Lab 7: Preprocessing Techniques in NLP Using NLTK package

In [1]:

```
#Import the necessary libraries
import nltk                                    # Python library for NLP
from nltk.corpus import twitter_samples    # sample Twitter dataset from NLTK
import matplotlib.pyplot as plt            # library for visualization
import random                              # pseudo-random number generator
import numpy as np
```

In [2]:

```
# downloads sample twitter dataset. execute the line below if running on a local ma
nltk.download('twitter_samples')
```

```
[nltk_data] Downloading package twitter_samples to
[nltk_data]     /home/yash/nltk_data...
[nltk_data]   Package twitter_samples is already up-to-date!
```

Out[2]:

```
True
```

We can load the text fields of the positive and negative tweets by using the module's `strings()` method like this:

In [3]:

```
# select the set of positive and negative tweets
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')
```

Next, we'll print a report with the number of positive and negative tweets. It is also essential to know the data structure of the datasets

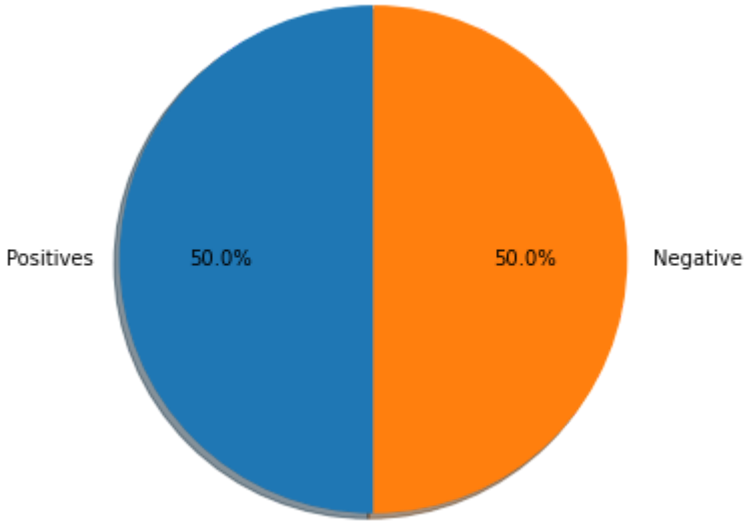
In [4]:

```
#TODO: Print the size of positive and negative tweets
print('Number of positive tweets: ', len(all_positive_tweets))
print('Number of negative tweets: ', len(all_negative_tweets))

#TODO:Print the type of positive and negative tweets, using type()
print('\nThe type of all_positive_tweets is: ', type(all_positive_tweets))
print('The type of a tweet entry is: ', type(all_negative_tweets))
```

```
Number of positive tweets:  5000
Number of negative tweets:  5000

The type of all_positive_tweets is:  <class 'list'>
The type of a tweet entry is:  <class 'list'>
```

In [5]:

```
#PLOT the positive and negative tweets in a pie-chart
# Declare a figure with a custom size
fig = plt.figure(figsize=(5, 5))

# labels for the two classes
labels = 'Positives', 'Negative'

# Sizes for each slide
sizes = [len(all_positive_tweets), len(all_negative_tweets)]

# Declare pie chart, where the slices will be ordered and plotted counter-clockwise
plt.pie(sizes, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)

# Equal aspect ratio ensures that pie is drawn as a circle.
plt.axis('equal')

# Display the chart
plt.show()
```



Looking at raw texts

Before anything else, we can print a couple of tweets from the dataset to see how they look. Understanding the data is responsible for 80% of the success or failure in data science projects. We can use this time to observe aspects we'd like to consider when preprocessing our data.

Below, you will print one random positive and one random negative tweet. We have added a color mark at the beginning of the string to further distinguish the two.

In [6]:

```
#TODO: Display a random tweet from positive and negative tweet.
#The size of positive and negative tweets are 5000 each.
#Generate a random number between 0 and 5000 using random.randint()
# print positive in greeen
print('\033[92m' + all_positive_tweets[420])

# print negative in red
print('\033[91m' + all_negative_tweets[69])
```

```
@applewriter yep, you *are* an expert on bisexuality :) Good luck!
@IzzyTailford that's true, I just want it soooooooner :(
```

Preprocess raw text for Sentiment analysis

In [7]:

```
# Our selected sample. Complex enough to exemplify each step
tweet = all_positive_tweets[2277]
print(tweet)
```

```
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #
favourites #happy #Friday off… https://t.co/3tfYom0N1i (https://t.co/3
tfYom0N1i)
```

