Chapter 10: Text Analytics

10.2 Sentiment Classification

10.2.1 Loading the dataset

```
import pandas as pd
import numpy as np

import warnings
warnings.filterwarnings('ignore')

train_ds = pd.read_csv( "sentiment_train", delimiter="\t" )
train_ds.head( 5 )
```

	sentiment	text
0	1	The Da Vinci Code book is just awesome.
1	1	this was the first clive cussler i've ever rea...
2	1	i liked the Da Vinci Code a lot.
3	1	i liked the Da Vinci Code a lot.
4	1	I liked the Da Vinci Code but it ultimatly did...

```
pd.set_option('max_colwidth', 800)
train_ds[train_ds.sentiment == 1][0:5]
```

	sentiment	text
0	1	The Da Vinci Code book is just awesome.
1	1	this was the first clive cussler i've ever read, but even books like Relic, and Da Vinci code were more plausible than this.
2	1	i liked the Da Vinci Code a lot.
3	1	i liked the Da Vinci Code a lot.
4	1	I liked the Da Vinci Code but it ultimatly didn't seem to hold it's own.

```
train_ds[train_ds.sentiment == 0][0:5]
```

	sentiment	text
3943	0	da vinci code was a terrible movie.
3944	0	Then again, the Da Vinci code is super shitty movie, and it made like 700 million.
3945	0	The Da Vinci Code comes out tomorrow, which sucks.
3946	0	i thought the da vinci code movie was really boring.
3947	0	God, Yahoo Games has this truly-awful looking Da Vinci Code-themed skin on it's chessboard right now.

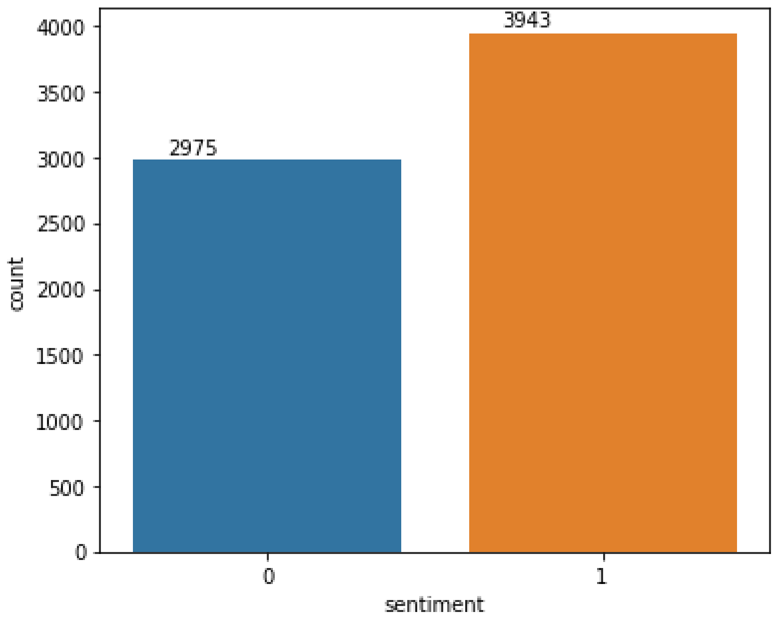
10.2.2 Exploring the dataset

```
train_ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6918 entries, 0 to 6917
Data columns (total 2 columns):
sentiment    6918 non-null int64
text         6918 non-null object
dtypes: int64(1), object(1)
memory usage: 108.2+ KB
```

```
import matplotlib.pyplot as plt
import seaborn as sn
%matplotlib inline

plt.figure( figsize=(6,5))
# create count plot
ax = sn.countplot(x='sentiment', data=train_ds)
# annotate
for p in ax.patches:
        ax.annotate(p.get_height(), (p.get_x()+0.1, p.get_height()+50))
```



