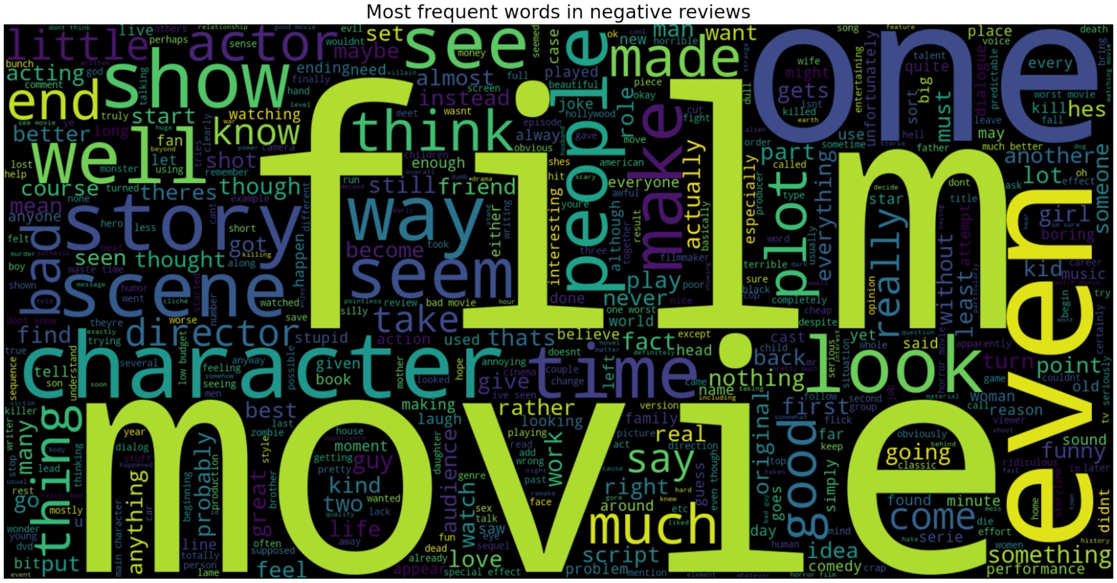
[70]:

	review	sentiment	word count
3	basically theres family little boy jake thinks…	0	67
7	show amazing fresh innovative idea 70s first a…	0	83
8	encouraged positive comments film looking forw…	0	64
10	phil alien one quirky films humour based aroun…	0	51

```
11  saw movie 12 came recall scariest scene big bi…          0          84
```

[71]:
```python
text = ' '.join([word for word in neg_reviews['review']])
plt.figure(figsize = (20,15), facecolor='None')
wordcloud = WordCloud(max_words=500, width = 1600, height=800).generate(text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Most frequent words in negative reviews', fontsize = 19)
plt.show()
```

Most frequent words in negative reviews



[72]:
```python
count = Counter()
for text in neg_reviews['review'].values:
    for word in text.split():
        count[word] += 1
count.most_common(15)
```

[72]:
```
[('movie', 47001),
 ('film', 34651),
 ('one', 24361),
 ('like', 21508),
 ('even', 14759),
 ('good', 13995),
 ('bad', 13903),
 ('would', 13482),
 ('really', 12084),
 ('time', 11349),
 ('see', 10412),
 ('dont', 9912),
```

```
 ('get', 9884),
 ('much', 9758),
 ('story', 9563)]
```

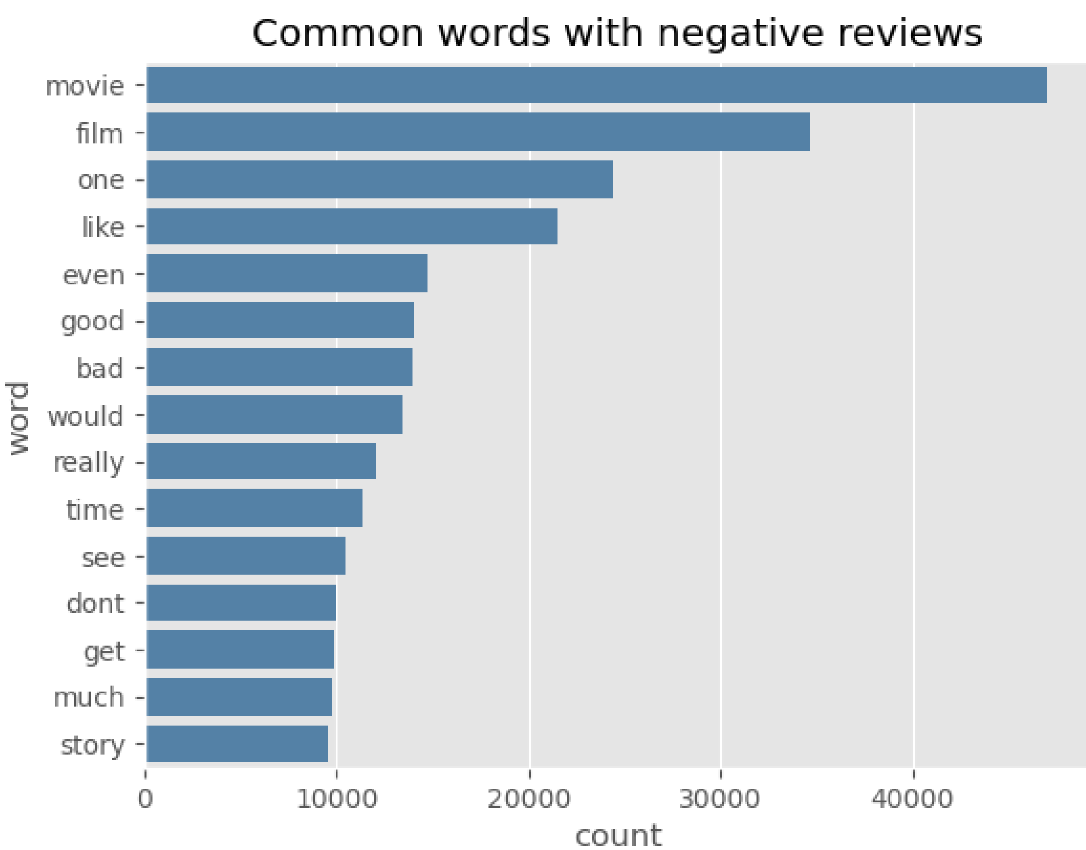
[73]:
```
neg_words = pd.DataFrame(count.most_common(15))
neg_words.columns = ['word','count']
neg_words.head()
```

