Natural Language Processing with `nltk`

`nltk` is the most popular Python package for Natural Language processing, it provides algorithms for importing, cleaning, pre-processing text data in human language and then apply computational linguistics algorithms like sentiment analysis.

Inspect the Movie Reviews Dataset

It also includes many easy-to-use datasets in the `nltk.corpus` package, we can download for example the `movie_reviews` package using the `nltk.download` function:

In [1]:

```
import nltk
```

In [2]:

```
#http://www.nltk.org/nltk_data/
nltk.download('movie_reviews')
```

```
[nltk_data] Downloading package movie_reviews to
[nltk_data]     /Users/johnny/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
```

Out[2]:

```
True
```

In [3]:

```
#nltk.download('pros_cons')
#nltk.download()
```

You can also list and download other datasets interactively just typing:

```
nltk.download()
```

in the Jupyter Notebook.

Once the data have been downloaded, we can import them from `nltk.corpus`

In [4]:

```
from nltk.corpus import movie_reviews
```

The `fileids` method provided by all the datasets in `nltk.corpus` gives access to a list of all the files available.

In particular in the movie_reviews dataset we have 2000 text files, each of them is a review of a movie, and they are already split in a `neg` folder for the negative reviews and a `pos` folder for the positive reviews:

In [5]:

```
movie_reviews.fileids()
```

Out[5]:

```
['neg/cv000_29416.txt',
 'neg/cv001_19502.txt',
 'neg/cv002_17424.txt',
 'neg/cv003_12683.txt',
 'neg/cv004_12641.txt',
 'neg/cv005_29357.txt',
 'neg/cv006_17022.txt',
 'neg/cv007_4992.txt',
 'neg/cv008_29326.txt',
 'neg/cv009_29417.txt',
 'neg/cv010_29063.txt',
 'neg/cv011_13044.txt',
 'neg/cv012_29411.txt',
 'neg/cv013_10494.txt',
 'neg/cv014_15600.txt',
 'neg/cv015_29356.txt',
 'neg/cv016_4348.txt',
 'neg/cv017_23487.txt',
```

In [6]:

```
len(movie_reviews.fileids())
```

Out[6]:

```
2000
```

In [7]:

```
movie_reviews.fileids()[:5]
```

Out[7]:

```
['neg/cv000_29416.txt',
 'neg/cv001_19502.txt',
 'neg/cv002_17424.txt',
 'neg/cv003_12683.txt',
 'neg/cv004_12641.txt']
```

In [8]:

```
movie_reviews.fileids()[-5:]
```

Out[8]:

```
['pos/cv995_21821.txt',
 'pos/cv996_11592.txt',
 'pos/cv997_5046.txt',
 'pos/cv998_14111.txt',
 'pos/cv999_13106.txt']
```

`fileids` can also filter the available files based on their category, which is the name of the subfolders they are located in. Therefore we can have lists of positive and negative reviews separately.

In [9]:

```
negative_fileids = movie_reviews.fileids('neg')
positive_fileids = movie_reviews.fileids('pos')
```

In [10]:

```
len(negative_fileids), len(positive_fileids)
```

Out[10]:

```
(1000, 1000)
```

We can inspect one of the reviews using the `raw` method of `movie_reviews`, each file is split into sentences, the curators of this dataset also removed from each review from any direct mention of the rating of the movie.