10.2.3 Text Preprocessing

10.2.3.2 Creating Count Vectors for sentiment_train dataset

```
from sklearn.feature_extraction.text import CountVectorizer

# Initialize the CountVectorizer
count_vectorizer = CountVectorizer()
# Create the dictionary from the corpus
feature_vector = count_vectorizer.fit( train_ds.text )
# Get the feature names
features = feature_vector.get_feature_names()
print( "Total number of features: ", len(features))
```

```
Total number of features:  2132
```

```
import random

random.sample(features, 10)
```

```
['mad',
 'ew',
 'sceneries',
 'awesome',
 'gary',
 'wept',
 'hope',
 'life',
 'television',
 'aimee']
```

```
train_ds_features = count_vectorizer.transform( train_ds.text )
type(train_ds_features)
```

```
scipy.sparse.csr.csr_matrix
```

```
train_ds_features.shape
```

```
(6918, 2132)
```

```
train_ds_features.getnnz()
```

```
65398
```

```
print( "Density of the matrix: ",
      train_ds_features.getnnz() * 100 /
      (train_ds_features.shape[0] * train_ds_features.shape[1]))
```

```
Density of the matrix:  0.4434010415225908
```

10.2.3.3 Displaying Document Vectors

```
# Converting the matrix to a dataframe
train_ds_df = pd.DataFrame(train_ds_features.todense())
# Setting the column names to the features i.e. words
train_ds_df.columns = features
```

```
train_ds[0:1]
```

	sentiment	text
0	1	The Da Vinci Code book is just awesome.

```
train_ds_df.iloc[0:1, 150:157]
```

	away	awesome	awesomely	awesomeness	awesomest	awful	awkward
0	0	1	0	0	0	0	0

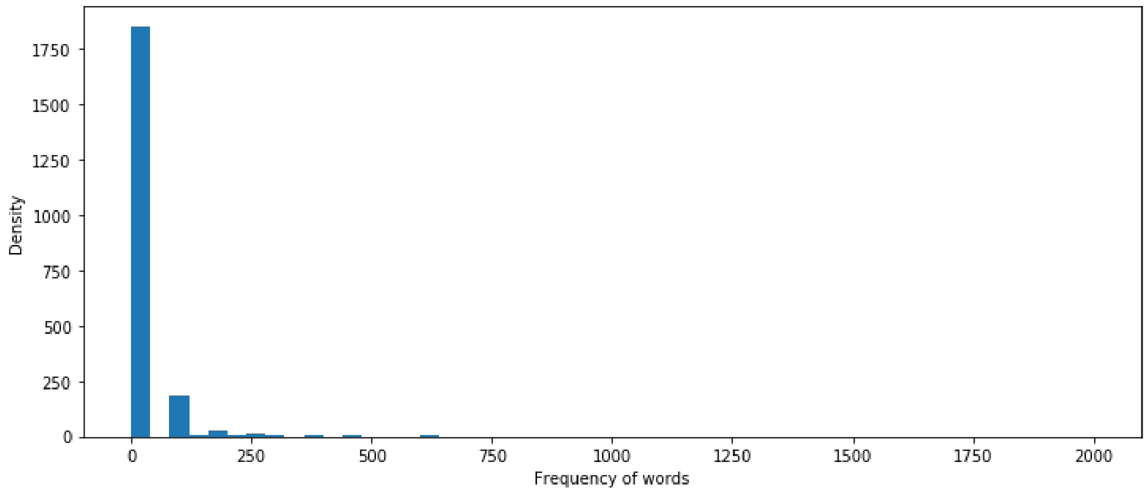
```
train_ds_df[['the', 'da', "vinci", "code", "book", 'is', 'just', 'awesome']][0:1
]
```

