Now that we're comfortable with NLTK, let's try to tackle text classification. The goal with text classification can be pretty broad. Maybe we're trying to classify text as about politics or the military. Maybe we're trying to classify it by the gender of the author who wrote it. A fairly popular text classification task is to identify a body of text as either spam or not spam, for things like email filters. In our case, we're going to try to create a sentiment analysis algorithm.

To do this, we're going to start by trying to use the movie reviews database that is part of the NLTK corpus. From there we'll try to use words as "features" which are a part of either a positive or negative movie review. The NLTK corpus movie\_reviews data set has the reviews, and they are labeled already as positive or negative. This means we can train and test with this data. First, let's wrangle our data.

import nltk

import random

from nltk.corpus import movie\_reviews

documents = [(list(movie\_reviews.words(fileid)), category)

for category in movie\_reviews.categories()

for fileid in movie\_reviews.fileids(category)]

random.shuffle(documents)

print(documents[1])

all\_words = []

for w in movie\_reviews.words():

all\_words.append(w.lower())

all\_words = nltk.FreqDist(all\_words)

print(all\_words.most\_common(15))

print(all\_words["stupid"])

It may take a moment to run this script, as the movie reviews dataset is somewhat large. Let's cover what is happening here.

After importing the data set we want, you see:

documents = [(list(movie\_reviews.words(fileid)), category)

for category in movie\_reviews.categories()

for fileid in movie\_reviews.fileids(category)]

Basically, in plain English, the above code is translated to: In each category (we have pos or neg), take all of the file IDs (each review has its own ID), then store the word\_tokenized version (a list of words) for the file ID, followed by the positive or negative label in one big list.

Next, we use random to shuffle our documents. This is because we're going to be training and testing. If we left them in order, chances are we'd train on all of the negatives, some positives, and then test only against positives. We don't want that, so we shuffle the data.

Then, just so you can see the data you are working with, we print out documents[1], which is a big list, where the first element is a list the words, and the 2nd element is the "pos" or "neg" label.

Next, we want to collect all words that we find, so we can have a massive list of typical words. From here, we can perform a frequency distribution, to then find out the most common words. As you will see, the most popular "words" are actually things like punctuation, "the," "a" and so on, but quickly we get to legitimate words. We intend to store a few thousand of the most popular words, so this shouldn't be a problem.

print(all\_words.most\_common(15))

The above gives you the 15 most common words. You can also find out how many occurences a word has by doing:

print(all\_words["stupid"])