[73]:

	word	count
0	movie	47001
1	film	34651
2	one	24361
3	like	21508
4	even	14759

[75]:
```
sns.barplot(data=neg_words, x='count', y='word',  color = 'steelblue')
plt.title('Common words with negative reviews')
```

[75]: `Text(0.5, 1.0, 'Common words with negative reviews')`



[76]:
```
X = df['review']
y = df['sentiment']
```

[77]:
```
vect = TfidfVectorizer()
```

[78]:
```
X = vect.fit_transform(df['review'])
```

[79]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
```

[80]:
```
print("Size of X_train: ", (X_train.shape))
print("Size of y_train: ", (y_train.shape))
print("Size of X_test: ", (X_test.shape))
print("Size of y_test: ", (y_test.shape))
```

```
Size of X_train:  (34704, 221700)
Size of y_train:  (34704,)
Size of X_test:  (14874, 221700)
Size of y_test:  (14874,)
```

[ ]:

[ ]:

[81]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix
import warnings
warnings.filterwarnings('ignore')
```

[83]:
```
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
logreg_pred = logreg.predict(X_test)
logreg_acc = accuracy_score(logreg_pred,y_test)
print("Test accuracy: {:.2f}%".format(logreg_acc*100))
```

```
Test accuracy: 89.00%
```

[84]:
```
print(confusion_matrix(y_test,logreg_pred))
print("\n")
print(classification_report(y_test,logreg_pred))
```

```
[[6453  908]
 [ 728 6785]]
```

```
              precision    recall  f1-score   support

           0       0.90      0.88      0.89      7361
           1       0.88      0.90      0.89      7513

    accuracy                           0.89     14874
   macro avg       0.89      0.89      0.89     14874
weighted avg       0.89      0.89      0.89     14874
```

[87]:
```
mnb = MultinomialNB()
mnb.fit(X_train,y_train)
mnb_pred = mnb.predict(X_test)
mnb_acc = accuracy_score(mnb_pred,y_test)
print("Test accuracy: {:.2f}%".format(mnb_acc*100))
```

```
Test accuracy: 86.43%
```

[88]:
```
print(confusion_matrix(y_test,mnb_pred))
print("\n")
print(classification_report(y_test,mnb_pred))
```

```
[[6418  943]
 [1076 6437]]


              precision    recall  f1-score   support

           0       0.86      0.87      0.86      7361
           1       0.87      0.86      0.86      7513

    accuracy                           0.86     14874
   macro avg       0.86      0.86      0.86     14874
weighted avg       0.86      0.86      0.86     14874
```

[89]:
```
svc = LinearSVC()
svc.fit(X_train,y_train)
svc_pred = svc.predict(X_test)
svc_acc = accuracy_score(svc_pred,y_test)
print("Test accuracy: {:.2f}%".format(svc_acc*100))
```

```
Test accuracy: 89.23%
```

[90]:
```
print(confusion_matrix(y_test,svc_pred))
print("\n")
print(classification_report(y_test,svc_pred))
```

```
[[6505  856]
 [ 746 6767]]
```

```
              precision    recall  f1-score   support

           0       0.90      0.88      0.89      7361
           1       0.89      0.90      0.89      7513

    accuracy                           0.89     14874
   macro avg       0.89      0.89      0.89     14874
weighted avg       0.89      0.89      0.89     14874
```