	the	da	vinci	code	book	is	just	awesome
0	1	1	1	1	1	1	1	1

10.2.3.4 Removing low frequency words

```
# summing up the occurances of features column wise
features_counts = np.sum( train_ds_features.toarray(), axis = 0 )
feature_counts_df = pd.DataFrame( dict( features = features,
                                        counts = features_counts ) )
```

```
plt.figure( figsize=(12,5))
plt.hist(feature_counts_df.counts, bins=50, range = (0, 2000));
plt.xlabel( 'Frequency of words' )
plt.ylabel( 'Density' );
```




```
len(feature_counts_df[feature_counts_df.counts == 1])
```

```
1228
```

```
# Initialize the CountVectorizer
count_vectorizer = CountVectorizer(max_features=1000)
# Create the dictionary from the corpus
feature_vector = count_vectorizer.fit( train_ds.text )
# Get the feature names
features = feature_vector.get_feature_names()
# Transform the document into vectors
train_ds_features = count_vectorizer.transform( train_ds.text )
# Count the frequency of the features
features_counts = np.sum( train_ds_features.toarray(), axis = 0 )
feature_counts = pd.DataFrame( dict( features = features,
                                    counts = features_counts ) )
```

```
feature_counts.sort_values('counts',
                           ascending = False)[0:15]
```

	counts	features
866	3306	the
37	2154	and
358	2093	harry
675	2093	potter
138	2002	code
934	2001	vinci
178	2001	da
528	2000	mountain
104	2000	brokeback
488	1624	love
423	1520	is
941	1176	was
60	1127	awesome
521	1094	mission
413	1093	impossible

10.2.3.5 Removing Stop Words

```
from sklearn.feature_extraction import text

my_stop_words = text.ENGLISH_STOP_WORDS

#Printing first few stop words
print("Few stop words: ", list(my_stop_words)[0:10])
```

```
Few stop words:  ['twenty', 'after', 'i', 'first', 'un', 'her', 'the
reupon', 'meanwhile', 'then', 'moreover']
```



```
# Adding custom words to the list of stop words
my_stop_words = text.ENGLISH_STOP_WORDS.union( ['harry', 'potter', 'code', 'vinc
i', 'da',
                                                'harry', 'mountain', 'movie', 'm
ovies'])
```

10.2.3.6 Creating Count Vectors

```
# Setting stop words list
count_vectorizer = CountVectorizer( stop_words = my_stop_words,
                                   max_features = 1000 )
feature_vector = count_vectorizer.fit( train_ds.text )
train_ds_features = count_vectorizer.transform( train_ds.text )
features = feature_vector.get_feature_names()
features_counts = np.sum( train_ds_features.toarray(), axis = 0 )
feature_counts = pd.DataFrame( dict( features = features,
                                    counts = features_counts ) )
```

```
feature_counts.sort_values( "counts", ascending = False )[0:15]
```

	counts	features
73	2000	brokeback
408	1624	love
39	1127	awesome
436	1094	mission
341	1093	impossible
390	974	like
745	602	sucks
743	600	sucked
297	578	hate
652	374	really
741	365	stupid
362	287	just
374	276	know
742	276	suck
409	256	loved

```
from nltk.stem.snowball import PorterStemmer

stemmer = PorterStemmer()
analyzer = CountVectorizer().build_analyzer()

#Custom function for stemming and stop word removal

def stemmed_words(doc):
    ### Stemming of words
    stemmed_words = (stemmer.stem(w) for w in analyzer(doc))
    ### Remove the words in stop words list
    non_stop_words = [ word for word in list(set(stemmed_words) - set(my_stop_wo
rds)) ]
    return non_stop_words
```