In [11]:

```
print(movie_reviews.raw(fileids = positive_fileids[0]))
```

```
films adapted from comic books have had plenty of success , whether th
ey're about superheroes ( batman , superman , spawn ) , or geared towa
rd kids ( casper ) or the arthouse crowd ( ghost world ) , but there's
never really been a comic book like from hell before .
for starters , it was created by alan moore ( and eddie campbell ) , w
ho brought the medium to a whole new level in the mid '80s with a 12-p
art series called the watchmen .
to say moore and campbell thoroughly researched the subject of jack th
e ripper would be like saying michael jackson is starting to look a li
ttle odd .
the book ( or " graphic novel , " if you will ) is over 500 pages long
and includes nearly 30 more that consist of nothing but footnotes .
in other words , don't dismiss this film because of its source .
if you can get past the whole comic book thing , you might find anothe
r stumbling block in from hell's directors , albert and allen hughes .

getting the hughes brothers to direct this seems almost as ludicrous a
s casting carrot top in , well , anything , but riddle me this : who b
etter to direct a film that's set in the ghetto and features really vi
olent street crime than the mad geniuses behind menace ii society ?
the ghetto in question is , of course , whitechapel in 1888 london's e
ast end .
it's a filthy , sooty place where the whores ( called " unfortunates "
) are starting to get a little nervous about this mysterious psychopat
h who has been carving through their profession with surgical precisio
n .
when the first stiff turns up , copper peter godley ( robbie coltrane
, the world is not enough ) calls in inspector frederick abberline ( j
ohnny depp , blow ) to crack the case .
abberline , a widower , has prophetic dreams he unsuccessfully tries t
o quell with copious amounts of absinthe and opium .
upon arriving in whitechapel , he befriends an unfortunate named mary
kelly ( heather graham , say it isn't so ) and proceeds to investigate
the horribly gruesome crimes that even the police surgeon can't stomac
h .
i don't think anyone needs to be briefed on jack the ripper , so i wo
n't go into the particulars here , other than to say moore and campbel
l have a unique and interesting theory about both the identity of the
killer and the reasons he chooses to slay .
in the comic , they don't bother cloaking the identity of the ripper ,
but screenwriters terry hayes ( vertical limit ) and rafael yglesias (
les mis ? rables ) do a good job of keeping him hidden from viewers un
til the very end .
it's funny to watch the locals blindly point the finger of blame at je
ws and indians because , after all , an englishman could never be capa
ble of committing such ghastly acts .
and from hell's ending had me whistling the stonecutters song from the
simpsons for days ( " who holds back the electric car/who made steve g
uttenberg a star ? " ) .
don't worry - it'll all make sense when you see it .
now onto from hell's appearance : it's certainly dark and bleak enough
, and it's surprising to see how much more it looks like a tim burton
film than planet of the apes did ( at times , it seems like sleepy hol
low 2 ) .
the print i saw wasn't completely finished ( both color and music had
not been finalized , so no comments about marilyn manson ) , but cinem
atographer peter deming ( don't say a word ) ably captures the drearin
```

```
ess of victorian-era london and helped make the flashy killing scenes
remind me of the crazy flashbacks in twin peaks , even though the viol
ence in the film pales in comparison to that in the black-and-white co
mic .
oscar winner martin childs' ( shakespeare in love ) production design
turns the original prague surroundings into one creepy place .
even the acting in from hell is solid , with the dreamy depp turning i
n a typically strong performance and deftly handling a british accent
.
ians holm ( joe gould's secret ) and richardson ( 102 dalmatians ) log
in great supporting roles , but the big surprise here is graham .
i cringed the first time she opened her mouth , imagining her attempt
at an irish accent , but it actually wasn't half bad .
the film , however , is all good .
2 : 00 - r for strong violence/gore , sexuality , language and drug co
ntent
```

In [12]:

```
print(movie_reviews.raw(fileids = negative_fileids[0]))
```

```
plot : two teen couples go to a church party , drink and then drive .
they get into an accident .
one of the guys dies , but his girlfriend continues to see him in her
life , and has nightmares .
what's the deal ?
watch the movie and " sorta " find out . . .
critique : a mind-fuck movie for the teen generation that touches on a
very cool idea , but presents it in a very bad package .
which is what makes this review an even harder one to write , since i
generally applaud films which attempt to break the mold , mess with yo
ur head and such ( lost highway & memento ) , but there are good and b
ad ways of making all types of films , and these folks just didn't sna
g this one correctly .
they seem to have taken this pretty neat concept , but executed it ter
ribly .
so what are the problems with the movie ?
well , its main problem is that it's simply too jumbled .
it starts off " normal " but then downshifts into this " fantasy " wor
ld in which you , as an audience member , have no idea what's going on
.
there are dreams , there are characters coming back from the dead , th
ere are others who look like the dead , there are strange apparitions
, there are disappearances , there are a looooot of chase scenes , the
re are tons of weird things that happen , and most of it is simply not
explained .
now i personally don't mind trying to unravel a film every now and the
n , but when all it does is give me the same clue over and over again
, i get kind of fed up after a while , which is this film's biggest pr
oblem .
it's obviously got this big secret to hide , but it seems to want to h
ide it completely until its final five minutes .
and do they make things entertaining , thrilling or even engaging , in
the meantime ?
not really .
the sad part is that the arrow and i both dig on flicks like this , so
we actually figured most of it out by the half-way point , so all of t
he strangeness after that did start to make a little bit of sense , bu
t it still didn't the make the film all that more entertaining .
i guess the bottom line with movies like this is that you should alway
s make sure that the audience is " into it " even before they are give
n the secret password to enter your world of understanding .
i mean , showing melissa sagemiller running away from visions for abou
t 20 minutes throughout the movie is just plain lazy ! !
okay , we get it . . . there
are people chasing her and we don't know who they are .
do we really need to see it over and over again ?
how about giving us different scenes offering further insight into all
of the strangeness going down in the movie ?
apparently , the studio took this film away from its director and chop
ped it up themselves , and it shows .
there might've been a pretty decent teen mind-fuck movie in here somew
here , but i guess " the suits " decided that turning it into a music
video with little edge , would make more sense .
the actors are pretty good for the most part , although wes bentley ju
st seemed to be playing the exact same character that he did in americ
an beauty , only in a new neighborhood .
but my biggest kudos go out to sagemiller , who holds her own througho
```

```
ut the entire film , and actually has you feeling her character's unra
veling .
overall , the film doesn't stick because it doesn't entertain , it's c
onfusing , it rarely excites and it feels pretty redundant for most of
its runtime , despite a pretty cool ending and explanation to all of t
he craziness that came before it .
oh , and by the way , this is not a horror or teen slasher flick . . .
it's
just packaged to look that way because someone is apparently assuming
that the genre is still hot with the kids .
it also wrapped production two years ago and has been sitting on the s
helves ever since .
whatever . . . skip
it !
where's joblo coming from ?
a nightmare of elm street 3 ( 7/10 ) - blair witch 2 ( 7/10 ) - the cr
ow ( 9/10 ) - the crow : salvation ( 4/10 ) - lost highway ( 10/10 ) -
memento ( 10/10 ) - the others ( 9/10 ) - stir of echoes ( 8/10 )
```