In [8]:

```
# download the stopwords from NLTK
nltk.download('stopwords')
```

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

Out[8]:

```
True
```

In [9]:

```
#Import the necessary libraries
import re                                      # library for regular expression operati
import string                                  # for string operations

from nltk.corpus import stopwords              # module for stop words that come with N
from nltk.stem import PorterStemmer            # module for stemming
from nltk.tokenize import TweetTokenizer       # module for tokenizing strings
```

Remove hyperlinks, Twitter marks and styles

Since we have a Twitter dataset, we'd like to remove some substrings commonly used on the platform like the hashtag, retweet marks, and hyperlinks. We'll use the re (https://docs.python.org/3/library/re.html) library to perform regular expression operations on our tweet. We'll define our search pattern and use the `sub()` method to remove matches by substituting with an empty character (i.e. `''`)

In [10]:

```
print('\033[92m' + tweet)
print('\033[94m')

# remove old style retweet text "RT"
tweet2 = re.sub(r'^RT[\s]+', '', tweet)

# remove hyperlinks
tweet2 = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet2)

# remove hashtags
# only removing the hash # sign from the word
tweet2 = re.sub(r'#', '', tweet2)

print(tweet2)
```

```
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #
favourites #happy #Friday off… https://t.co/3tfYom0N1i (https://t.co/3
tfYom0N1i)

My beautiful sunflowers on a sunny Friday morning off :) sunflowers fa
vourites happy Friday off…
```

Tokenize the string

To tokenize means to split the strings into individual words without blanks or tabs. In this same step, we will also convert each word in the string to lower case. The tokenize (https://www.nltk.org/api/nltk.tokenize.html#module-nltk.tokenize.casual) module from NLTK allows us to do these easily:

In [11]:

```
print()
print('\033[92m' + tweet2)
print('\033[94m')

# instantiate tokenizer class
tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True,
                           reduce_len=True)

# tokenize tweets
tweet_tokens = tokenizer.tokenize(tweet2)

print()
print('Tokenized string:')
print(tweet_tokens)
```

```
My beautiful sunflowers on a sunny Friday morning off :) sunflowers fa
vourites happy Friday off…


Tokenized string:
['my', 'beautiful', 'sunflowers', 'on', 'a', 'sunny', 'friday', 'morni
ng', 'off', ':)', 'sunflowers', 'favourites', 'happy', 'friday', 'of
f', '…']
```

Remove stop words and punctuations

The next step is to remove stop words and punctuation. Stop words are words that don't add significant meaning to the text. You'll see the list provided by NLTK when you run the cells below.

In [12]:

```
#Import the english stop words list from NLTK
stopwords_english = stopwords.words('english')

print('Stop words\n')
print(stopwords_english)

print('\nPunctuation\n')
print(string.punctuation)
```

```
Stop words

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
"you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'y
ourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'her
s', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',
'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
t', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'b
e', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'di
d', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
s', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'again
st', 'between', 'into', 'through', 'during', 'before', 'after', 'abov
e', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov
er', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'wh
en', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'mor
e', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'ow
n', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'jus
t', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o',
're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'did
n', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't",
'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'must
n', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
dn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn',
"wouldn't"]

Punctuation

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

In [13]:

```
print()
print('\033[92m')
print(tweet_tokens)
print('\033[94m')

#TODO: Create the empty list to store the clean tweets after removing stopwords and
tweets_clean = []

#TODO: Remove stopwords and punctuation from the tweet_tokens
for word in tweet_tokens: # Go through every word in your tokens list
    if (word not in stopwords_english and  # remove stopwords
        word not in string.punctuation):  # remove punctuation
        #TODO: Append the clean word in the tweets_clean list
        tweets_clean.append(word)

print('removed stop words and punctuation:')
print(tweets_clean)
```

```
['my', 'beautiful', 'sunflowers', 'on', 'a', 'sunny', 'friday', 'morni
ng', 'off', ':)', 'sunflowers', 'favourites', 'happy', 'friday', 'of
f', '…']