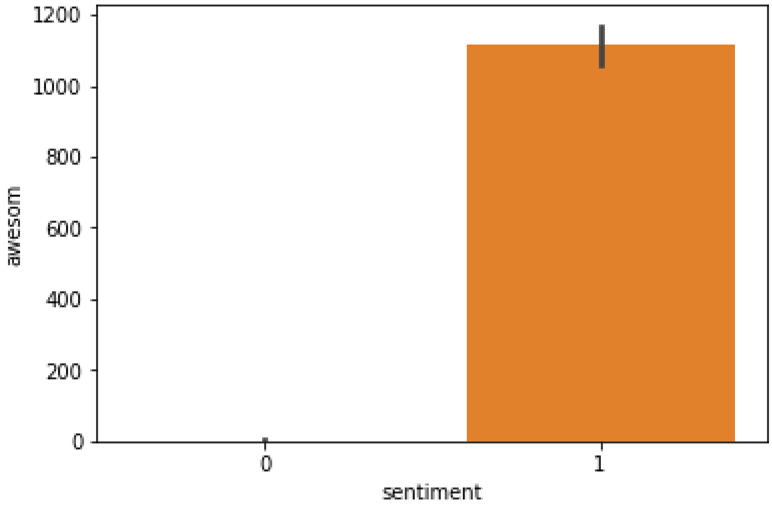
```
count_vectorizer = CountVectorizer( analyzer=stemmed_words,
                                    max_features = 1000)
feature_vector = count_vectorizer.fit( train_ds.text )
train_ds_features = count_vectorizer.transform( train_ds.text )
features = feature_vector.get_feature_names()
features_counts = np.sum( train_ds_features.toarray(), axis = 0 )
feature_counts = pd.DataFrame( dict( features = features,
                                     counts = features_counts ) )
feature_counts.sort_values( "counts", ascending = False )[0:15]
```

	counts	features
80	1930	brokeback
297	1916	harri
407	1837	love
803	1378	suck
922	1142	wa
43	1116	awesom
345	1090	imposs
433	1090	mission
439	1052	movi
393	823	like
299	636	hate
54	524	becaus
604	370	realli
796	364	stupid
379	354	know

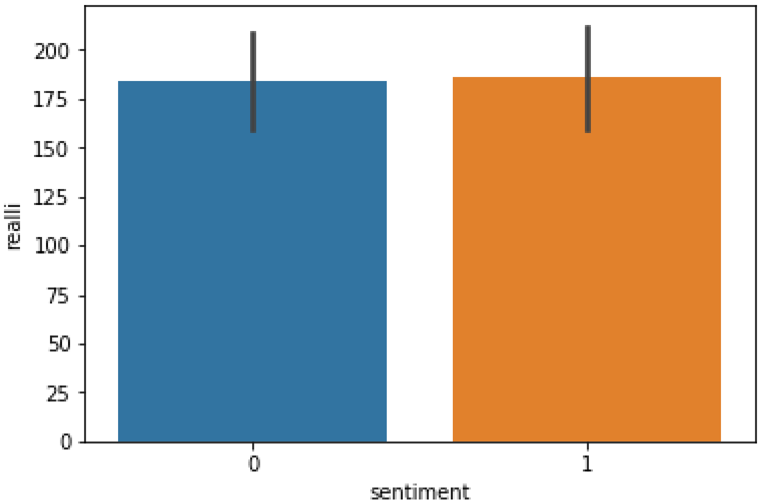
10.2.3.7 Distribution of words across different sentiment

```
# Convert the document vector matrix into dataframe
train_ds_df = pd.DataFrame(train_ds_features.todense())
# Assign the features names to the column
train_ds_df.columns = features
# Assign the sentiment labels to the train_ds
train_ds_df['sentiment'] = train_ds.sentiment
```