Tokenize Text in Words

In [13]:

```
romeo_text = """Why then, O brawling love! O loving hate!
O any thing, of nothing first create!
O heavy lightness, serious vanity,
Misshapen chaos of well-seeming forms,
Feather of lead, bright smoke, cold fire, sick health,
Still-waking sleep, that is not what it is!
This love feel I, that feel no love in this."""
```

The first step in Natural Language processing is generally to split the text into words, this process might appear simple but it is very tedious to handle all corner cases, see for example all the issues with punctuation we have to solve if we just start with a split on whitespace:

In [14]:

```
romeo_text.split()
```

Out[14]:

```
['Why',
 'then,',
 'O',
 'brawling',
 'love!',
 'O',
 'loving',
 'hate!',
 'O',
 'any',
 'thing,',
 'of',
 'nothing',
 'first',
 'create!',
 'O',
 'heavy',
 'lightness,',
 'serious',
 'vanity,',
 'Misshapen',
 'chaos',
 'of',
 'well-seeming',
 'forms,',
 'Feather',
 'of',
 'lead,',
 'bright',
 'smoke,',
 'cold',
 'fire,',
 'sick',
 'health,',
 'Still-waking',
 'sleep,',
 'that',
 'is',
 'not',
 'what',
 'it',
 'is!',
 'This',
 'love',
 'feel',
 'I,',
 'that',
 'feel',
 'no',
 'love',
 'in',
 'this.']
```

`nltk` has a sophisticated word tokenizer trained on English named `punkt`, we first have to download its

parameters:

In [15]:

```
nltk.download("punkt")
```

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

Out[15]:

```
True
```

Then we can use the `word_tokenize` function to properly tokenize this text, compare to the whitespace splitting we used above:

In [16]:

```
romeo_words = nltk.word_tokenize(romeo_text)
```

In [17]:

```
romeo_words
```

Out[17]:

```
['Why',
 'then',
 ',',
 'O',
 'brawling',
 'love',
 '!',
 'O',
 'loving',
 'hate',
 '!',
 'O',
 'any',
 'thing',
 ',',
 'of',
 'nothing',
 'first',
 'create',
 '!',
 'O',
 'heavy',
 'lightness',
 ',',
 'serious',
 'vanity',
 ',',
 'Misshapen',
 'chaos',
 'of',
 'well-seeming',
 'forms',
 ',',
 'Feather',
 'of',
 'lead',
 ',',
 'bright',
 'smoke',
 ',',
 'cold',
 'fire',
 ',',
 'sick',
 'health',
 ',',
 'Still-waking',
 'sleep',
 ',',
 'that',
 'is',
 'not',
 'what',
 'it',
 'is',
```

```
 '!',
 'This',
 'love',
 'feel',
 'I',
 ',',
 'that',
 'feel',
 'no',
 'love',
 'in',
 'this',
 '.']
```

Good news is that the `movie_reviews` corpus already has direct access to tokenized text with the `words` method:

In [18]:

```
movie_reviews.words(fileids = positive_fileids[0])
```

Out[18]:

```
['films', 'adapted', 'from', 'comic', 'books', 'have', ...]
```

In [19]:

```
movie_reviews.words(fileids = negative_fileids[0])
```

Out[19]:

```
['plot', ':', 'two', 'teen', 'couples', 'go', 'to', ...]
```

Build a bag-of-words model

The simplest model for analyzing text is just to think about text as an unordered collection of words (bag-of-words). This can generally allow to infer from the text the category, the topic or the sentiment.

From the bag-of-words model we can build features to be used by a classifier, here we assume that each word is a feature that can either be `True` or `False`. We implement this in Python as a dictionary where for each word in a sentence we associate `True`, if a word is missing, that would be the same as assigning `False`.