removed stop words and punctuation:
['beautiful', 'sunflowers', 'sunny', 'friday', 'morning', ':)', 'sunfl
owers', 'favourites', 'happy', 'friday', '…']
```

Stemming

Stemming is the process of converting a word to its most general form, or stem. This helps in reducing the size of our vocabulary.

Consider the words:

- **learn**
- **learn**ing
- **learn**ed
- **learn**t

All these words are stemmed from its common root **learn**. However, in some cases, the stemming process produces words that are not correct spellings of the root word. For example, **happi** and **sunni**. That's because it chooses the most common stem for related words. For example, we can look at the set of words that comprises the different forms of happy:

- **happ**y
- **happi**ness
- **happi**er

We can see that the prefix **happi** is more commonly used. We cannot choose **happ** because it is the stem of unrelated words like **happen**.

NLTK has different modules for stemming and we will be using the [PorterStemmer (https://www.nltk.org/api/nltk.stem.html#module-nltk.stem.porter)](https://www.nltk.org/api/nltk.stem.html#module-nltk.stem.porter) module which uses the [Porter Stemming Algorithm (https://tartarus.org/martin/PorterStemmer/)](https://tartarus.org/martin/PorterStemmer/). Let's see how we can use it in the cell below.

In [14]:

```
print()
print('\033[92m')
print(tweets_clean)
print('\033[94m')

# Instantiate stemming class
stemmer = PorterStemmer()

#TODO: Create an empty list to store the stems
tweets_stem = []

#TODO: Itearate over the tweets_clean fot stemming
for word in tweets_clean:
    #TODO:call the stem function for stemming the word
    stem_word = stemmer.stem(word)               # stemming word
    #TODO:Append the stem_word in tweets_stem list
    tweets_stem.append(stem_word)

print('stemmed words:')
print(tweets_stem)
```

```
['beautiful', 'sunflowers', 'sunny', 'friday', 'morning', ':)', 'sunfl
owers', 'favourites', 'happy', 'friday', '…']

stemmed words:
['beauti', 'sunflow', 'sunni', 'friday', 'morn', ':)', 'sunflow', 'fav
ourit', 'happi', 'friday', '…']
```

process_tweet()

As shown above, preprocessing consists of multiple steps before you arrive at the final list of words. We will not ask you to replicate these however. You will use the function `process_tweet(tweet)` available below.

To obtain the same result as in the previous code cells, you will only need to call the function `process_tweet()`. Let's do that in the next cell.

In [15]:

```
def process_tweet(tweet):
    """Process tweet function.
    Input:
        tweet: a string containing a tweet
    Output:
        tweets_clean: a list of words containing the processed tweet

    """
    stemmer = PorterStemmer()
    stopwords_english = stopwords.words('english')
    # remove stock market tickers like $GE
    tweet = re.sub(r'\$\w*', '', tweet)
    # remove old style retweet text "RT"
    tweet = re.sub(r'^RT[\s]+', '', tweet)
    # remove hyperlinks
    tweet = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet)
    # remove hashtags
    # only removing the hash # sign from the word
    tweet = re.sub(r'#', '', tweet)
    # tokenize tweets
    tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True,
                               reduce_len=True)
    tweet_tokens = tokenizer.tokenize(tweet)

    tweets_clean = []
    for word in tweet_tokens:
        if (word not in stopwords_english and  # remove stopwords
                word not in string.punctuation):  # remove punctuation
            # tweets_clean.append(word)
            stem_word = stemmer.stem(word)  # stemming word
            tweets_clean.append(stem_word)

    return tweets_clean
```

In [16]:

```
# choose the same tweet
tweet = all_positive_tweets[2277]

print()
print('\033[92m')
print(tweet)
print('\033[94m')

#TODO: call the process_tweet function
tweets_stem = process_tweet(tweet)

print('preprocessed tweet:')
print(tweets_stem) # Print the result
```

```
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #
favourites #happy #Friday off… https://t.co/3tfYom0N1i (https://t.co/3
tfYom0N1i)

preprocessed tweet:
['beauti', 'sunflow', 'sunni', 'friday', 'morn', ':)', 'sunflow', 'fav
ourit', 'happi', 'friday', '…']
```

Building and Visualizing word frequencies

In this lab, we will focus on the `build_freqs()` helper function and visualizing a dataset fed into it. In our goal of tweet sentiment analysis, this function will build a dictionary where we can lookup how many times a word appears in the lists of positive or negative tweets. This will be very helpful when extracting the features of the dataset in the week's programming assignment. Let's see how this function is implemented under the hood in this notebook.

In [17]:

```
#TODO: Concatenate the lists, 1st part is the positive tweets followed by the negat
tweets = all_positive_tweets + all_negative_tweets

# let's see how many tweets we have
print("Number of tweets: ", len(tweets))
```

```
Number of tweets:  10000
```

In [18]:

```
# make a numpy array representing labels of the tweets
labels = np.append(np.ones((len(all_positive_tweets))), np.zeros((len(all_negative_
```

Dictionaries

In Python, a dictionary is a mutable and indexed collection. It stores items as key-value pairs and uses [hash tables (https://en.wikipedia.org/wiki/Hash_table)](https://en.wikipedia.org/wiki/Hash_table) underneath to allow practically constant time lookups. In NLP, dictionaries are essential because it enables fast retrieval of items or containment checks even with thousands of entries in the collection.

Definition

A dictionary in Python is declared using curly brackets. Look at the next example:

In [19]:

```
dictionary = {'key1': 1, 'key2': 2}
```

The former line defines a dictionary with two entries. Keys and values can be almost any type (with a few restriction on keys (https://docs.python.org/3/tutorial/datastructures.html#dictionaries)), and in this case, we used strings. We can also use floats, integers, tuples, etc.

Adding or editing entries

New entries can be inserted into dictionaries using square brackets. If the dictionary already contains the specified key, its value is overwritten.

In [20]:

```
# Add a new entry
dictionary['key3'] = -5

# Overwrite the value of key1
dictionary['key1'] = 0

print(dictionary)
```

```
{'key1': 0, 'key2': 2, 'key3': -5}
```

Accessing values and lookup keys

Performing dictionary lookups and retrieval are common tasks in NLP. There are two ways to do this:

- Using square bracket notation: This form is allowed if the lookup key is in the dictionary. It produces an error otherwise.
- Using the get() (https://docs.python.org/3/library/stdtypes.html#dict.get) method: This allows us to set a default value if the dictionary key does not exist.

Let us see these in action:

In [21]:

```
# Square bracket lookup when the key exist
print(dictionary['key2'])
```

```
2
```

However, if the key is missing, the operation produce an erro

In [22]:

```
# The output of this line is intended to produce a KeyError
print(dictionary['key8'])
```

```
---------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
 last)
<ipython-input-22-8d63520997fb> in <module>
      1 # The output of this line is intended to produce a KeyError
----> 2 print(dictionary['key8'])

KeyError: 'key8'
```

When using a square bracket lookup, it is common to use an if-else block to check for containment first (with the keyword `in`) before getting the item. On the other hand, you can use the `.get()` method if you want to set a default value when the key is not found. Let's compare these in the cells below:

This prints a value

if 'key1' in dictionary: print("item found: ", dictionary['key1']) else: print('key1 is not defined')

Same as what you get with get

print("item found: ", dictionary.get('key1', -1))

In [23]:

```
# This prints a message because the key is not found
if 'key7' in dictionary:
    print(dictionary['key7'])
else:
    print('key does not exist!')

# This prints -1 because the key is not found and we set the default to -1
print(dictionary.get('key7', -1))
```

```
key does not exist!
-1
```

Word frequency dictionary

In [24]:

```
def build_freqs(tweets, ys):
    """Build frequencies.
    Input:
        tweets: a list of tweets
        ys: an m x 1 array with the sentiment label of each tweet
            (either 0 or 1)
    Output:
        freqs: a dictionary mapping each (word, sentiment) pair to its
        frequency
    """
    # Convert np array to list since zip needs an iterable.
    # The squeeze is necessary or the list ends up with one element.
    # Also note that this is just a NOP if ys is already a list.
    yslist = np.squeeze(ys).tolist()

    # Start with an empty dictionary and populate it by looping over all tweets
    # and over all processed words in each tweet.
    #TODO:Create the empty dictionary
    freqs = {}
    for y, tweet in zip(yslist, tweets):
        #TODO: Iterate over all the words returned by calling the process_tweet() fu
        for word in process_tweet(tweet):
            pair = (word, y)
            #TODO: If pair matches in the dictionary, then increment the count of co
            if pair in freqs:
                freqs[pair] += 1
            else:
                #TODO: If pair does not matches in the dictionary, then set the coun
                freqs[pair] = 1

    #TODO: Return the dictionary
    return freqs
```

In [25]:

```
#TODO: Call the build_freqs function to create frequency dictionary based on tweets
freqs = build_freqs(tweets, labels)

#TODO: Display the data type of freqs
print(f'type(freqs) = {freqs}')