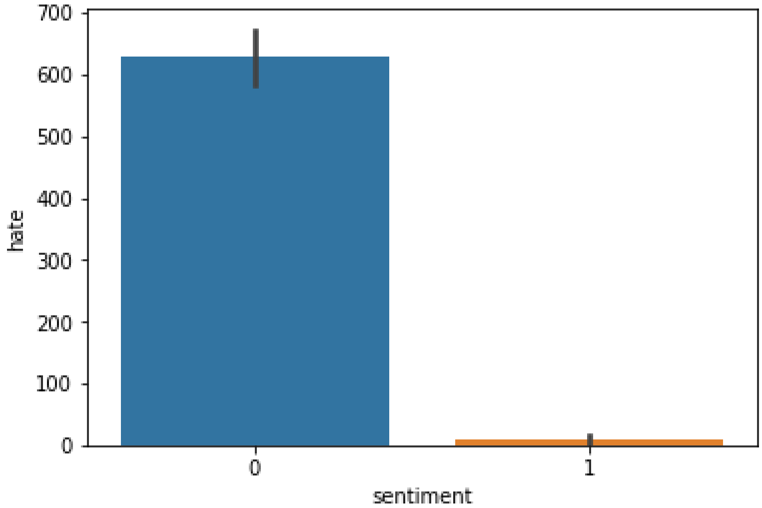
```
sn.barplot( x = 'sentiment', y = 'awesom', data = train_ds_df, estimator=sum );
```




```
sn.barplot( x = 'sentiment', y = 'realli', data = train_ds_df, estimator=sum );
```



```
sn.barplot( x = 'sentiment', y = 'hate', data = train_ds_df, estimator=sum );
```



10.3 Naive Bayes Model for Sentiment Classification

10.3.1 Split the dataset

```
from sklearn.model_selection import train_test_split

train_X, test_X, train_y, test_y = train_test_split( train_ds_features,
                                                     train_ds.sentiment,
                                                     test_size = 0.3,
                                                     random_state = 42 )
```

10.3.2 Build Naive Bayes Model

```
from sklearn.naive_bayes import BernoulliNB

nb_clf = BernoulliNB()
nb_clf.fit( train_X.toarray(), train_y )
```

```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=Tru
e)
```

10.3.3 Make prediction on test case

```
test_ds_predicted = nb_clf.predict( test_X.toarray() )
```

10.3.4 Print clasification report

```
from sklearn import metrics

print( metrics.classification_report( test_y, test_ds_predicted ) )
```

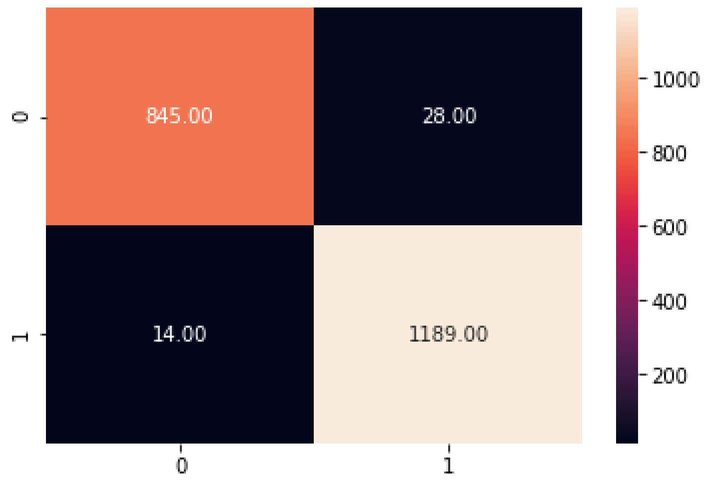
```
              precision    recall  f1-score   support

           0       0.98      0.97      0.98       873
           1       0.98      0.99      0.98      1203

   micro avg       0.98      0.98      0.98      2076
   macro avg       0.98      0.98      0.98      2076
weighted avg       0.98      0.98      0.98      2076
```

```
from sklearn import metrics

cm = metrics.confusion_matrix( test_y, test_ds_predicted )
sn.heatmap(cm, annot=True,  fmt='.2f' );
```



10.4 Using TF-IDF Vectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer( analyzer=stemmed_words,
                                    max_features = 1000)
feature_vector = tfidf_vectorizer.fit( train_ds.text )
train_ds_features = tfidf_vectorizer.transform( train_ds.text )
features = feature_vector.get_feature_names()
```