In [20]:

```
# word is True when word in romeo_words
{word: True for word in romeo_words}
```

Out[20]:

```
{'!': True,
 ',': True,
 '.': True,
 'Feather': True,
 'I': True,
 'Misshapen': True,
 'O': True,
 'Still-waking': True,
 'This': True,
 'Why': True,
 'any': True,
 'brawling': True,
 'bright': True,
 'chaos': True,
 'cold': True,
 'create': True,
 'feel': True,
 'fire': True,
```

In [21]:

```
type(_) # _ the underscore here is the last output that goes into the standard out
```

Out[21]:

```
dict
```

In [22]:

```
def build_bag_of_words_features(words):
    return {word : True for word in words}
```

In [23]:

```
build_bag_of_words_features(romeo_words)
```

Out[23]:

```
{'!': True,
 ',': True,
 '.': True,
 'Feather': True,
 'I': True,
 'Misshapen': True,
 'O': True,
 'Still-waking': True,
 'This': True,
 'Why': True,
 'any': True,
 'brawling': True,
 'bright': True,
 'chaos': True,
 'cold': True,
 'create': True,
 'feel': True,
 'fire': True,
 'first': True,
 'forms': True,
 'hate': True,
 'health': True,
 'heavy': True,
 'in': True,
 'is': True,
 'it': True,
 'lead': True,
 'lightness': True,
 'love': True,
 'loving': True,
 'no': True,
 'not': True,
 'nothing': True,
 'of': True,
 'serious': True,
 'sick': True,
 'sleep': True,
 'smoke': True,
 'that': True,
 'then': True,
 'thing': True,
 'this': True,
 'vanity': True,
 'well-seeming': True,
 'what': True}
```

This is what we wanted, but we notice that also punctuation like "!" and words useless for classification purposes like "of" or "that" are also included. Those words are named "stopwords" and `nltk` has a convenient corpus we can download:

In [24]:

```
nltk.download("stopwords")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/johnny/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[24]:

```
True
```

In [25]:

```
import string
```

In [26]:

```
string.punctuation
```

Out[26]:

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [27]:

```
nltk.corpus.stopwords.words('english') # It's a combination of stopwords for Englis
```

Out[27]:

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

Using the Python `string.punctuation` list and the English stopwords we can build better features by filtering out those words that would not help in the classification:

In [28]:

```
useless_words = nltk.corpus.stopwords.words('english') + list(string.punctuation)
useless_words
```

Out[28]:

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

In [29]:

```
type(useless_words)
```

Out[29]:

```
list
```

In [30]:

```
# if the words exists in useless_words, skip that word
def build_bag_of_words_features_filtered(words):
    return {
        word: 1 for word in words \
        if not word in useless_words # if the word not in useless_words, add that to
    }
```

In [31]:

```
build_bag_of_words_features_filtered(romeo_words)
```

Out[31]:

```
{'Feather': 1,
 'I': 1,
 'Misshapen': 1,
 'O': 1,
 'Still-waking': 1,
 'This': 1,
 'Why': 1,
 'brawling': 1,
 'bright': 1,
 'chaos': 1,
 'cold': 1,
 'create': 1,
 'feel': 1,
 'fire': 1,
 'first': 1,
 'forms': 1,
 'hate': 1,
 'health': 1,
 'heavy': 1,
 'lead': 1,
 'lightness': 1,
 'love': 1,
 'loving': 1,
 'nothing': 1,
 'serious': 1,
 'sick': 1,
 'sleep': 1,
 'smoke': 1,
 'thing': 1,
 'vanity': 1,
 'well-seeming': 1}
```

Plotting Frequencies of Words

It is common to explore a dataset before starting the analysis, in this section we will find the most common words and plot their frequency.

Using the `.words()` function with no argument we can extract the words from the entire dataset and check that it is about 1.6 millions.

In [32]:

```
all_words = movie_reviews.words()
len(all_words)/1e6 # print it in the millions format
```

Out[32]:

```
1.58382
```

First we want to filter out `useless_words` as defined in the previous section, this will reduce the length of the dataset by more than a factor of 2:

In [33]:

```
# do this if word is not in the useless_words
filtered_words = [word for word in movie_reviews.words() if not word in useless_word
type(filtered_words)
```

Out[33]:

```
list
```

In [34]:

```
len(filtered_words)/1e6
```

Out[34]:

```
0.710579
```

The `collection` package of the standard library contains a `Counter` class that is handy for counting frequencies of words in our list:

In [35]:

```
from collections import Counter

word_counter = Counter(filtered_words)
```

It also has a `most_common()` method to access the words with the higher count:

In [36]:

```
most_common_words = word_counter.most_common()[:10]
```

In [37]:

```
most_common_words
```

Out[37]:

```
[('film', 9517),
 ('one', 5852),
 ('movie', 5771),
 ('like', 3690),
 ('even', 2565),
 ('good', 2411),
 ('time', 2411),
 ('story', 2169),
 ('would', 2109),
 ('much', 2049)]
```

Then we would like to have a visualization of this using `matplotlib` .

First we want to use the Jupyter magic function

```
%matplotlib inline
```

to setup the Notebook to show the plot embedded into the Jupyter Notebook page, you can also test:

```
%matplotlib notebook
```

for a more interactive plotting interface which however is not as well supported on all platforms and browsers.

In [38]:

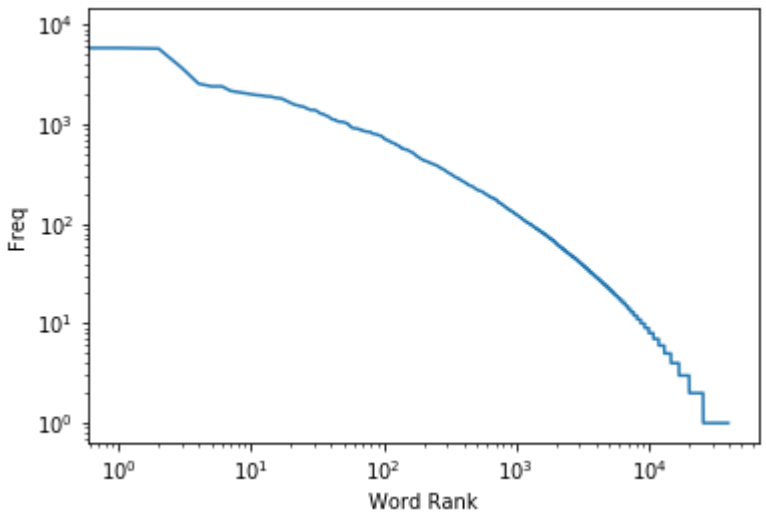
```
import matplotlib.pyplot as plt
%matplotlib inline
```

We can sort the word counts and plot their values on Logarithmic axes to check the shape of the distribution. This visualization is particularly useful if comparing 2 or more datasets, a flatter distribution indicates a large vocabulary while a peaked distribution a restricted vocabulary often due to a focused topic or specialized language.

In [39]:

```
sorted_word_counts = sorted(list(word_counter.values()), reverse = True)

plt.loglog(sorted_word_counts)
plt.xlabel('Word Rank')
plt.ylabel('Freq');
```

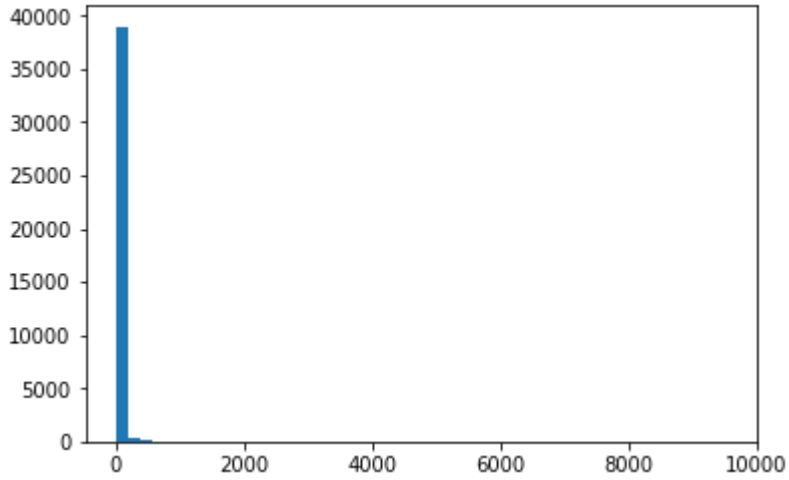


Another related plot is the histogram of `sorted_word_counts` , which displays how many words have a count in a specific range.

Of course the distribution is highly peaked at low counts, i.e. most of the words appear which a low count, so we better display it on semilogarithmic axes to inspect the tail of the distribution.

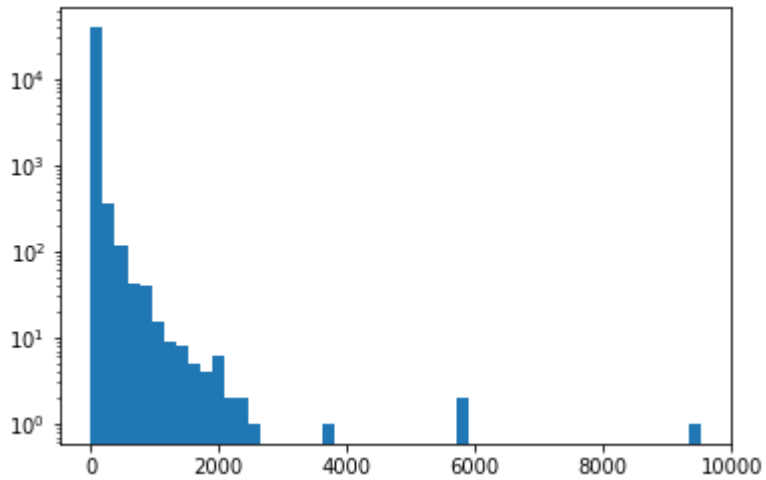
In [40]:

```
plt.hist(sorted_word_counts, bins = 50);
```



In [41]:

```
# Displaying this at log scale
plt.hist(sorted_word_counts, bins=50, log = True);
```



Train a Classifier for Sentiment Analysis

Using our `build_bag_of_words_features` function we can build separately the negative and positive features. Basically for each of the 1000 negative and for the 1000 positive review, we create one dictionary of the words and we associate the label "neg" and "pos" to it.

In [42]:

```
negative_features = [
    (build_bag_of_words_features_filtered(movie_reviews.words(fileids = [f])), 'neg
    for f in negative_fileids
]
```

In [43]:

```
# Let's look at negative review #3
print(negative_features[3])
```

```
({'quest': 1, 'camelot': 1, 'warner': 1, 'bros': 1, 'first': 1, 'featu
re': 1, 'length': 1, 'fully': 1, 'animated': 1, 'attempt': 1, 'steal':
1, 'clout': 1, 'disney': 1, 'cartoon': 1, 'empire': 1, 'mouse': 1, 're
ason': 1, 'worried': 1, 'recent': 1, 'challenger': 1, 'throne': 1, 'la
st': 1, 'fall': 1, 'promising': 1, 'flawed': 1, '20th': 1, 'century':
1, 'fox': 1, 'production': 1, 'anastasia': 1, 'hercules': 1, 'lively':
1, 'cast': 1, 'colorful': 1, 'palate': 1, 'beat': 1, 'hands': 1, 'cam
e': 1, 'time': 1, 'crown': 1, '1997': 1, 'best': 1, 'piece': 1, 'anima
tion': 1, 'year': 1, 'contest': 1, 'pretty': 1, 'much': 1, 'dead': 1,
'arrival': 1, 'even': 1, 'magic': 1, 'kingdom': 1, 'mediocre': 1, '--
': 1, 'pocahontas': 1, 'keeping': 1, 'score': 1, 'nearly': 1, 'dull':
1, 'story': 1, 'revolves': 1, 'around': 1, 'adventures': 1, 'free': 1,
'spirited': 1, 'kayley': 1, 'voiced': 1, 'jessalyn': 1, 'gilsig': 1,
'early': 1, 'teen': 1, 'daughter': 1, 'belated': 1, 'knight': 1, 'kin
g': 1, 'arthur': 1, 'round': 1, 'table': 1, 'dream': 1, 'follow': 1,
'father': 1, 'footsteps': 1, 'gets': 1, 'chance': 1, 'evil': 1, 'warlo
rd': 1, 'ruber': 1, 'gary': 1, 'oldman': 1, 'ex': 1, 'member': 1, 'gon
e': 1, 'bad': 1, 'steals': 1, 'magical': 1, 'sword': 1, 'excalibur':
1, 'accidentally': 1, 'loses': 1, 'dangerous': 1, 'booby': 1, 'trappe
d': 1, 'forest': 1, 'help': 1, 'hunky': 1, 'blind': 1, 'timberland':
1, 'dweller': 1, 'garrett': 1, 'carey': 1, 'elwes': 1, 'two': 1, 'head
ed': 1, 'dragon': 1, 'eric': 1, 'idle': 1, 'rickles': 1, 'always': 1,
'arguing': 1, 'might': 1, 'able': 1, 'break': 1, 'medieval': 1, 'sexis
t': 1, 'mold': 1, 'prove': 1, 'worth': 1, 'fighter': 1, 'side': 1, 'mi
ssing': 1, 'pure': 1, 'showmanship': 1, 'essential': 1, 'element': 1,
'ever': 1, 'expected': 1, 'climb': 1, 'high': 1, 'ranks': 1, 'nothin
g': 1, 'differentiates': 1, 'something': 1, 'see': 1, 'given': 1, 'sat
urday': 1, 'morning': 1, 'subpar': 1, 'instantly': 1, 'forgettable':
1, 'songs': 1, 'poorly': 1, 'integrated': 1, 'computerized': 1, 'foota
ge': 1, 'compare': 1, 'run': 1, 'angry': 1, 'ogre': 1, 'herc': 1, 'bat
tle': 1, 'hydra': 1, 'rest': 1, 'case': 1, 'characters': 1, 'stink':
1, 'none': 1, 'remotely': 1, 'interesting': 1, 'film': 1, 'becomes':
1, 'race': 1, 'one': 1, 'bland': 1, 'others': 1, 'end': 1, 'tie': 1,
'win': 1, 'comedy': 1, 'shtick': 1, 'awfully': 1, 'cloying': 1, 'leas
t': 1, 'shows': 1, 'signs': 1, 'pulse': 1, 'fans': 1, "-'": 1, '90s':
1, 'tgif': 1, 'television': 1, 'line': 1, 'thrilled': 1, 'find': 1, 'j
aleel': 1, 'urkel': 1, 'white': 1, 'bronson': 1, 'balki': 1, 'pincho
t': 1, 'sharing': 1, 'scenes': 1, 'nicely': 1, 'realized': 1, 'thoug
h': 1, 'loss': 1, 'recall': 1, 'enough': 1, 'specific': 1, 'actors':
1, 'providing': 1, 'voice': 1, 'talent': 1, 'enthusiastic': 1, 'paire
d': 1, 'singers': 1, 'sound': 1, 'thing': 1, 'like': 1, 'big': 1, 'mus
ical': 1, 'moments': 1, 'jane': 1, 'seymour': 1, 'celine': 1, 'dion':
1, 'must': 1, 'strain': 1, 'mess': 1, 'good': 1, 'aside': 1, 'fact':
1, 'children': 1, 'probably': 1, 'bored': 1, 'watching': 1, 'adults':
1, 'grievous': 1, 'error': 1, 'complete': 1, 'lack': 1, 'personality':
1, 'learn': 1, 'goes': 1, 'long': 1, 'way': 1}, 'neg')
```

In [44]:

```
positive_features = [
    (build_bag_of_words_features_filtered(movie_reviews.words(fileids=[f])), 'pos')
    for f in positive_fileids
]
```

In [45]:

```
# Let's look at positive review #6
print(positive_features[6])
```

```
({'apparently': 1, 'director': 1, 'tony': 1, 'kaye': 1, 'major': 1, 'b
attle': 1, 'new': 1, 'line': 1, 'regarding': 1, 'film': 1, 'american':
1, 'history': 1, 'x': 1, 'know': 1, 'details': 1, 'fight': 1, 'seems':
1, 'happy': 1, 'final': 1, 'product': 1, 'nearly': 1, 'removed': 1, 'n
ame': 1, 'credits': 1, 'altogether': 1, 'heard': 1, 'kind': 1, 'thin
g': 1, 'happening': 1, 'makes': 1, 'wonder': 1, 'much': 1, 'input': 1,
'studio': 1, 'films': 1, 'produce': 1, 'found': 1, 'extremely': 1, 'go
od': 1, 'focused': 1, 'look': 1, 'touchy': 1, 'subject': 1, 'racism':
1, 'powerful': 1, 'charismatic': 1, 'performance': 1, 'edward': 1, 'no
rton': 1, 'hard': 1, 'believe': 1, 'two': 1, 'years': 1, 'since': 1,
'fantastic': 1, 'role': 1, 'primal': 1, 'fear': 1, 'starring': 1, 'mak
ing': 1, 'star': 1, 'one': 1, 'performers': 1, 'becomes': 1, 'characte
r': 1, 'work': 1, 'best': 1, 'performances': 1, 'year': 1, 'plays': 1,
'young': 1, 'man': 1, 'named': 1, 'derek': 1, 'vinyard': 1, 'skinhea
d': 1, 'living': 1, 'venice': 1, 'beach': 1, 'brother': 1, 'danny': 1,
'furlong': 1, 'mother': 1, 'beverly': 1, 'angelo': 1, 'sister': 1, 'da
vin': 1, 'jennifer': 1, 'lien': 1, 'opens': 1, 'flashback': 1, 'brutal
ly': 1, 'kills': 1, 'black': 1, 'men': 1, 'vandalizing': 1, 'car': 1,
'find': 1, 'lands': 1, 'prison': 1, 'point': 1, 'seen': 1, 'eyes': 1,
'present': 1, 'time': 1, 'high': 1, 'school': 1, 'eager': 1, 'follow':
1, 'footsteps': 1, 'told': 1, 'see': 1, 'path': 1, 'leads': 1, 'adopti
on': 1, 'white': 1, 'supremacy': 1, 'released': 1, 'served': 1, 'thre
e': 1, 'finds': 1, 'full': 1, 'blown': 1, 'however': 1, 'given': 1, 'v
iolence': 1, 'tries': 1, 'get': 1, 'understand': 1, 'comes': 1, 'bad':
1, 'things': 1, 'interesting': 1, 'stupid': 1, 'thoughtless': 1, 'peop
le': 1, '--': 1, 'intelligent': 1, 'articulate': 1, 'voice': 1, 'belie
fs': 1, 'disturbingly': 1, 'straightforward': 1, 'terms': 1, 'make':
1, 'controversial': 1, 'movie': 1, 'preach': 1, 'right': 1, 'note': 1,
'material': 1, 'mainstream': 1, 'redemption': 1, 'phase': 1, 'main':
1, 'may': 1, 'think': 1, 'way': 1, 'sympathetic': 1, 'partially': 1,
'disagree': 1, 'although': 1, 'advocate': 1, 'presents': 1, 'loud': 1,
'obnoxious': 1, 'also': 1, 'smart': 1, 'reasons': 1, 'believable': 1,
'father': 1, 'arbitrarily': 1, 'killed': 1, 'group': 1, 'clear': 1, 'p
assionate': 1, 'punk': 1, 'looking': 1, 'excuse': 1, 'beat': 1, 'cours
e': 1, 'helps': 1, 'actor': 1, 'talented': 1, 'play': 1, 'part': 1, 'a
stonishing': 1, 'frightening': 1, 'looks': 1, 'shaved': 1, 'head': 1,
'swastika': 1, 'chest': 1, 'addition': 1, 'getting': 1, 'perfect': 1,
'requires': 1, 'intelligence': 1, 'depth': 1, 'whole': 1, 'lot': 1, 's
houting': 1, 'ease': 1, 'even': 1, 'meanest': 1, 'likable': 1, 'qualit
y': 1, 'gutsy': 1, 'approach': 1, 'telling': 1, 'story': 1, 'adds': 1,
'subplot': 1, 'principal': 1, 'avery': 1, 'brooks': 1, 'obsessed': 1,
'purging': 1, 'hatred': 1, 'terrific': 1, 'standouts': 1, 'visually':
1, 'indulges': 1, 'artistic': 1, 'choices': 1, 'nicely': 1, 'lots': 1,
'slow': 1, 'motion': 1, 'strange': 1, 'camera': 1, 'angles': 1, 'add':
1, 'moody': 1, 'atmosphere': 1, 'like': 1, 'movies': 1, 'lately': 1,
'skims': 1, 'past': 1, 'greatness': 1, 'last': 1, 'minutes': 1, 'clima
ctic': 1, 'scene': 1, 'moving': 1, 'picture': 1, 'ends': 1, 'pretentio
us': 1, 'preachy': 1, 'resolution': 1, 'featuring': 1, 'brief': 1, 'na
rration': 1, 'subtle': 1, 'felt': 1, 'slap': 1, 'face': 1, 'hand': 1,
'fed': 1, 'theme': 1, 'simplistic': 1, 'exactly': 1, 'disliked': 1, 'v
ersion': 1, 'perhaps': 1, 'problem': 1, 'imagine': 1, 'least': 1, 'ple
ased': 1, 'many': 1, 'timid': 1, 'weak': 1, 'manages': 1, 'compellin
g': 1, 'argument': 1, 'without': 1, 'advocating': 1}, 'pos')
```

In [46]:

```
from nltk.classify import NaiveBayesClassifier
```

One of the simplest supervised machine learning classifiers is the Naive Bayes Classifier, it can be trained on 80% of the data to learn what words are generally associated with positive or with negative reviews.

In [47]:

```
# 1000 * 80% = 800
split = 800
```

In [48]:

```
sentiment_classifier = NaiveBayesClassifier.train(positive_features[:split] + negati
```

We can check after training what is the accuracy on the training set, i.e. the same data used for training, we expect this to be a very high number because the algorithm already "saw" those data. Accuracy is the fraction of the data that is classified correctly, we can turn it into percent:

In [49]:

```
nltk.classify.util.accuracy(sentiment_classifier, positive_features[:split] + negati
```

Out[49]:

```
98.0625
```

The accuracy above is mostly a check that nothing went very wrong in the training, the real measure of accuracy is on the remaining 20% of the data that wasn't used in training, the test data:

In [50]:

```
nltk.classify.util.accuracy(sentiment_classifier,positive_features[split:] + negativ
```

Out[50]:

```
71.75
```

Accuracy here is around 70% which is pretty good for such a simple model if we consider that the estimated accuracy for a person is about 80%. We can finally print the most informative features, i.e. the words that mostly identify a positive or a negative review:

In [51]:

```
sentiment_classifier.show_most_informative_features()
```

```
Most Informative Features
             outstanding = 1                pos : neg    =     13.9 :
1.0
               insulting = 1                neg : pos    =     13.7 :
1.0
              vulnerable = 1                pos : neg    =     13.0 :
1.0
               ludicrous = 1                neg : pos    =     12.6 :
1.0
             uninvolving = 1                neg : pos    =     12.3 :
1.0
              astounding = 1                pos : neg    =     11.7 :
1.0
                  avoids = 1                pos : neg    =     11.7 :
1.0
             fascination = 1                pos : neg    =     11.0 :
1.0
                   anna = 1                pos : neg    =     10.3 :
1.0
               animators = 1                pos : neg    =     10.3 :
1.0
```