[93]:

```python
from sklearn.model_selection import GridSearchCV
param_grid = {'C':[0.1,1,10,100], 'loss':['hinge','squared_hinge']}
grid = GridSearchCV(svc, param_grid, refit = True, verbose = 3)
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[CV 1/5] END …C=0.1, loss=hinge;, score=0.872 total time=   0.3s
[CV 2/5] END …C=0.1, loss=hinge;, score=0.875 total time=   0.2s
[CV 3/5] END …C=0.1, loss=hinge;, score=0.871 total time=   0.2s
[CV 4/5] END …C=0.1, loss=hinge;, score=0.878 total time=   0.3s
[CV 5/5] END …C=0.1, loss=hinge;, score=0.874 total time=   0.3s
[CV 1/5] END …C=0.1, loss=squared_hinge;, score=0.892 total time=   0.4s
[CV 2/5] END …C=0.1, loss=squared_hinge;, score=0.895 total time=   0.4s
[CV 3/5] END …C=0.1, loss=squared_hinge;, score=0.888 total time=   0.4s
[CV 4/5] END …C=0.1, loss=squared_hinge;, score=0.894 total time=   0.4s
[CV 5/5] END …C=0.1, loss=squared_hinge;, score=0.890 total time=   0.4s
[CV 1/5] END …C=1, loss=hinge;, score=0.896 total time=   0.7s
[CV 2/5] END …C=1, loss=hinge;, score=0.894 total time=   1.9s
[CV 3/5] END …C=1, loss=hinge;, score=0.892 total time=   0.7s
[CV 4/5] END …C=1, loss=hinge;, score=0.894 total time=   0.7s
[CV 5/5] END …C=1, loss=hinge;, score=0.894 total time=   0.5s
[CV 1/5] END …C=1, loss=squared_hinge;, score=0.892 total time=   0.7s
[CV 2/5] END …C=1, loss=squared_hinge;, score=0.895 total time=   0.7s
[CV 3/5] END …C=1, loss=squared_hinge;, score=0.889 total time=   0.7s
[CV 4/5] END …C=1, loss=squared_hinge;, score=0.896 total time=   0.7s
[CV 5/5] END …C=1, loss=squared_hinge;, score=0.894 total time=   0.7s
[CV 1/5] END …C=10, loss=hinge;, score=0.876 total time=   2.9s
[CV 2/5] END …C=10, loss=hinge;, score=0.882 total time=   9.7s
[CV 3/5] END …C=10, loss=hinge;, score=0.875 total time=  11.1s
[CV 4/5] END …C=10, loss=hinge;, score=0.881 total time=   5.3s
[CV 5/5] END …C=10, loss=hinge;, score=0.878 total time=   5.5s
[CV 1/5] END …C=10, loss=squared_hinge;, score=0.881 total time=   1.8s
[CV 2/5] END …C=10, loss=squared_hinge;, score=0.885 total time=   2.4s
[CV 3/5] END …C=10, loss=squared_hinge;, score=0.879 total time=   2.4s
```

```
[CV 4/5] END …C=10, loss=squared_hinge;, score=0.885 total time=   2.1s
[CV 5/5] END …C=10, loss=squared_hinge;, score=0.883 total time=   2.0s
[CV 1/5] END …C=100, loss=hinge;, score=0.876 total time=   2.9s
[CV 2/5] END …C=100, loss=hinge;, score=0.881 total time=   8.5s
[CV 3/5] END …C=100, loss=hinge;, score=0.874 total time=  10.2s
[CV 4/5] END …C=100, loss=hinge;, score=0.880 total time=   3.8s
[CV 5/5] END …C=100, loss=hinge;, score=0.878 total time=  10.4s
[CV 1/5] END …C=100, loss=squared_hinge;, score=0.877 total time=   2.5s
[CV 2/5] END …C=100, loss=squared_hinge;, score=0.881 total time=   5.6s
[CV 3/5] END …C=100, loss=squared_hinge;, score=0.875 total time=   7.6s
[CV 4/5] END …C=100, loss=squared_hinge;, score=0.881 total time=   6.5s
[CV 5/5] END …C=100, loss=squared_hinge;, score=0.878 total time=   6.5s
```

[93]:
```
GridSearchCV(estimator=LinearSVC(),
             param_grid={'C': [0.1, 1, 10, 100],
                         'loss': ['hinge', 'squared_hinge']},
             verbose=3)
```

[94]:
```
print("best cross validation score: {:.2f}".format(grid.best_score_))
print("best parameters: ",grid.best_params_)
```

```
best cross validation score: 0.89
best parameters:  {'C': 1, 'loss': 'hinge'}
```

[95]:
```
svc = LinearSVC(C = 1, loss = 'hinge')
svc.fit(X_train,y_train)
svc_pred = svc.predict(X_test)
svc_acc = accuracy_score(svc_pred,y_test)
print("Test accuracy: {:.2f}%".format(svc_acc*100))
```

```
Test accuracy: 89.40%
```

[96]:
```
print(confusion_matrix(y_test,svc_pred))
print("\n")
print(classification_report(y_test,svc_pred))
```

```
[[6510  851]
 [ 725 6788]]


              precision    recall  f1-score   support

           0       0.90      0.88      0.89      7361
           1       0.89      0.90      0.90      7513

    accuracy                           0.89     14874
   macro avg       0.89      0.89      0.89     14874
weighted avg       0.89      0.89      0.89     14874
```

```
[ ]:
```