```
from sklearn.naive_bayes import GaussianNB

train_X, test_X, train_y, test_y = train_test_split( train_ds_features,
                                                     train_ds.sentiment,
                                                     test_size = 0.3,
                                                     random_state = 42 )

nb_clf = GaussianNB()
nb_clf.fit( train_X.toarray(), train_y )
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
test_ds_predicted = nb_clf.predict( test_X.toarray() )
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       873
           1       0.97      0.97      0.97      1203

   micro avg       0.97      0.97      0.97      2076
   macro avg       0.97      0.97      0.97      2076
weighted avg       0.97      0.97      0.97      2076
```

10.5.1 Using N-grams

```
import nltk
from nltk.stem import PorterStemmer
# library for regular expressions
import re

stemmer = PorterStemmer()

def get_stemmed_tokens( doc ):
    # Tokenize the documents to words
    all_tokens = [word for word in nltk.word_tokenize(doc)]
    clean_tokens = []
    # remove the all characters other than alphabets. It takes a regex for match
ing.
    for each_token in all_tokens:
        if re.search('[a-zA-Z]', each_token):
            clean_tokens.append(each_token)
    # Stem the words
    stemmed_tokens = [stemmer.stem(t) for t in clean_tokens]
    return stemmed_tokens
```

```
tfidf_vectorizer = TfidfVectorizer(max_features=500,
                                   stop_words='english',
                                   tokenizer=get_stemmed_tokens,
                                   ngram_range=(1,2))

feature_vector = tfidf_vectorizer.fit( train_ds.text )
train_ds_features = tfidf_vectorizer.transform( train_ds.text )
features = feature_vector.get_feature_names()
```

```
/Users/manaranjan/anaconda/lib/python3.5/site-packages/sklearn/featu
re_extraction/text.py:286: UserWarning: Your stop_words may be incon
sistent with your preprocessing. Tokenizing the stop words generated
tokens ['abov', 'afterward', 'alon', 'alreadi', 'alway', 'ani', 'ano
th', 'anyon', 'anyth', 'anywher', 'becam', 'becaus', 'becom', 'befo
r', 'besid', 'cri', 'describ', 'dure', 'els', 'elsewher', 'empti',
'everi', 'everyon', 'everyth', 'everywher', 'fifti', 'formerli', 'fo
rti', 'ha', 'henc', 'hereaft', 'herebi', 'hi', 'howev', 'hundr', 'in
de', 'latterli', 'mani', 'meanwhil', 'moreov', 'mostli', 'nobodi',
'noon', 'noth', 'nowher', 'onc', 'onli', 'otherwis', 'ourselv', 'per
hap', 'pleas', 'seriou', 'sever', 'sinc', 'sincer', 'sixti', 'someo
n', 'someth', 'sometim', 'somewher', 'themselv', 'thenc', 'thereaf
t', 'therebi', 'therefor', 'thi', 'thu', 'togeth', 'twelv', 'twent
i', 'veri', 'wa', 'whatev', 'whenc', 'whenev', 'wherea', 'whereaft',
'wherebi', 'wherev', 'whi', 'yourselv'] not in stop_words.
  sorted(inconsistent))
```

10.5.2 Build the model using n-grams

```
train_X, test_X, train_y, test_y = train_test_split( train_ds_features,
                                                     train_ds.sentiment,
                                                     test_size = 0.3,
                                                     random_state = 42 )

nb_clf = BernoulliNB()
nb_clf.fit( train_X.toarray(), train_y )
test_ds_predicted = nb_clf.predict( test_X.toarray() )
print( metrics.classification_report( test_y, test_ds_predicted ) )
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97       873
           1       0.96      1.00      0.98      1203

   micro avg       0.97      0.97      0.97      2076
   macro avg       0.98      0.97      0.97      2076
weighted avg       0.97      0.97      0.97      2076
```