#TODO: Display the length of the dictionary
print(f'len(freqs) = {freqs}')
```

```
type(freqs) = {('followfriday', 1.0): 25, ('top', 1.0): 32, ('enga
g', 1.0): 7, ('member', 1.0): 16, ('commun', 1.0): 33, ('week', 1.
0): 83, (':)', 1.0): 3568, ('hey', 1.0): 76, ('jame', 1.0): 7, ('od
d', 1.0): 2, (':/', 1.0): 5, ('pleas', 1.0): 97, ('call', 1.0): 37,
('contact', 1.0): 7, ('centr', 1.0): 2, ('02392441234', 1.0): 1, ('a
bl', 1.0): 8, ('assist', 1.0): 1, ('mani', 1.0): 33, ('thank', 1.0):
620, ('listen', 1.0): 16, ('last', 1.0): 47, ('night', 1.0): 68, ('b
leed', 1.0): 2, ('amaz', 1.0): 51, ('track', 1.0): 5, ('scotland',
1.0): 2, ('congrat', 1.0): 21, ('yeaaah', 1.0): 1, ('yipppi', 1.0):
1, ('accnt', 1.0): 2, ('verifi', 1.0): 2, ('rqst', 1.0): 1, ('succee
d', 1.0): 1, ('got', 1.0): 69, ('blue', 1.0): 9, ('tick', 1.0): 1,
('mark', 1.0): 1, ('fb', 1.0): 6, ('profil', 1.0): 2, ('15', 1.0):
5, ('day', 1.0): 246, ('one', 1.0): 129, ('irresist', 1.0): 2, ('fli
pkartfashionfriday', 1.0): 17, ('like', 1.0): 233, ('keep', 1.0): 6
8, ('love', 1.0): 400, ('custom', 1.0): 4, ('wait', 1.0): 70, ('lon
g', 1.0): 36, ('hope', 1.0): 141, ('enjoy', 1.0): 75, ('happi', 1.
0): 211, ('friday', 1.0): 116, ('lwwf', 1.0): 1, ('second', 1.0): 1
0, ('thought', 1.0): 29, ('’', 1.0): 21, ('enough', 1.0): 18, ('tim
e', 1.0): 127, ('dd', 1.0): 1, ('new', 1.0): 143, ('short', 1.0): 7,
```

In [26]:

```
#TODO: prinf all the key-value pair of frequency dictionary
freqs
```

Out[26]:

```
{('followfriday', 1.0): 25,
 ('top', 1.0): 32,
 ('engag', 1.0): 7,
 ('member', 1.0): 16,
 ('commun', 1.0): 33,
 ('week', 1.0): 83,
 (':)', 1.0): 3568,
 ('hey', 1.0): 76,
 ('jame', 1.0): 7,
 ('odd', 1.0): 2,
 (':/', 1.0): 5,
 ('pleas', 1.0): 97,
 ('call', 1.0): 37,
 ('contact', 1.0): 7,
 ('centr', 1.0): 2,
 ('02392441234', 1.0): 1,
 ('abl', 1.0): 8,
 ('assist', 1.0): 1,
```

Table of word counts

In [27]:

```
# select some words to appear in the report. we will assume that each word is uniqu
keys = ['happi', 'merri', 'nice', 'good', 'bad', 'sad', 'mad', 'best', 'pretti',
        '❤', ':)', ':(', '😒', '😬', '😄', '😍', '♛',
        'song', 'idea', 'power', 'play', 'magnific']



#TODO: Create the empty list as list representing our table of word counts, where
#each element consist of a sublist with this pattern: [<word>, <positive_count>, <n
data = []

#TODO:Iterate over each word in keys
for word in keys:

    # initialize positive and negative counts
    pos = 0
    neg = 0

    # retrieve number of positive counts
    if (word, 1) in freqs:
        pos = freqs[(word, 1)]

    # retrieve number of negative counts
    if (word, 0) in freqs:
        neg = freqs[(word, 0)]

    # append the word counts to the table
    data.append([word, pos, neg])

data
```

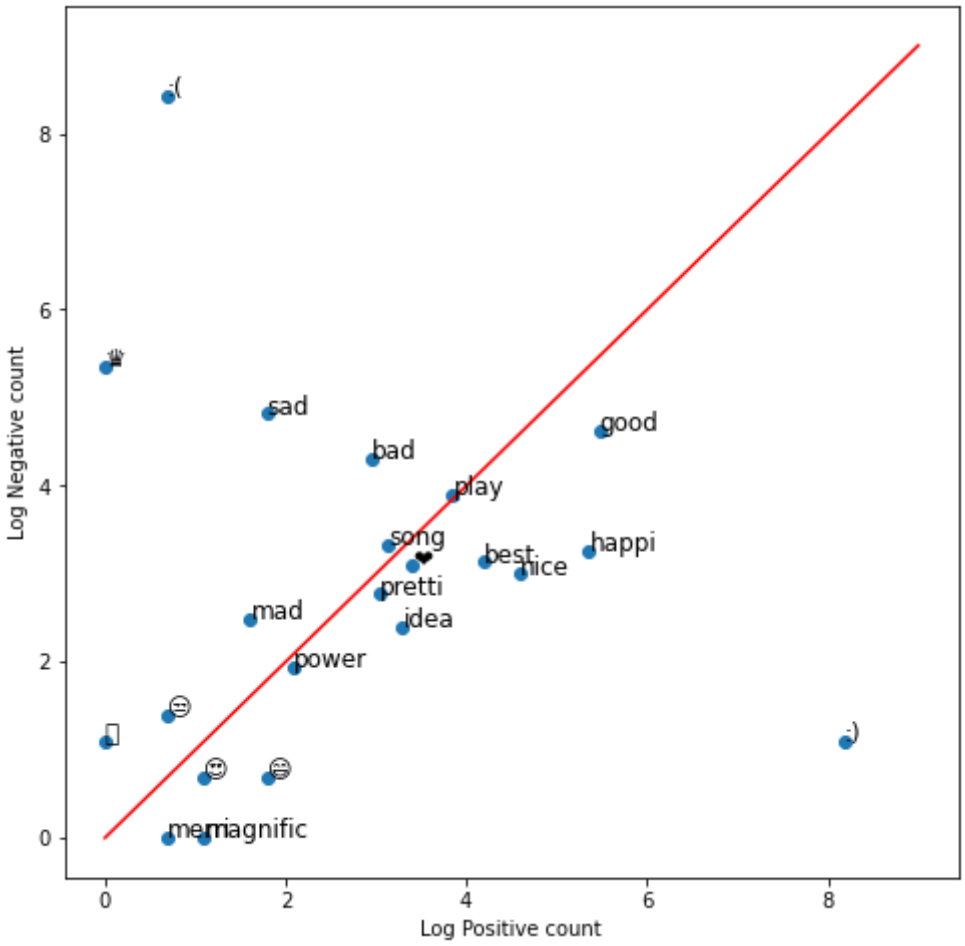
Out[27]:

```
[['happi', 211, 25],
 ['merri', 1, 0],
 ['nice', 98, 19],
 ['good', 238, 101],
 ['bad', 18, 73],
 ['sad', 5, 123],
 ['mad', 4, 11],
 ['best', 65, 22],
 ['pretti', 20, 15],
 ['❤', 29, 21],
 [':)', 3568, 2],
 [':(', 1, 4571],
 ['😒', 1, 3],
 ['😬', 0, 2],
 ['😄', 5, 1],
 ['😍', 2, 1],
 ['♛', 0, 210],
 ['song', 22, 27],
 ['idea', 26, 10],
 ['power', 7, 6],
 ['play', 46, 48],
 ['magnific', 2, 0]]
```

In [28]:

```
fig, ax = plt.subplots(figsize = (8, 8))

# convert positive raw counts to logarithmic scale. we add 1 to avoid log(0)
x = np.log([x[1] + 1 for x in data])

# do the same for the negative counts
y = np.log([x[2] + 1 for x in data])

# Plot a dot for each pair of words
ax.scatter(x, y)

# assign axis labels
plt.xlabel("Log Positive count")
plt.ylabel("Log Negative count")

# Add the word as the label at the same position as you added the points just befor
for i in range(0, len(data)):
    ax.annotate(data[i][0], (x[i], y[i]), fontsize=12)

ax.plot([0, 9], [0, 9], color = 'red') # Plot the red line that divides the 2 areas
plt.show()
```

```
/home/yash/anaconda3/lib/python3.8/site-packages/matplotlib/backends/b
ackend_agg.py:238: RuntimeWarning: Glyph 128556 missing from current f
ont.
  font.set_text(s, 0.0, flags=flags)
/home/yash/anaconda3/lib/python3.8/site-packages/matplotlib/backends/b
ackend_agg.py:201: RuntimeWarning: Glyph 128556 missing from current f
ont.
  font.set_text(s, 0, flags=flags)
```


Log Negative count
Log Positive count
0
2
4
6
8
sad
good
bad
play
song
best
nice
happi
pretti
mad
idea
power
:)
magnific